Technische Universität München
TUM School of Computation, Information and Technology

Efficient and High Performance Algorithms for Context Modeling in Learning-based Image Compression

Ahmet Burakhan Koyuncu

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr.-Ing. Klaus Diepold
Prüfende der Dissertation:	1.	Prof. Dr.-Ing. Eckehard Steinbach
	2.	Prof. Dr.-Ing. André Kaup
	3.	Prof. Dr.-Ing. João Duarte Ascenso

Die Dissertation wurde am 16.12.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 01.05.2025 angenommen.

Abstract

The exponential growth of digital imagery—propelled by advances in camera technology and the ubiquity of social media—has led to an unprecedented demand for efficient image compression techniques capable of reducing data sizes without sacrificing quality. To cope with this demand, learned image compression (LIC) methods have emerged as a promising alternative, leveraging deep learning to surpass traditional compression approaches. However, a critical challenge remains: developing accurate and efficient entropy models that effectively capture the intricate dependencies within compressed representations without incurring prohibitive computational costs. This thesis addresses this challenge by presenting a series of advancements in context modeling of entropy models, introducing two key architectures—**Contextformer** and **Efficient Contextformer**—that leverage the flexibility and power of transformer-based models to significantly enhance compression performance and computational efficiency. These contributions lay the foundation for novel context models aimed at achieving high compression ratios and low computational overhead.

The **Contextformer** innovates upon widely adopted context models by employing a Spatio-Channel Multi-Head Attention (SC-MHA) mechanism. By generalizing the traditional attention mechanism to jointly process spatial and channel dimensions, the Contextformer dynamically captures complex spatial and inter-channel dependencies in the latent space with a content-adaptive approach. Additionally, it includes several optimizations that accelerate encoding and decoding while maintaining precision. These innovations significantly enhance entropy modeling, achieving up to 11% bitrate savings over the state-of-the-art Versatile Video Coding (VVC) Test Model (VTM) 16.2. Extensive evaluations on standard datasets such as Kodak, CLIC, and Tecnick demonstrate that the Contextformer outperforms existing LIC models in terms of Peak Signal-to-Noise Ratio (PSNR) and Multi-Scale Structural Similarity Index Measure (MS-SSIM).

The **Efficient Contextformer** (eContextformer) builds upon the foundations of the Contextformer, prioritizing computational efficiency while preserving performance to tackle the computational demands inherent in transformer-based models. The eContextformer introduces Spatio-Channel Window and Shifted-Window Attention (SC-W-MHA and SC-SW-MHA) mechanisms, effectively approximating global spatio-channel attention with drastically reduced complexity. By integrating parallel context modeling techniques—including patch-wise, checkered, and channel-wise grouping—it reduces the number of autoregressive steps required, enabling efficient processing. The eContextformer employs several additional optimizations, such as dynamic attention scaling and caching mechanisms, resulting

in a remarkable 145x reduction in model complexity and a 210x increase in decoding speed compared to the Contextformer. Additionally, its lightweight design enables the use of online rate-distortion optimization (oRDO), further refining compression results on a per-image basis and achieving bitrate savings of up to 17% against VTM 16.2 on standard benchmark datasets.

In summary, this thesis contributes a robust framework for high-performance, efficient LIC, validated across diverse datasets and complexity levels, offering a scalable solution that balances visual fidelity and resource constraints. The proposed models not only establish new benchmarks in LIC performance but also introduce scalable transformer-based techniques that push the boundaries of context modeling in compression, addressing the pressing need to manage the ever-expanding volume of visual data in the digital age.

Kurzfassung

Das exponentielle Wachstum digitaler Bilder, angetrieben durch Fortschritte in der Kameratechnologie und die Allgegenwart sozialer Medien, hat zu einer signifikanten Nachfrage nach effizienten Bildkompressionstechniken geführt. Dabei ist es wichtig, dass die Datenmengen reduzieren werden können, ohne die Qualität zu beeinträchtigen. Um dieser Nachfrage gerecht zu werden, sind gelernte Bildkompressionsmethoden (LICs) als vielversprechende Alternative entstanden, die Deep Learning nutzen, um traditionelle Kompressionsansätze zu übertreffen. Allerdings bleibt eine kritische Herausforderung bestehen: die Entwicklung genauer und effizienter Entropiemodelle, die komplexe Abhängigkeiten innerhalb komprimierter Repräsentationen effektiv erfassen, ohne prohibitive Rechenkosten zu verursachen. Diese Arbeit adressiert dieses Problem, indem sie eine Reihe von Fortschritten in der Kontextmodellierung von Entropiemodellen präsentiert und zwei Schlüsselarchitekturen einführt: den **Contextformer** und den **Efficient Contextformer**. Diese nutzen die Flexibilität und Leistungsfähigkeit transformerbasierter Modelle, um die Kompressionsleistung und Recheneffizienz signifikant zu verbessern. Die Beiträge dieser Arbeit sind Basis für die Entwicklung neuartiger Kontextmodelle, die darauf abzielen, hohe Kompressionsraten bei geringem Rechenaufwand zu erreichen.

Der **Contextformer** verbessert weit verbreitete Kontextmodelle, indem er einen Spatio-Channel Multi-Head Attention (SC-MHA)-Mechanismus verwendet. Durch die Verallgemeinerung des traditionellen Aufmerksamkeitsmechanismus zur gemeinsamen Verarbeitung von Raum- und Kanaldimensionen erfasst der Contextformer dynamisch komplexe räumliche und kanalübergreifende Abhängigkeiten im latenten Raum mit einem inhaltsadaptiven Ansatz. Darüber hinaus enthält er mehrere Optimierungen, die das Kodieren und Dekodieren beschleunigen, ohne die Präzision zu beeinträchtigen. Diese Innovationen verbessern die Entropiemodellierung erheblich und erreichen bis zu 11 % Bitratenersparnis gegenüber dem modernen Versatile Video Coding (VVC) Test Model (VTM) 16.2. Umfangreiche Bewertungen auf Standard-Datensätzen wie Kodak, CLIC und Tecnick zeigen, dass der Contextformer bestehende LIC-Modelle in Bezug auf das Spitzen-Signal-Rausch-Verhältnis (PSNR) und den Mehrskalen-Index struktureller Ähnlichkeit (MS-SSIM) übertrifft.

Der **Efficient Contextformer** (eContextformer) baut auf den Grundlagen des Contextformers auf und priorisiert die Recheneffizienz, während er die Leistung beibehält, um den rechnerischen Anforderungen transformerbasierter Modelle gerecht zu werden. Der eContextformer führt Spatio-Channel Window und Shifted-Window Attention (SC-W-MHA und SC-SW-MHA)-Mechanismen ein, welche die globale räumliche und kanalübergreifende

Aufmerksamkeit mit drastisch reduzierter Komplexität effektiv annähern. Durch die Integration paralleler Kontextmodellierungstechniken, einschließlich patchweiser, schachbrettartiger und kanalweiser Gruppierung, reduziert er die Anzahl der erforderlichen autoregressiven Schritte und ermöglicht eine effiziente Verarbeitung. Der eContextformer verwendet mehrere zusätzliche Optimierungen wie dynamische Aufmerksamkeits-Skalierung und Caching-Mechanismen, was zu einer bemerkenswerten 145-fachen Reduktion der Modellkomplexität und einer 210-fachen Erhöhung der Dekodiergeschwindigkeit im Vergleich zum Contextformer führt. Darüber hinaus ermöglicht sein leichtgewichtiges Design die Verwendung von Online-Rate-Distortion-Optimierung (oRDO), welche die Kompressionsergebnisse pro Bild weiter verbessert und durchschnittliche Bitrateneinsparungen von bis zu 17 % gegenüber VTM 16.2 auf Benchmark-Datensätzen erzielt.

Zusammenfassend liefert diese Arbeit ein robustes Framework für leistungsstarke, effiziente LICs, validiert über verschiedene Datensätze und Komplexitätsstufen. Sie bietet eine skalierbare Lösung, die visuelle Qualität und Ressourcenbeschränkungen ausbalanciert. Die vorgeschlagenen Modelle setzen nicht nur neue Maßstäbe in der LIC-Leistung, sondern führen auch skalierbare, transformerbasierte Techniken ein, die die Grenzen der Kontextmodellierung in der Kompression erweitern und dem dringenden Bedarf gerecht werden, das ständig wachsende Volumen an visuellen Daten im digitalen Zeitalter zu bewältigen.

Acknowledgements

The research presented in this thesis was conducted during my time as a PhD candidate at the Chair of Media Technology (LMT) at Technical University of Munich and the Audiovisual Technology Lab (AVTL) at Huawei Munich Research Center. I would like to express my deepest gratitude to those who have supported me throughout this challenging journey.

First and foremost, I am immensely grateful to my supervisor, Prof. Dr.-Ing. Eckehard Steinbach, for his exceptional guidance and unwavering support. Despite his busy schedule, he always provided fruitful discussions and insightful feedback that helped me sharpen my thinking and elevate my work to a higher level. His mentorship has been invaluable, and his encouragement has inspired me to overcome numerous obstacles.

I would also like to thank Dr. Atanas Boev for his immense support and motivation. His positive outlook and encouragement provided me with the necessary balance and inspiration during difficult times. Moreover, his meticulous proofreading of all my publications and this thesis significantly improved the quality of my writing, and his guidance greatly enhanced my presentation skills.

Furthermore, I would like to thank Prof. Dr.-Ing. André Kaup and Prof. Dr.-Ing. João Ascenso for agreeing to become the second and third examiners, and Prof. Dr.-Ing. Klaus Diepold for chairing the thesis committee.

I am fortunate to have had wonderful colleagues at both the LMT and the AVTL, whose camaraderie made the long days more enjoyable. Their friendship and shared moments brought light to my work environment.

My heartfelt appreciation goes to my friends whose encouragement and continuous mental support were crucial in navigating the complexities of this endeavor.

I am deeply indebted to my parents and family for their unconditional love and support, both emotionally and financially. They have always believed in me and have been instrumental in shaping the person I am today. I also wish to honor the memory of my grandmother, Saadet Pınarcı, who played a significant role in my upbringing. Her passing during the pandemic was a profound loss, and her influence continues to guide me.

Above all, I want to express my deepest love and gratitude to my wife, Ülkü Karaduman Koyuncu, and our two cats, Kafka and Mocha. Without their unwavering support, patience, and love, I would not have been able to endure the challenges of these past years. My wife has been my rock, tirelessly working to make my life easier and standing by me every step of the way. She is everything to me.

Contents

Notation

Abbreviations

Abbreviation	Description	Definition
2D	two-dimensional	page 8
3D	three-dimensional	page 8
3D-WPP	3D Wavefront Coding	page 61
ANN, NN	Artificial Neural Networks	page 16
AGM	Asymmetric Gaussian Model	page 27
AGMM	Asymmetric Gaussian Mixture Model	page 27
API	Application Programming Interface	page 44
BD-Rate	Bjøntegaard Delta-Rate	page 43
BDS	Batched Dynamic Sequence processing	page 60
BPG	Better Portable Graphics	page 8
Bpp	Bits per pixel	page 41
CABAC	Context-based Binary Arithmetic Coding	page 14
CAVLC	Context-Adaptive Variable-Length Coding	page 12
CfO	Channel-first-Order	page 56
ChARM	Channel-wise autoregressive context model	page 50
CLIC	Challenge on Learned Image Compression	page 32
CLIC-M	CLIC Mobile dataset	page 32
CLIC-P	CLIC Professional dataset	page 32
CNN	Convolutional Neural Network	page 3
COCO	Microsoft Common Objects in Context	page 33
CPU	Central Processing Unit	page 44
CTU	Coding Tree Unit	page 13
CUDA	Compute Unified Device Architecture	page 44
CV	Computer Vision	page 21
DCT	Discrete Cosine Transform	page 13
DS	Dynamic Sequence processing	page 59
DST	Discrete Sine Transform	page 13
DWT	Discrete Wavelet Transforms	page 13
EGR	Efficient coding Group Rearrangement	page 91
FLOPS	FLoating-point Operations Per Second	page 45
FSIM	Feature Similarity Index Measure	page 41
f.t.	Fine-tuned	page 101
GDN	Generalized Divisive Normalization	page 16
GELU	Gaussian Error Linear Unit	page 23
GIF	Graphics Interchange Format	page 8
GLCM	Gray-Level Co-occurrence Matrix	page 34
GMM	Gaussian Mixture Model	page 27
GPU	Graphics Processing Unit	page 44

Abbreviation	Description	Definition
HEIF	High Efficiency Image File Format	page 2
HEVC	High-Efficiency Video Coding	page 2
HDR	High Dynamic Range	page 1
HSV	Hue, Saturation, Value	page 41
IGDN	Inverse Generalized Divisive Normalization	page 16
i.i.d.	independent and identically distributed	page 12
ISB	Information Sharing Block	page 83
ITU	International Telecommunication Union	page 9
IW-MS-SSIM	Information content Weighted MS-SSIM	page 41
JPEG	Joint Photographic Experts Group	page 8
kMACpp	kilo MAC operations per pixel	page 45
LIC	Learned (lossy) Image Compression	page 2
MAC	Multiply-ACccumulate	page 45
MACpp	MAC operations per pixel	page 45
MDLC	Minimum Description Length Clustering	page 34
MHA	Multi-Head Attention	page 21
MLP	Multi-Layer Perceptron	page 22
MP	Mega Pixel	page 1
MSE	Mean-Squared Error	page 41
MS-SSIM	Multi-Scale SSIM	page 41
NLP	Natural Language Processing	page 21
NLPD	Normalized Laplacian Pyramid Distance	page 41
NNVC	Neural Network-based Video Coding	page 2
NPU	Neural Processing Unit	page 44
OOM	Out of Memory	page 71
oRDO	online Rate-Distortion Optimization	page 28
PCM	Parallel Context Model	page 52
PNG	Portable Network Graphics	page 8
PSNR	Peak Signal-to-Noise Ratio	page 41
RAB	Residual Attention Block	page 53
RD	Rate-Distortion	page 41
RDO	Rate-Distortion Optimization	page 14
ReLU	Rectified Linear Unit	page 23
ResBlock	Residual Block	page 53
RGB	Red, Green, Blue	page 8
RNN	Recurrent Neural Network	page 83
S-MHA	Spatial-MHA	page 55
SCM	Serial Context Model	page 52
SC-MHA	Spatio-Channel MHA	page 56
SCS	Skipping intermediate Channel Segments	page 60
SFG	Skipping First coding Group	page 91
SfO	Spatial-first-Order	page 56
SGA	Stochastic Gumbel Annealing	page 97
SI	Spatial Information	page 34
SIMD	Single Instruction, Multiple Data	page 71
sRAB	simplified RAB	page 53
SSIM	Structural Similarity Index Measure	page 41
STE	Straight-Through Estimator	page 17
SwinT	Hierarchical Vision Transformer using Shifted Windows	page 24
SC-(S)W-MHA	spatio-channel (shifted-)window MHA	page 90
TCQ	Trellis-Coded Quantization	page 14
TIC	Traditional (lossy) Image Compression	page 7

Abbreviation	Description	Definition
TPE	Tree-Structured Parzen Estimator	page 107
TPU	Tensor Processing Unit	page 44
VIF	Visual Information Fidelity	page 41
ViT	Vision Transformer	page 24
VLC	Variable-Length Coding	page 12
VMAF	Video Multimethod Assessment Fusion	page 41
VVC	Versatile Video Coding	page 2
WPP	Wavefront Parallel Processing	page 61
YC_BC_R	Luminance, Blue-difference Chroma, Red-difference Chroma	page 9

Scalars and vectors

x	Scalar
$\boldsymbol{x}$	Tensor
$\mathcal{X}$	Set

Subscripts and superscripts

$\hat{x}$	Reconstructed or quantized value of x
$\bar{x}$	Dequantized value of x
$\tilde{x}$	Noised value of x

Symbols

α_0	The initial learning rate in oRDO
B	Bit depth
C	Number of channels of a tensor or image
C_M	Number of channels in the bottleneck of the autoencoder
C_N	Number of channels in the transforms of the autoencoder
$\mathbb{C}$	Arithmetic complexity in terms of MAC
$\mathbb{C}_{\text{conv}}$	Arithmetic complexity of a convolutional layer in terms of MAC
$\mathbb{C}_{\text{ctx}}$	Arithmetic complexity of a context model in terms of MAC
$\mathbb{C}_{\text{trans}}$	Arithmetic complexity of a transformer layer in terms of MAC
d_e	Embedding layer size
d_{mlp}	MLP size of a transformer layer
D	Distortion
$g_a, \boldsymbol{\psi_a}$	Analysis transform network and its parameters
$g_c, \boldsymbol{\theta_c}$	Context model network and its parameters
$g_{ep}, \boldsymbol{\theta_{ep}}$	Entropy parameters network and its parameters
$g_s, \boldsymbol{\psi_s}$	Synthesis transform network and its parameters
γ	The decay rate of the learning rate in oRDO
h	Number of heads
$h_a, \boldsymbol{\theta_a}$	Hyperprior analysis transform network and its parameters
$h_s, \boldsymbol{\theta_s}$	Hyperprior synthesis transform network and its parameters
H	Height of a tensor or image
$\mathcal{H}$	Shannon entropy
$\mathcal{H}_C$	Color Variety
$\mathcal{H}_{GLCM}$	Average Entropy of Gray-Level Co-occurrence Matrix
k_m	Number of mixtures
K	Spatial kernel size
L	Number of transformer layers
λ	Lagrangian multiplier controlling the trade-off between bitrate and distortion
λ_{MSE}	Lagrangian multiplier of MSE trained models controlling the trade-off between bitrate and distortion
$\lambda_{\text{MS-SSIM}}$	Lagrangian multiplier of MS-SSIM trained models controlling the trade-off between bitrate and distortion
N_{cs}	Number of channel segments
N_{oRDO}	The number of iterations in oRDO
N_w	Number of elements within a window
$\boldsymbol{\Omega}$	Output of the context model
$\boldsymbol{p_e}$	Positional encodings
$\boldsymbol{\Psi}$	Output of the hyperprior's synthesis transform

Q	Priority queue
$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$	Query, Key, Value
R	Bitrate
$\mathbb{R}$	Set of reel numbers
S	Sequence length
$\mathcal{S}_c$	Coding strategy
SI_{mean}	Mean Spatial Information
S_{MDLC}	Minimum Description Length Clustering score
W	Width of a tensor or image

Mathematical Operations

$\lvert x \rvert$	Absolute value of scalar x
$\approx$	Approximately equal
$\arg\min_{x \in \mathcal{X}} f(x)$	Argument of the minimum
$\boldsymbol{x} +\!\!+ \boldsymbol{y}$	Concatenation of the tensors $\boldsymbol{x}$ and $\boldsymbol{y}$ along their last dimension
$p(x \mid y)$	Conditional probability of x given y
$Q^{-1}(\cdot)$	Dequantization
$\dim(\boldsymbol{x})$	Dimensionality of $\boldsymbol{x}$
$\boldsymbol{x} \oplus \boldsymbol{y}$	Element-wise addition of tensors $\boldsymbol{x}$ and $\boldsymbol{y}$
$\boldsymbol{x} \odot \boldsymbol{y}$	Element-wise multiplication (Hadamard product) of tensors $\boldsymbol{x}$ and $\boldsymbol{y}$
$\lVert \boldsymbol{x} \rVert$	Euclidean norm of vector $\boldsymbol{x}$
$f(\cdot)$	Function
$f(\cdot\,; \boldsymbol{\theta})$	Function parameterized by the parameters $\boldsymbol{\theta}$
$\boldsymbol{x}_{a,b,c}$	Indexing the first three dimensions of N dimensional tensor $\boldsymbol{x}$ with the indices a, b and c
$f^{-1}(\cdot)$	Inverse function
$Q(\cdot)$	Quantization
$\lfloor \cdot \rfloor$	Rounding down toward negative infinity
$\lfloor \cdot \rceil$	Rounding to nearest integer value
$\lceil \cdot \rceil$	Rounding up toward positive infinity
$\boldsymbol{x}_{a \ldots b, c \ldots d, e \ldots f}$	Slicing a N dimensional tensor $\boldsymbol{x}$ along its first three dimensions with the ranges of $[a, b]$, $[c, d]$ and $[e, f]$
$x \ll y$	x is orders of magnitude lesser than y
$x \gg y$	x is orders of magnitude greater than y
$x \sim y$	x is proportional to y
$x \sim \mathcal{N}(\mu, \sigma^2)$	x is sampled from a Gaussian distribution with the mean μ and variance σ^2
$x \sim \mathcal{U}(a, b)$	x is sampled from an uniform distribution bounded by a and b
$x \mod y$	x modulo y

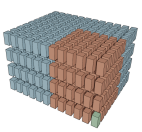
Chapter 1

Introduction

With the increasing accessibility of smartphones and digital cameras, the number of images captured and shared online continues to grow at an exponential rate. Recent statistics [13] reveal that approximately 1.94 trillion photos are taken globally every year, with over 136 billion images indexed by Google Image Search as of 2024. Social media platforms see approximately 14 billion images shared daily, which is expected to continue rising as visual content dominates online communication. This explosion in image creation and sharing places immense pressure on storage and transmission systems, highlighting the critical need for scalable and efficient solutions to manage vast quantities of visual data.

The improving quality of smartphone cameras has further intensified this challenge. Modern cameras typically have sensors ranging from 10 to 20 megapixels (MP); some devices now feature resolutions as high as 100 MP. For instance, a 24-bit color image captured by a 10 MP camera requires approximately 30 megabytes of storage in its uncompressed, raw format. If images shared on social media were transmitted in raw form, the daily data load would easily exceed 400 petabytes—equivalent to 400,000 terabytes. Additionally, technological advancements enhancing color gamut and visual vividness, such as High Dynamic Range (HDR) imaging, require up to 48 bits per pixel to represent an image [14], [15], which further inflates the data footprint. These trends highlight the pressing need for advanced compression technologies to reduce both storage and bandwidth demands while preserving high image quality.

Traditional lossy image compression algorithms, such as Joint Photographic Experts Group (JPEG) [16], JPEG 2000 [17], and WebP [18], were developed to address the growing need for efficient image storage and transmission. These algorithms work by reducing spatial, statistical, and psychovisual redundancies in images through a combination of block-based transform coding, quantization, and entropy coding, aiming to lower the bitrate—i.e., the number of bits needed to represent the image—while maintaining the highest possible reconstruction quality. For instance, JPEG [16] decomposes the image into frequency components, which are quantized to eliminate perceptually less important components. These quantized components are then encoded using lossless entropy coding techniques, such as Huffman coding [19], into a compact bit representation by reducing the statistical dependencies within them. Over time, image and video compression standards have often improved in parallel. Many image compression algorithms borrow principles from video coding al-



gorithms such as adaptive block-partitioning [20], [21] and predictive coding [22], [23] since video coding standards like High-Efficiency Video Coding (HEVC) [24] and its successor, Versatile Video Coding (VVC) [25], use similar techniques for both still images and video frames. In fact, modern smartphones from major manufacturers, including Android and Apple devices, now support the High Efficiency Image File Format (HEIF) [26], which employs the still image (intra-frame) coding framework of HEVC [24] to deliver more efficient image compression. While traditional codecs have achieved significant improvements in compression efficiency, they face limitations in handling increasingly complex visual data and adapting to diverse content, especially as image sizes and qualities continue to rise. Furthermore, their reliance on hand-engineered coding decisions, such as a fixed set of linear transforms and quantization parameters, restricts their flexibility and efficiency in dynamically changing environments. Moreover, recent algorithms can suffer from very long encoding times since they search for optimal parameters from a set of different coding decisions during encoding.

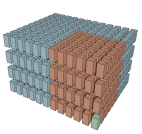
Driven by the availability of powerful computing resources and vast amounts of training data, the rapid advancement of deep learning techniques has led to significant breakthroughs in fields such as computer vision, autonomous driving, and natural language processing [27]–[30]. In recent years, this trend has extended into image and video compression, where learned image compression (LIC) methods [1]–[4], [31]–[76] have emerged as a promising alternative to traditional methods [77]. Instead of relying on fixed, hand-engineered transforms and entropy coding schemes used in traditional codecs, LIC employs neural network models, particularly autoencoders [78], to learn complex non-linear transforms [36] that map input images into compact latent representations. These latent variables are then compressed using entropy models, which learn the probability distribution of each latent element, allowing for more efficient compression by better estimating the dependencies within the data. A critical component of LIC is its ability to jointly optimize both bitrate and reconstruction quality by learning powerful transforms and entropy models end-to-end during training, unlike traditional methods where these are optimized separately. This flexibility has allowed LIC to surpass the compression performance of traditional codecs like JPEG [16], and even challenge HEVC [24] and VVC [25] in intra-frame coding scenarios. Notably, these deep learning-driven approaches are not limited to academic research; standardization bodies have recognized their potential. Initiatives such as JPEG AI [79] for image coding and Neural Network-based Video Coding (NNVC) [80] for video coding explore the integration of deep learning into future compression standards, further highlighting the impact of learned methods on the field.

An accurate entropy model is crucial in achieving high performance in the LIC framework. Two key techniques are employed in entropy modeling [36]—forward adaptation and backward adaptation. Forward adaptation, often implemented via a hyperprior network [33], leverages a side-channel of auxiliary data to model the distribution of latent elements. In contrast, backward adaptation, commonly referred to as context modeling [34], estimates the distribution of the current latent element by utilizing contextual information from previously decoded elements with an autoregressive process. This makes context mod-

eling particularly efficient as it enhances the entropy model without requiring additional signaling overhead. Given its effectiveness, context modeling has become a critical part of LIC research, exploring a variety of context model architectures to further improve compression performance and efficiency. Ideally, a context model processes each latent element sequentially and builds contextual information for them based on all previous elements. However, practically, this is infeasible due to the increased computational cost, and it might even result in reduced performance due to noise introduced by irrelevant relations added to the computation. Therefore, one research track focuses on high-performance context models by investigating what kind of contextual information in latent space optimizes the accuracy of probability estimation. These mechanisms include models exploiting spatial local dependencies [34], [37]–[40], [47], [49] and inter-channel dependencies [1], [2], [35], [42], non-local models capturing long-range dependencies [44], [45], and dynamic models leveraging content-adaptive modeling [1], [2], [46], [59], [61]–[63]. Another research track focuses on more efficient context models achieved by decreasing the model's latency due to the serial processing of autoregressive operations, which increases significantly with the image resolution. For more efficient context modeling, studies have introduced parallelized context modeling approaches such as grouped latent coding. Rather than processing each latent element in a strictly sequential order, the elements within a group are processed in parallel, significantly accelerating the context modeling process. While grouped latent coding reduces processing time and enhances parallelism, it often comes at the cost of slightly reduced compression performance, as inter-group dependencies may not be fully captured. Therefore, various grouping mechanisms have been explored, such as spatial patches [3], channel segments [35], or even checkered patterns [52], to reduce the performance drop from the parallelization and achieve computationally efficient models, particularly for real-time applications.

Attention mechanisms have emerged as a powerful tool in deep learning due to their ability to selectively focus on relevant parts of the input and suppress irrelevant information [81]. An advanced implementation of attention mechanisms was proposed in transformer networks for natural language processing [82], which later became the backbone of successful algorithms for various computer vision tasks, from object detection to image generation [83]–[88]. Unlike traditional convolutional neural networks (CNNs), transformers excel in capturing long-range dependencies and are dynamically adaptable to varying contexts through their flexible receptive fields [89]. By leveraging the attention mechanism, transformer-based context models can dynamically refine their predictions based on the spatial, channel, and semantic relations within the latent space, making them ideal candidates for efficient and high-performance entropy models.

This thesis seeks to further advance the field of context modeling in learned image compression by focusing on transformer-based architectures. By exploiting the dynamic and adaptive nature of transformers, this work aims to invent more high-performance and computationally efficient entropy models, capable of achieving state-of-the-art compression performance while maintaining a scalable solution for both high-resolution and resource-constrained scenarios.



1.1 Main Contributions

This thesis introduces several contributions in the field of LIC, focusing on advanced context modeling techniques. The two main innovations—**Contextformer** and **Efficient Contextformer**—advance the state of entropy modeling by enhancing both compression performance and computational efficiency.

Contextformer delivers a highly content-adaptive context model by leveraging a novel transformer-based architecture, which significantly improves compression performance upon previous LIC methods. The core innovation is the **Spatio-Channel Multi-Head Attention (SC-MHA)** mechanism, which extends the traditional Multi-Head Attention (MHA) to operate in higher dimensions, capturing both spatial and channel-wise dependencies dynamically. This mechanism allows for two distinct coding strategies, which prioritize either spatial or channel-wise dependencies depending on the processing order of the latent tensor elements. Additionally, Contextformer incorporates several optimizations aimed at improving encoding and decoding efficiency. To handle high-resolution images efficiently, a three-dimensional sliding window strategy is introduced, ensuring that the computational cost of SC-MHA remains within reach of the current hardware. Encoder-side optimizations exploit the parallelizable nature of transformer architectures, significantly enhancing throughput during encoding. Finally, three-dimensional wavefront coding is proposed to accelerate the decoding process by allowing multiple sliding windows to be processed in parallel, further improving the system's efficiency.

Efficient Contextformer builds upon the Contextformer, aiming at high performance while drastically reducing computational complexity. By introducing a range of optimizations, it addresses the latency and resource demands of context modeling, making it well-suited for resource-constrained applications. At the core of Efficient Contextformer is the **Spatio-Channel Window and Shifted-Window Attention (SC-W-MHA and SC-SW-MHA)** mechanism, which maintains the dynamic and content-adaptive nature of the original SC-MHA while significantly reducing its computational overhead. This mechanism approximates global attention by dividing the latent space into non-overlapping windows and shifting the windows across layers to capture both local and global spatio-channel dependencies. The proposed mechanism also integrates several grouped latent coding techniques to further reduce the number of autoregressive steps required in context modeling. One of these techniques is based on previously published work [3]. Moreover, Efficient Contextformer introduces a few novel runtime optimizations. On the encoder side, a reordering technique is proposed which maximizes parallel processing and efficiency of the attention computation. On the decoder side, caching mechanisms are adopted, allowing the model to store intermediate attention computations for reuse, which minimizes redundant calculations and reduces decoding time. The proposed optimization techniques enable the implementation of online optimization strategies, adjusting the latent representation during encoding to optimize the bitrate and reconstruction quality for each individual image. This adaptability allows more refined compression results across diverse datasets and use cases.

Additionally, an evaluation of various training and test datasets using multiple complexity criteria—such as texture, color variety, and structural segmentation—is provided,

offering a detailed guideline for future research. This approach ensures that both fairness and generalization are addressed when developing and testing new compression algorithms. Similarly, a large variety of state-of-the-art LIC frameworks is investigated for compression performance and efficiency, providing a benchmark for further studies.

This dissertation is based on the scientific works and results described in the following international peer-reviewed journal and conference publications:

- [1] A. B. Koyuncu, P. Jia, A. Boev, E. Alshina, and E. Steinbach, "Efficient Contextformer: Spatio-Channel Window Attention for Fast Context Modeling in Learned Image Compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 8, pp. 7498–7511, 2024

- [2] A. B. Koyuncu, H. Gao, A. Boev, G. Gaikov, E. Alshina, and E. Steinbach, "Contextformer: A Transformer with Spatio-Channel Attention for Context Modeling in Learned Image Compression," in *European Conference on Computer Vision (ECCV)*, Springer, 2022, pp. 447–463

- [3] A. B. Koyuncu, K. Cui, A. Boev, and E. Steinbach, "Parallelized Context Modeling for Faster Image Coding," in *2021 International Conference on Visual Communications and Image Processing (VCIP)*, IEEE, 2021, pp. 1–5

Furthermore, the following peer-reviewed publications are not directly included in this thesis. However, the results they present have been obtained and published during the course of this work:

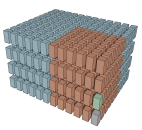
- [4] P. Jia, A. B. Koyuncu, G. Gaikov, A. Karabutov, E. Alshina, and A. Kaup, "Learning-based Conditional Image Coder Using Color Separation," in *2022 Picture Coding Symposium (PCS)*, IEEE, 2022, pp. 49–53

- [5] H. B. Dogaroglu, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "Adapting Learned Image Codecs to Screen Content via Adjustable Transformations," in *2024 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2024, pp. 1774–1780

- [6] P. Jia, A. B. Koyuncu, J. Mao, Z. Cui, Y. Ma, T. Guo, T. Solovyev, A. Karabutov, Y. Zhao, J. Wang, *et al.*, "Bit Rate Matching Algorithm Optimization in JPEG-AI Verification Model," in *2024 Picture Coding Symposium (PCS)*, IEEE, 2024, pp. 1–5

- [7] P. Jia, J. Mao, E. Koyuncu, A. B. Koyuncu, T. Solovyev, A. Karabutov, Y. Zhao, E. Alshina, and A. Kaup, "Bit Distribution Study and Implementation of Spatial Quality Map in the JPEG-AI Standardization," in *2024 International Conference on Visual Communications and Image Processing (VCIP)*, IEEE, 2024, pp. 1–5

- [8] N. Giuliani, H. You, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "CALC-VFS: Content-Adaptive Low-Complexity Video Frame Synthesis," in *25th IEEE International Symposium on Multimedia*, IEEE, 2023, pp. 57–61



- [9] K. Cui, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "Quality-Blind Compressed Color Image Enhancement with Convolutional Neural Networks," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5
- [10] K. Cui, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "Convolutional Neural Network-based Post-Filtering for Compressed YUV420 Images and Video," in *2021 Picture Coding Symposium (PCS)*, IEEE, 2021, pp. 1–5
- [11] A. B. Koyuncu, E. Erçelik, E. Comulada-Simpson, J. Venrooij, M. Kaboli, and A. Knoll, "A Novel Approach to Neural Network-based Motion Cueing Algorithm for a Driving Simulator," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 2118–2125

1.2 Thesis Organization

This thesis is organized as follows:

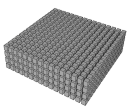
Chapter 2 presents a comprehensive background on image compression, covering both traditional and learned image compression techniques. It introduces key concepts such as transform coding, quantization, entropy modeling, and recent advancements involving attention mechanisms and transformers.

Chapter 3 outlines the performance evaluation methodology, including an analysis of the testing and training datasets, the performance and complexity metrics used to assess the models, and the experimental setup employed throughout the thesis.

Chapter 4 introduces Contextformer, highlighting its architecture based on spatial and spatio-channel attention mechanisms, along with several architectural and algorithmic optimizations to improve computational efficiency. Moreover, this chapter provides detailed experiments and ablation studies to validate the performance of the proposed methods.

Chapter 5 presents the Efficient Contextformer, highlighting the spatio-channel shifted-window attention mechanism, which fuses several parallelization techniques to reduce model complexity while retaining a high performance. It presents additional runtime optimizations, including encoder-side reordering and decoder-side caching mechanisms. Furthermore, this chapter includes extensive experiments and ablation studies to demonstrate the effectiveness of the proposed methods.

Finally, Chapter 6 concludes the thesis by summarizing the contributions, discussing limitations, and outlining potential future research directions.



Chapter 2

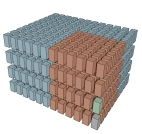
Background and Related Work

This chapter introduces the essential concepts and advancements in image compression. Section 2.1 covers the definition and types of image compression, setting the foundation for understanding the techniques discussed. Section 2.2 introduces traditional lossy image compression (TIC), detailing key components such as block-based transform coding, quantization, entropy coding, and rate-distortion optimization, and offers an overview of widely used traditional compression frameworks. Section 2.3 transitions to learned lossy image compression (LIC), highlighting advancements such as non-linear transform coding, differentiable quantization, learned entropy modeling, and online rate-distortion optimization, while also providing an overview of modern learned frameworks. Additionally, this section introduces the transformer neural network architecture, which forms the basis of the proposed learned entropy models in this thesis. Finally, Section 2.4 concludes the chapter with a summary.

2.1 Fundamentals of Image Compression

With the growing demand for higher-quality visuals, particularly in applications such as streaming, online storage, and media sharing, the management of vast quantities of data has become an important aspect of industrial applications. Image compression is the process of reducing the amount of data required to represent an image while maintaining the highest possible perceptual quality. Since image data typically contains significant redundancies, such as spatial, statistical, and psychovisual redundancies, various decorrelation operations are performed during the encoding process to map the input image into a more compact binary representation, called a bitstream, which is used for efficient storage and transmission of the data. On demand, the bitstream is converted back to a reconstructed image through the decoding process. Image compression techniques are broadly classified into two categories:

1. **Lossless Image Compression:** This technique refers to algorithms allowing perfect reconstruction of an original image from the compressed data. Since no information is lost during the process, the ability to achieve small file sizes is limited. Lossless techniques are typically used in applications where data fidelity and integrity are crucial, such as medical imaging or archival purposes. Widely used examples of lossless al-



gorithms are Portable Network Graphics (PNG) [14] and Graphics Interchange Format (GIF) [90].

2. **Lossy Image Compression:** In contrast to lossless algorithms, lossy approaches allow for the permanent removal of certain information from the original image, usually details that are less perceptually important. This results in an even smaller file size, yet some image quality is sacrificed. Lossy compression algorithms aim to balance the trade-off between visual quality and file size depending on the use case. For example, professional photography typically requires minimal compression to preserve image quality for detailed editing or high-resolution displays. Mobile photography allows for more compression to optimize storage and transmission efficiency, balancing file size and quality. Instant messaging applications require even more aggressive compression to minimize bandwidth usage, resulting in much smaller file sizes but with more noticeable quality degradation. Widely used examples of lossy compression algorithms include Joint Photographic Experts Group (JPEG) [16], JPEG 2000 [17], and WebP [18]. Additionally, intra-frame coding algorithms of video coding standards are also used for image compression, such as High Efficiency Image File Format (HEIF) [26] and Better Portable Graphics (BPG) [91], based on High Efficiency Video Coding (HEVC) standard [24].

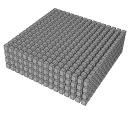
This thesis focuses on lossy image compression, exploring modern techniques and advancements for neural network-based learned image compression (LIC) methods, which provide more efficient solutions compared to widely used traditional image compression (TIC) techniques. Hereafter, whenever "image compression" is referred to in this thesis, it denotes lossy image compression.

2.2 Traditional Lossy Image Compression

2.2.1 Image Representation

An image is a visual representation of information that is captured, scanned, generated, or processed in various forms, such as digital photographs, scanned pictures, or computer-generated graphics. Digital images typically consist of picture elements, i.e., pixels, each conveying information about color intensities. The pixels are arranged in a three-dimensional (3D) tensor $\boldsymbol{x} \in \mathbb{R}^{H \times W \times C}$ with two-dimensional (2D) spatial coordinates, referred to as height H and width W, and the last dimension C encompasses the color components. The resolution of an image ($W \times H$) is historically expressed as the number of pixels in the horizontal and vertical dimensions, e.g., 3840×2160 pixels for a 4K image. Higher-resolution images contain more pixels and generally have finer details, whereas lower-resolution images may appear pixelated or blocky.

In a grayscale image, each pixel has a single value ($C=1$) that represents a shade of gray ranging from black to white. In a color image, each pixel contains multiple values ($C>1$) corresponding to the intensities of primary color components. When displayed or captured, these components are typically represented using the RGB model, which combines Red (R),



Green (G), and Blue (B) channels to approximate human color perception reasonably well. Each color component in an image has a digital representation using a certain number of bits, referred to as bit depth B. The bit depth determines the number of distinct shades or intensity levels that can be assigned to each color component. For instance, in the case of RGB images, a 24-bit image with 8-bit bit depth per channel allows 2^8 possible shades for each of the red, green, and blue components, resulting in 16.7 million possible colors.

Another commonly used color space is the YC_BC_R, which breaks the image into one luminance (Y) and two chrominance (C_B, C_R) components. Y represents brightness, while C_B and C_R reflect the deviation of blue and red color components from the luminance. The human eye is more sensitive to changes in brightness than to changes in color [92]. YC_BC_R is preferred in traditional image and video compression since separating luminance from chrominance allows compression algorithms to take advantage of this property by reducing the amount of color information without noticeably affecting the perceived image quality. For instance, chrominance channels can be downsampled to reduce the amount of data. In 4:2:0 subsampling, the chrominance channels are stored at half the horizontal and vertical resolution of the luminance channel. When luminance and chrominance components have the same number of samples, the scheme is referred to as the 4:4:4 YC_BC_R. The RGB and YC_BC_R formats can be seamlessly converted back and forth through linear transformations following the International Telecommunication Union (ITU) Recommendations such as BT.601 [93], BT.709 [94], and BT.2100 [15].

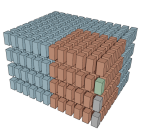
For further information about the human vision system and fundamentals of image representations, the reader is referred to [92], [95]–[98].

2.2.2 Block-based Transform Coding

A fundamental principle behind image compression is that neighboring pixels in an image are often highly correlated, as summarized in layman's terms in [96]:

> *'The Principle of Image Compression. If we select a pixel in an image at random, there is a high probability that its neighbors will have the same or very similar colors.'*
>
> (Data Compression: The Complete Reference, 2002)

Due to a significant amount of redundancy, the compression of images in the spatial domain is not always the most efficient. To remedy this, a fundamental technique called transform coding [99] uses a linear transform to convert image data from the spatial domain into a transformation coefficients domain—often the frequency domain—where the data is highly decorrelated. The frequency domain represents the rate of variation in pixel intensities across space, rather than the pixel values themselves. Low-frequency components correspond to slow, smooth changes in pixel intensity, while high-frequency components correspond to areas with sharp changes containing complex edges and fine details. Due to the high correlations in the spatial domain, most of the energy of natural images is concentrated in low-frequency components in the frequency domain, which allows for better compression by prioritizing these low-frequency components after the transform.



Given the monochrome input image $\boldsymbol{x} \in \mathbb{R}^{H\times W\times 1}$, the linear transformation $T(\cdot)$ can be mathematically formulated by:

$$\boldsymbol{y} = T(\boldsymbol{x}) \tag{2.1a}$$
$$= r^{-1}(\boldsymbol{A}\, r(\boldsymbol{x})), \tag{2.1b}$$

where $r : \mathbb{R}^{H\times W\times 1} \to \mathbb{R}^{HW\times 1}$ stands for the flattening transform vectorizing the input image, and $r^{-1} : \mathbb{R}^{HW\times 1} \to \mathbb{R}^{H\times W\times 1}$ is its inverse. The transformation matrix $\boldsymbol{A} \in \mathbb{R}^{HW\times HW}$ is an invertible matrix to guarantee a lossless transform. The reconstruction $\hat{\boldsymbol{x}}$ is obtained by applying the inverse transform defined as:

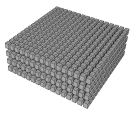
$$\hat{\boldsymbol{x}} = T^{-1}(\boldsymbol{y}) \tag{2.2a}$$
$$= r^{-1}(\boldsymbol{A}^{-1}\, r(\boldsymbol{y})). \tag{2.2b}$$

Images contain different levels of complexity and detail across different regions. Given an image of a landscape, some regions (e.g., trees and grassland) may have complex textures and sharp edges, whereas other regions (e.g., sky and lake) consist of smooth gradients. Performing a single transform on the entire image limits the ability to capture local frequency features, which may result in suboptimal compression. Moreover, processing a full-resolution image in a single step is computationally expensive due to large matrix multiplications, making transform coding infeasible on devices with limited resources such as mobile phones. To remedy both limitations, modern compression algorithms use block partitioning to perform block-based transform coding, which divides the image into smaller blocks of $N\times M$ pixels and separately applies transform coding to each block. Depending on the traditional image compression (TIC) algorithm, the block size can be fixed [16], i.e., agnostic to the content, or adaptive, adjusting to the local content within the image [24], [25]. Complex regions may use smaller blocks for better detail capture, while smoother areas can use larger blocks to efficiently reduce redundancy.

2.2.3 Quantization

Once an image has been transformed into the frequency domain, the transform coefficients typically have the same precision as those in the spatial domain, which is inefficient for high degrees of compression. Therefore, a quantization operation is applied to the transform coefficients to reduce their precision by mapping them into a set of discrete values. The alphabet size of the quantized values is significantly lower than that of the coefficients, which leads to fewer bits being required to store them. Quantization is the key step where data loss occurs, and it plays a central role in the lossy compression process.

Quantization operations can be broadly categorized into two groups—scalar and vector quantization. Scalar quantization maps each coefficient independently to a scalar value, providing a simple, low-complexity procedure. On the other hand, vector quantization maps a set of coefficients to a vector from a finite list of possible vectors. This allows capturing more complex relationships between the coefficients but at the cost of significantly higher



computational complexity, making it less practical for many applications. A simple scalar quantization $Q(\cdot)$ of transform coefficients and its inverse operation $Q^{-1}(\cdot)$ can be formulated by:

$$\hat{\boldsymbol{y}} = Q(\boldsymbol{y}) = \left\lfloor \frac{\boldsymbol{y}}{\Delta} \right\rceil, \tag{2.3a}$$

$$\bar{\boldsymbol{y}} = Q^{-1}(\hat{\boldsymbol{y}}) = \Delta\hat{\boldsymbol{y}}, \tag{2.3b}$$

where Δ defines the quantization step size. Larger Δ values correspond to a coarser quantization, and Δ=1 corresponds to rounding values to their nearest integer. Δ can be a single value uniformly quantizing the entire input range (uniform quantizer), or can have different values throughout the input range (non-uniform quantizer).

In image compression, more optimal quantization can be achieved by exploiting the limitations of the human visual system to reduce psychovisual redundancies. Humans are generally less sensitive to high-frequency details (rapid changes in intensity, such as sharp edges) than to low-frequency information (smooth gradients), and are more sensitive to brightness than to color [92]. Therefore, TIC algorithms can leverage these limitations by applying separate scaling factors to the coefficients prior to quantization, such as:

$$\hat{y}_{i,j} = Q\left(\frac{y_{i,j}}{q_{i,j}}\right), \tag{2.4a}$$

$$\bar{y}_{i,j} = q_{i,j}\, Q^{-1}\left(\hat{y}_{i,j}\right), \tag{2.4b}$$

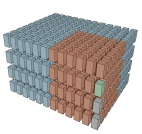
where the quantization table $\boldsymbol{q}$ determines how aggressively each transform coefficient at coordinate (i, j) is quantized, and its values are selected based on psychovisual experiments on the human visual system's sensitivity to different frequencies [100]. The choice of quantization values $q_{i,j}$ can significantly impact the compression ratio and image quality. For larger quantization values, the reconstructed image will have more artifacts but a smaller file size. On the other hand, smaller quantization values preserve more detail at the cost of a higher compressed file size.

2.2.4 Entropy Coding

After transform coding and quantization reduce the spatial and psychovisual redundancies, the next step in the compression process is to eliminate any remaining statistical redundancy in the quantized transform coefficients. This is achieved with a lossless compression technique called entropy coding. It reduces the number of bits required to represent the quantized coefficients by taking advantage of their statistical properties and maps them into a bitstream.

The concept of information entropy $\mathcal{H}(\cdot)$ is defined in information theory [101]. It measures the average amount of information produced by a stochastic source of events or data in bits, quantifying the uncertainty or randomness in a signal. The information entropy of a source signal $\boldsymbol{X}$ with a symbol alphabet $\mathcal{X} = \{x_0, x_1, \ldots, x_{N-1}\}$ can be mathematically defined as:

$$\mathcal{H}(\boldsymbol{X}) = -\sum_{x \in \mathcal{X}} p(x) \log_2(p(x)), \tag{2.5}$$



where $\mathcal{X}$ encapsulates all possible outcomes, with each symbol having a probability $p(x)$ and an information content of $-\log_2(p(x))$. The more probable observing a symbol is, the less the symbol contributes to the overall entropy.

In the context of image compression, the information entropy represents the theoretical lower bound on the number of bits required to represent the quantized coefficients. For efficient compression, entropy coding follows the principle of Variable-Length Coding (VLC) [97], which assigns shorter codewords to more frequent symbols and longer codewords to less frequent ones, ensuring that the average number of bits used approaches the entropy $\mathcal{H}(\boldsymbol{X})$. Two common VLC methods are Huffman coding [19] and arithmetic coding [102]–[104]. Huffman coding [19] uses a prefix-free binary tree structure where the deepest leaves represent the least frequent symbols. More frequent symbols appear higher in the tree, meaning they require fewer bits to code. Huffman coding [19] is optimal for independent and identically distributed (i.i.d.) signals with a dyadic probability distribution. On the other hand, arithmetic coding encodes an entire sequence of symbols as a fractional number in the range $[0, 1)$. It iteratively subdivides the interval $[0, 1)$ based on symbol probabilities $p(x)$, narrowing the range as symbols are processed. Compared to Huffman coding [19], this method can achieve non-integer bit lengths for each symbol, allowing the average bit length to come closer to $\mathcal{H}(\boldsymbol{X})$ for skewed and non-dyadic distributions. However, it requires higher computational complexity than Huffman coding [19].

Another important technique in entropy coding is context-adaptive variable-length coding (CAVLC). Although transform coding reduces spatial correlations between the coefficients, some underlying dependencies may remain, which can reduce the efficiency of entropy coding. Given two source signals $\boldsymbol{X}$ and $\boldsymbol{Y}$ with their corresponding symbol alphabets $\mathcal{X} = \{x_0, x_1, \ldots, x_{N-1}\}$ and $\mathcal{Y} = \{y_0, y_1, \ldots, y_{M-1}\}$, the joint information entropy can be mathematically defined as:

$$\mathcal{H}(\boldsymbol{X}, \boldsymbol{Y}) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x, y)\log_2\left(p(x, y)\right) \tag{2.6a}$$

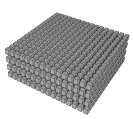
$$= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x, y)\log_2\left(p(y)\cdot p(x \mid y)\right) \tag{2.6b}$$

$$= -\sum_{y\in\mathcal{Y}} p(y)\log_2\left(p(y)\right) - \sum_{y\in\mathcal{Y}} p(y)\sum_{x\in\mathcal{X}} p(x \mid y)\log_2\left(p(x \mid y)\right) \tag{2.6c}$$

$$= \mathcal{H}(\boldsymbol{Y}) + \mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y}) \leq \mathcal{H}(\boldsymbol{Y}) + \mathcal{H}(\boldsymbol{X})^{\dagger}, \tag{2.6d}$$

where the conditional entropy $\mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y})$ is lower than or equal to the marginal entropy $\mathcal{H}(\mathcal{X})$. This implies that conditioning on an additional signal $\boldsymbol{Y}$ can only reduce entropy or, at worst, keep the entropy unchanged. CAVLC leverages the properties of conditional entropy by employing a context model. The context model considers previously encoded symbols $\boldsymbol{X}_{<i}$ as the additional signal $\boldsymbol{Y}$ and uses them for coding the current symbol X_i at coding step i. It improves upon traditional entropy coding by adjusting the conditional probability $p(X_i \mid \boldsymbol{X}_{<i})$ of $\boldsymbol{X}_i$ which is updated based on previously encoded data $\boldsymbol{X}_{<t}$, also referred to as the context.

† A detailed proof can be found in [105]



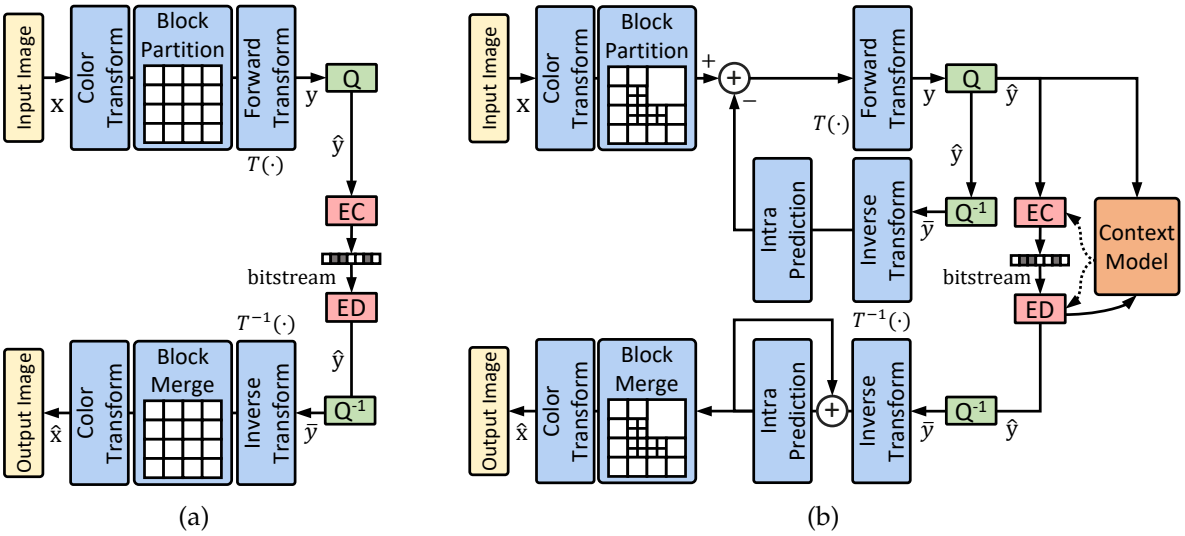


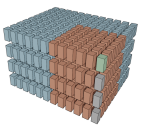
Figure 2.1: Illustration of traditional image compression frameworks: (a) early approaches, such as JPEG [16], and (b) more advanced methods, including the intra-coding of HEVC [24] and VVC [25]. $T(\cdot)$, $Q(\cdot)$, $T^{-1}(\cdot)$, and $Q^{-1}(\cdot)$ stand for the linear transformation, quantization, and their inverse operations, respectively. EC and ED denote lossless entropy encoder and decoder.

2.2.5 Overview of Traditional Image Compression Frameworks

Fig. 2.1 illustrates the overall workflow of TIC frameworks, integrating the concepts explained in the previous sections (Sections 2.2.1 to 2.2.4). The encoder-side pipeline begins with the conversion of the image from the RGB color space into the YC_BC_R color space. Algorithms like JPEG [16] apply 4:2:0 chrominance subsampling to reduce data size and computational complexity. More modern video coding standards such as HEVC [24] and VVC [25] support multiple subsampling formats, such as 4:4:4 and 4:2:0, depending on the specific use case.

Following color conversion, the image undergoes block-based transform coding to reduce spatial redundancies. JPEG [16] applies the type-II Discrete Cosine Transform (DCT-II) [106] to fixed 8×8 blocks of pixels. JPEG 2000 [17] adopts a different approach by using Discrete Wavelet Transform (DWT) [107] instead of the DCT, without a theoretical limitation on the block size. On the other hand, HEVC [24] and VVC [25] introduce more flexible transform coding techniques. For instance, HEVC [24] replaces fixed blocks with hierarchical partitioning [108], where the image is split into Coding Tree Units (CTUs) with sizes ranging from 8×8 to 64×64 pixels. Quad-tree partitioning [20] is then applied to recursively split the CTUs into even smaller blocks for transform coding, allowing the codec to adapt block sizes based on local image details. VVC [25] extends this by introducing more advanced partitioning structures, such as binary-tree and ternary-tree [21], which provide even more adaptive and flexible block partitioning. Moreover, HEVC [24] and VVC [25] can also use different types of transforms such as type-VIII DCT and type-VII Discrete Sine Transform (DST-VII) [109], depending on the content of the image.

Following the transform stage, the precision of the transform coefficients is reduced by the quantization step. JPEG [16] and JPEG 2000 [17] use a simple scalar quantization, where



each coefficient is divided by a value from a quantization matrix. In contrast, HEVC [24] and VVC [25] allow for more sophisticated quantization schemes, adapting quantization parameters based on local image characteristics to achieve a better balance between compression and visual quality. Moreover, VVC [25] adopts a vector quantization technique called Trellis-Coded Quantization [110] (TCQ) for higher compression performance.

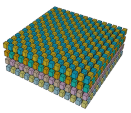
Another key innovation in modern compression algorithms like HEVC [24] and VVC [25] is the use of predictive coding techniques such as intra-prediction [22], [23], which further reduces spatial redundancy by predicting the pixel values of the current block using neighboring blocks within the same picture. Instead of encoding an entire block of pixels, the codec predicts the values from already decoded adjacent block samples, and only the differences between the predicted and actual values, also called *residuals*, are encoded.

The final step in the encoder-side process is entropy coding, which compresses the quantized coefficients into a bitstream. JPEG [16] traditionally uses Huffman coding [19], while JPEG 2000 [17] employs a type of CAVLC, called the MQ Arithmetic Coder [111]. In contrast, HEVC [24] and VVC [25] adopt a more advanced entropy coding technique known as Context-based Binary Arithmetic Coding (CABAC) [112], maximizing compression efficiency through detailed context modeling and adaptive probability estimation.

On the decoder side, the process mirrors the encoder's workflow. The compressed bitstream undergoes entropy decoding to retrieve the quantized coefficients, which are then dequantized to restore their original scale. The next step involves applying the inverse transforms, such as the inverse DCT, inverse DST, or inverse DWT, to convert the data back from the frequency domain to the spatial domain. For HEVC [24] and VVC [25], intra-prediction is reversed by adding the residuals back to the predicted values to reconstruct the original block. Finally, the color components are merged back into the original RGB color space, yielding the final reconstructed image. While the overall process is symmetrical, some loss of quality is inherent due to the lossy nature of the quantization employed during the compression. By leveraging these techniques—color space conversion, chroma subsampling, block-based transform coding, intra-prediction, quantization, and entropy coding—TIC frameworks can achieve substantial reductions in image size while maintaining acceptable visual quality.

2.2.6 Rate-Distortion Optimization

Rate-Distortion Optimization (RDO) [113] plays a fundamental role in modern lossy compression frameworks, balancing the trade-off between the size of the compressed bitstream (i.e., the bitrate R) and the loss of quality in the reconstructed image (i.e., the distortion D). The strength of modern TIC algorithms, such as HEVC [24] and VVC [25], lies in their ability to perform computationally expensive encoding tasks, allowing the encoder to search through various encoding configuration parameters to achieve the optimal balance between bitrate and quality. These parameters involve selecting appropriate transform coding types, partitioning strategies for dividing the image into blocks, and determining the best quantization parameters, among others. The optimal parameter setting may vary for different images depending on their content.



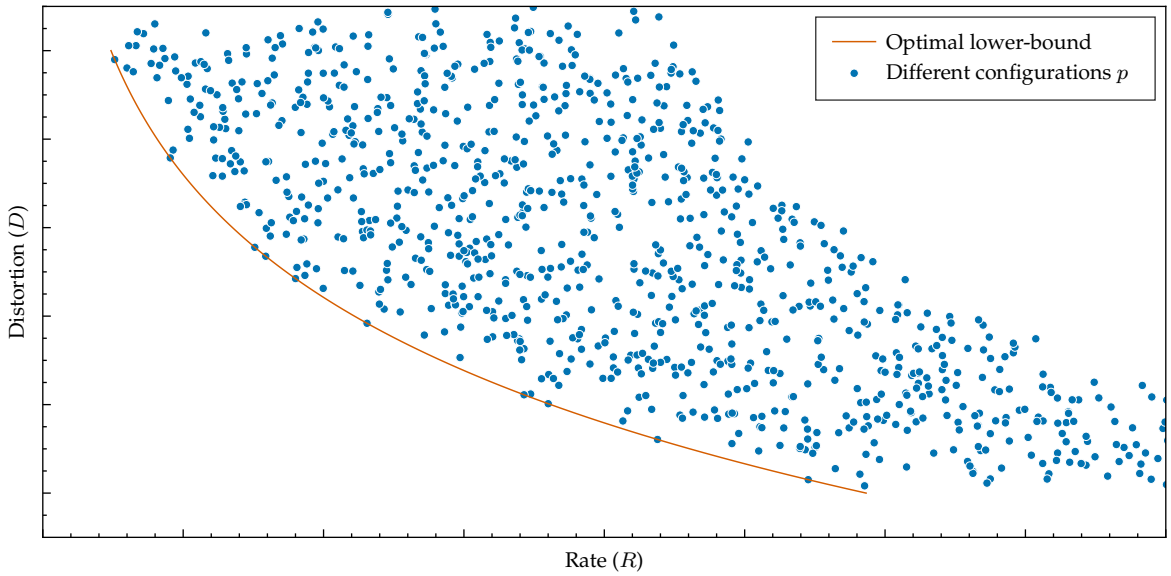


Figure 2.2: Rate-distortion plot for different configurations of the encoding parameters. Solid curve shows the hypothetical lower-bound for the best compression performance for each bitrate. (Adapted from [114])

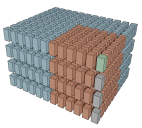
Fig. 2.2 illustrates the bitrate as a function of distortion $R(D)$, where different configurations of encoding parameters result in different points on the rate-distortion curve for a single image. The dashed Pareto curve represents the optimal lower bound, providing the highest possible quality for each bitrate. Finding points on the lower bound is often formulated as a Lagrangian optimization problem, which can be mathematically expressed as:

$$\boldsymbol{P}_{\text{opt}} = \arg\min_{\boldsymbol{P}\in\mathcal{P}} J(\boldsymbol{P} \mid \lambda) \tag{2.7a}$$

$$= \arg\min_{\boldsymbol{P}\in\mathcal{P}} \left[D(\boldsymbol{P}) + \lambda R(\boldsymbol{P})\right], \tag{2.7b}$$

where $J(\boldsymbol{P} \mid \lambda)$ represents the cost function for a given configuration $\boldsymbol{P}$ from a set of configurations $\mathcal{P}$. Each configuration $\boldsymbol{P}$ contains a set of parameters $\{p_1, \ldots, p_N\}$ and can differ from other configurations by one or many parameters depending on the complexity requirements. Typically, combinatorial optimization of multiple parameters results in a larger search space, leading to much higher computational complexity in encoding. The Lagrangian multiplier λ controls the trade-off between bitrate and distortion by adjusting the cost function.

Since the parameters of the optimal configuration $\boldsymbol{P}_{\text{opt}}$ are signaled to the decoder, no RDO process is required during decoding. This results in asymmetric computational complexity in TIC frameworks, where the encoder requires time and memory complexity orders of magnitude higher than that of the decoder.



2.3 Learned Lossy Image Compression

2.3.1 Non-linear Transform Coding

In lossy learned image compression frameworks (LIC), non-linear transform coding [36] has emerged as a powerful alternative to traditional linear transform coding methods such as DCT [106] or DWT. The core concept behind non-linear transform coding is to replace manually designed linear transforms with learned non-linear transformations, typically implemented using Artificial Neural Networks (ANN, or shortly NN). Known for their universal approximation capabilities [115]–[118], NNs can approximate highly complex functions, making them suitable for learning more sophisticated transforms. Unlike fixed traditional transforms, learned transforms adapt to the content of the image and capture intricate non-linear dependencies in a high dimensional space.

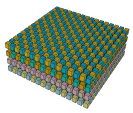
At the heart of LIC is an autoencoder structure [78], [119], which implements the non-linear transforms. The encoder side of the autoencoder employs an *analysis transform* $g_a(\cdot\,;\,\boldsymbol{\phi}_a)$, composed of several convolutional neural network (CNN) layers with learnable parameters $\boldsymbol{\phi}_a$. Each layer of the CNN applies a combination of linear operations (such as convolutions and downsampling) and non-linear activations. Convolutional layers help capture local spatial correlations, while non-linear activations enhance the network's modeling capacity. One common activation function is Generalized Divisive Normalization (GDN) [120], inspired by processes in the human sensory system, where neural responses are normalized based on surrounding activity. GDN [120] normalizes activations across channels by dividing each activation by a weighted sum of other activations, allowing higher image decorrelation.

The analysis transform $g_a(\cdot\,;\,\boldsymbol{\phi}_a)$ projects the input image $\boldsymbol{x} \in \mathbb{R}^{H\times W\times 3}$ into a lower-dimensional latent space representation $\boldsymbol{y} \in \mathbb{R}^{\frac{H}{2^D}\times\frac{W}{2^D}\times C}$, where D defines the number of downsampling layers. This reduces spatial redundancies similarly to traditional transform coding, but with the added flexibility of learning the transformation during training. The dimensionality ratio between the input image and the latent space can be computed as:

$$\frac{\dim(\boldsymbol{x})}{\dim(\boldsymbol{y})} = \frac{3HW}{\frac{CHW}{2^{2D}}} = \frac{3\cdot 2^{2D}}{C}, \tag{2.8}$$

which results in a factor of 4 for typical values of $D{=}4$ and $C{=}192$. However, this does not necessarily lead to information loss, as the latent space is typically represented with a much higher bit depth (e.g., 32-bit or 64-bit) compared to the bit depth of the original image space (e.g., 8-bit).

On the decoder side, the *synthesis transform* $g_s(\cdot \mid \boldsymbol{\phi}_s)$, analogous to the inverse transform in traditional coding, reconstructs the image $\hat{\boldsymbol{X}}$ from the quantized latent representation $\hat{\boldsymbol{y}}$. Here, $\boldsymbol{\phi}_s$ represents the learnable parameters of the synthesis transform. Typically, it mirrors the structure of the analysis transform, using deconvolution layers to upsample the data and activation functions like inverse GDN (IGDN) [120] to denormalize the activations. Both transforms are learned end-to-end using optimization techniques such as backpropagation, as detailed in later sections.



2.3.2 Differentiable Quantization

As introduced in Section 2.2.3, quantization plays a crucial role in lossy image compression by reducing the precision of variables. In LIC frameworks, the continuous latent space representation $\boldsymbol{y}$ must be quantized into discrete variables $\hat{\boldsymbol{y}}$ to encode them into a bitstream efficiently. Traditional quantization methods, such as rounding in Eq. (2.3), introduce discontinuities that prevent gradient calculation during training, making them unsuitable for the end-to-end training of NNs through gradient descent. To address this issue, several techniques have been developed, enabling quantization to become differentiable and allowing the use of gradient-based optimization.

A widely used method for handling non-differentiable operations in NNs like quantization is the Straight-Through Estimator (STE) [121]. In this method, non-differentiable operations are applied directly during the forward pass through the NN. However, during backpropagation, gradients simply pass through the non-differentiable operator unchanged, as if the operation did not exist. Formally, this can be described for hard quantization from Eq. (2.3) as:

$$\hat{\boldsymbol{y}} = Q(\boldsymbol{y}), \quad \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{y}} \approx 1, \tag{2.9}$$

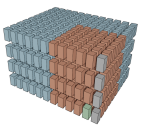
where the gradients at the final layer of the analysis transform and the first layer of the synthesis transform are approximated the same.

While STE [121] provides a simple yet effective solution for achieving differentiable quantization during training, it introduces some limitations in the context of efficient LIC frameworks. Studies have shown [31]–[33] that training with discretized latent variables $\hat{\boldsymbol{y}}$ results in learning a discrete probability distribution and, consequently, a discrete entropy model, which limits the capacity of learned entropy coding (explained in the next section). To remedy this, the additive uniform noise strategy was introduced by [31]. Instead of performing hard rounding (see Eq. (2.3)), small random uniform noise $U \sim \mathcal{U}(-0.5, 0.5)$ is added to $\boldsymbol{y}$, approximating the effect of quantization. Mathematically, this approximation can be expressed as:

$$\tilde{\boldsymbol{y}} \approx \boldsymbol{y} + U, \quad U \sim \mathcal{U}(-0.5, 0.5). \tag{2.10}$$

This smooth gradient approximation ensures that gradients can be propagated and a differential entropy model can be learned instead of a discrete one.

Another technique to address the differentiability issue is soft assignment quantization [122]. Instead of rounding to a single nearest value, continuous latent variables are probabilistically assigned to a scalar-valued cluster center from a codebook [42]. This allows for smooth gradient propagation while training, eventually converging toward hard decisions as the network optimizes the latent space. The codebook of the cluster centers is also learned via backpropagation and gradient descent, allowing for non-uniform quantization. Moreover, using vectors as cluster centers achieves differentiable vector quantization [70], [123]. Given the cluster centers from the learned codebook $\mathcal{C} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_K\}$, where K is



the number of discrete vectors in the codebook, the soft quantization process is defined as:

$$\tilde{\boldsymbol{y}} = \sum_{i=1}^{K} p(\boldsymbol{c_i} \mid \boldsymbol{y}) \cdot \boldsymbol{c}_i, \tag{2.11a}$$

$$p(\boldsymbol{c_i} \mid \boldsymbol{y}) = \mathrm{softmax}(\|\boldsymbol{y} - \boldsymbol{c}_i\|) \tag{2.11b}$$

$$= \frac{\exp\left(\frac{-\|\boldsymbol{y}-\boldsymbol{c}_i\|}{\tau}\right)}{\sum_{j=1}^{K} \exp\left(\frac{-\|\boldsymbol{y}-\boldsymbol{c}_j\|}{\tau}\right)}, \tag{2.11c}$$

where $p(\boldsymbol{c_i} \mid \boldsymbol{y})$ is the probability weighting the membership of the latent tensor $\boldsymbol{y}$ to each codebook vector $\boldsymbol{c}$. The temperature parameter τ determines the "smoothness" of the probabilistic assignment to codebook vectors, while lower values of τ approach hard assignment. After training, the soft vector quantization is replaced with a hard one as:

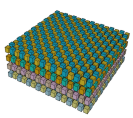
$$\hat{\boldsymbol{y}} = \arg\min_{\boldsymbol{c}_i \in \mathcal{C}} \|\boldsymbol{y} - \boldsymbol{c}_i\|. \tag{2.12}$$

While soft vector quantization [70], [123] can offer better compression performance, especially for complex distributions, it is computationally expensive due to soft assignments and gradual transitions. STE [121] provides a more computationally efficient approach, though it may lead to sub-optimal compression performance due to ill-conditioned gradients. The smooth gradient approximation [31] offers smoother training than STE [121] with a similar computational cost. However, it is less flexible than soft quantization, and training the nonlinear transforms with approximate values $\tilde{\boldsymbol{y}}$ may lower performance due to its stochastic nature. Therefore, multiple studies [32]–[34] suggest combining both approximation techniques Eqs. (2.9) and (2.10) for training different parts of the same LIC framework. For instance, STE [121] can be used for training the analysis and synthesis transforms, i.e., the synthesis transform receiving hard quantized values $\hat{\boldsymbol{y}}$, while the learned entropy coding model uses the approximated values $\tilde{\boldsymbol{y}}$. Consequently, both methods are replaced with hard quantization during inference.

2.3.3 Learned Entropy Modeling

The entropy coding techniques such as Huffman coding [19] and arithmetic coding [102], [103] rely on probability distributions of the quantized transform coefficients, which in TIC algorithms are often hand-engineered. These probability tables are built on heuristic assumptions, such as that the distribution is either i.i.d., or follows a simple predefined probability model, treating the transform coding and entropy modeling stages as separate components. While this approach may be effective for simpler linear transforms like DCT [106], the high-dimensional latent representations of LIC frameworks are far more complex, with intricate dependencies and correlations. Moreover, hand-engineering a probability model during the training of LIC is practically infeasible due to the continuous updates of nonlinear transform coding parameters.

To address these limitations, LIC frameworks replace the predefined probability models with learned probability models that are trained to minimize entropy in the latent space,



which reduces the statistical redundancies of the latent representation. By learning the entropy model alongside the transform coding parameters, LIC frameworks can find better transforms that not only decorrelate the data more effectively but also result in a more compact latent representation that is easier to compress. The improved transforms produce more structured latents with lower entropy, easing the learning of more accurate entropy models. After training, the entropy model provides the probability distribution of the latent representation, which is used to encode latents into the bitstream with lossless entropy coding methods such as Huffman coding [19] and arithmetic coding [102], [103].

In recent learned image compression models, the entropy modeling process has evolved into several advanced techniques. One of the earliest innovations was the *factorized entropy model* [32]. This model consists of NN layers with non-negative weights and activations, learning the marginal probability distribution of the quantized latent representation $p(\hat{\boldsymbol{y}})$. In essence, it treats each latent element as conditionally independent from the others, building local histograms for entropy modeling, thereby providing a simple yet effective method for entropy coding. The factorized entropy model can be mathematically expressed as:

$$p(\hat{\boldsymbol{y}}\,;\,\theta_f) = \prod_i p(\hat{y}_i\,;\,\theta_f), \tag{2.13}$$

where θ_f stands for the parameters of the NN[†]. Although this method captures some of the redundancies in the latent space, it cannot exploit the dependencies between latent variables, which could lead to suboptimal compression, particularly when statistical dependencies exist in the actual distribution of the latent representation.

To improve upon the factorized entropy model, the concept of the *hyperprior network* was introduced in [33] to implement hierarchical coding. In this framework, a second autoencoder, called the hyperprior, processes the latent representation ($\boldsymbol{y}$ or $\hat{\boldsymbol{y}}$) to generate a side information representation $\boldsymbol{z}$. The side information $\hat{\boldsymbol{z}}$ is quantized, modeled using a factorized entropy model, and transmitted along with the compressed bitstream. The quantized side information $\hat{\boldsymbol{z}}$ serves as context for modeling the conditional probability distribution of the latent variables $p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$. The hyperprior network can be viewed as a non-linear transform coding scheme applied to the latent tensor. It can capture the spatial dependencies within the latent tensor $\hat{\boldsymbol{y}}$, and signaling those dependencies helps decouple the latent tensor, resulting in a more refined entropy model. The conditional probability distribution $p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$ learned by the hyperprior is mathematically formulated as:

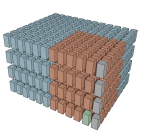
$$p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}}) \leftarrow h_s(\hat{\boldsymbol{z}}\,;\,\boldsymbol{\theta_s}), \tag{2.14a}$$

$$\hat{\boldsymbol{z}} = Q(\boldsymbol{z}), \tag{2.14b}$$

$$\boldsymbol{z} = h_a(\hat{\boldsymbol{y}}\,;\,\boldsymbol{\theta_a}), \tag{2.14c}$$

where h_a and h_s represent the analysis and synthesis transforms of the hyperprior with their learnable parameters $\boldsymbol{\theta_a}$ and $\boldsymbol{\theta_s}$. The operator $\leftarrow$ indicates that the hyperprior provides the parameters, or priors, for modeling the conditional distribution $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$. For instance, if the

[†] For more information about the design of the NN, the reader is referred to Section 6.1 of the Appendix in [33].



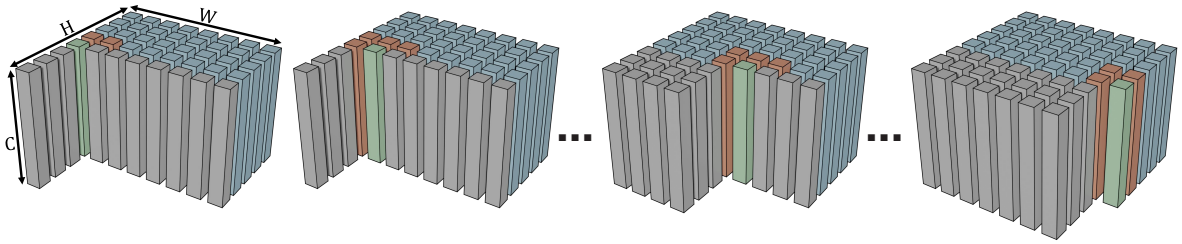


Figure 2.3: Illustration of the context model proposed in Minnen *et al.* [34], featuring a 3×3 masked convolution that processes one row of the latent tensor from left to right. The context model aggregates information from neighboring latent elements (■) to estimate the symbol probability of the current latent element to be coded (■). Elements already coded in previous steps are shown in (■), while those yet to be coded are displayed in (■). For simplicity, only part of the convolutional kernel is shown at the edges, where the latent tensor is padded with zeros for actual coding.

latent tensor $\hat{\boldsymbol{y}}$ is modeled with a Gaussian distribution $\mathcal{N}(\cdot)$, Eq. (2.14a) becomes:

$$p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}}) = \prod_i \mathcal{N}(\hat{y}_i \,;\, \mu_i, \sigma_i^2) \tag{2.15a}$$

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = h_s(\hat{\boldsymbol{z}} \,;\, \boldsymbol{\theta_s}), \tag{2.15b}$$

where each latent element $\hat{y}_i$ is parametrized by a dedicated mean μ_i and variance σ_i^2 signaled via the hyperprior network.

Even greater compression gains in LIC frameworks can be achieved by learned context modeling, which is inspired by the principles behind CAVLCs used in TIC frameworks such as HEVC [24] and VVC [25]. Unlike the fixed and manually designed methods in TIC frameworks, LIC frameworks can adaptively learn the context from data during training. This allows for a more powerful and flexible estimation of the entropy of each latent element, enhancing the ability to capture complex dependencies in the latent space. By autoregressively updating entropy predictions based on both the local spatial structure and previously encoded information, learned context models significantly improve compression efficiency.

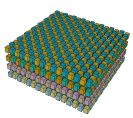
The first implementation of learned context models [34], [49] used masked convolutions proposed in PixelCNN [124]. In this approach, $K\times K$ convolutional kernels are applied, where the lower triangle of the coefficients is set to zero to ensure causality in the raster-scan coding process by masking future, unseen data. This guarantees that the model only has access to previously encoded data from the spatial local neighborhood during the prediction process. Fig. 2.3 illustrates the context model with a masked convolution processing the latent tensor in raster-scan order. A typical high-performance entropy model combines the hyperprior with the context model, which can be expressed as:

$$p_{\hat{\boldsymbol{y}}_i}(\hat{\boldsymbol{y}}_i \mid \hat{\boldsymbol{z}}) \leftarrow g_{ep}(\Omega, \Psi \,;\, \boldsymbol{\theta_{ep}}), \tag{2.16a}$$

$$\boldsymbol{\Omega} \quad = g_c(\hat{\boldsymbol{y}}_{<i} \,;\, \boldsymbol{\theta_c}), \tag{2.16b}$$

$$\boldsymbol{\Psi} \quad = h_s(\hat{\boldsymbol{z}} \,;\, \boldsymbol{\theta_s}), \tag{2.16c}$$

where $g_c(\cdot \,;\, \boldsymbol{\theta_c})$ is the context model with learnable parameters $\boldsymbol{\theta_c}$, and $g_{ep}(\cdot \,;\, \boldsymbol{\theta_{ep}})$ is the entropy parameters network with learnable parameters $\boldsymbol{\theta_{ep}}$, which combines the outputs of the context model ($\boldsymbol{\Omega}$) and the hyperprior ($\boldsymbol{\Psi}$).



In this combined framework, the hyperprior is referred to as *forward adaptation* [36], as it transmits global side information to update conditional entropy estimation. As the bitstream size increases with the size of the side information, the performance gain that can be achieved by forward adaptation alone is limited. In contrast, the context model performs *backward adaptation* [36], conditioning the current latent $\hat{\boldsymbol{y}}_i$ on previously coded elements $\hat{\boldsymbol{y}}_{<i}$, without requiring additional side information. However, context modeling decreases computational efficiency due to its autoregressive nature. Combining forward and backward adaptation allows the model to capture both global and local dependencies, resulting in a more accurate entropy model but with higher latency.

2.3.4 Transformers

The transformer architecture, initially introduced for natural language processing (NLP) tasks [82], has since been adapted for various computer vision (CV) applications due to its ability to build dynamic relationships between elements in a sequence. Unlike traditional CNN-based architectures, which use fixed coefficients after training to process inputs, transformers use mechanisms to weigh interactions between different parts of the input, making them more flexible and powerful for modeling complex spatial and semantic relationships within images.

One of the core mechanisms of the transformer is the *self-attention* mechanism. Inspired by information retrieval methods [125], self-attention operates by projecting an input sequence of vectors $\boldsymbol{x} \in \mathbb{R}^{S\times C}$, called *tokens*, where S is the sequence length and C is the channel dimension, into an embedding $\boldsymbol{e} \in \mathbb{R}^{S\times d_e}$ using learnable linear transformations. Next, the embedding representation $\boldsymbol{e}$ is mapped into three distinct representations: *queries* ($\boldsymbol{Q}$), *keys* ($\boldsymbol{K}$), and *values* ($\boldsymbol{V}$), all typically in $\mathbb{R}^{S\times d_e}$. This can be formulated as:

$$\boldsymbol{Q}(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W}_{\boldsymbol{Q}}, \quad \boldsymbol{K}(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W}_{\boldsymbol{K}}, \quad \boldsymbol{V}(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W}_{\boldsymbol{V}}, \tag{2.17}$$

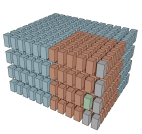
where $\boldsymbol{W}_{\boldsymbol{Q}} \in \mathbb{R}^{d_e\times d_e}$, $\boldsymbol{W}_{\boldsymbol{K}} \in \mathbb{R}^{d_e\times d_e}$, and $\boldsymbol{W}_{\boldsymbol{V}} \in \mathbb{R}^{d_e\times d_e}$ are the weights of the learned linear transformations, applied to each token separately.

The transformer then computes the *attention* using the scaled dot-product between the query and key pairs, which determines how strongly each token in the sequence relates to others. The similarity scores are then used to weight the values $\boldsymbol{V}$, determining how much attention each input element should pay to others in the sequence. The attention mechanism is formulated as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_e}}\right)\boldsymbol{V}, \tag{2.18}$$

where $\text{softmax}(\cdot)$ normalizes the attention scores across the sequence, ensuring a probabilistic output.

To enhance the model's ability to capture multiple relationships, transformers implement Multi-Head Attention (MHA). This involves splitting the query, key, and value tensors into h smaller sub-representations across their embedding dimension. Each sub-representation $\boldsymbol{Q}_i$, $\boldsymbol{K}_i$, and $\boldsymbol{V}_i$ has the dimensionality of $\mathbb{R}^{S\times\frac{d_e}{h}}$, and they are also referred to as *heads*. The attention of each head independently processes part of the sequence, allowing the model to focus



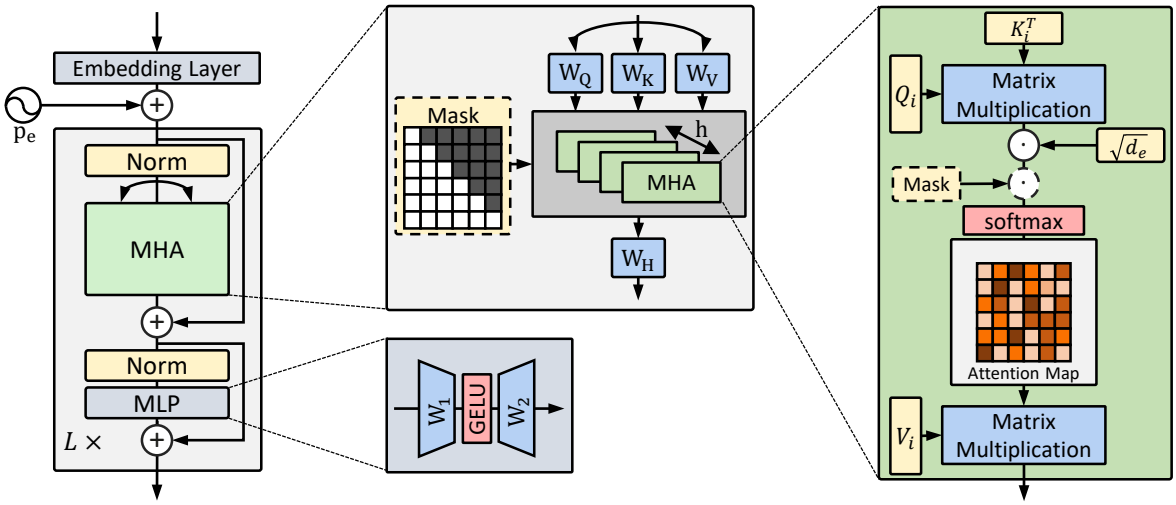


Figure 2.4: Illustration of transformer network with self-attention [82]. The Multi-head Attention mechanism is displayed in details.

on different aspects of the input, such as global context or fine-grained local details. The results of each attention head are then concatenated and combined using a learned weight matrix $\boldsymbol{W_H}$, producing the final MHA output:

$$\text{MHA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = (\text{Head}_1 \mathbin{+\!\!+} \text{Head}_2 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} \text{Head}_h)\boldsymbol{W_H}, \tag{2.19a}$$

$$\text{Head}_i(\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i) = \text{softmax}\left(\frac{\boldsymbol{Q}_i \boldsymbol{K}_i^T}{\sqrt{d_e}}\right)\boldsymbol{V}_i, \tag{2.19b}$$

where $\mathbin{+\!\!+}$ denotes to concatenation of heads along their last dimension as defined in [126].

Moreover, in autoregressive tasks such as image generation and context modeling, the attention mechanism applies a mask to prevent access to elements that appear later in the sequence. This ensures that each position only considers previous or current information, preserving the causal flow and preventing any unintended "look-ahead" to future elements. The masked attention extends Eq. (2.19b) to:

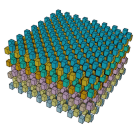
$$\text{Head}_i(\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i) = \text{softmax}\left(\frac{\boldsymbol{Q}_i \boldsymbol{K}_i^T}{\sqrt{d_e}} \odot \boldsymbol{M}\right)\boldsymbol{V}_i, \tag{2.20}$$

where $\boldsymbol{M} \in \mathbb{R}^{S \times S}$ is a mask that blocks access to unprocessed elements by assigning $-\infty$ to the positions representing future interactions, and $\odot$ is the Hadamard product [127]. Typically, the mask is implemented as a lower triangular matrix, where the upper triangle has values of $-\infty$.

Following the attention mechanism, the transformer applies a Multi-Layer Perceptron (MLP) with a non-linear activation function $\gamma(\cdot)$ to the output of the attention mechanism $\boldsymbol{z}$, as:

$$\text{MLP}(\boldsymbol{z}) = \gamma(\boldsymbol{z}\boldsymbol{W_1} + \boldsymbol{b_1})\boldsymbol{W_2} + \boldsymbol{b_2}, \tag{2.21}$$

where MLP is a point-wise network, i.e., applied to each position in the sequence independently. $\boldsymbol{W_1} \in \mathbb{R}^{d_e \times d_{mlp}}$, $\boldsymbol{W_2} \in \mathbb{R}^{d_{mlp} \times d_e}$, $\boldsymbol{b_1} \in \mathbb{R}^{d_{mlp}}$, and $\boldsymbol{b_2} \in \mathbb{R}^{d_e}$ are the learnable weights and biases of the network, where d_{mlp} defines the inner-layer dimensionality. The non-linear



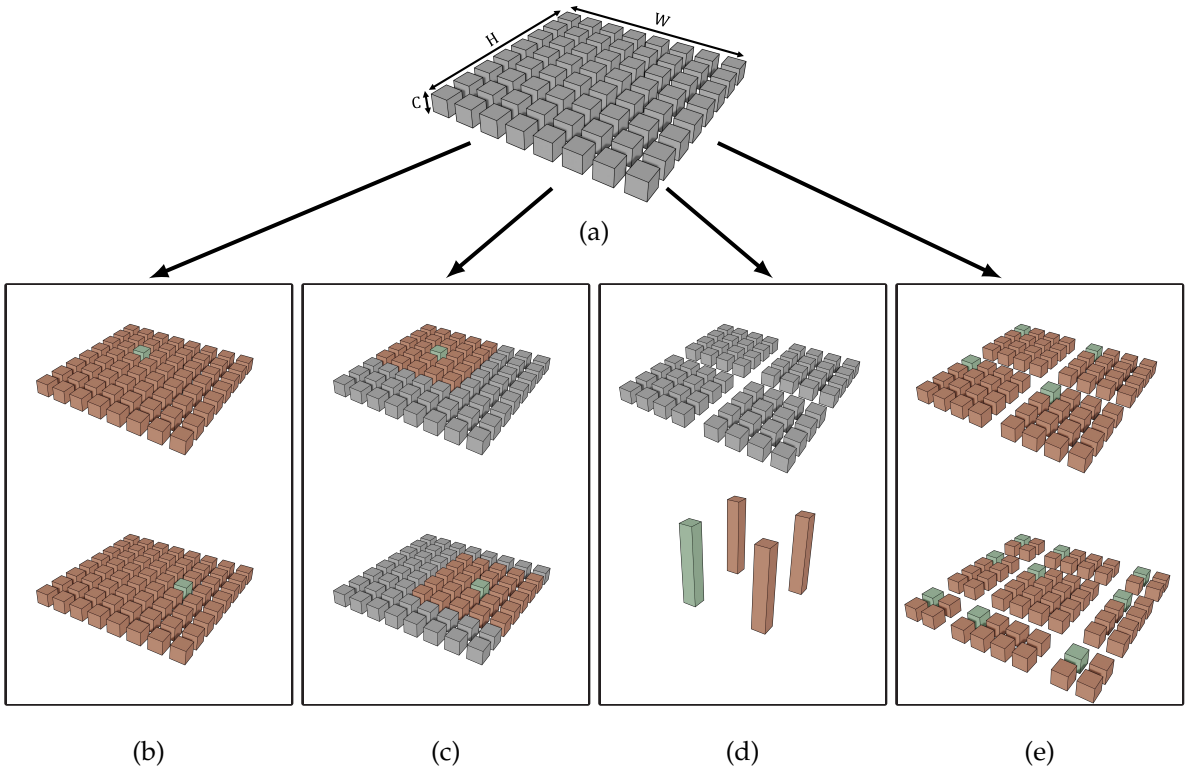
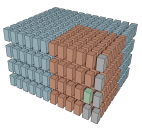


Figure 2.5: Illustration of different attention mechanisms developed for CV tasks applied to a tensor shown in (a). In each illustration, the attention is computed between the elements depicted as (■) and their neighboring elements (■). (b) shows the global attention computation for two different elements. (c) show the sliding-window attention mechanisms for two different spatial positions. In (d), the patch-wise attention mechanism in ViT [85] is illustrated, where the non-overlapping patches are flattened into a single element, and the attention is computed between the flattened patches. (e) shows the application of the (top) window and (bottom) shifted-window attention proposed in SwinT [88], where the attention is computed within each window.

activation function $\gamma(\cdot)$ is typically implemented by Rectified Linear Unit (ReLU) [128] or Gaussian Error Linear Unit (GELU) [129]. The MLP introduces non-linearity, enabling the model to learn more complex mappings beyond linear combinations of input features.

Furthermore, residual connections [130] and layer normalization [131] are also employed for both attention and MLP layers to stabilize the training process and allow for deeper architectures. Learnable or fixed positional encodings $\boldsymbol{p}_e$ are also input to transformer to introduce permutation-variance [82], [85]. The transformer network consists of multiple layers (L), where each layer has an attention mechanism followed by an MLP layer, as illustrated in Fig. 2.4.

One of the key strengths of transformers is their ability to dynamically capture long-distance dependencies in the input, which makes them effective for a wide range of CV tasks [84]–[87]. However, the computational complexity of transformers grows quadratically with the sequence length S, since attention computes the outer-product of tensors. As images typically contain a large number of pixels, directly applying a vanilla transformer to high-resolution images can be computationally infeasible. To address this, various adaptations of transformers have been developed for vision tasks, illustrated in Fig. 2.5.



The Vision Transformer (ViT) [85] mitigates the complexity by dividing an image into non-overlapping patches, treating each patch as a token in a sequence. Instead of computing a global attention for the whole input tensor, ViT [85] computes attention between these patches, reducing the overall number of tokens while still capturing global relationships within the image. This approach allows transformers to handle higher-resolution images more efficiently while benefiting from long-range dependencies. ViT [85] demonstrates strong performance, particularly in tasks such as image classification. However, it presents a trade-off between patch size and computational complexity. Tasks that rely on pixel-level details, such as image segmentation or reconstruction, require smaller patch sizes, which in turn increases the number of patches and, thus, the quadratic complexity of the attention mechanism. This inherent limitation in ViT [85] makes it less computationally efficient for tasks where precise, pixel-level decisions are necessary.

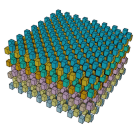
To address these issues, several alternatives to global attention have been proposed. One such approach is sliding window attention, where attention is computed locally within smaller windows [2], [83], [86]. This limits the attention span and significantly reduces the computational complexity compared to global attention. However, sliding window attention must be re-computed for each position in the sequence, which can introduce latency and reduce efficiency in real-time applications.

A more refined solution for local attention computation is the Hierarchical Vision Transformer using Shifted Windows [88], also known as the Swin Transformer (SwinT). The SwinT [88] computes the attention within non-overlapping windows, reducing the overall complexity. To approximate global attention, the windows are cyclically shifted between layers, allowing information to be shared across windows. This shifted window mechanism alternates between non-shifted and shifted attention layers, leading the model to progressively capture both local and global information while maintaining manageable computational costs. This scalable hierarchical structure enables efficient handling of high-resolution images while maintaining accuracy in pixel-level tasks.

Transformers have shown great potential in advancing learned image compression due to their high modeling capacity and flexibility in dynamically capturing both local and global structures across different regions of an image. They are particularly suited for improving various components of compression frameworks. Transformers can achieve better analysis and synthesis transforms with higher decorrelation properties for the latent tensor, and more accurate entropy and context modeling, exploiting complex statistical dependencies in the latent space, which ultimately leads to higher compression efficiency. Additionally, the recent developments in computationally efficient versions of transformers, such as ViT [85] and SwinT [88], have made it feasible to integrate these architectures into image compression pipelines.

2.3.5 Overview of Learned Frameworks

Fig. 2.6 illustrates the historical evolution of LIC architecture, focusing on the major advancements in entropy modeling. The figure highlights three key architectures: models based on a factorized entropy model, those utilizing a hyperprior network, and the most advanced



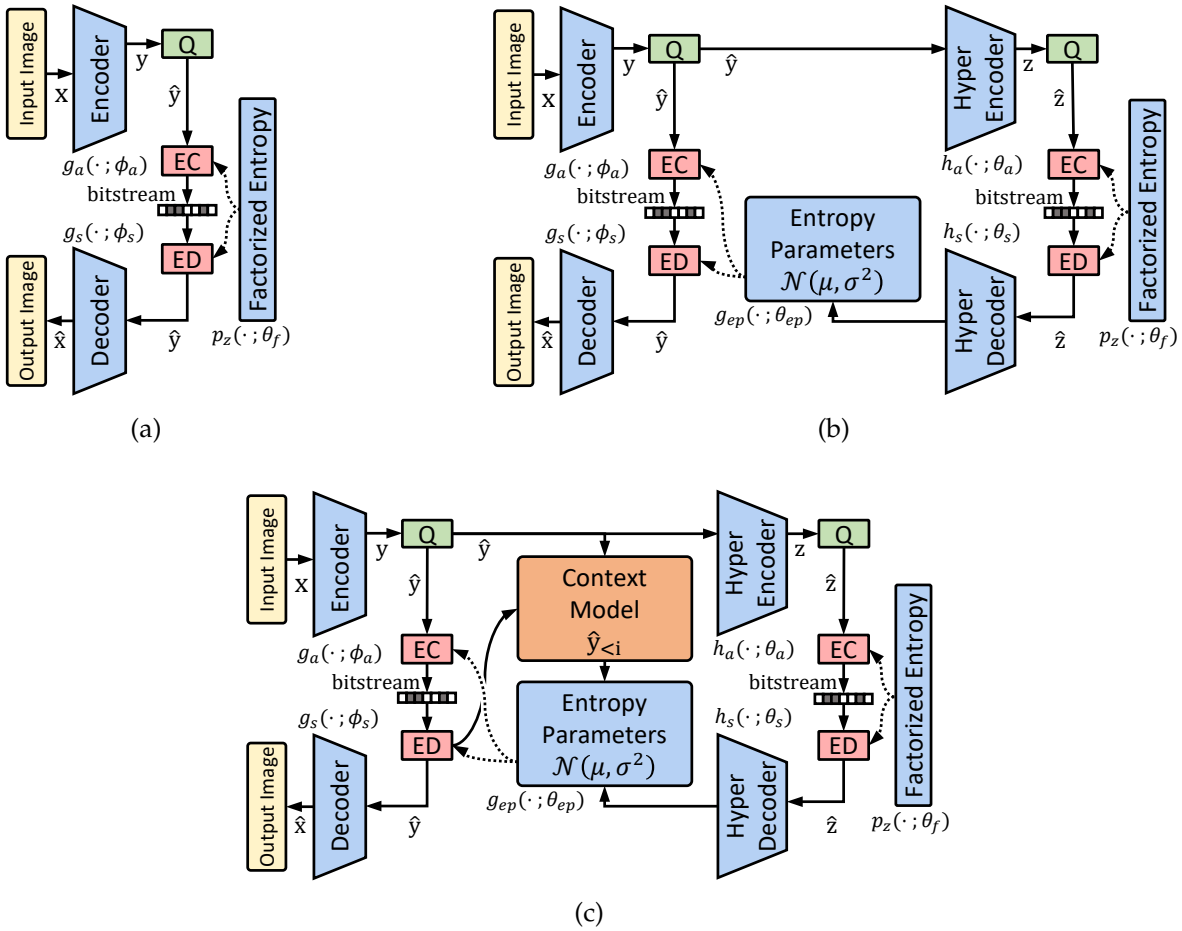
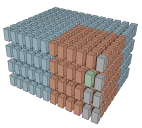


Figure 2.6: Illustration of learned image compression frameworks ordered with respect to their improvements in entropy modeling: (a) Ballé *et al.* [32] with factorized entropy model, (b) Ballé *et al.* [33] with a hyperprior network, and Minnen *et al.* [34] employing a hyperprior and context model. $g_a(\cdot;\boldsymbol{\phi_a})$ and $g_s(\cdot;\boldsymbol{\phi_s})$ stand for the analysis and synthesis transforms, and $h_a(\cdot;\boldsymbol{\theta_a})$, $h_s(\cdot;\boldsymbol{\theta_s})$ and $g_c(\cdot;\boldsymbol{\theta_c})$ denote the analysis and synthesis transforms in the hyper prior and the context model respectively. Entropy parameters block $g_{ep}(\cdot;\boldsymbol{\theta_{ep}})$ provides probability estimation of the latent tensor $\hat{\boldsymbol{y}}$ based on the given probability model, e.g., Gaussian distribution, modeled by the outputs of the hyperprior and context model. $\boldsymbol{\phi_a}$, $\boldsymbol{\phi_s}$, $\boldsymbol{\theta_a}$, $\boldsymbol{\theta_s}$, $\boldsymbol{\theta_c}$, $\boldsymbol{\theta_{ep}}$ and $\boldsymbol{\theta_f}$ correspond to the learnable parameters of each block in the non-linear transforms and the entropy model. EC and ED denote lossless entropy encoder and decoder.

models combining both a hyperprior and a context model. Each represents a significant leap in performance and complexity, as these frameworks evolved to better capture dependencies and reduce entropy. The first generation of learned models predominantly relied on factorized entropy models and hyperpriors, providing a simple yet effective method for entropy coding, which is particularly well-suited for computationally restricted hardware. In contrast, models that employ a context model offer the best compression performance, making them the preferred choice for high-performance implementations. However, they typically require higher computational resources, making them less suitable for environments with strict hardware limitations.

At a high level, the process of encoding an image in LIC starts with the analysis trans-



form, which projects the input image into a latent space. These latent variables are then quantized and entropy-coded using a learned entropy model, resulting in a compressed bitstream. On the decoder side, the process is reversed: the latent representation is entropy-decoded, and the synthesis transform reconstructs the image. These concepts were covered extensively in Sections 2.3.1 to 2.3.3, detailing the role of each module in compression. The overall lossy LIC framework from Fig. 2.6c during inference can be summarized based on Eqs. (2.9) to (2.16):

$$\hat{\boldsymbol{y}}, \tilde{\boldsymbol{y}} = Q(g_a(\boldsymbol{x}\,;\,\boldsymbol{\phi_a})), \tag{2.22a}$$

$$\hat{\boldsymbol{x}} = g_s(\hat{\boldsymbol{y}}\,;\,\boldsymbol{\phi_s}), \tag{2.22b}$$

$$\hat{\boldsymbol{z}}, \tilde{\boldsymbol{z}} = Q(h_a(\hat{\boldsymbol{y}}\,;\,\boldsymbol{\theta_a})), \tag{2.22c}$$

$$p_{\hat{\boldsymbol{y}}_{\boldsymbol{i}}}(\hat{\boldsymbol{y}}_{\boldsymbol{i}} \mid \hat{\boldsymbol{z}}) \leftarrow g_{ep}(g_c(\hat{\boldsymbol{y}}_{<\boldsymbol{i}}\,;\,\boldsymbol{\theta_c}), h_s(\hat{\boldsymbol{z}}\,;\,\boldsymbol{\theta_s})\,;\,\boldsymbol{\theta_{ep}}), \tag{2.22d}$$

$$p_{\hat{\boldsymbol{z}}}(\hat{\boldsymbol{z}}\,;\,\boldsymbol{\theta_f}) = \prod_i p(\hat{z}_i; \boldsymbol{\theta_f}), \tag{2.22e}$$

where each module contains learnable parameters $\boldsymbol{\phi}=\{\boldsymbol{\phi_a}, \boldsymbol{\phi_s}\}$ and $\boldsymbol{\theta}=\{\boldsymbol{\theta_a}, \boldsymbol{\theta_s}, \boldsymbol{\theta_f}, \boldsymbol{\theta_c}, \boldsymbol{\theta_{ep}}\}$. The fundamental goal of LIC is to find optimal parameters that minimize a rate-distortion trade-off during training, achieved through a joint loss function $\mathcal{L}$:

$$\boldsymbol{\phi}_{\text{opt}}, \boldsymbol{\theta}_{\text{opt}} = \arg\min_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}), \tag{2.23a}$$

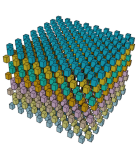
$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathbf{R}(\tilde{\boldsymbol{y}}) + \mathbf{R}(\tilde{\boldsymbol{z}}) + \lambda \cdot \mathbf{D}(\boldsymbol{x}, \hat{\boldsymbol{x}}), \tag{2.23b}$$

$$= \mathbb{E}[\log_2(p_{\tilde{\boldsymbol{y}}}(\tilde{\boldsymbol{y}} \mid \hat{\boldsymbol{z}}))] + \mathbb{E}[\log_2(p_{\tilde{\boldsymbol{z}}}(\tilde{\boldsymbol{z}}))] + \lambda \cdot \mathbf{D}(\boldsymbol{x}, \hat{\boldsymbol{x}}), \tag{2.23c}$$

where $\mathbf{D}(\cdot)$ denotes the distortion between the input and reconstructed images, and $\mathbf{R}(\cdot)$ represents the bitrate required to encode the latent variables and side information. The Lagrange multiplier λ regulates the trade-off between distortion and rate. End-to-end training of LIC frameworks uses backpropagation and gradient descent to minimize this loss function, which simultaneously optimizes each component, such as the analysis and synthesis transforms, the entropy model (including the hyperprior and context model), and the quantization process.

Over the years, there have been significant improvements in each of these components, resulting in progressively more efficient learned codecs:

- **Analysis and Synthesis Transforms:** The early LIC models [32]–[34] used simple CNNs for the analysis and synthesis transforms with GDN [120] activation function. However, more recent architectures [38], [41], [45], [47], [48], [56], [71], [75] have investigated alternative non-linearities such as ReLU [128], LeakyReLU [132], and GELU [129], and more complex CNN-based architectures such as residual layers [130] and attention modules [133], [134]. Transformer-based transforms have also been explored in multiple studies [59], [62], [63], [68] to implement more sophisticated attention mechanisms, capturing more long-distance dependencies and improving coding efficiency.



- **Hyperprior:** Parallel to the improvements in analysis and synthesis transforms, the hyperprior network has also seen numerous refinements. Multiple studies have investigated more sophisticated hyperprior architectures with deeper and wider CNN-based networks with different non-linearities [38], [41], [45], [47], [48], [56], [71], and transformer-based models [46], [59], [61]–[63], [68]. Additionally, other studies proposed multiple sequential hyperprior networks [73], [74] to provide multiple side-information with different levels of detail, and parallel hyperprior networks [72] to signal variable quantization steps as additional side information.

- **Context Modeling:** Context models can substantially enhance coding efficiency without adding signaling overhead but introduce computational costs due to their autoregressive nature. In the literature, numerous model architectures have been explored to improve the context model's coding performance and its computational complexity. These architectures extend context modeling beyond basic masked convolutions, which are limited to capturing local spatial dependencies. More advanced techniques enable the exploration of broader and more complex dependencies in the latent space, and efficiently reduce the required autoregressive steps. A detailed discussion of these architectures and their efficient implementations is provided in Chapter 4 and Chapter 5.

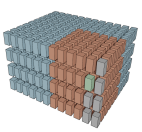
- **Conditional Probability Model:** Several studies [34], [35], [75] explored the choice of conditional probability models $p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$, such as Gaussian distributions [34], [35], Logistic distributions [34], and Laplacian distributions [34], [75]. Other studies [39], [47], [135] investigated more complex probability models such as the Asymmetric Gaussian Model (AGM) [39] and mixture models like the Gaussian Mixture Model (GMM) [47], Asymmetric Gaussian Mixture Model (AGMM) [1], [2], Logistic Mixture Model [135], and even a combined mixture of Gaussian-Laplacian-Logistic mixtures [76].

Overall, learned image compression frameworks have evolved into highly sophisticated systems with complex neural network architecture, advanced entropy modeling, and improved optimization strategies. Each component in non-linear transform coding and entropy modeling has seen continual advancements, driving the field toward more efficient and high-performance compression algorithms.

2.3.6 Online Rate-Distortion Optimization

Autoencoder-based frameworks, such as LIC, are typically trained on large datasets to perform well on average, leading to globally optimal but locally sub-optimal results for specific images. This discrepancy, referred to as the amortization gap [136], arises because the model parameters are optimized across the entire training set, sacrificing per-image performance in favor of generalizability.

As discussed in Section 2.2.6, traditional codecs face a similar issue, where a generic configuration may not be ideal for all images. To address this, traditional codecs employ an encoder that searches for the best configuration from a set of encoding parameters to achieve



an optimal rate-distortion balance. In LIC, these "configurations" are analogously embedded in the model's parameters $\{\boldsymbol{\phi}, \boldsymbol{\theta}\}$. Different configurations in TIC frameworks, such as quantization levels or transform types, correspond to different rate-distortion trade-offs. Similarly, different model parameters in an LIC framework result in different compression performance. However, unlike traditional frameworks, LIC models contain millions of parameters. Optimizing the entire set of model parameters for each individual image would be computationally expensive and impractical due to the signaling overhead required to transmit the updated parameters.

To address this challenge, several online Rate-Distortion Optimization (oRDO) methods [53]–[55], [137] have been introduced in LIC frameworks. Instead of re-optimizing the network for every image, oRDO focuses on adjusting the latent representations $\{\boldsymbol{y}, \boldsymbol{z}\}$ during encoding. The oRDO process starts by passing the input image $\boldsymbol{x}$ through the encoder network to generate initial latent representations $\boldsymbol{y}$ and side information $\boldsymbol{z}$, while keeping the network parameters fixed. Once the latent variables are initialized, they are iteratively refined through optimization to minimize the Lagrangian cost function:

$$\boldsymbol{y}_{\text{opt}}, \boldsymbol{z}_{\text{opt}} = \underset{\boldsymbol{y},\boldsymbol{z}}{\arg\min} \left[D(\boldsymbol{x}, g_s(Q(\boldsymbol{y}))) + \lambda R(Q(\boldsymbol{y}), Q(\boldsymbol{z})) \right], \tag{2.24}$$

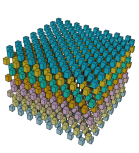
where $\boldsymbol{y}_{\text{opt}}$ and $\boldsymbol{z}_{\text{opt}}$ correspond to the optimal latent variables, which are quantized and encoded into the bitstream after optimization. The oRDO can refine the latent tensor with backpropagation and gradient descent by updating the residuals [53], [137], adjusting the quantization step for each latent element [55], or even deciding whether to round elements up or down [54].

While oRDO enables image-specific optimization and significantly narrows the amortization gap, it comes with increased computational complexity on the encoder side. However, the decoder remains unaffected, maintaining a low complexity similar to the one in TIC frameworks. Given the higher complexity of the encoding process, oRDO is most suited for LIC frameworks that use simpler entropy models with low-complexity context models, or even without applying any context modeling, such as in [32], [33]. A comprehensive example of an oRDO algorithm based on [53], applied to an efficient context model, is provided in Section 5.5.

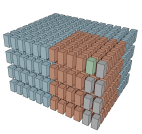
2.4 Chapter Summary

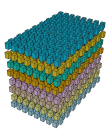
This chapter provided an overview of the image compression techniques, with a focus on lossy TIC and LIC methods. Core components such as transform coding, quantization, and entropy coding were explained, alongside a review of widely used TIC frameworks such as JPEG [16], HEVC [24], and VVC [25], as well as learned methods Ballé *et al.* [32], Ballé *et al.* [33] and Minnen *et al.* [34]. A detailed comparison was drawn between TIC and LIC techniques, highlighting the strengths of learned methods.

Special attention was given to recent advancements in transformers, emphasizing their efficiency in handling complex image data for computer vision tasks. The chapter also introduced context modeling, providing a foundational understanding of its critical role in LIC



methods. However, more in-depth analysis and exploration of context modeling, especially transformer-based approaches, is reserved for Chapters 4 and 5, where these concepts will be pivotal to the proposed innovations.





Chapter 3

Performance Evaluation Methodology

This chapter outlines the methodology used to evaluate the performance and complexity of traditional and learned image compression frameworks. Section 3.1 introduces the testing and training datasets, offering a detailed analysis of their content complexity and emphasizing the importance of dataset selection in ensuring fair and effective model evaluation and training. Section 3.2 defines the metrics used for performance and complexity evaluation, covering key benchmarks such as rate-distortion performance, perceptual quality, and complexity metrics that account for the computational and runtime demands of different algorithms. Section 3.3 details the performance evaluation setup, specifying the hardware, software, and frameworks used to guarantee consistent and reproducible benchmarking across various compression methods. This consistent setup ensures reliable results, forming a solid foundation for the evaluations and comparisons presented in subsequent chapters. Finally, Section 3.4 concludes the chapter with a summary.

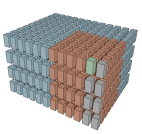
3.1 Testing and Training Datasets

3.1.1 Test Datasets

The selection of an appropriate test dataset is of great importance for having a meaningful evaluation of both TIC and LIC models. Test datasets must represent a broad spectrum of scenarios to evaluate whether the models generalize well and perform robustly across different image types. An inadequate or biased dataset could lead to skewed results, potentially overestimating or underestimating a model's actual capabilities. To achieve accurate assessments, it is essential that the test datasets include diverse image content types. Moreover, the complexity of the images should reflect real-world use cases to ensure that the model performs efficiently in various scenarios, covering different resolutions, formats, and a wide range of textures, colors, and structures.

In this work, the following test datasets are used to represent a variety of content properties and ensure robust evaluation of the proposed models:

1. **Kodak Dataset [138]:** The Kodak Lossless True Color Image Suite, commonly referred



to as the Kodak Dataset, consists of 24 uncompressed true-color images in 24-bit PNG format, each with a resolution of either 768×512 or 512×768 pixels. The images cover real-world scenes such as nature, buildings, outdoor environments, and human portraits, and are widely used for benchmarking image compression algorithms. Although historically popular, the dataset's small size and limited diversity reduce its effectiveness in evaluating models across a broad range of conditions. The dataset is available for download at http://r0k.us/graphics/kodak/.

2. **Tecnick Dataset [139]:** The Tecnick Dataset is a subset of the TESTIMAGES archive, which is a large collection of images designed for quality assessment of displays and image processing techniques. This subset includes 100 PNG images, each with a resolution of 1200×1200 pixels and 24-bit color depth. The images cover a wide range of real-world scenes, including both indoor and outdoor environments, animal portraits, and challenging lighting conditions such as sunsets and low-exposure scenes. The dataset's diversity and higher resolution make it a popular choice for image compression research. The dataset is available at https://sourceforge.net/projects/testimages/files/OLD/OLD_SAMPLING/.

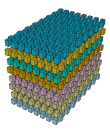
3. **CLIC Test Dataset [140]**: The CLIC dataset was created for the Challenge on Learned Image Compression 2020 (CLIC) [140], and is used to benchmark rate-distortion performance in lossy image compression. The test dataset consists of two subsets: CLIC Professional (CLIC-P), which contains 250 images captured with professional cameras, and CLIC Mobile (CLIC-M), which includes 178 images taken with mobile devices, with all images from both subsets having 24-bit RGB PNG format. The dataset offers a broad variety of scenes, such as indoor and outdoor environments, night scenes, low-exposure conditions, and images of humans, animals, and nature. Together, the subsets provide a comprehensive evaluation of compression models in diverse, real-world conditions. The images range from 0.5 MP to 5 MP in resolution, with 97 unique resolutions and 75 different aspect ratios. The datasets are available online:

   - CLIC-P: https://data.vision.ee.ethz.ch/cvl/clic/test/CLIC2020Professional_test.zip
   - CLIC-M: https://data.vision.ee.ethz.ch/cvl/clic/test/CLIC2020Mobile_test.zip

3.1.2 Training Datasets

The selection of an appropriate training dataset is critically important for achieving high performance in learning-based models. As numerous studies have emphasized [141]–[143], the quality, representativeness, and diversity of the training data significantly influence the learning based model's performance. Poorly selected datasets can lead to suboptimal models that exhibit decreased accuracy, overfitting, or failure to generalize to real-world conditions.

In LIC frameworks, robust datasets are essential to ensure that models can handle diverse content, varying resolutions, and structural and entropy-based complexities. Moreover, to enable fair comparisons between different models, it is crucial that all models are trained on



the same dataset. A more sophisticated model trained on a poor-quality dataset may underperform compared to a simpler model trained on a higher-quality dataset. Therefore, using consistent and representative datasets is key to accurately determining performance dominance between models. In this work, the following training datasets are utilized to train the proposed models discussed in Chapters 4 and 5:

1. **Vimeo90K [144]**: This dataset comprises 89,800 video shots sourced from Vimeo, captured using professional cameras across diverse scenes, including both indoor and outdoor environments. Only videos without inter-frame compression were selected to ensure that each frame is compressed independently, preventing distortions introduced by video codecs. Each frame is resized to a resolution of 256×448 and stored as a separate RGB 24-bit PNG file. This diversity makes the dataset particularly suitable for training models in video and image compression tasks. Additionally, the open-source CompressAI library [145] has implemented and trained many state-of-the-art LIC models using the *triplets subset* of Vimeo90K [144], which contains 73,171 3-frame sequences extracted from 14,777 video clips. Replication studies using this dataset have closely reproduced results from the original works, making Vimeo90K [144] a reliable standard for fair comparisons between different compression models.

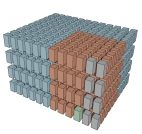
2. **COCO [146]**: The Microsoft Common Objects in Context (COCO) dataset is a large-scale dataset primarily designed for object detection, segmentation, and captioning tasks. The training subset, *train2017*, consists of 118,287 images featuring a wide range of resolutions and everyday scenes, including indoor and outdoor environments, objects, human activities, and complex backgrounds. Due to its rich content, COCO [146] has been used in recent LIC studies [57], [66]–[68], demonstrating its applicability beyond object detection.

This work primarily utilizes the Vimeo90K [144] dataset to ensure a fair comparison of the proposed models with state-of-the-art models. Additionally, the complexity of the dataset and the impact of training with the COCO [146] dataset on model performance are also investigated in Sections 3.1.3 and 5.7, addressing specific limitations of the Vimeo90K [144].

3.1.3 Complexity Assessment of Datasets

An ideal image dataset for image compression evaluation should encompass scenes with varied content complexity. Various methods [147]–[155] exist which evaluate different aspects of dataset complexity, including color diversity, structural and semantic richness, and image dimensionality. In this work, the following metrics are used to quantify the complexity of both the test and training datasets, focusing on entropy, structural, and semantic complexities, as well as dimensionality analysis:

1. **Color Variety ($\mathcal{H}_C$) [152]:** This metric measures the diversity of colors within an image by calculating the entropy based on the distribution of color values. Images with a large number of uniformly distributed unique colors exhibit higher entropy, posing



challenges for compression algorithms. The metric is formulated as:

$$\mathcal{H}_C = -\sum_{i=0}^{N_C-1} n_i \log \frac{n_i}{HW}, \tag{3.1}$$

where N_C represents the number of unique colors within the image, and n_i is the number of occurrences of the i-th unique color.

2. **Average Entropy of Gray-Level Co-occurrence Matrix ($\mathcal{H}_{GLCM}$) [153], [154]:** This metric captures texture and spatial relationships by analyzing how often pairs of grayscale pixel values occur at specific spatial distances, called offsets. It is particularly useful for evaluating the complexity of natural textures and fine details in images. For a normalized co-occurrence matrix $G_k \in [0,1]^{N\times N}$, with K different grayscale values at offset k, the metric is expressed as:

$$\mathcal{H}_{GLCM} = -\frac{1}{M}\sum_{k=0}^{K-1}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} G_{k,i,j}\log G_{k,i,j}, \tag{3.2}$$

where M represents the number of different offsets used. Offsets of $k \in \{1,4,8,16,32\}$ are employed in this work, following [155]. For a detailed explanation of GLCM computation, refer to [156].

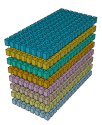
3. **Mean Spatial Information (SI_{mean}) [149], [150]:** This metric measures the average edge energy in an image, indicating structural complexity. It computes the average gradient magnitude of the luminance channel using horizontal and vertical Sobel operators [157]. Higher values indicate more complex structures and edges in the image. SI_{mean} is formulated as:

$$SI_{mean} = -\frac{1}{HW}\sum_{i=0}^{H-1}\sum_{j=0}^{W-1}\sqrt{\boldsymbol{s_h}(Y)_{i,j}^2 + \boldsymbol{s_v}(Y)_{i,j}^2}, \tag{3.3}$$

where $\boldsymbol{s_h}(\boldsymbol{Y})$ and $\boldsymbol{s_v}(\boldsymbol{Y})$ are the horizontal and vertical gradient images, and the luminance channel Y of RGB images is computed using the BT.709 standard [94].

4. **Minimum Description Length Clustering (MDLC) [155]:** This metric identifies the minimum number of clusters required to represent an image by solving an optimization problem across multiple hierarchical levels. At the first level, clustering is applied at the pixel level to generate a pixel-wise segmentation map. The subsequent levels perform clustering on patches of the previous level's map. This metric is optimized for measuring so-called "meaningful complexity", introduced in [155], aligning with human perception—assigning higher complexity to scenes with multiple objects, a variety of edges, and rich structures, while assigning the lowest complexity to white noise and repetitive patterns.

5. **Image Dimensionality:** The dimensionality of an image is assessed by analyzing its resolution and aspect ratio. A dataset with diverse resolutions and aspect ratios is



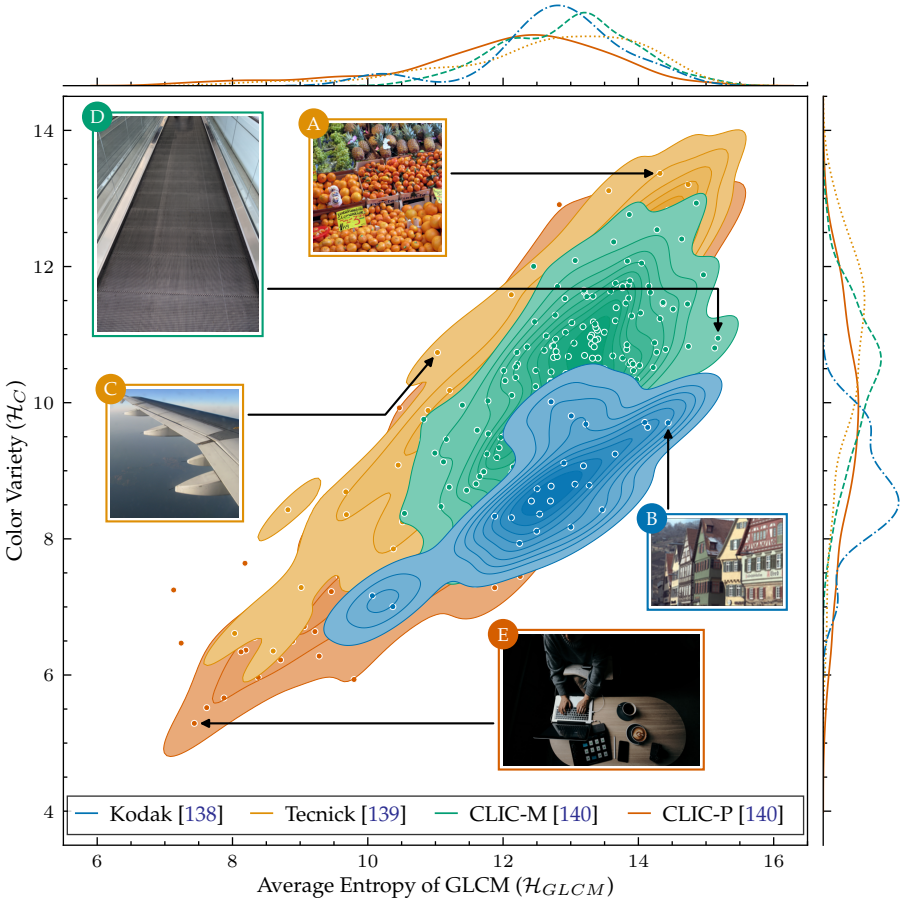
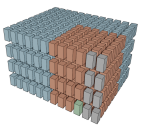


Figure 3.1: Density estimation of images from the test datasets, including Kodak [138], Tecnick [139], and CLIC-P/-M [140], with respect to content complexity metrics: Average Entropy of Average Entropy of Gray-Level Co-occurrence Matrix ($\mathcal{H}_{GLCM}$) [153], [154] and Color Variety ($\mathcal{H}_C$) [152]. Example images, labeled A, B, C, D, and E, are shown to illustrate how each metric works. Images A and B have similar $\mathcal{H}_{GLCM}$ but differ in $\mathcal{H}_C$. Conversely, images C and D share a similar color palette and $\mathcal{H}_C$, yet differ in $\mathcal{H}_{GLCM}$. Image A has the highest score in $\mathcal{H}_C$, while image D has the highest score in $\mathcal{H}_{GLCM}$. Image E has the lowest score on both metrics.

important for evaluating model robustness. Low-resolution images often lose more data during compression, leading to visible artifacts, especially when LIC algorithms use downsampling layers. Conversely, high-resolution images present challenges due to their higher frequency content, making them harder to compress, while the larger number of pixels increases computational complexity. The diversity of aspect ratios is also crucial, as certain content types (e.g., large human faces in portrait format) tend to appear in specific ratios. Additionally, horizontal and vertical frequency distributions can vary with aspect ratio, further influencing compression performance.

Figs. 3.1 to 3.3 illustrate the image complexity metrics across four test datasets: Kodak [138], Tecnick [139], and CLIC-P/-M [140]. Fig. 3.1 compares these datasets using GLCM entropy ($\mathcal{H}_{GLCM}$) and color variety ($\mathcal{H}_C$), and includes exemplary images, labeled A, B, C, D, and E, to help visualize how these metrics reflect image content. For instance, the images A and B share similar $\mathcal{H}_{GLCM}$, yet the image A has significantly higher color variety,



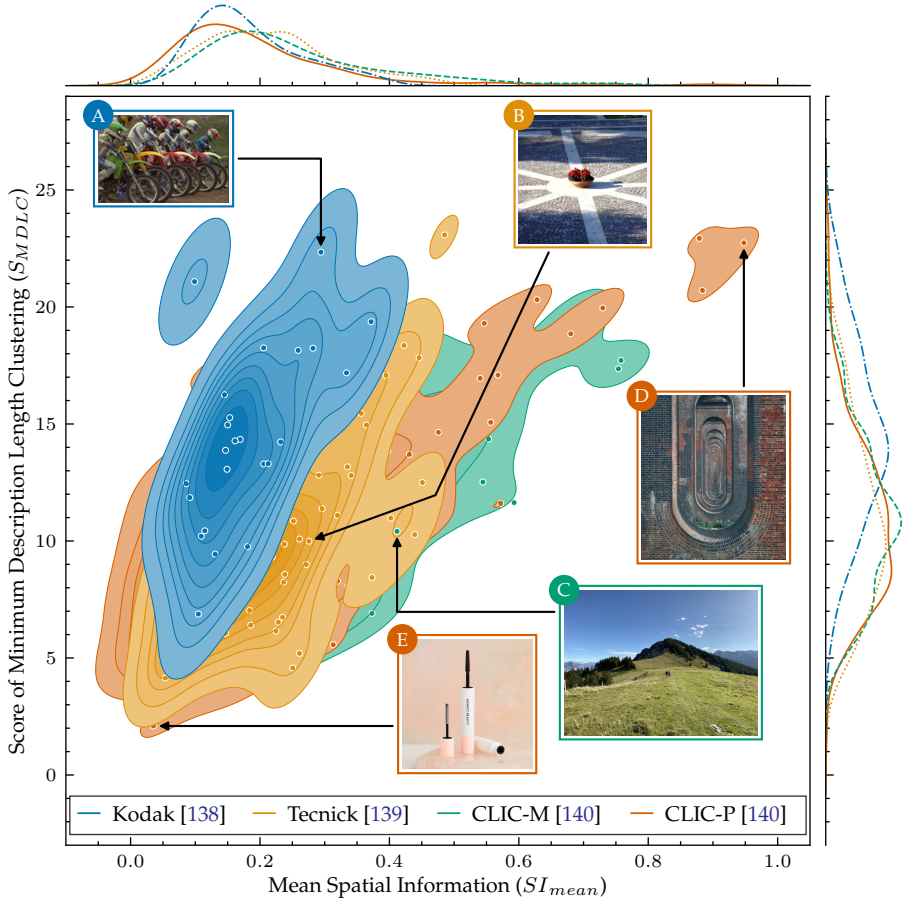
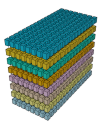


Figure 3.2: Density estimation of images from the test datasets, including Kodak [138], Tecnick [139], and CLIC-P/-M [140], with respect to content complexity metrics: Mean Spatial Information (SI_{mean}) [149], [150] and Score of Minimum Description Length Clustering (S_{MDLC}) [155]. Example images, labeled A, B, C, D, and E, are provided to demonstrate the behavior of each metric. While images A and B share a similar SI_{mean}, they differ in S_{MDLC}. Similarly, image pairs B, C and A, D have comparable S_{MDLC} values but differ in SI_{mean}. Image D achieves the highest scores in both metrics, whereas image E exhibits the lowest values in both.

making it appear more colorful despite comparable textural complexity. The images C and D have similar $\mathcal{H}_C$, as both feature predominantly cold color palettes. The image D exhibits the highest $\mathcal{H}_{GLCM}$ of all examples, depicting an escalator floor with intricate details and high-frequency components. Conversely, the image E has the lowest $\mathcal{H}_{GLCM}$ and $\mathcal{H}_C$, illustrating a dark, low-exposure scene with minimal color and content, making it the simplest example.

On average, the Kodak dataset [138] shows lower color variety and moderate textural complexity, indicating simpler image content. CLIC-M [140] exhibits slightly higher average $\mathcal{H}_C$ and $\mathcal{H}_{GLCM}$ compared to Kodak [138], indicating more intricate textures and colors. However, CLIC-M's narrower complexity range compared to Tecnick [139] or CLIC-P [140] suggests a more consistent level of difficulty for compression. Tecnick [139] and CLIC-P [140] datasets cover a broad spectrum of scene complexity according to those metrics—with Tecnick [139] showing highest average complexity in terms of both metrics, and CLIC-P [140] having the highest variance.



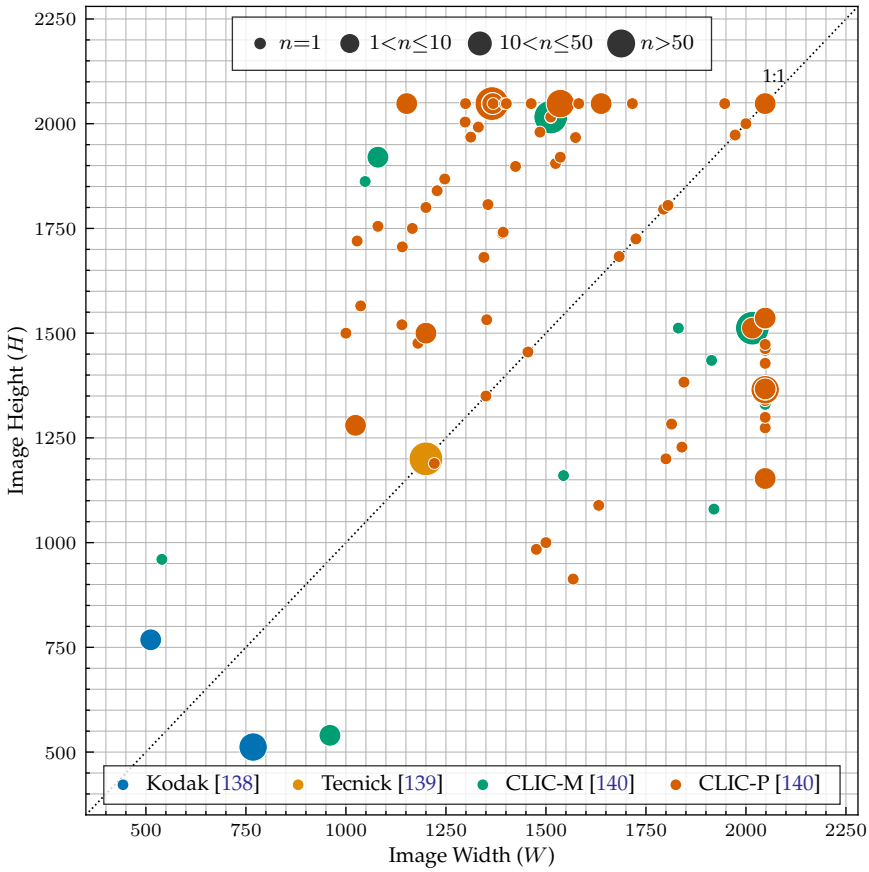
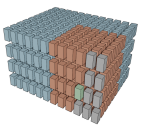


Figure 3.3: Resolution of images from the test datasets, including Kodak [138], Tecnick [139], and CLIC-P/-M [140]. Images with identical height (H) and width (W) are represented as a single point, with the size of each point corresponding to the number of samples (n). The diagonal line indicates a 1:1 aspect ratio (square resolution). CLIC-P [140] displays the widest range in both resolution and aspect ratio, while Tecnick [139] comprises only images with a resolution of 1200×1200. The Kodak [138] dataset contains the smallest images in terms of resolution."

Fig. 3.2 compares complexity using mean spatial information (SI_{mean}) and the Minimum Description Length Clustering score (S_{MDLC}), along with exemplary images, labeled A, B, C, D, and E, to increase metric explainability. Images A and B exhibit similar SI_{mean} values, indicating comparable edge variety. However, image B has a much lower S_{MDLC} due to its repetitive pattern, requiring fewer groups to be clustered. The images A and D share similar S_{MDLC}, yet the image D has a significantly higher SI_{mean} due to its complex edge patterns (e.g., an infinity mirror-like brick arrangement), which increases the number of clusters required for MDLC. In contrast, the images C and B have similar, moderate S_{MDLC} values since the pavement pattern in B and the landscape in C can be segmented relatively easy. However, the grassland in the image C increases spatial complexity due to much complex edge structures. The image E is the simplest in terms of both SI_{mean} and S_{MDLC}, with just a few simple objects against a plain background, making it the least complex example.

On average, the Kodak dataset [138] has the highest average S_{MDLC}, indicating complex structural segmentation with relatively low spatial complexity. Recall that MDLC metric assigns the lowest score to white noise, Kodak's high score indicates lower noise, mak-



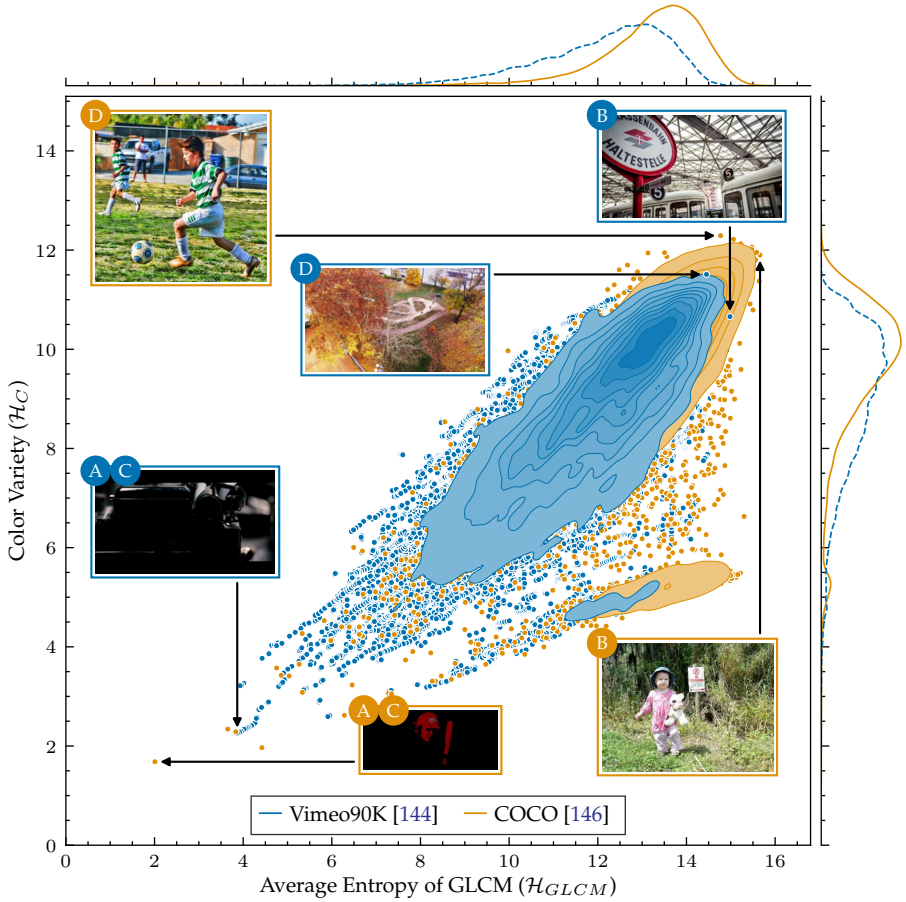
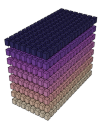


Figure 3.4: Density estimation of images from the training datasets, including Vimeo90K [144] and COCO [146], with respect to content complexity metrics: Average Entropy of Average Entropy of Gray-Level Co-occurrence Matrix ($\mathcal{H}_{GLCM}$) [153], [154] and Color Variety ($\mathcal{H}_C$) [152]. Compared to Vimeo90K [144], COCO [146] exhibits higher complexity on average and includes more extreme samples across both metrics. Example images, labeled A, B, C, and D, illustrate the extremes within each dataset. Images labeled A and C have the lowest $\mathcal{H}_{GLCM}$ and $\mathcal{H}_C$, respectively, while images labeled B have the highest $\mathcal{H}_{GLCM}$ and those labeled D exhibit the highest $\mathcal{H}_C$.

ing compression less challenging. However, the high S_{MDLC} also suggests the retention of meaningful structural segments, making Kodak dataset [138] valuable for the tasks such as compression for machine vision [69], [158], where the compressed representations are used for different downstream tasks. The Tecnick [139] and CLIC-P/-M [140] datasets span a wide range of both metrics, with while CLIC-P/-M datasets [140] covering more extreme ranges in spatial complexity.

Fig. 3.3 illustrates the dimensionality of images in each dataset, plotting image width and height, with scatter point sizes representing the number of samples at each resolution. The Kodak dataset [138] contains fewer images, clustered at lower resolutions with two aspect ratios, indicating less complexity in size. The Tecnick [139] dataset features higher-resolution images, though they all share the same resolution and aspect ratio. The CLIC-M [140] dataset covers a wider range of resolutions, from low to high, with a moderate aspect ratio variety. The CLIC-P [140] dataset shows the highest complexity, covering a broad range of resolutions



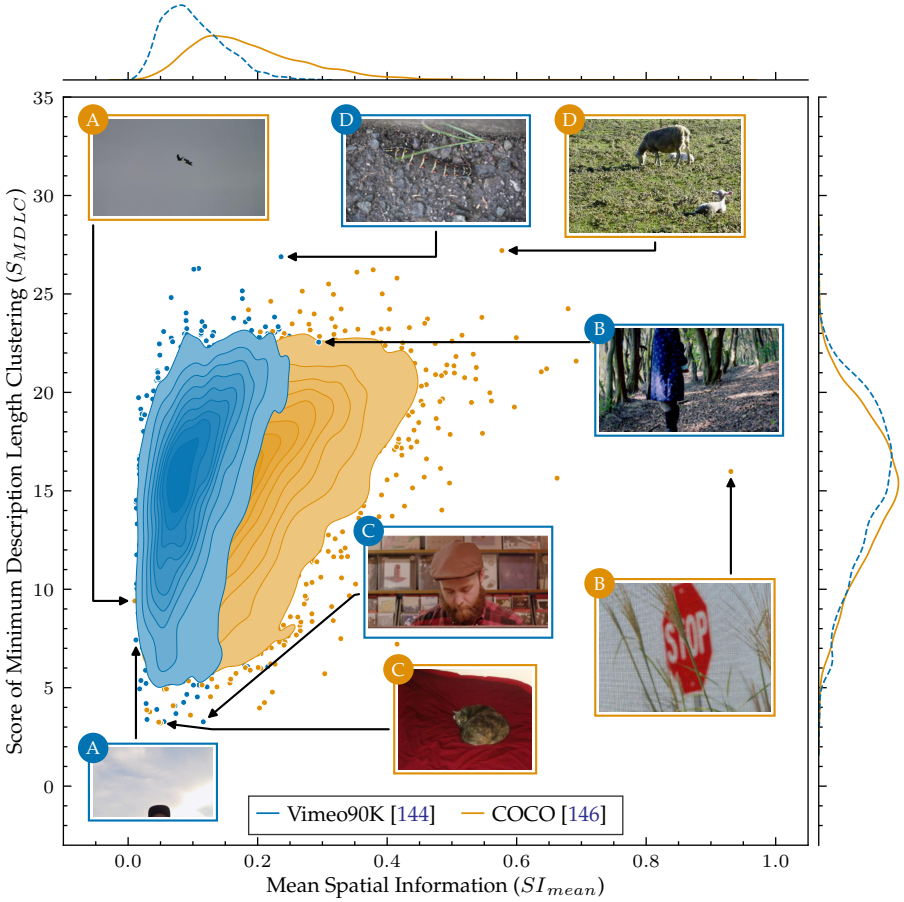
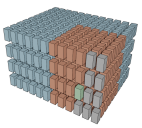


Figure 3.5: Density estimation of images from the training datasets, including Vimeo90K [144] and COCO [146], with respect to content complexity metrics: Mean Spatial Information (SI_{mean}) [149], [150] and Score of Minimum Description Length Clustering (S_{MDLC}) [155]. Compared to Vimeo90K [144], COCO [146] exhibits higher average spatial complexity and greater variety in SI_{mean}, with more extreme samples. Both datasets show similar complexity in terms of S_{MDLC}. Example images, labeled A, B, C, and D, illustrate the extremes within each dataset. Images labeled A and C have the lowest SI_{mean} and S_{MDLC}, respectively, while images labeled B exhibit the highest SI_{mean} and those labeled D show the highest S_{MDLC}.

and aspect ratios, including many high-resolution images that demand more computational resources.

Figs. 3.4 to 3.6 provide a similar analysis for the training datasets: Vimeo90K [144] and COCO [146]. Fig. 3.4 compares both datasets using $\mathcal{H}_{GLCM}$ and $\mathcal{H}_C$. The COCO [146] dataset has higher average complexity than Vimeo90K [144], exhibiting more challenging content with higher color variety and textural complexity. This complexity makes COCO [146] particularly useful for training models aimed at improving performance on test datasets such as Tecnick [139] and CLIC-P/-M [140] that have similarly high average complexity.

Fig. 3.5 compares the training datasets based on SI_{mean} and S_{MDLC}. Both datasets have similar average S_{MDLC}, suggesting that they contain images with a comparable number of meaningful segments, making them equally useful for machine vision tasks relying on compressed representations. However, the COCO [146] dataset has higher average SI_{mean} and greater variance, indicating more complex textures and spatial details. The dataset contains



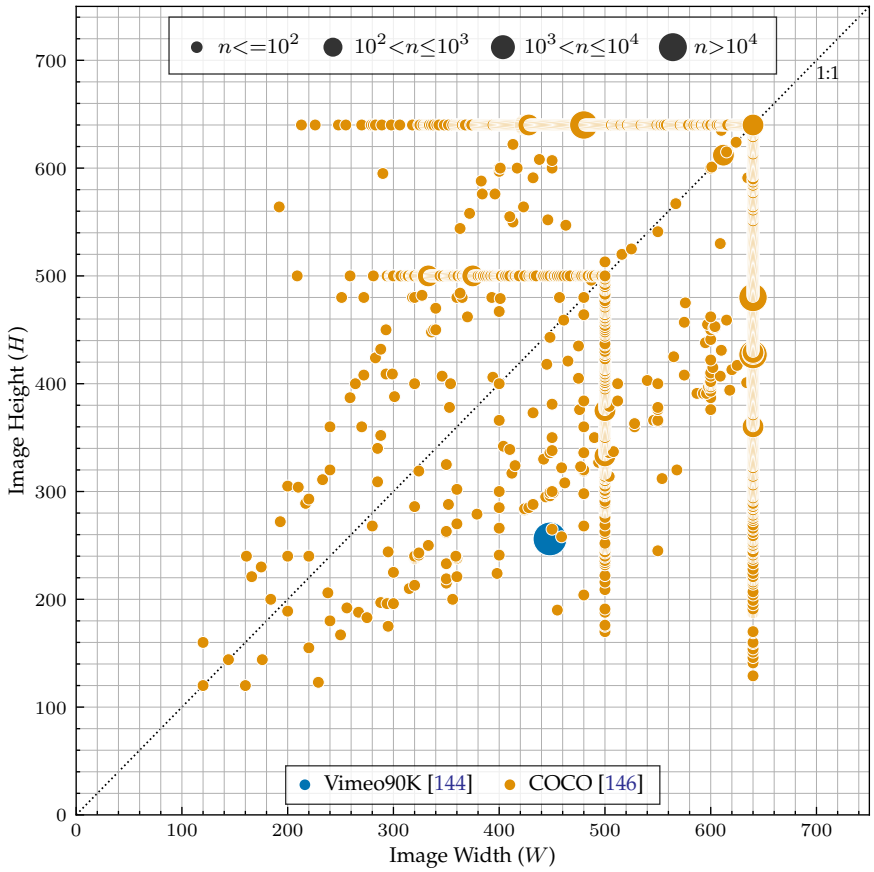
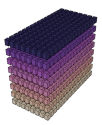


Figure 3.6: Resolution of images from the training datasets, including Vimeo90K [144] and COCO [146]. Images with the same height (H) and width (W) are clustered as a single point, with the size of each point representing the sample size (n). The diagonal line corresponds to a 1:1 aspect ratio (square resolution). COCO [146] has large variety of images with different resolution and aspect ratios, exhibiting an asymmetrical distribution around 1:1 line, whereas Vimeo90K [144] comprises only images with a resolution of 256×448.

images approaching the spatial complexity found in test datasets, such as Tecnick [139] and CLIC-P/-M [140].

Figs. 3.4 and 3.5 contain exemplary images, labeled A, B, C, and D, which covers the extremities of each dataset in terms of the complexity metrics. The images labeled as A and C from each dataset shows the images with the lowest score according to the metric displayed in x and y-axes, respectively. Similarly, the images B and D have the highest score on the metrics of the axes x and y, respectively.

Fig. 3.6 compares the dimensionality of images in the training datasets. Vimeo90K [144] contains only one resolution and aspect ratio, making it uniform and less varied in terms of size. In contrast, COCO [146] features a wide variety of resolutions and aspect ratios, with image heights up to 2.4x larger and widths up to 1.4x larger than those in Vimeo90K [144]. This variety makes COCO [146] more useful for training, as it exposes the model to higher-resolution images, whose benefits will be discussed in Section 5.7.



3.2 Performance and Complexity Metrics

3.2.1 Performance Metrics

The performance of an image compression algorithm is determined by two key factors: the size of the compressed data to be transmitted (measured, e.g., as bitrate) and the quality or distortion of the reconstructed image (measured as the difference from the original image). This relationship is often referred to as rate-distortion (RD) performance throughout this thesis. Since the majority of benchmark test datasets contain 3-channel RGB images (see Section 3.1), the bitrate is computed as bits per pixel (Bpp)[†], formulated as:

$$\text{Bpp} = \frac{N_{bits}}{N_{samples}} = \frac{N_{bits}}{HW}, \tag{3.4}$$

where N_{bits} is the total number of bits in the bitstream and $N_{samples}$ refers to the total number of pixel samples in the image, calculated as the product of the image height (H) and width (W). Notably, Bpp is computed for each pixel coordinate without differentiating between the number of color channels.

To assess the quality of the reconstructed image, either objective quality metrics such as Peak Signal-to-Noise Ratio (PSNR), or perceptually optimized metrics modeling the subjective quality, such as the Structural Similarity Index Measure (SSIM) [159] and Multi-scale Structural Similarity Index Measure (MS-SSIM) [160] are used in TIC frameworks and standardization activities [16], [24], [25], [91], [161]. Additionally, the JPEG AI standardization activity [79] explores several other perceptually optimized metrics, including Visual Information Fidelity (VIF) [162], Feature Similarity Index Measure (FSIM) [163], Normalized Laplacian Pyramid Distance (NLPD) [164], Information Content Weighted MS-SSIM (IW-MS-SSIM) [165], Video Multimethod Assessment Fusion (VMAF) [166], and a modified PSNR metric based on Hue Saturation Value (PSNR-HSV) [167]. While these advanced perceptual metrics are actively being explored, state-of-the-art LIC frameworks primarily rely on PSNR and MS-SSIM [160] in the RGB color space for quality evaluation. They also provide publicly available models trained for these two metrics. Therefore, to ensure fair comparisons, this thesis focuses on PSNR and MS-SSIM [160] metrics for both objective and perceptually optimized quality assessments.

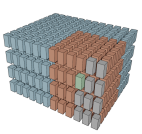
Given an original image $\boldsymbol{x} \in \mathbb{N}^{H\times W\times 3}$ and its reconstructed version $\hat{\boldsymbol{x}} \in \mathbb{N}^{H\times W\times 3}$ in the RGB space, the PSNR in decibel (dB) is formulated as:

$$\text{PSNR}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = 10 \cdot \log_{10}\left(\frac{2^B - 1}{\text{MSE}(\boldsymbol{x}, \hat{\boldsymbol{x}})}\right), \tag{3.5a}$$

$$\text{MSE}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{3HW}\sum_{c=0}^{2}\sum_{i=0}^{H-1}\sum_{j=0}^{W-1}[\boldsymbol{x}_{i,j,c} - \hat{\boldsymbol{x}}_{i,j,c}]^2, \tag{3.5b}$$

where $\text{MSE}(\boldsymbol{x}, \hat{\boldsymbol{x}})$ represents the mean-squared error. B refers to the bit depth used to calculate the maximum possible pixel value, i.e., $2^B - 1$. For all test and training datasets used in this thesis, the bit depth is 8, meaning the maximum possible pixel value is 255.

[†] In the literature, the terms bpp, BPP, and Bpp are used interchangeably. This thesis uses Bpp.



MS-SSIM [160] refines SSIM [159] by computing it over multiple scales and combining these into a single measure. Given a set of original and reconstructed image pairs $\{(\boldsymbol{x}_1, \hat{\boldsymbol{x}}_1), \ldots, (\boldsymbol{x}_M, \hat{\boldsymbol{x}}_M)\}$, where the images are progressively down-sampled by a factor of 2^{m-1} for each scale m, the SSIM [159] at scale m is expressed as:

$$\text{SSIM}_m(\boldsymbol{x}_m, \hat{\boldsymbol{x}}_m) = \frac{1}{3}\sum_{c=0}^{2} \text{SSIM}_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}) \tag{3.6a}$$

$$= \frac{1}{3}\sum_{c=0}^{2} L_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}) \cdot C_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}) \cdot S_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}), \tag{3.6b}$$

$$L_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}) = \frac{2\mu_{\boldsymbol{x}_{m,c}}\mu_{\hat{\boldsymbol{x}}_{m,c}} + C_1}{\mu^2_{\boldsymbol{x}_{m,c}} + \mu^2_{\hat{\boldsymbol{x}}_{m,c}} + C_1}, \tag{3.6c}$$

$$C_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}) = \frac{2\sigma_{\boldsymbol{x}_{m,c}\hat{\boldsymbol{x}}_{m,c}} + C_2}{\sigma^2_{\boldsymbol{x}_{m,c}} + \sigma^2_{\hat{\boldsymbol{x}}_{m,c}} + C_2}, \tag{3.6d}$$

$$S_{m,c}(\boldsymbol{x}_{m,c}, \hat{\boldsymbol{x}}_{m,c}) = \frac{\sigma_{\boldsymbol{x}_{m,c}\hat{\boldsymbol{x}}_{m,c}} + C_3}{\sigma_{\boldsymbol{x}_{m,c}}\sigma_{\hat{\boldsymbol{x}}_{m,c}} + C_3}, \tag{3.6e}$$

where $L_{m,c}(\cdot)$, $C_{m,c}(\cdot)$, and $S_{m,c}(\cdot)$ denote the luminance, contrast, and structure comparison measures for each channel c and scale m. The variables $\mu_{\boldsymbol{x}_{m,c}}$ and $\mu_{\hat{\boldsymbol{x}}_{m,c}}$ represent the local means per channel; $\sigma^2_{\boldsymbol{x}_{m,c}}$, $\sigma^2_{\hat{\boldsymbol{x}}_{m,c}}$ is the variances per channel; and $\sigma_{\boldsymbol{x}_{m,c}\hat{\boldsymbol{x}}_{m,c}}$ stands for the covariance of the original and reconstructed images per channel. Constants C_1, C_2, and C_3 are used to stabilize the division when the denominator is small. Finally, MS-SSIM [160] is computed by combining these values:

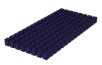
$$\text{MS-SSIM}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{3}\sum_{c=0}^{2} \left([\text{SSIM}_{M,c}]^{\alpha_M} \prod_{m=1}^{M-1} [C_{m,c}]^{\alpha_m} \cdot [S_{m,c}]^{\alpha_m} \right), \tag{3.7a}$$

where α_m weights are assigned to different scales—higher scales are generally given more importance, reflecting their contribution to perceived changes in structural information.

In this thesis, the implementation in PyTorch MS-SSIM package [168] is used for the computation of MS-SSIM [160], following the JPEG AI standardization activity [79] and the CompressAI benchmarking platform [145]. At higher compression qualities, MS-SSIM [160] values typically range between 0.98 and 1.00. Within this narrow range, even small differences can signify significant quality changes. This makes it difficult to visually interpret these variations in linear graphs or effectively compare the performance of different codecs. Therefore, the MS-SSIM [160] is transformed into a log-scale metric following [18], [65], which is defined as:

$$\text{Log-MS-SSIM}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = -10\log_{10}(1 - \text{MS-SSIM}(\boldsymbol{x}, \hat{\boldsymbol{x}})). \tag{3.8a}$$

MS-SSIM is designed to model the human visual system's sensitivity to structural information in images. It evaluates image quality based on luminance, contrast, and structural changes across multiple scales, which are crucial to how humans perceive image quality [159], [160]. Specifically, it focuses on structural elements like edges, textures, and patterns, which are more critical to perceived image quality than pixel-level errors, such as



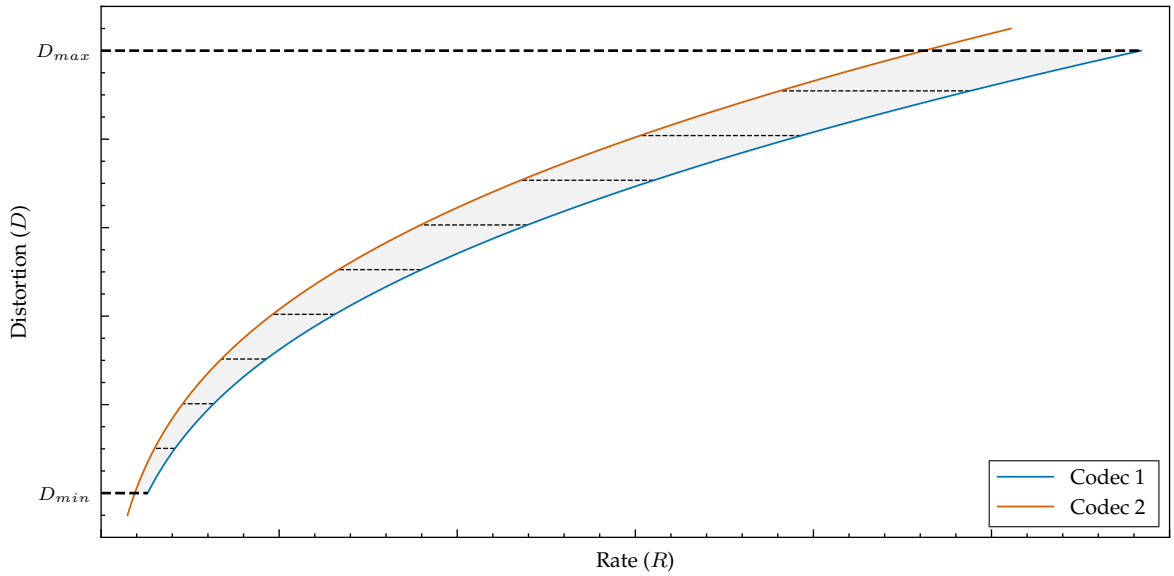
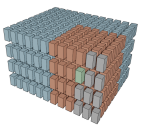


Figure 3.7: Rate-distortion plots used to compare the performance of two different codecs. The entire shaded area between two curves corresponds to the overall BD-Rate [169] metric, which quantifies the average bitrate savings or penalties across a range of quality points. The generalized BD-Rate metric, as proposed by Minnen&Singh [35], is calculated by determining the BD-Rate for each specific interval, represented by the dashed lines, independently.

those measured by PSNR. On the other hand, PSNR can be more advantageous than MS-SSIM [160] in specific scenarios such as compression of sensitive data, medical and scientific imaging, where high data fidelity is required, i.e., exact pixel value accuracy is crucial. It is also useful for benchmarking algorithms that require a straightforward, uniform measure of pixel error. Therefore, despite its limited perceptual relevance, PSNR remains valuable in contexts that prioritize exact error quantification over visual quality.

Once the rate and distortion metrics are available, the efficiency of different compression frameworks can be evaluated using the Bjøntegaard Delta-Rate [169] (BD-Rate) metric. This metric quantifies the average bitrate difference for the same level of distortion. It involves encoding an image at various quality levels to generate rate-distortion (RD) points, interpolating these points to obtain $R(D)$ curves, and comparing the average performance across a specified bitrate range by integrating the difference of these curves. The BD-Rate is then calculated as a percentage difference, indicating how much more or less efficient one codec is over another in terms of bitrate savings for equivalent quality. Notably, discrepancies between the curves become more pronounced at higher bitrates. As bitrate increases, the gap between the curves tends to widen. Therefore, performing the integration on a logarithmic bitrate scale is preferred, as this approach balances potential biases toward higher bitrates [169]. Fig. 3.7 illustrates BD-Rate [169] by showing the interpolated curves of two coding algorithms for an image. Given the first algorithm (Codec 1) as the baseline, the shaded area corresponds to the BD-Rate [169], mathematically defined as:

$$\text{BD-Rate} = \left(\exp\left(\frac{\int_{D_{\min}}^{D_{\max}} \log(R_1(D)) - \log(R_2(D)) dD}{D_{\max} - D_{\min}} \right) - 1 \right) \times 100\%, \tag{3.9}$$



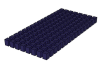
where $R_1(D)$ and $R_2(D)$ are the interpolated rate-distortion curves of the two codecs, and $D_{\min}$ and $D_{\max}$ define the distortion range over which the integrals are computed. The percentage indicates how much more or less bitrate is required by the Codec 2 to achieve the same level of distortion as the Codec 1. A positive BD-Rate [169] suggests that the Codec 2 requires more bitrate to achieve the same quality as Codec 1, while a negative value indicates greater efficiency of the Codec 2.

To evaluate codec performance across a test dataset, BD-Rate [169] is calculated by first averaging distortions and bitrates for each codec at predefined bitrates across all test images. This yields a unified set of average RD points for each codec, from which interpolated RD curves are derived. The BD-Rate [169] is subsequently computed using these curves, serving as a metric that quantifies the average performance difference between the codecs throughout the dataset, thereby providing a holistic assessment of codec efficiency across diverse content and encoding scenarios.

Visualizing RD curves is useful for comparing multiple compression algorithms. However, as the number of algorithms increases, particularly when their performances are close, the plots can become cluttered and difficult to interpret. To address this, comparing all algorithms to a single baseline using the BD-Rate metric [169] offers a clearer score that quantifies how much better or worse each method is relative to the baseline. However, BD-Rate [169] provides only a single aggregated score across all quality levels, which can obscure performance variations at specific quality intervals. To overcome this limitation, Minnen&Singh [35] introduced a generalized BD-Rate computation. This approach calculates BD-Rate over smaller, non-overlapping intervals of the RD curves, as illustrated by dashed lines in Fig. 3.7. The negative BD-Rate per interval, which represents rate savings, is then plotted as a function of quality, offering more detailed insights into the quality ranges where one codec outperforms another. In this thesis, the generalized BD-Rate [35] for PSNR is computed using an interval length $\Delta PSNR$ of 0.25 dB, providing four quality evaluations per 1 dB change in distortion.

3.2.2 Complexity Metrics

In this thesis, the complexity of compression frameworks is evaluated based on three key criteria: runtime complexity, computational complexity, and model complexity. Traditional compression frameworks [16], [24], [25], [91], [161] typically report complexity in terms of runtime, measured during both encoding and decoding, using a Central Processing Unit (CPU) in a single-threaded configuration. In contrast, LIC frameworks rely heavily on large matrix multiplications due to the use of NN-based modules, which are more efficiently deployed on accelerated architectures like Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Neural Processing Units (NPUs). These architectures make use of parallel computation Application Programming Interfaces (APIs) such as Compute Unified Device Architecture (CUDA) [170], PyTorch [171], and TensorFlow [172]. Even within the same parallel computing platform, runtime measurements can vary across LIC frameworks due to differences in API versions and optimizations. To address this variability, runtime complexity for different LIC frameworks in this thesis is measured under uniform condi-



tions using the DeepSpeed Profiler [173], ensuring consistency across all frameworks.

While runtime measurements are useful, they are closely tied to the specific hardware on which they are performed, limiting information about their general applicability. For example, an LIC implementation may perform efficiently on high-end GPUs but struggle or be infeasible on resource-constrained devices, e.g. mobile phones, due to the high number of parallel arithmetic operations. Therefore, it is essential to report more generalized computational complexity metrics, such as the number of multiply-accumulate (MAC) operations or floating point operations per second (FLOPS).

For instance, given an input $\boldsymbol{x} \in \mathbb{R}^{H_{in} \times W_{in} \times C_{in}}$ and an output $\boldsymbol{y} \in \mathbb{R}^{H_{out} \times W_{out} \times C_{out}}$, the number of MAC operations for a CNN layer with kernel size $K \times K$ is computed as:

$$\mathbb{C}_{\text{conv}} = K^2 C_{in} C_{out} H_{out} W_{out}, \tag{3.10}$$

Similarly, for a vanilla transformer layer with MHA (see Section 2.3.4) applied to a flattened input $\boldsymbol{x} \in \mathbb{R}^{S \times C_{in}}$, where $S{=}H_{in}W_{in}$, the complexity is given by:

$$\begin{aligned} \mathbb{C}_{\text{trans}} &= \mathbb{C}_{\text{QKV}} + \mathbb{C}_{\text{MHA}} + \mathbb{C}_{\text{MLP}}, && \text{(3.11a)} \\ &= (3S{d_e}^2) + (2S^2 d_e + S{d_e}^2) + (2S d_e d_{mlp}), && \text{(3.11b)} \\ &= (2S^2 d_e + 4S{d_e}^2) + (2S d_e d_{mlp}), && \text{(3.11c)} \end{aligned}$$

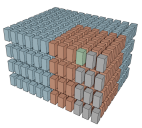
where $\mathbb{C}_{\text{QKV}}$, $\mathbb{C}_{\text{MHA}}$, and $\mathbb{C}_{\text{MLP}}$ represent the MAC operations for the query-key-value mapping, the attention mechanism, and the MLP layers, respectively. Here, the complexity of CNN layers scales linearly with input resolution ($H_{in} \times W_{in}$), while the complexity of transformer layers grows quadratically with the input size due to the S^2 term in the MHA.

Recent standardization efforts, such as JPEG AI [79], normalize MAC computations by image resolution, resulting in the complexity metric MAC per pixel (MACpp). Following these guidelines, computational complexity in this work is measured in MACpp or kilo MACpp (kMACpp) by using the ptflops package [174] and DeepSpeed [173].

Another important aspect is model complexity, which refers to the number of learnable parameters in the LIC model. Generally, models with more parameters have greater capacity for complex tasks [175], but they also require more storage and memory, which can limit their usability on certain hardware platforms. Deep learning APIs, such as PyTorch [171] and TensorFlow [172], provide tools to calculate the number of learnable parameters, which are used in this thesis to measure model complexity of each LIC model.

3.3 Performance Evaluation Setup

In this work, a wide variety of state-of-the-art learned image compression frameworks [32]–[35], [39], [40], [43]–[47], [52], [56]–[63] are evaluated for their compression performance. The evaluation includes models with publicly available source code as well as those without. For models with source code and pretrained models, official implementations were used with default configurations. Evaluations were conducted on hardware comprising a single NVIDIA Titan RTX with CUDA Toolkit 11.4 [170] and Intel Core i9-10980XE, ensuring consistency



across experiments. For models without open-source implementations and pretrained models, results were extracted using WebPlotDigitizer [176], [177], a tool that semi-automatically extracts data from plots. The accuracy of the extraction process was validated by comparing the extracted results with those obtained from models with available source code, showing negligible differences (<0.1% in BD-Rate difference). Table 3.1 provides an overview of all evaluated LIC models, including details on source code and pretrained model availability, with links to respective repositories.

In addition to the LIC methods, the TIC frameworks were used for comparison, including Better Portable Graphics (BPG) [91], which is based on the intra-frame coding of the High Efficiency Video Coding (HEVC) standard [24], and the intra-frame coding framework of Versatile Video Coding (VVC) [25], the latest video coding standard.

For BPG [91], the official executable is used, with the following command-line parameters for encoding:

```
bpgenc -m 444 -b 8 -c ycbcr -e x265 -q <Q> -o <Bitstream> <InputPNG>
```

Command 3.1: Command used for encoding with BPG [91]

Here, `InputPNG`, and `Bitstream` stand for the file paths of the input PNG image and its encoded binary file, respectively. The encoding uses 4:4:4 chroma subsampling (`-m 444`) and converts the RGB images to YC_BC_R (`-c ycbcr`) for optimized compression. The BPG encoder is executed at different quality values $Q \in [12, 47]$ in increments of 5 to explore a wide range of RD trade-offs.

The reconstruction of images (`ReconstructedPNG`) is performed by decoding binary bitstream (`Bitstream`) with the following command:

```
bpgdec -o <ReconstructedPNG> <Bitstream>
```

Command 3.2: Command used for decoding with BPG [91]

For VVC [25], the VVC Test Model (VTM) [178] version 16.2 was used. Since VTM 16.2 [178] requires images in YC_BC_R format, the input RGB images (`InputPNG`) is first converted to YC_BC_R (`InputYCbCr`) using FFmpeg [179] with the BT.709 standard [94]:

```
ffmpeg -i <InputPNG> -pix_fmt yuv444p -vf scale=in_range=full:
   in_color_matrix=bt709:out_range=full:out_color_matrix=bt709
   -color_primaries bt709 -color_trc bt709 -colorspace bt709
   -y <InputYCbCr>
```

Command 3.3: Command for FFmpeg [179], applying color conversion from RGB to YC_BC_R according to the BT.709 [94]

After conversion, the intra-frame coding framework of VTM 16.2 [178] is executed for encoding the input image using the following command:

```
EncoderApp -c encoder_intra_vtm.cfg -i <InputYCbCr> -wdt <W> -hgt <H>
   -b <Bitstream> -f 1 -fr 1 -q <Q> -InputBitDepth=8
   -InputChromaFormat=444 -ConformanceMode=1
```

Command 3.4: Command used for encoding with VTM 16.2 [178]

Same as the setup for BPG [91], the encoding is performed for quality values $\mathtt{Q} \in [12, 47]$ in increments of 5. The binary bitstream (`Bitstream`) was decoded back into reconstructed image (`ReconstructedYCbCr`) in YC_BC_R format with the following command:

```
DecoderApp -d 8 -b <Bitstream> -o <ReconstructedYCbCr>
```

Command 3.5: Command used for decoding with VTM 16.2 [178]

Finally, the reconstructed YC_BC_R image is converted back to RGB using FFmpeg [179]:

```
ffmpeg -f rawvideo -vcodec rawvideo -s <W>x<H> -r 1 -pix_fmt yuv444p
   -i <ReconstructedYUV> -pix_fmt rgb24 -vf scale=in_range=full:
   in_color_matrix=bt709:out_range=full:out_color_matrix=bt709
   -color_primaries bt709 -color_trc bt709 -colorspace bt709
   -y <ReconstructedPNG>
```

Command 3.6: Command for FFmpeg [179], applying color conversion from YC_BC_R to RGB conversion according to the BT.709 [94]

3.4 Chapter Summary

This chapter outlined the evaluation methodologies used throughout this thesis. It provided a detailed analysis of widely adopted testing datasets such as Kodak [138], Tecnick [139], and CLIC [140], with a particular emphasis on their content complexity. The discussion highlighted the critical importance of selecting diverse datasets for robust and fair evaluations of image compression models. Additionally, the chapter addressed the limitations of the Vimeo90K [144] dataset, which serves as the primary training dataset for many state-of-the-art approaches, including those discussed in this thesis. To mitigate these limitations, the COCO [146] dataset was introduced as a supplementary resource, enhancing the diversity of training data. This investigation into training datasets forms the foundation for experiments in Chapter 5.

Furthermore, the chapter presented a set of performance and complexity metrics, which are essential for evaluating the trade-offs between bitrate and reconstruction quality, and computational efficiency, which are a key focus throughout this thesis. These metrics are used to benchmark state-of-the-art TIC and LIC methods, as well as the proposed approaches. Finally, a detailed explanation of the performance evaluation setup, including both hardware and software configurations, was provided to ensure consistent and reproducible benchmarking across state-of-the-art compression methods used in this thesis.

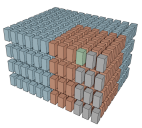


Table 3.1: Details of the state-of-the-art LIC frameworks used for the experimental evaluation

	Availabilty		
Model	Code	Model	Link to Official Implementation
Ballé *et al.* [32]	✓	✓	https://github.com/tensorflow/compression
Ballé *et al.* [33]	✓	✓	https://github.com/tensorflow/compression
Chen *et al.* [48]	✓	✓	https://github.com/NJUVISION/NIC/tree/main/code
Cheng *et al.* [47]	✓	✓	https://github.com/ZhengxueCheng/Learned-Image-Compression-with-GMM-and-Attention
Cui *et al.* [39]	✗	✗	–
Fu *et al.* [58]	✓	✗	https://github.com/fengyurenpingsheng/Asymmetric-Learned-Image-Compression-with-MRSB-IM-and-PQF
Guo *et al.* [45]	✗	✗	–
He *et al.* [52]	✗	✗	–
He *et al.* [56]	✗	✗	–
Kim *et al.* [61]	✓	✗	https://github.com/junhyukk/informer
Liu *et al.* [59] (L)	✓	✗	https://github.com/jmliu206/LIC_TCM
Liu *et al.* [59] (S)	✓	✓	https://github.com/jmliu206/LIC_TCM
Lu *et al.* [62]	✓	✗	https://github.com/lumingzzz/TIC
Lu *et al.* [63]	✓	✓	https://github.com/lumingzzz/TinyLIC
Minnen *et al.* [34]	✓	✓	https://github.com/tensorflow/compression
Minnen&Singh [35]	✓	✗	https://github.com/tensorflow/compression
Qian *et al.* [44]	✓	✓	https://github.com/damo-cv/img-comp-reference
Qian *et al.* [46] (PCM)	✓	✓	https://github.com/damo-cv/entroformer
Qian *et al.* [46] (SCM)	✓	✓	https://github.com/damo-cv/entroformer
Tang *et al.* [43]	✓	✗	https://mic.tongji.edu.cn/1f/e3/c9778a270307/page.htm
Wang *et al.* [60] (LL)	✓	✓	https://github.com/microsoft/dcvc
Yang&Mandt [57]	✓	✓	https://github.com/mandt-lab/shallow-ntc

Chapter 4

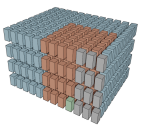
High-performance Learned Image Compression Framework

This chapter introduces a high-performance context model, named the *Contextformer*, which significantly improves the compression performance of learned image compression frameworks. Unlike CNN-based approaches, the Contextformer employs a transformer-based architecture to achieve context modeling that is highly adaptive to the input content. Moreover, the proposed architecture presents a novel application of the attention mechanism, extending its functionality beyond conventional use to include spatial and channel-wise dependencies in the latent space. In this chapter, both the performance and complexity of the Contextformer are comprehensively analyzed through empirical evaluations conducted on commonly used test datasets. Additionally, ablation studies are presented to demonstrate the impact of model design and hyperparameters.

Parts of the work presented in this chapter have been published in [2], [3].

4.1 Motivation

Forward and backward adaptation for entropy modeling, as explained in Chapter 2, play an important role in the performance of the LIC framework. Entropy modeling of the primary bitstream of $\hat{y}$ necessitates the refinement of forward adaptation to provide accurate side information, which inherently increases the amount of the secondary bitstream information of $\hat{z}$ [34]. Conversely, backward adaptation uses a context model to condition the entropy modeling of the primary bitstream elements on the preceding ones, or their *contextual data*. The context model reduces the necessity for additional signaling, significantly increasing the compression performance. Early approaches [34], [47], [49] utilized 2D masked convolutions as context models, inspired by the PixelCNN [180], which autoregressively models the joint image distribution pixel by pixel. However, 2D masked convolutions limit the capability of exploiting complex contextual relations in the latent space to only the spatial dimension, constrained by the kernel size of the convolutions. Therefore, recent literature has investigated a large variety of context modeling schemes and architectures (see Fig. 4.1), which can be categorized into three groups according to the limitations they address:

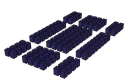


1. Underexplored spatial dependencies in the latent space [3], [37]–[39]
2. Underexplored cross-channel dependencies in the latent space [35], [40]–[43]
3. Underexploited content-adaptivity in the entropy estimation [44]–[46], [61]

To address the first problem, studies such as [3], [37]–[39] implemented a multi-scale 2D context model extending the context model introduced by Minnen *et al.* [34]. Instead of using a single 2D masked convolution, they employ multiple masked convolutions with varying kernel sizes, aiming to capture short- and long-distance dependencies in the latent space.

To exploit the cross-channel dependencies along with the spatial ones, studies such as [40]–[43], [48] introduced 3D masked convolutions to the context model. Their studies show that 3D convolutions improve the coding performance significantly, at the expense of increased complexity due to the additional dimension for the autoregressive process. To reduce the complexity, Minnen&Singh [35] introduced a channel-wise autoregressive model (ChARM). In their model, the latent tensor is split across the channel dimension, and preceding channel segments are used as context for the current segment to be coded. The proposed method drastically reduces the complexity; however, it completely discards the spatial dependencies in the latent space.

In a trained CNN, each layer may have unique convolutional kernels that progressively extract complex image features through successive layers. However, the weights of these kernels remain fixed for every input, making models based solely on classic CNN architectures agnostic to input content [89], [181]. To achieve a content-adaptive context model, early proposals [44], [45] combine 2D masked convolutions and simplified attention mechanisms, capturing local and global correlations of latent elements. Inspired by template matching [182], Qian *et al.* [44] implemented a global reference approach for increasing context adaptivity. They employ a 2D masked convolution in context modeling similar to [34]. Additionally, they use the masked convolution's output as a template for searching for a similar patch in the previously coded positions, which is also included in the entropy modeling. Guo *et al.* [45] combines multiple advancements from prior works. Similar to ChARM [35], they split the latent tensor into two channel segments for exploiting cross-channel dependencies. The first segment is processed by a 2D masked convolution without any adaptive mechanism. For coding the second segment, they use two different mechanisms: first, an "improved" version of the 2D masked convolutions, named MaskConv+, that uses the spatially co-located elements from the first segment in addition to the local neighbors; second, an advanced global reference. They extend the patch-based global reference to a more precise element-based one. They first compute the causal similarity between all elements from the first segment. Based on the spatial position of the top-k similar elements in the first segment, corresponding k elements from the second segment are used in the entropy model. Moreover, Kim *et al.* [61] proposed two different hyperprior networks: a CNN-based local hyperprior and a transformer-based global hyperprior. The transformer-based model extracts eight fixed-size global features signaled to the decoder as secondary bitstream. The local hyperprior applies point-wise operations without downsampling, i.e., 1×1 convolutions, to



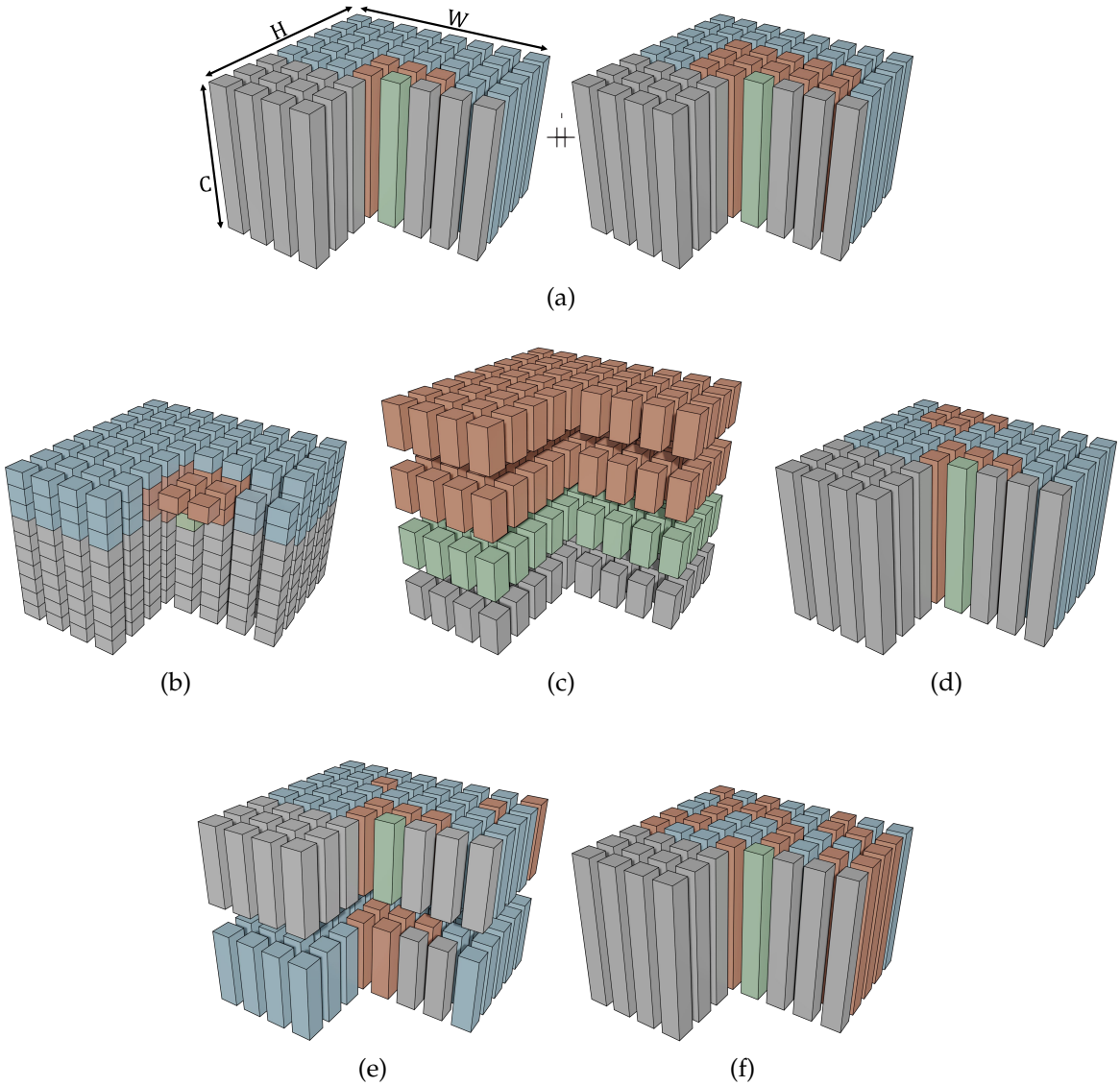
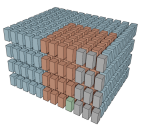


Figure 4.1: Illustration of various context modeling processes for one step in: (a) multi-scale 2D context model [3], [37]–[39], (b) 3D context model, (c) channel-wise autoregressive model (ChARM) [35], (d) context model with a global reference [44], (e) context model with an advanced global reference [45], and (f) transformer-based context model with spatial attention [46]. The context model gathers information from neighboring latent elements (■) to predict the probability of the current latent element(s) (■). Elements that have already been encoded are shown in (■), while those still to be encoded are represented in (■). Notably, in (a), $+\!\!+$ stands for the concatenation of the outputs from each masked convolution in the channel dimension; in (c), all latent elements depicted as (■) are coded independently of each other; and in (e), the primary channel segment is displayed at the bottom for better visibility.

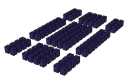
generate a third latent space which is also signaled. Additionally, they also employ the context model of [34] with 2D masked convolutions. Furthermore, Qian *et al.* [46] implemented a transformer-based hyperprior and context model instead of a CNN-based approach. They used a global attention mechanism, which scales with the whole latent space to obtain a large degree of content-adaptivity for the spatial dimension. Additionally, their model computes



the attention between the current element and all previously coded elements, but then distills context information by selecting the top-k of them, to prevent a large number of irrelevant latent elements from dominating the softmax computation, which can result in an unstable training procedure [183], [184].

Although these advancements in context modeling increase the performance and capabilities of entropy modeling, the aforementioned limitations are only partially addressed by the proposed methods. For instance, the multi-scale 2D context models [3], [37]–[39] focus only on the spatial dependencies without any content-adaptive behavior. The 3D context models [40]–[42], [48] are only feasible for small kernel sizes due to the significantly high model complexity; therefore, they cannot build long-distance relations along the channel dimension. The ChARM architecture simplifies channel-wise contextual information aggregation by sacrificing the exploration of the spatial dependencies. Additionally, both 3D context models and ChARM are CNN-based architectures without any attention mechanism. The architectures using a global reference [44], [45] provide limited content adaptivity, as the computation of the spatial relations is limited to the size of the template or k-similar elements, and the cross-channel elements are partially or not considered for the adaptive modeling. According to the experiments of [45], the primary channel segment contains 60% of the information on average, while the context model utilizes only 2D masked convolutions for modeling it. Due to the intrinsic properties of transformers, such as a dynamic receptive field [89], transformer-based context models can support a large degree of content adaptivity, making them a promising candidate for high-performance entropy modeling. However, the proposed methods [46], [61] have several issues in reaching that goal. First, those models partially explore the channel-wise adaptive modeling due to the limitations of the multi-head attention (MHA) mechanism, which is discussed in Section 4.2.2. Second, both methods have limitations on the number of elements or features to attend. Lastly, they apply global attention to the whole latent space, where the performance and complexity scale poorly with increased image resolution, and training and testing with large-resolution images may become infeasible. Additionally, the model in [61] only uses transformer-based architectures as hyperpriors while still using a CNN for the context modeling.

Notably, in all mentioned entropy modeling algorithms except those of [3], [35], the context model separately processes each element of the quantized latent tensor $\hat{y}$ in an autoregressive fashion, resulting in a high-performance model with potential to capture each single dependency in the latent space. These models are referred to as serial context models (SCMs). When multiple latent elements are processed simultaneously, the resulting, more computationally efficient context models are referred to as parallel context models (PCMs). This chapter is mainly focused on the serial approaches, and more detailed explanations for the parallel approaches are reserved for Chapter 5. For instance, Qian *et al.* [46] proposed SCM and PCM variants of their architecture; in the following, only the SCM version is addressed.



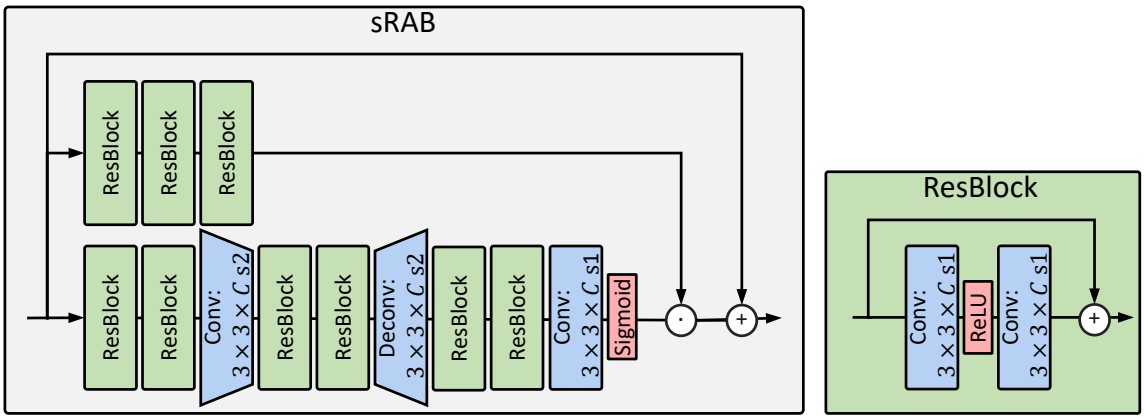


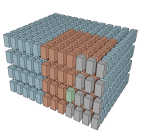
Figure 4.2: Illustration of the ResBlock [40], [48] and simplified residual attention blocks (sRAB) [38]–[40], [47], [48] used in the proposed framework with Contextformer. "Conv: $3\times3\times C$ sX" stands for a convolutional layer with kernel size of 3×3, number of C output channels, and stride "s" of X. Similarly, "Deconv" is an abbreviation for transposed convolutions. Sigmoid and ReLU [128] are the activation functions. $\oplus$ and $\odot$ correspond to element-wise addition and multiplication (Hadamard product [127]), respectively.

4.2 Contextformer

4.2.1 Contextformer with Spatial Attention

The compression framework of the Contextformer is inspired by advanced autoencoder architectures introduced by Guo *et al.* [38], Cui *et al.* [39], and Cheng *et al.* [47]. On the encoder side, the analysis transform g_a incorporates 3×3 convolutional layers with a stride of 2, each followed by GDN activation functions [120]. On the decoder side, g_s mirrors the architecture of g_a, using 3×3 deconvolutional layers with IGDN [120]. Additionally, residual blocks (ResBlocks) [40], [48] are integrated into the first layer to broaden the receptive field and reduce quantization error. Recent studies [38]–[40], [47], [48] have demonstrated that using residual attention blocks (RABs) [133] in the autoencoder improves compression performance by redistributing bit allocation—assigning more bits to salient areas and high-frequency edges, and fewer bits elsewhere. Moreover, Cheng *et al.* [47] showed that RABs can be simplified by omitting the non-local mechanism [133] to reduce model complexity with minimal performance degradation. Following these studies, a single layer of simplified RAB (sRAB) is used in both g_a and g_s (see Fig. 4.2). The entropy model employs a hyperprior network [33] and universal quantization [185]. The entropy of the secondary bitstream $\hat{z}$ is modeled by a factorized entropy model [32], while the conditional entropy of the primary bitstream $p_{\hat{\boldsymbol{y}}_i}(\hat{\boldsymbol{y}}_i \mid \hat{\boldsymbol{z}})$ is estimated using an Asymmetric Gaussian Mixture Model (AGMM) [186] with $k_m{=}3$ mixtures. This approach combines the methods in [38], [39] and [47], providing a skewed distribution with two separate variances per mixture.

The primary difference between the proposed framework and the architectures of [38], [39] is that the multi-scale context model is replaced by the Contextformer. The Contextformer implements L transformer layers with an architecture similar to the ViT [85]. Since ViT [85] requires the input in a sequential format (i.e., flattened patches) for attention computation (see Section 2.3.4), the quantized latent tensor $\hat{\boldsymbol{y}} \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}\times C}$ is rearranged into a



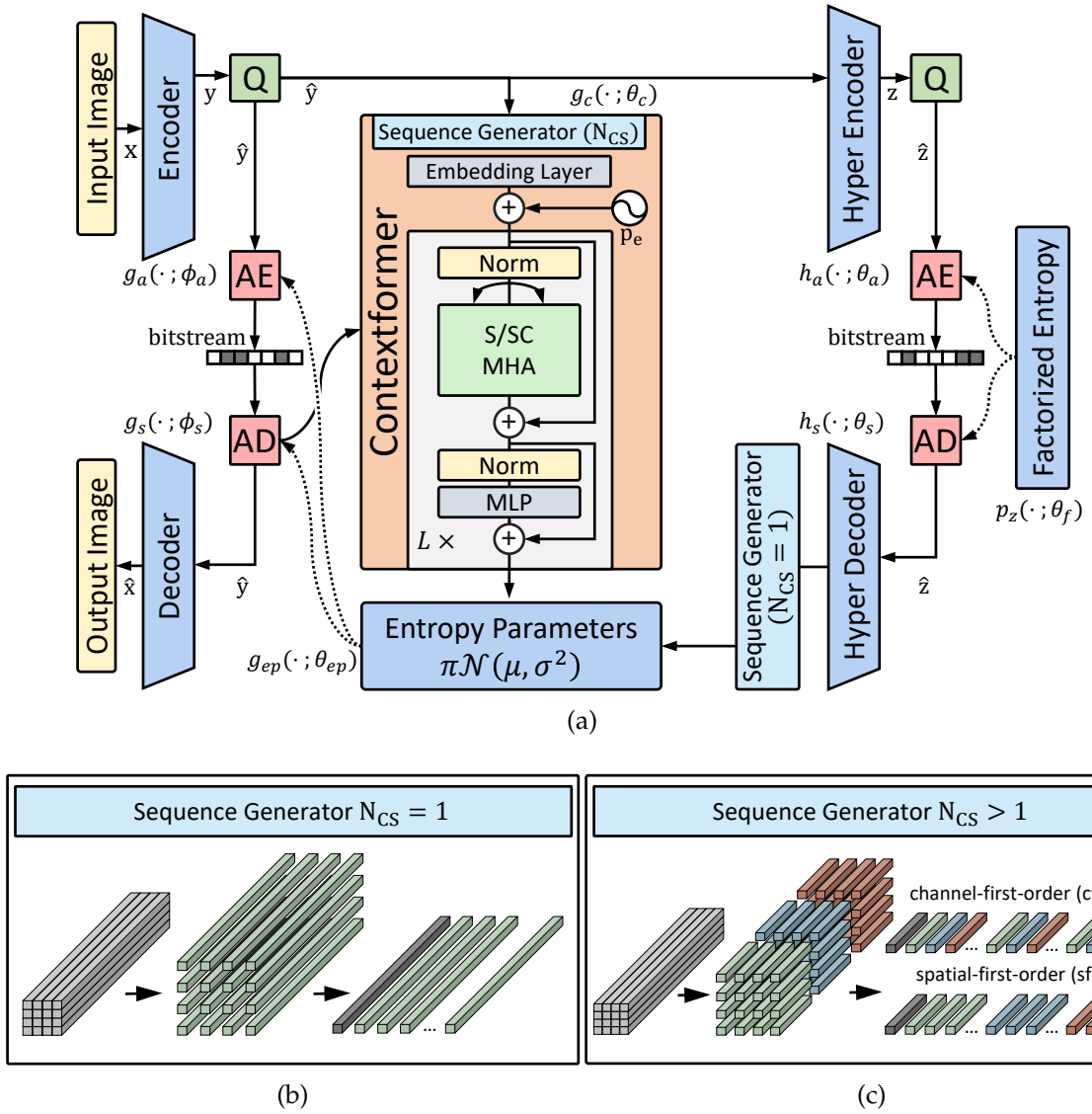


Figure 4.3: Illustration of (a) the proposed LIC framework utilizing Contextformer, (b) the sequence generation process for Contextformer with spatial attention (N_{cs}=1), and (c) the sequence generation process for Contextformer with spatio-channel attention (N_{cs}>1). In (a), S-MHA and SC-MHA stand for the spatial and spatio-channel multi-head attention mechanisms as explained in Section 4.2.1 and Section 4.2.2, respectively. The sequence generation method shown in (b) is applied to the output of the hyperdecoder, regardless of the chosen N_{cs} value. In both (b) and (c), prepended start token is depicted as a dark gray element.

sequence of spatial patches $\hat{\boldsymbol{s}} \in \mathbb{R}^{N\times(p_h p_w C)}$, where $N=\frac{HW}{256 p_h p_w}$. Here, (p_h, p_w) represents the shape of each spatial patch, and N corresponds to the number of sequential elements. ViT [85] involves a trade-off between its complexity and the patch size—larger patch sizes reduce the complexity of attention computation but degrade the granularity of exploiting correlations. Therefore, ViT [85] is originally designed for tasks such as object recognition, which do not require pixel-level decisions. However, context modeling requires dense predictions, and the spatial relations of neighboring elements within a patch cannot be fully exploited when using large patches. To address this, the spatial patch size of the Context-

former is set to 1×1. The resulting sequential representation is formulated as:

$$\hat{\boldsymbol{s}} = \{\mathbf{0}, \hat{\boldsymbol{y}}_{0,0}, \hat{\boldsymbol{y}}_{0,1}, \ldots, \hat{\boldsymbol{y}}_{i,j}, \ldots, \hat{\boldsymbol{y}}_{\frac{H}{16}-1,\frac{W}{16}-2}, \hat{\boldsymbol{y}}_{\frac{H}{16}-1,\frac{W}{16}-1}\}, \tag{4.1}$$

where each sequential element corresponds to one latent element $\hat{\boldsymbol{y}}_{i,j} \in \mathbb{R}^{1\times1\times C}$ with spatial coordinates (i, j) rearranged in raster-scan order. The index of the last dimension is omitted for simplicity. To ensure the autoregressive process is not violated, a zero-valued latent element (also called a start token) is prepended to $\hat{\boldsymbol{s}}$. Fig. 4.3a illustrates the overall compression framework employing the Contextformer, and Fig. 4.3b shows the spatial sequence generation process.

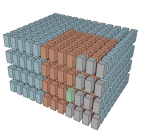
After obtaining the sequential representation $\hat{\boldsymbol{s}}$, it is projected to vector embeddings using a learnable linear transformation $\mathbb{R}^{N\times C} \rightarrow \mathbb{R}^{N\times d_e}$ (embedding layer), where d_e is the embedding dimension. Positional variance is introduced by applying a learned positional encoding (p_e) to the first layer, similar to the approaches in [85], [86]. The Contextformer uses Multi-Head Attention (MHA), which allows it to focus independently on different parts of the channel dimension of $\hat{\boldsymbol{y}}$. However, due to the sequential representation in Eq. (4.1), the attention primarily focuses on the spatial dimension. Therefore, the proposed mechanism in this section is referred to as S-MHA (Spatial Multi-Head Attention) to distinguish it from other proposed techniques. Additionally, a coding mask is applied to the S-MHA layers to ensure causal coding, as explained in Section 2.3.4.

4.2.2 Contextformer with Spatio-Channel Attention

As described in Section 2.3.4, the S-MHA mechanism enables each head, or part of a sequential element, to focus efficiently on distinct parts of other sequential elements. In the context of entropy modeling, each head of the Contextformer corresponds to a channel segment, and applying attention between different heads implements a context model that can explore cross-channel dependencies in the latent space. However, the S-MHA mechanism has several limitations in fully exploiting channel-wise dependencies when used for context modeling.

To simplify the analysis, consider a Contextformer with a single transformer layer. The first limitation is that intra-channel information cannot be exploited during attention computation (see Fig. 4.4d). Given an element from the sequential representation $\hat{\boldsymbol{s}}_n \in \mathbb{R}^{1\times C}$ at coding step n and its corresponding latent tensor element $\hat{\boldsymbol{y}}_{i,j,0\ldots C-1} \in \mathbb{R}^{1\times1\times C}$, each element $\hat{\boldsymbol{y}}_{i,j,c}$ with the same spatial coordinates (i, j) but different channel index c is entropy-coded independently. Therefore, the context modeling of $\hat{\boldsymbol{y}}_{i,j,c}$ cannot utilize information from spatially co-located elements in other channels $\hat{\boldsymbol{y}}_{i,j,\neq c}$.

The second limitation arises from how S-MHA is performed. Given sub-representations $\hat{\boldsymbol{h}} \in \mathbb{R}^{N\times h\times C_h}$ of the sequential representation $\hat{\boldsymbol{s}} \in \mathbb{R}^{N\times C}$ used for S-MHA (where h represents the number of heads and $C_h = \frac{C}{h}$), the n-th sub-representation $\hat{\boldsymbol{h}}_{n,i,0\ldots C_h-1}$ with head index i can only apply attention to preceding elements in the coding order $\hat{\boldsymbol{h}}_{<n,i,0\ldots C_h-1}$ with the same head index. Since each sub-representation also corresponds to a channel segment, S-MHA strictly limits the Contextformer's ability to exploit inter-channel information.



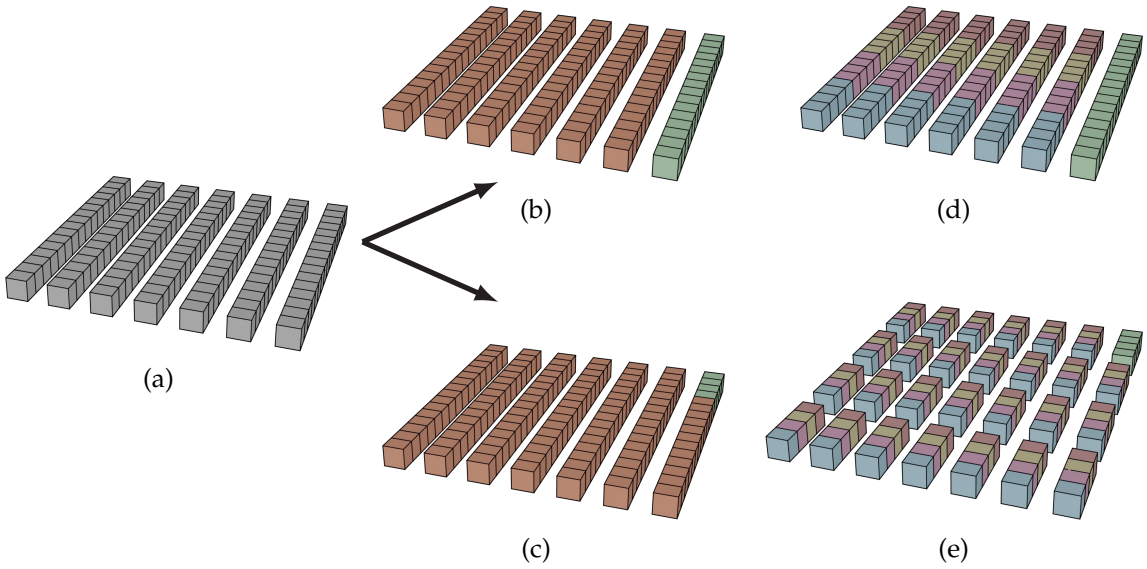


Figure 4.4: Illustration of different attention mechanisms for context modeling: (a,b,d) spatial multi-head attention (S-MHA), and (a,c,d) spatio-channel multi-head attention (SC-MHA). In (a), a portion of the flattened latent tensor is shown. (b) and (c) highlight the current latent element to be coded (■), along with the neighboring elements participating in its context modeling (■), where N_{cs}=4 in (c). (d-e) illustrate the application of MHA, where attention is computed across each head (indicated by shared colors) and aggregated to form the final attention used in context modeling of the current element (■). In this example, the number of heads h is set to 4.

To address these limitations, the sequence generation process presented in Section 4.2.1 is extended to spatio-channel patch generation, as illustrated in Fig. 4.3c. The proposed sequence generator rearranges $\hat{\boldsymbol{y}} \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}\times C}$ into a sequence of spatio-channel patches $\hat{\boldsymbol{s}} \in \mathbb{R}^{\frac{HWC}{256 p_h p_w p_c}\times(p_h p_w p_c)}$, where each channel segment has a size of p_c, and the total number of channel segments is $N_{cs}{=}\frac{C}{p_c}$. Following the reasoning in Section 4.2.1, the spatial patch size is selected as $1{\times}1$, providing $N{=}HWN_{cs}$ sequential elements to the context model, ensuring more accurate entropy coding. Combined with the MHA, the resulting context model can apply attention with higher granularity, including sub-segments of each channel segment, as illustrated in Fig. 4.4e, resulting in a novel Spatio-Channel Multi-Head Attention (SC-MHA). Depending on the arrangement of the spatio-channel patches, the proposed segmentation method enables two different coding strategies ($\mathcal{S}_c$), termed *Spatial-first Order* (SfO) and *Channel-first Order* (CfO).

Let $\hat{\boldsymbol{r}} \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}\times N_{cs}\times p_c}$ be a proxy tensor for $\hat{\boldsymbol{y}}$, where the third dimension corresponds to the indices of the channel segments of $\hat{\boldsymbol{y}}$. By extending Eq. (4.1) with channel segmentation, the sequential representation for different coding strategies can be formulated as:

$$\hat{\boldsymbol{s}}_{SfO} = \{\mathbf{0}, \hat{\boldsymbol{r}}_{0,0,0}, \hat{\boldsymbol{r}}_{0,1,0}, \ldots, \hat{\boldsymbol{r}}_{i,j,k}, \ldots, \hat{\boldsymbol{r}}_{\frac{H}{16}-1,\frac{W}{16}-2,N_{cs}-1}, \hat{\boldsymbol{r}}_{\frac{H}{16}-1,\frac{W}{16}-1,N_{cs}-1}\}, \tag{4.2a}$$

$$\hat{\boldsymbol{s}}_{CfO} = \{\mathbf{0}, \hat{\boldsymbol{r}}_{0,0,0}, \hat{\boldsymbol{r}}_{0,0,1}, \ldots, \hat{\boldsymbol{r}}_{i,j,k}, \ldots, \hat{\boldsymbol{r}}_{\frac{H}{16}-1,\frac{W}{16}-1,N_{cs}-2}, \hat{\boldsymbol{r}}_{\frac{H}{16}-1,\frac{W}{16}-1,N_{cs}-1}\}, \tag{4.2b}$$

where $\hat{\boldsymbol{r}}_{i,j,k} \in \mathbb{R}^{1\times1\times1\times p_c}$ corresponds to a vector with spatial coordinates (i, j) and channel segment index k, with the last dimension omitted for simplicity.

Fig. 4.3c illustrates how the Contextformer processes the sequential representation of the latent tensor according to both coding strategies. Notably, the Contextformer with SfO codes all sequential elements from the same channel segment before starting the next channel segment. When coding the sequential element corresponding to the latent tensor $\hat{\boldsymbol{r}}_{i,j,k}$, the Contextformer with SfO can use the information from the latent elements $\hat{\boldsymbol{r}}_{0\ldots\frac{H}{16}-1,0\ldots\frac{W}{16}-1,<k}$ from all spatial locations in the preceding channel segments. This enables prioritization of the spatial dimensions over the channel dimension for entropy modeling. On the other hand, the Contextformer with CfO first codes all channel segments of the same spatial location before moving to the next spatial location. In this case, the context of the latent tensor $\hat{\boldsymbol{r}}_{i,j,k}$ can contain information from the latent elements $\hat{\boldsymbol{r}}_{<i,0\ldots\frac{W}{16}-1,0\ldots N_{cs}-1}$ from all channel segments in preceding spatial locations. This enables prioritization of the channel dimension for entropy modeling.

As explained in Chapter 2, the entropy parameters network g_{ep} combines the outputs of the context model and the hyperprior network for entropy modeling. Since the output of the hyperprior network $\boldsymbol{\Psi} \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}\times C}$ is fully available during the entropy modeling of $\hat{\boldsymbol{y}}$, the hyperprior provides a future lookup for context modeling. Studies by Lee *et al.* [49] and Koyuncu *et al.* [3] showed that exploiting the available future information from hyperprior networks yields higher compression performance. They proposed using not only the spatially co-located element of the hyperprior output $\boldsymbol{\Psi}_{i,j} \in \mathbb{R}^{1\times1\times C}$ but also including local hyperprior neighbors $\boldsymbol{\Psi}_{(i-K)\ldots(i+K),(j-K)\ldots(j+K)}$ for entropy coding of $\hat{\boldsymbol{y}}_{i,j}$, where K defines the kernel size of the neighboring elements included in the computation. Inspired by these studies, the Contextformer implements channel-wise local hyperprior neighbor aggregation to improve compression performance. This is achieved by rearranging $\boldsymbol{\Psi}$ into a sequential format $\boldsymbol{\Gamma}$, similar to Eq. (4.2), where $\boldsymbol{\Gamma}_{SfO}$ can be formulated as:

$$\boldsymbol{\Gamma}_{SfO} = \overbrace{\{\boldsymbol{\psi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\psi}, \ldots, \boldsymbol{\psi}, \boldsymbol{\psi}\}}^{\times N_{cs}}, \tag{4.3a}$$

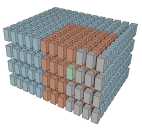
$$\boldsymbol{\psi} = \{\boldsymbol{\Psi}_{0,0}, \boldsymbol{\Psi}_{0,1}, \ldots, \boldsymbol{\Psi}_{i,j}, \ldots, \boldsymbol{\Psi}_{\frac{H}{16}-1,\frac{W}{16}-2}, \boldsymbol{\Psi}_{\frac{H}{16}-1,\frac{W}{16}-1}\}, \tag{4.3b}$$

and $\boldsymbol{\Gamma}_{CfO}$ can be formulated as:

$$\boldsymbol{\Gamma}_{CfO} = \{\boldsymbol{\psi}_{0,0}, \boldsymbol{\psi}_{0,1}, \ldots, \boldsymbol{\psi}_{i,j}, \ldots, \boldsymbol{\psi}_{\frac{H}{16}-1,\frac{W}{16}-2}, \boldsymbol{\psi}_{\frac{H}{16}-1,\frac{W}{16}-1}\}, \tag{4.4a}$$

$$\boldsymbol{\psi}_{i,j} = \underbrace{\{\boldsymbol{\Psi}_{i,j}, \boldsymbol{\Psi}_{i,j}, \ldots, \boldsymbol{\Psi}_{i,j}, \ldots, \boldsymbol{\Psi}_{i,j}, \boldsymbol{\Psi}_{i,j}\}}_{\times N_{cs}}. \tag{4.4b}$$

This means that $\boldsymbol{\Psi}$ is not split into channel segments but is instead replicated according to the number of channel segments N_{cs}, and the entropy modeling of the spatio-channel segment $\hat{\boldsymbol{r}}_{i,j,k}$ uses $\boldsymbol{\Psi}_{i,j}$ with all its channels, regardless of the selected coding strategy.



4.3 Complexity Optimizations

4.3.1 Architectural Optimizations

Given one of the sequential representations from Eq. (4.2), the complexity $\mathbb{C}_{\text{ctx}}$ of a Contextformer layer, in terms of the number of MAC operations, can be formulated as follows:

$$\mathbb{C}_{\text{ctx}} \sim N\mathbb{C}_{\text{trans}}, \tag{4.5a}$$

$$\mathbb{C}_{\text{trans}} = (2N^2 d_e + 4N{d_e}^2) + (2N d_e d_{mlp}), \tag{4.5b}$$

$$N = \frac{HWN_{cs}}{256}, \tag{4.5c}$$

where $\mathbb{C}_{\text{trans}}$ is the complexity of the spatio-channel attention mechanism for one iteration of the autoregressive context modeling, and N represents the number of elements to be entropy coded, which is equivalent to the receptive field size of the transformer-based model. For simplicity of the explanation, the complexity of the embedding layer is omitted, as it can be applied to all latent elements at once, prior to context modeling. For the same reason, the context model is assumed to have a fixed receptive field size ($\frac{HWN_{cs}}{256}$) for all coding steps.

The Contextformer is designed with a receptive field that, in theory, can process inputs of any size. However, in practical applications, the processing of long input sequences, such as high-resolution images, is limited by the exponential growth of $\mathbb{C}_{\text{trans}}$ in both memory usage and computational load as input sequence length increases. This limitation necessitates constraining the receptive field size, leading to the adoption of a sliding-window attention mechanism, similar to the approach proposed in studies such as [83], [86].

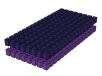
Inspired by the principles of 3D convolutions, a 3D sliding-window approach is proposed to navigate through the spatio-channel data. Unlike traditional 3D convolutions, this receptive field is specifically tailored to traverse spatial dimensions while expanding to encompass all elements in the channel dimension. Fig. 4.5 illustrates the application of the sliding-window attention mechanism across different configurations of Contextformer[†]. To avoid unnecessary padding at the tensor boundaries, the number of elements involved in the attention computation increases without moving the sliding-window to the next spatial coordinate. The proposed sliding-window attention reduces the complexity of the attention mechanism to:

$$\mathbb{C}_{\text{trans}} = (2N_w^2 d_e + 4N_w{d_e}^2) + (2N_w d_e d_{mlp}), \tag{4.6a}$$

$$N_w = K^2 N_{cs}, \tag{4.6b}$$

where N_w is the number of elements within a spatio-channel window, with a channel-wise kernel size of N_{cs} and a spatial kernel size of $K\times K$, where $K \ll \frac{H}{16}$ and $K \ll \frac{W}{16}$. Here, for simplicity, the context model is assumed to have a fixed receptive field size ($K\times K$) for all coding steps. In reality, only half of the elements within the kernel is processed for most of the coding steps, while number of processed elements only grows at the edges.

[†] The sliding-window processing of Contextformer with CfO is available as a flipbook animation in reverse in pp. 1–145, bottom right.



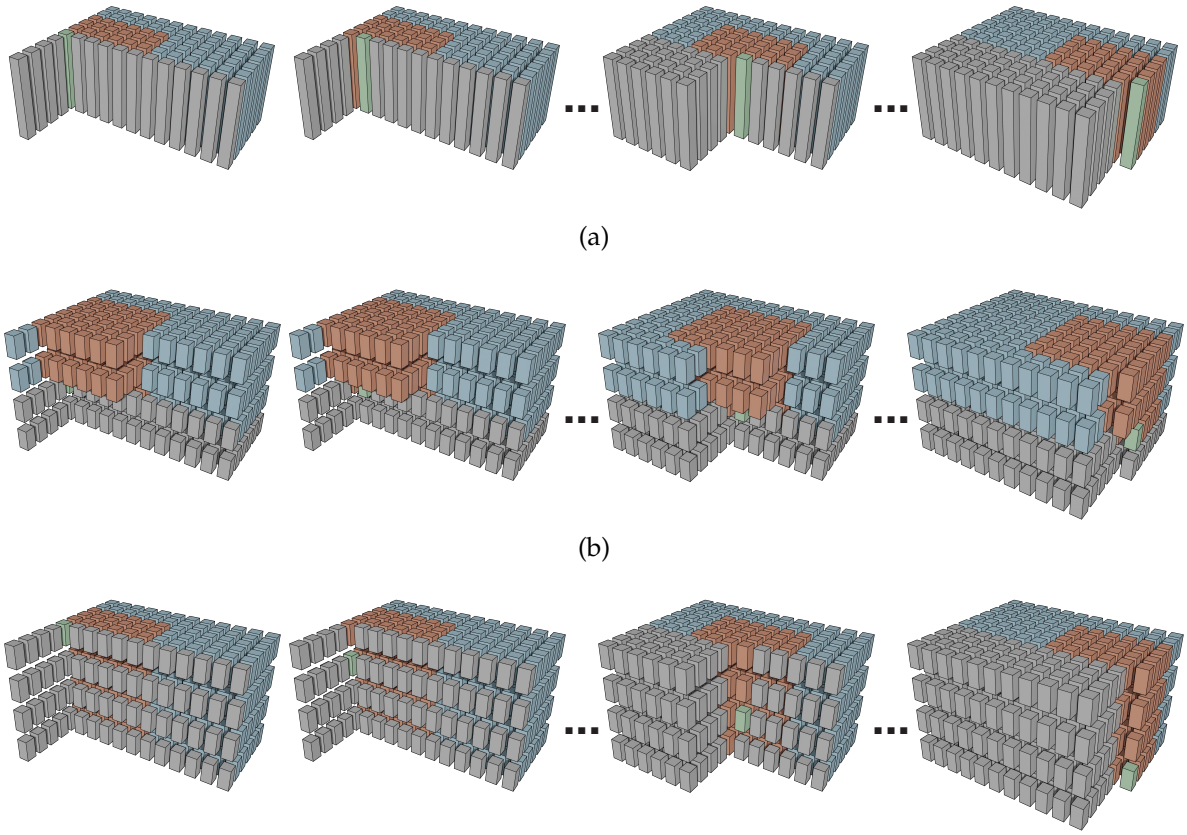


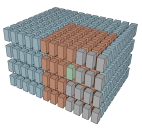
Figure 4.5: Illustration of the sliding-window process in different Contextformer variants, which process one row of the latent tensor from left to right: (a) Contextformer with spatial multi-head attention (S-MHA) for N_{cs}=1, (b) Contextformer with spatio-channel multi-head attention (SC-MHA) and spatial-first-order (SfO) coding for N_{cs}=4, and (c) Contextformer with SC-MHA and channel-first-order (CfO) coding for N_{cs}=4. In all cases, the context model gathers information from neighboring latent elements (■) to predict the symbol probability of the current element being coded (■). Elements already coded are shown in (■), while elements yet to be coded are displayed in (■). Notably, at the tensor edges, the kernel size grows rather than sliding, which eliminates the need for padding.

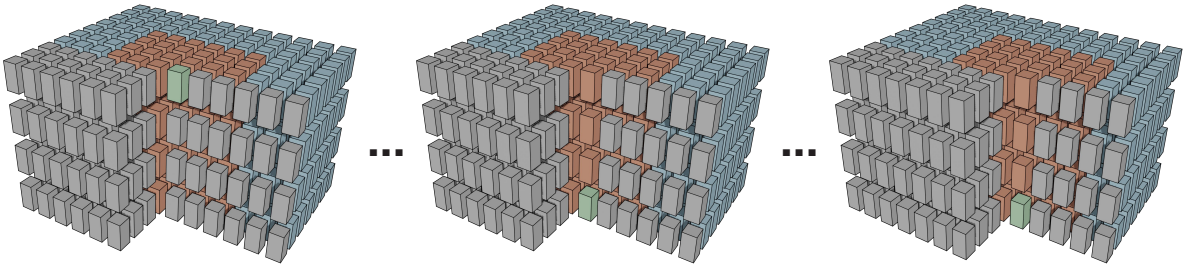
4.3.2 Algorithmic Optimizations

4.3.2.1 Encoder-side Optimizations

The serialized nature of autoregressive processes inherently complicates their efficient deployment on hardware architectures optimized for parallel execution, such as GPUs [3], [35], [52]. A commonly used solution, as described in [82], [187], involves standardizing sequence lengths through padding, thereby enabling parallel sequence processing during training by stacking them into the batch dimension. This method is referred to as the *Pad&Batch* method. However, when integrating Pad&Batch with sliding-window attention, a computational bottleneck arises due to the attention being computed for every padded element.

On the other hand, sliding-window attention can be directly implemented without attention masking by varying sequence lengths at each window's position, as in [86]. This method, referred to as *Dynamic Sequence processing* (DS), can avoid the inefficiency caused by





(a) Step n (b) Step $n+N_{cs}-1$ (c) Step $n+2N_{cs}-1$

Figure 4.6: Illustration of the context modeling process in a Contextformer with channel-first-order (CfO) processing and sliding-window attention across different steps: (a) n, (b) $n+N_{cs}-1$, and (c) $n+2N_{cs}-1$. The latent tensor is color-coded to highlight the different elements: the current latent elements to be coded (■), the elements contributing to the context model (■), the previously coded elements (■), and the elements yet to be coded (■). Notably, in step (b), the attention computation already includes all necessary calculations from step (a), which enables the proposed SCS method. Additionally, both steps (b) and (c) involve the same number of elements participating in the context modeling process (■), allowing to use the proposed BDS method.

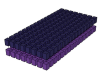
padded elements. However, a straightforward implementation of DS lacks parallelism due to the use of variable sequence lengths, which complicates efficient hardware utilization.

To address these challenges and optimize the parallelization of spatio-channel sliding-window attention, sophisticated algorithms have been proposed by analyzing the inherent properties of the Contextformer. Fig. 4.6 illustrates several processing steps of Contextformer with CfO. Transformers, being sequence-to-sequence models, compute outputs for each element within a sliding window simultaneously. Processing step $n+N_{cs}-1$ also computes the output for step n, ensuring causality in context modeling through attention masking as described in Section 2.3.4. Consequently, intermediate channel segments from step n to $n+N_{cs}-2$ can be skipped, directly calculating the last channel segment's output for each spatial coordinate. This method is referred to as *Skipping intermediate Channel Segments* (SCS).

Furthermore, as highlighted in Fig. 4.6, different spatial coordinates can share the same number of sequential elements joining the context modeling within a sliding-window. All sliding-windows with different spatial coordinates, but the same number of sequence elements can be grouped together in batches for parallel processing. This method is named *Batched Dynamic Sequence processing* (BDS).

Algorithm 1 outlines the implementation of these optimizations. The algorithm begins by determining a processing order or *priority* (Algorithm 2) for each sliding-window position, while grouping windows with identical priorities for batched processing. Once priorities are computed for all window positions, context modeling is applied to groups of windows starting from the highest-priority group. For implementing BDS, the priority is determined by the number of elements in each window, allowing them to be processed from the smallest to the largest number of elements. BDS can be disabled by simply setting the priority to the linear index of each window. Additionally, SCS is implemented by skipping the priority computation for all channel segments except the last one.

It is important to note that this algorithm requires simultaneous access to distant parts of the latent tensor. Therefore, BDS and SCS methods can only be applied on the encoder side.



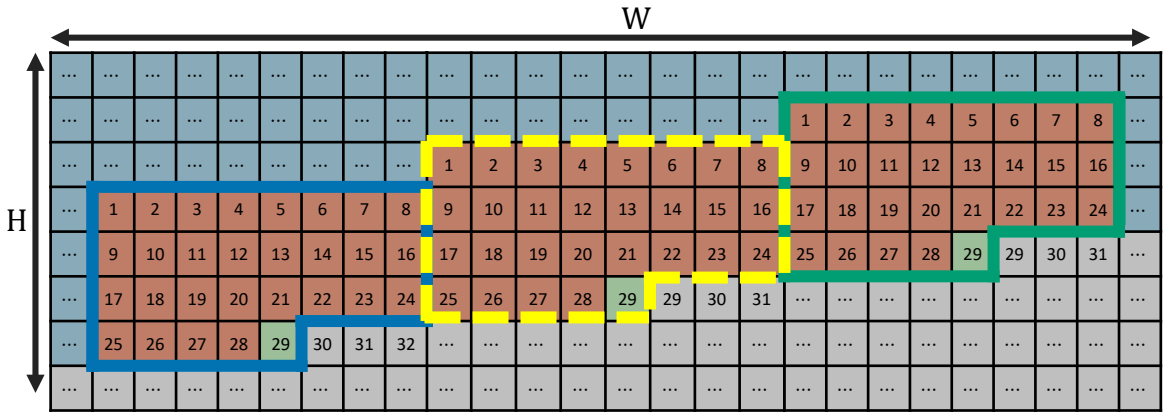


Figure 4.7: Top-down illustration of the context modeling process in a Contextformer using sliding-window attention and wavefront processing for an arbitrary step n. Each latent element is processed in the order $n + N$, where N corresponds to the number assigned to each latent element in the figure. Simultaneously processed windows for step $n + 29$ are highlighted in different colors. For clarity, the channel dimension has been omitted.

4.3.2.2 Decoder-side Optimizations

To optimize decoder-side computations, the studies [51], [12] implement wavefront coding for a 2D context model, which is similar to wavefront parallel processing (WPP) [188] in HEVC [24] and the partitioning slices used in VVC [189]. In their approach, all masked convolutions along the same diagonal can be processed simultaneously, as they do not contain overlapping elements that would compromise the causality of autoregressive modeling.

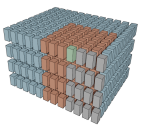
Inspired by this, a 3D wavefront parallel processing strategy (3D-WPP) is adopted for Contextformer by computing non-overlapping spatio-channel sliding windows along the diagonal of the spatial dimensions with the same channel segment (see Fig. 4.7). The same algorithm described in Algorithm 1 is used on the decoder side, where the priority of each window is set based on the index of the diagonal in the spatio-channel grid (see Algorithm 2).

Utilizing wavefront processing in the decoder requires that the sequence of encoding the latent tensor elements into the bitstream aligns with the sequence used by the encoder. This alignment ensures the accurate computation of entropy parameters, regardless of the encoding optimization algorithm employed. Consequently, the parameters need to be re-ordered according to the wavefront processing index before being incorporated into the bitstream.

The primary requirement for using wavefront processing in the decoder is that the latent tensor elements must be encoded into the bitstream in the same order as in the encoder. This alignment allows the entropy parameters to be computed correctly, independent of the specific algorithm used for encoding optimization, and ensures that they are ordered according to the wavefront processing index before bitstream encoding.

4.4 Comparison to the Prior Art Context Models

The proposed spatio-channel sequence generation extends the default spatial attention mechanism of the transformer architecture into the channel dimension, creating a more general-



ized context model. By parameterizing the number of channel segments N_{cs} and coding strategies $\mathcal{S}_c$ (SfO and CfO), the Contextformer can replicate the behavior of various state-of-the-art context models discussed in Section 4.1, while enhancing their context modeling capabilities with additional features. Notably, Figs. 4.1 and 4.5 provide visual references for the following comparisons:

- In case of $N_{cs} = 1$: The attention mechanism is confined to the spatial dimensions, and only the SfO coding strategy is available, as elaborated in Section 4.2.1. Despite these limitations, the Contextformer exhibits faster encoding and decoding speeds due to fewer autoregressive steps compared to its versions with $N_{cs} > 1$. Even with this restriction, it can surpass certain prior models in context modeling capabilities. Compared to other models that enhance spatial dependency support for context modeling [3], [37]–[39], the Contextformer uses a much larger kernel size and applies attention to increase content adaptivity. When compared to architectures using a global reference [44], [45], the Contextformer's receptive field is not limited to a template or a fixed number of k elements; it can theoretically expand to cover the entire latent tensor. Moreover, the Contextformer with $N_{cs} = 1$ closely resembles the transformer-based model in [46], but it is more expressive due to employing full attention instead of a sparse top-k attention mechanism.

- In case of $C > N_{cs} > 1$: The attention mechanism leverages both spatial and cross-channel dependencies, with different coding strategies prioritizing specific dimensions, as explained in Section 4.2.1. The number of channel segments is proportional to the context model's ability to exploit channel-wise information but inversely proportional to computational efficiency. The Contextformer with SfO exhibits operational characteristics similar to [45]. However, unlike [45], which limits adaptive entropy estimation to non-primary channel segments, the Contextformer's receptive field adaptively encompasses all channel segments, offering a more versatile approach to context modeling. Additionally, other transformer-based approaches [46], [61] have limited capacity to explore channel-wise relationships due to the standard MHA mechanism (see Fig. 4.4 and Section 4.2.1). In contrast, the proposed SC-MHA provides a more granular exploration of these relations. Unlike ChARM [35], the Contextformer's modeling is not restricted to the channel dimension alone; it adapts to both spatial and channel dimensions. Moreover, the Contextformer does not require additional modules to aggregate decoded channel segments (further details are discussed in Chapter 5). The Contextformer with CfO also introduces a unique context modeling mechanism that cannot be replicated by previous approaches.

- In the extreme case $N_{cs} = C$: The Contextformer fully exploits spatio-channel attention by processing each latent element individually. This represents the most comprehensive use of the model's capabilities, with other configurations functioning as scaled-down versions designed to balance efficiency and computational complexity. In this configuration, the Contextformer with SfO operates as a sophisticated version of the 3D context models [40]–[42], featuring an expansive and dynamic receptive field.

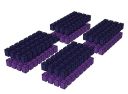


Table 4.1: The architecture of the proposed model using the Contextformer. Each row depicts one layer or component of the model. "Conv: $K_c \times K_c \times C$ s2" stands for a convolutional layer with kernel size of $K \times K$, number of C output channels, and stride "s" of 2. Similarly, "Deconv" is an abbreviation for transposed convolutions. The fully-connected "Dense" layers are specified by their output dimension, whereas $D_1 = 2C_m + d_e$, $D_2 = (4k_m C_M)/N_{cs}$. ResBlock and sRAB are the residual blocks and simplified residual attention mechanisms, shown in Fig. 4.2.

Encoder (g_a)	Decoder (g_s)	Hyperencoder (h_a)	Hyperdecoder (h_s)
Conv: $3\times3\times C_N$ s2	2×ResBlock: $3\times3\times C_M$	Conv: $3\times3\times C_N$ s1	Conv: $3\times3\times C_M$ s1
GDN [120]	Deconv: $3\times3\times C_N$ s2	Leaky ReLU [132]	Deconv: $3\times3\times C_M$ s2
sRAB [39], [47], [133]	IGDN [120]	Conv: $3\times3\times C_N$ s1	Leaky ReLU [132]
Conv: $3\times3\times C_N$ s2	Deconv: $3\times3\times C_N$ s2	Conv: $3\times3\times C_N$ s2	Conv: $3\times3\times C_M$ s1
GDN [120]	IGDN [120]	Leaky ReLU [132]	Deconv: $3\times3\times\frac{3}{2}M$ s2
Conv: $3\times3\times C_N$ s2	Deconv: $3\times3\times C_N$ s2	Conv: $3\times3\times C_N$ s1	Leaky ReLU [132]
GDN [120]	sRAB [39], [47], [133]	Conv: $3\times3\times C_N$ s2	Deconv: $3\times3\times2C_M$ s2
Conv: $3\times3\times C_N$ s2	IGDN [120]		Leaky ReLU [132]
Conv: $1\times1\times C_M$ s2	Deconv: $3\times3\times3$ s2		

Context Model (g_c)	Entropy Parameters (g_e)
Contextformer:	Dense: $(2D_1+D_2)/3$
$\{L, d_e, d_{mlp}, h, N_{cs}, K, \mathcal{S}_c\}$	GELU [129]
	Dense: $(D_1+2D_2)/3$
	GELU [129]
	Dense: D_2

4.5 Experiments

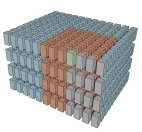
4.5.1 Experimental Setup

4.5.1.1 Training

To investigate the effect of different hyperparameters, various compression frameworks with different Contextformer architectures were trained, defined by the following configuration:

$$\{C_N,\, C_M,\, L,\, d_e,\, d_{mlp},\, h,\, N_{cs},\, \mathcal{S}_c,\, K\}, \tag{4.7}$$

where C_N corresponds to the intermediate layer size of the encoder and decoder, and C_M is the bottleneck size. The rest of the parameters define the Contextformer, where L, d_e, d_{mlp}, and h correspond to the number of layers, embedding size, MLP size, and the number of heads, respectively (see Section 2.3.4). N_{cs} and $\mathcal{S}_c$ stand for the number of channel segments and the coding strategy—either SfO or CfO (see Section 4.2.2). K is the spatial kernel size of



the sliding-window attention (see Section 4.3.1). Table 4.1 provides a detailed description of the compression framework utilizing the Contextformer.

For all models, h is set to 12, and d_{mlp} is selected as $4d_e$ to simplify the search space of the experiments, and the base configuration of the Contextformer is selected as:

$$\{L=8,\ d_e=\frac{8C_M}{N_{cs}},\ d_{mlp}=4d_e,\ h=12,\ N_{cs}=4,\ \mathcal{S}_c=\text{CfO}\} \tag{4.8}$$

which is used to compare the performance of the proposed framework using the Contextformer to other state-of-the-art approaches.

For the training of all variants, random 256×256 image crops from the Vimeo90K [144] dataset were utilized, along with a batch size of 16 and the ADAM optimizer [190] set to default settings. The models were trained for 120 epochs (around 1.2 million iterations). For computational efficiency during the training phase, the sliding-window operations were omitted by selecting K as 16, which is equal to the latent space size. Following prior work [145], an initial learning rate of 10^{-4} was employed, which was halved whenever the validation loss remained nearly constant for a duration of 20 epochs. For validation, the validation set of Vimeo90K [144] was used. The distortion metric $\mathbf{D}(\cdot)$ was chosen as MSE, and a series of models were trained across a range of $\lambda_{\text{MSE}} \in \{0.002, 0.004, 0.007, 0.014, 0.026, 0.034, 0.070\}$ to span various bitrates. Models optimized for MS-SSIM [160] were also developed by fine-tuning the MSE-trained models for approximately 500,000 iterations with the distortion function complementary MS-SSIM [160] ($1-$MS-SSIM) and $\lambda_{\text{MS-SSIM}}=1300\,\lambda_{\text{MSE}}$.

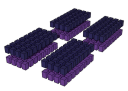
By default, the size of the intermediate layer C_N for both the encoder and decoder, as well as the bottleneck size C_M, were set to 192. To boost the model's capacity at higher bitrates, corresponding to λ_5, λ_6, and λ_7 for both MSE and MS-SSIM optimized models, C_M was increased to 312, aligning with state-of-the-art approaches [34], [35], [47].

4.5.1.2 Evaluation

The compression framework utilizing the proposed Contextformer is evaluated through empirical studies on the Kodak [138], Tecnick [139], and CLIC-P/-M [140] datasets, all of which are described in detail in Section 3.1. Unless specified otherwise, the base configuration of the Contextformer is used for comparative studies, with the spatial receptive field size K set to 16×16 for the sliding-window attention. Ablation studies on different configurations are presented in Section 4.5.2.3.

Various learned image compression frameworks with different context model architectures are used for performance comparison, as summarized in Table 4.2. For a fair comparison, the model of Guo *et al.* [45] without the GRDN post-filter [191] is used, since none of the other frameworks utilize a post-filter. Similarly, the serial context model (SCM) from Qian *et al.* [46] is selected for comparison, as it achieves higher rate-distortion performance, and Contextformer is also a serial context modeling approach. Additionally, TIC algorithms such as BPG [91] and VTM 16.2 [178] are included in the comparative study.

Depending on the published implementation, the number of parameters in the compression frameworks, especially in the entropy models, are measured using the summary functions of PyTorch [171] or TensorFlow [172]. The model complexity in kilo MACpp (kMACpp)



is computed using the ptflops package [174], following the recent JPEG-AI [79] standards, and confirmed by re-evaluating them with DeepSpeed [173]. Similarly, DeepSpeed [173] is used for runtime measurements, where the time for arithmetic coding is excluded due to variations in its implementation across different compression frameworks. All tests, including runtime measurements, are performed on hardware with a single NVIDIA Titan RTX and Intel Core i9-10980XE, using PyTorch 1.10.2 [171] and CUDA Toolkit 11.4 [170].

More details on the experimental setup and how results of the state-of-the-art approaches were obtained are described in Section 3.3.

4.5.2 Experimental Results

4.5.2.1 Model Performance

Fig. 4.8a presents the RD performance of the Contextformer with the SC-MHA mechanism, alongside the comparative performance of prior methods on the Kodak [138] dataset. The Contextformer, using its base configuration, consistently surpasses competing approaches in PSNR values across all tested bitrates. According to the BD-Rate [169], the method achieves an average saving of 6.9% over VTM 16.2 [178] on the Kodak [138] dataset. It is noteworthy that the major difference between the Contextformer and the framework using the multi-scale 2D context model from [39] is the context and entropy modeling, with the Contextformer saving an average of 10% more bits compared to [39]. Figs. 4.8b and 4.8c display the Contextformer's performance against prior methods on the Tecnick [139], CLIC-P [140], and CLIC-M [140] datasets in terms of PSNR. On these datasets, the Contextformer outperforms the previous models, achieving BD-Rate [169] performances of -10.6%, -8.8%, and -5.9% over VTM 16.2 [178].

Fig. 4.9 highlights the Contextformer's performance using the generalized BD-Rate [35] metric, which is explained in Section 3.2. On the Kodak dataset [138], the Contextformer delivers 4% bitrate savings at the highest quality point, while reaching a remarkable 9.2% improvement in bitrate savings at the lowest quality point. Even larger performance gains are observed on the other datasets, confirming the generalization ability of the proposed model across various image contents. The Contextformer demonstrates up to 12.4%, 11.9%, and 6.6% relative bitrate savings over VTM 16.2 [178] on the Tecnick [139], CLIC-P [140], and CLIC-M [140] datasets, respectively.

Fig. 4.10 shows the RD performance of the Contextformer with respect to MS-SSIM [160] across different datasets. The evaluation includes only the versions of the Contextformer and related prior models that were optimized for MS-SSIM [160] during training. The Contextformer consistently outperforms prior-art models for this perceptual quality metric, saving 49.4%, 47.0%, 43.8%, and 36.3% more bits than VTM 16.2 [178] on the Tecnick [139], CLIC-P [140], and CLIC-M [140] datasets, respectively. On the test datasets, the proposed model provides 3%–7% more bitrate savings than the commonly used baseline from Cheng *et al.* [47].

A comprehensive report on BD-Rate [169] performance is available in Table 5.7.

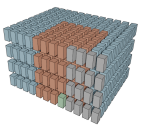
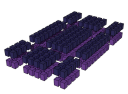
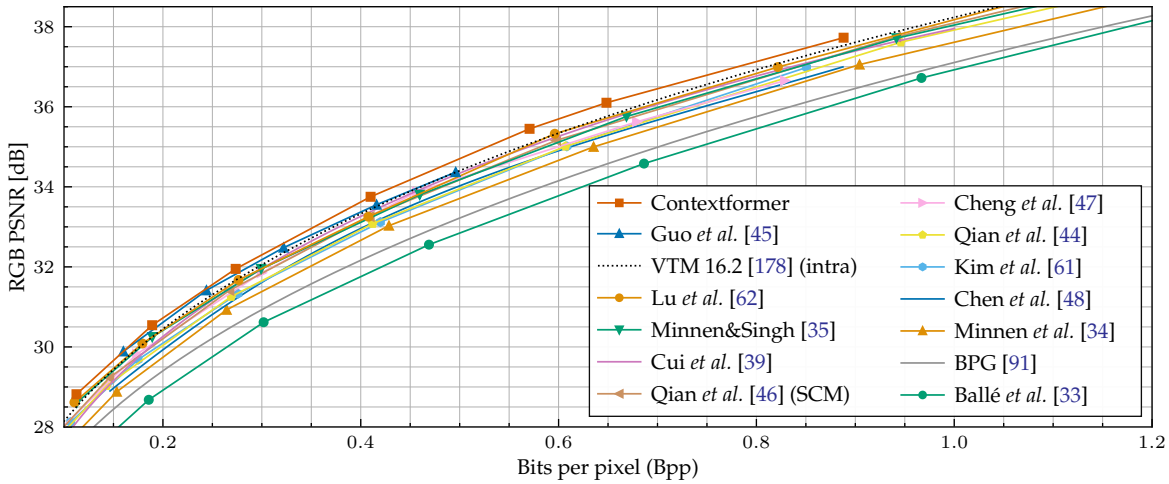


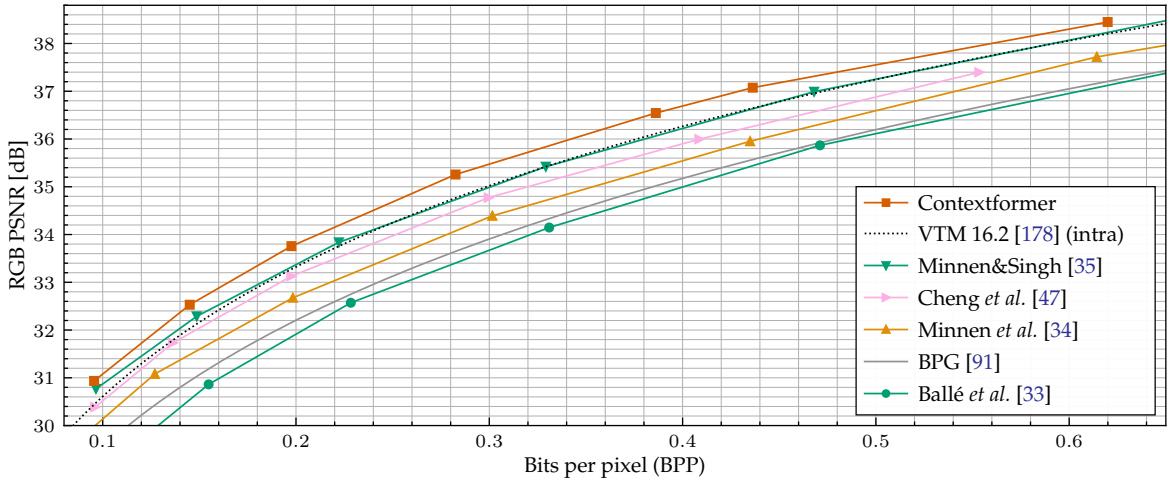
Table 4.2: Key properties of the state-of-the-art LIC frameworks compared to the Contextformer.

	Model Details	
Model	Encoder (g_a) and Decoder (g_s)	Entropy Model
Ballé *et al.* [33]	A similar framework as Contextformer (see Table 4.1) is used, but without sRABs and ResBlocks. It uses 5×5 convolutions instead of 3×3 and has lower channel capacity ($C_N = 128$ and $C_M = 192$).	A similar hyperprior network as in Contextformer is used (see Table 4.1), but with only one convolutional layer with stride 1 instead of three. It uses 5×5 convolutions instead of 3×3 and employs ReLU activation. The entropy model only employs a hyperprior network without a context model and uses a zero-mean Gaussian distribution as the probability model for $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$.
Minnen *et al.* [34]	A similar framework as Ballé *et al.* [33] is used, but with higher channel capacity ($C_N = 192$ and $C_M = 192$).	The same hyperprior network as Ballé *et al.* [33] is used, but it employs a context model using 2D masked convolutions and models the mean and variance of $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$ using a Gaussian distribution.
Minnen&Singh [35]	A similar framework as Ballé *et al.* [33] and Minnen *et al.* [34], with higher channel capacity ($C_N = 192$ and $C_M = 320$).	A similar hyperprior network as Minnen *et al.* [34] is used, but larger with higher channel sizes. Additionally, it contains the channel-wise autoregressive context model (ChARM) with multiple CNN layers modeling each channel segment (10 segments in total) using a Gaussian distribution probability model. More details available in Section 5.1.
Cheng *et al.* [47]	A similar framework as Contextformer (see Table 4.1) is used, but every convolutional layer is replaced by residual layers with a stride. The GDN and IGDN are replaced by residual CNNs with Leaky ReLU activation. It contains four attention modules similar to sRAB and has lower channel capacity ($C_N = 128$ and $C_M = 192$).	A similar hyperprior network as Contextformer is used (see Table 4.1), where the deconvolutions are replaced by subpixel convolutions. It employs a context model using 2D masked convolutions and models $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$ using a Gaussian Mixture Model (GMM) with 3 mixtures.
Cui *et al.* [39]	The same framework is used as Contextformer (see Table 4.1).	The same hyperprior network as Contextformer is used (see Table 4.1), but it employs a context model using multi-scale masked convolutions and an Asymmetric Gaussian Mixture (AGM) for modeling $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$.
Chen *et al.* [48]	A similar framework as Contextformer (see Table 4.1) is used, but it uses 5×5 convolutions instead of 3×3. It contains four attention modules with non-local mechanisms, which are more complex than sRAB, and includes four ResBlocks each having three layers.	The hyperprior implements a similar architecture as the analysis and synthesis transforms with attention modules and ResBlocks. The context model contains 3D masked convolutions, and a Gaussian distribution is used as the probability model.
Qian *et al.* [44]	A similar framework as Ballé *et al.* [33] and Minnen *et al.* [34] is used, but with higher channel capacity ($C_N = 192$ and $C_M = 384$) and implements a slightly more complex variant of GDN with a residual connection.	The same hyperprior network as Minnen *et al.* [34] is used, but it employs a context model using 2D masked convolutions with template matching-based global reference. A Gaussian distribution is used for probability modeling of $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$.
Guo *et al.* [45]	A similar framework as Cheng *et al.* [47] is used, but it contains seven ResBlocks.	The same hyperprior network as Cheng *et al.* [47] is used. The context model contains MaskConv+ layers and an advanced global reference using top-k attention. A GMM with 3 mixtures is implemented for modeling $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$.
Qian *et al.* [46] (SCM)	A similar framework as Ballé *et al.* [33] and Minnen *et al.* [34] is used, but with higher channel capacity ($C_N = 192$ and $C_M = 384$).	The hyperprior network implements transformer-based layers with global attention. The context model has a similar architecture as the hyperprior layers, containing six transformer layers with only a global attention mechanism applied to spatial dimensions. Similar to Minnen *et al.* [34], $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$ is modeled with a Gaussian distribution.
Lu *et al.* [62]	A similar framework as Minnen *et al.* [34] is used, where all the activation functions are replaced by multiple transformer layers (Swin Transformer) with residual connections.	The hyperprior implements a similar architecture as the analysis and synthesis transforms with multiple Swin Transformer layers and residual connections. The context model uses a single-layer ViT-based transformer with a patch size of 5×5, computing attention for only spatial dimensions. The conditional probability $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$ is modeled with a Gaussian distribution, similar to Minnen *et al.* [34].
Kim *et al.* [61]	The same framework as Minnen *et al.* [34] is used.	The entropy model contains the context model using 2D masked convolutions from Minnen *et al.* [34], and two hyperprior networks. One contains only 1×1 convolutional layers, and the other has a single-layer ViT-based transformer with a patch size of 1×1, computing global attention for only spatial dimensions. Similar to Minnen *et al.* [34], a Gaussian distribution is used for modeling the conditional probability $p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}})$.

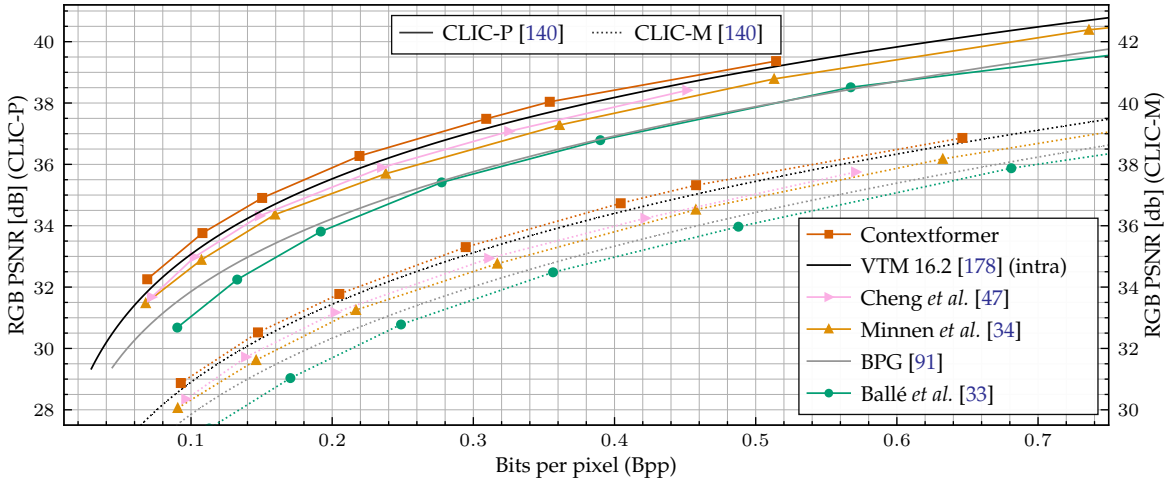




(a)

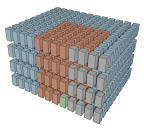


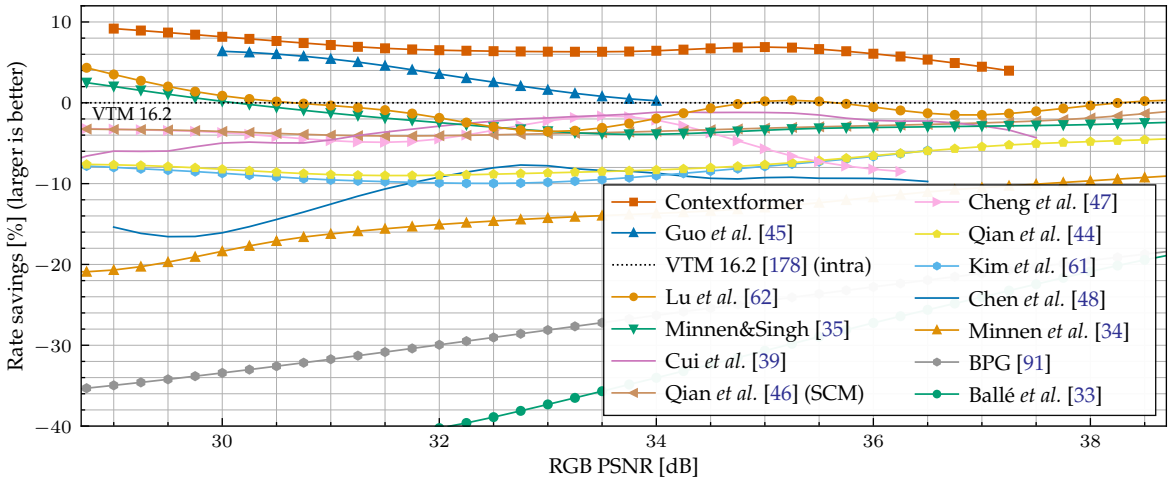
(b)



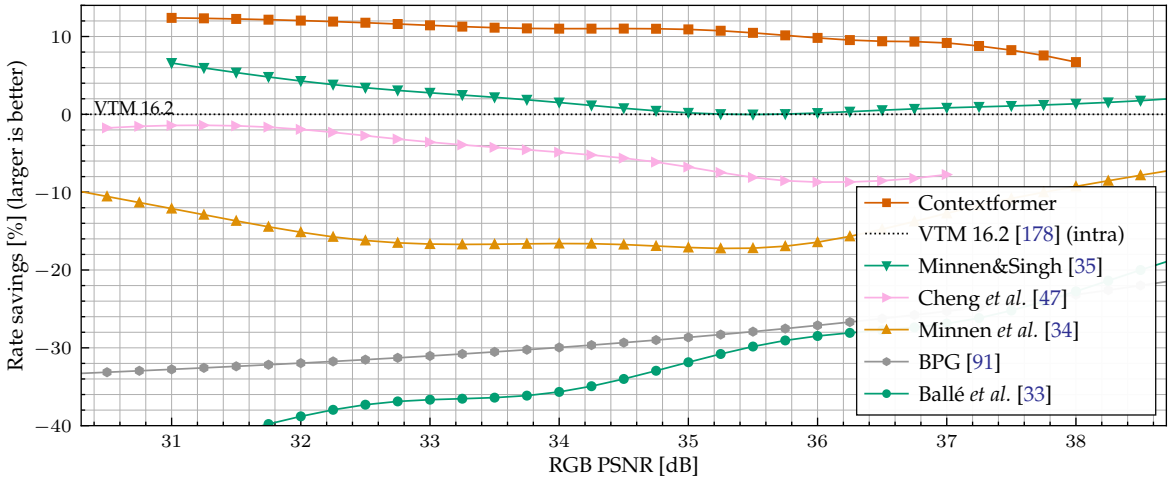
(c)

Figure 4.8: RD performance in terms of PSNR, comparing various frameworks to Contextformer across: (a) Kodak [138], (b) Tecnick [139], and (c) CLIC-P/-M [140] datasets.

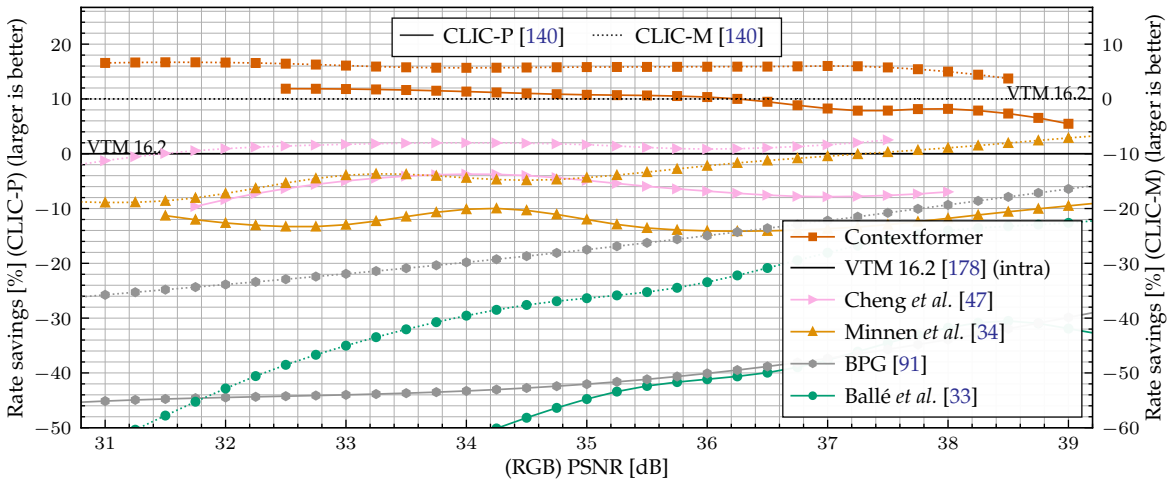




(a)

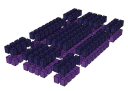


(b)



(c)

Figure 4.9: Rate savings relative to VTM 16.2 [178] as a function of PSNR, comparing various frameworks to Contextformer across: (a) Kodak [138], (b) Tecnick [139], and (c) CLIC-P/-M [140] datasets.



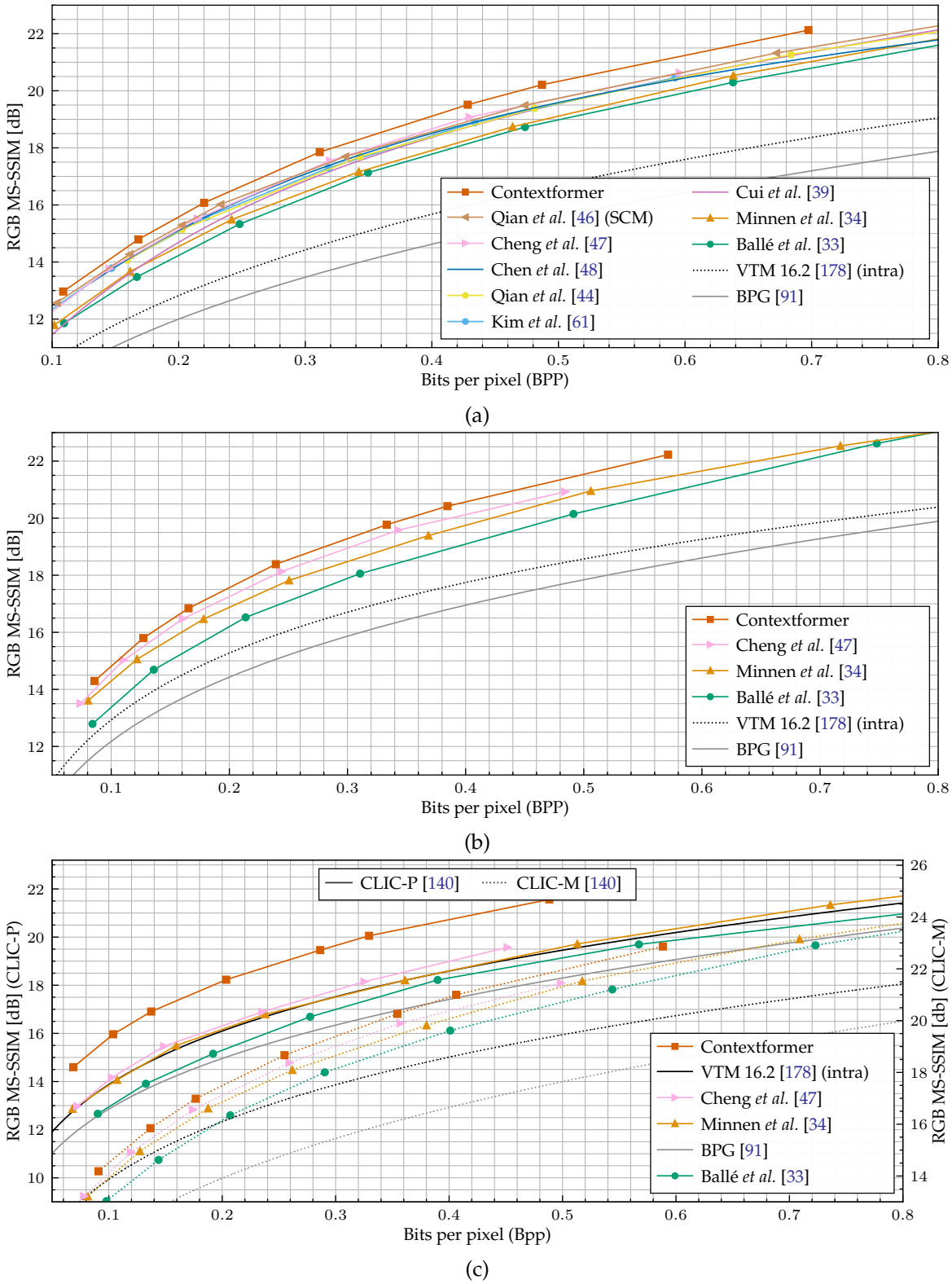


Figure 4.10: RD performance in terms of MS-SSIM [160], comparing various frameworks to Contextformer across: (a) Kodak [138], (b) Tecnick [139], and (c) CLIC-P/-M [140] datasets.

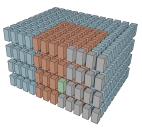


Table 4.4: Number of parameters of different modules and the entropy model complexity of various LIC frameworks compared to proposed model with Contextformer.

	Number of Parameters $[\times 10^6]$				
			Hyperprior		Context+Entropy
	Encoder	Decoder	Encoder	Decoder	Parameters
Method	(g_a)	(g_s)	(h_a)	(h_s)	(g_c+g_{ep})
Contextformer	8.1	9.4	1.6	2.4	15.9
Cui *et al.* [39]	8.1	9.4	1.6	2.4	17.2
Qian *et al.* [46]	3.8	3.8	8.2	16.0	13.2
Minnen *et al.* [34]	2.8	2.8	2.1	2.1	2.8
Minnen&Singh [35]	4.2	4.2	5.2	5.8	101.9

4.5.2.2 Model and Runtime Complexity

Table 4.4 shows the number of parameters in the proposed framework utilizing the Contextformer compared to various other frameworks. Notably, the proposed spatio-channel context modeling method contains an order of magnitude fewer parameters than the ChARM mechanism of Minnen&Singh [35], as it does not rely on additional modules to combine previously coded channel segments. Additionally, the proposed framework with the Contextformer has a similar total complexity to the baseline [39]. Compared to other transformer-based models such as Qian *et al.* [46], the Contextformer framework exhibits a different distribution of complexity across the entire framework. The analysis and synthesis transforms in the proposed framework are 2.3x more complex than those in [46], while the framework employs entropy modeling (h_a+h_s+g_c+g_e) that is 1.8x simpler, leading to lower overall complexity compared to [46].

Table 4.5 presents the encoding and decoding time complexities of the Contextformer using both existing and proposed sliding-window computation methods, alongside comparisons with several prior-art LIC frameworks and VTM 16.2 [178]. The results indicate that padding-based approaches (Pad), even when combined with batch processing (Pad&Batch), are less efficient compared to dynamic sequence length methods (DS). The proposed BDS algorithm, which leverages both dynamic sequence lengths and batch-wise parallelism, achieves up to a 2x speed-up over DS. When combined with SCS, the encoding complexity of BDS approaches that of the 3D context model [40] and the transformer-based SCM with global spatial attention from [46]. This combined parallelization approach shows even greater benefits when applied to 4K images, where the encoding time increases by only 15x relative to the Kodak [138] dataset, despite a 37x increase in pixel count. Additionally, the implementation of wavefront coding (3D-WPP) results in decoding times that are 9x faster than a typical 3D context model.

One issue encountered by frameworks like [46] is the quadratic increase in memory re-

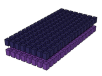


Table 4.5: Encoding and decoding time of different compression frameworks compared to the framework with Contextformer using proposed optimization methods.

	Encoding Time [s]		Decoding Time [s]	
Method	Kodak	4K*	Kodak	4K*
Pad	97	2100	114	2480
Pad&Batch	72	1510	–	–
DS	56	1240	62	1440
BDS	32	600	–	–
BDS&SCS	8	120	–	–
3D-WPP	40	760	44	820
Cheng *et al.* [47]	2	54	6	140
Qian *et al.* [46] (SCM)	2	–**	74	–**
Chen *et al.* [48]	4	28	316	7486
VTM 16.2 [178]	420	950	0.8	2.5

* The time is measured on 4K images with 3840x2160 resolution.
** Can not be computed due to out of memory (OOM)

quirements for their global attention mechanism and hyperprior, which made experiments with 4K images infeasible on GPUs with 24 GB of memory, resulting in out-of-memory (OOM). Another observation is that frameworks such as [40], [46], [47] implement vectorized operations, i.e., Single Instruction, Multiple Data (SIMD), optimized by CUDA [170] for the encoder's autoregressive process, helping to accelerate encoding. However, they still rely on explicit loops for the decoder, which leads to higher time complexity during decoding compared to encoding. Conversely, the Contextformer applies explicit loops on both the encoder and decoder, due to the straightforward implementation of sliding-window attention. Therefore, a more optimized encoder-side implementation could further reduce the time complexity in the future.

Table 4.6 highlights the impact of various sliding-window computation methods on the algorithmic complexity of the Contextformer. Methods that support dynamic sequence lengths (DS, BDS, and 3D-WPP) roughly halve the complexity compared to padding-based methods (Pad and Pad&Batch) by eliminating redundant computations in the attention mechanism. Although BDS and 3D-WPP offer significant runtime improvements, they do not reduce algorithmic complexity compared to DS, as they only increase the number of coordinates processed simultaneously without changing the total number of coordinates. In contrast, SCS reduces complexity by 75% as it processes a single channel segment at a time, rather than all four. However, this reduction is only applicable on the encoder side, leaving the decoder-side complexity unchanged. Overall, the Contextformer's computational

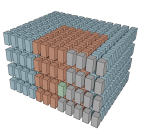


Table 4.6: Ablation study of the proposed optimization methods for Contextformer with respect to the effects on the context model complexity

Optimization Methods						Complexity [kMACpp]*
Pad	Pad&Batch	DS	BDS	SCS	3D-WPP	Context Model (g_c)
✓	✗	✗	✗	✗	✗	$227 \cdot 10^3$
✗	✓	✗	✗	✗	✗	$227 \cdot 10^3$
✗	✗	✓	✗	✗	✗	$120 \cdot 10^3$
✗	✗	✗	✓	✗	✗	$120 \cdot 10^3$
✗	✗	✗	✓	✓	✗	$30 \cdot 10^3$
✗	✗	✗	✗	✗	✓	$120 \cdot 10^3$

* The time is measured on 4K images with 3840x2160 resolution.

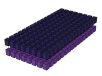
complexity remains high, limiting its deployment to high-end hardware. The limitations are further explored in Chapter 5, where more efficient architectures based on the Contextformer are proposed.

4.5.2.3 Ablation Studies

Fig. 4.11a depicts the performance variations of the Contextformer with respect to differing numbers of channel segments (N_{cs}), and the coding strategy ($\mathcal{S}_c$). It is evident that performance enhancements correlate with an increase in channel segments, attributed to a more comprehensive exploration of cross-channel dependencies. Nevertheless, a balance must be maintained, as complexity and computational costs rise quadratically with an increase in N_{cs} (see Eq. (4.6)). Observations indicate that exceeding four channel segments during training leads to increased complexity with diminishing gains in performance.

Fig. 4.11b presents the effect of different model sizes on the Contextformer's performance, compared to the base configuration. Modifications to both the network depth (L), and the embedding dimension (d_e), similarly affect model performance, with the most significant improvements observed when both parameters are increased. However, an enhancement plateau is reached beyond a network depth of 8 layers. Given the state-of-the-art performance of the Contextformer with the base configuration, further upscaling of the model, e.g., L>12, was not pursued to avoid unnecessary complexity. Notably, the base model of the Contextformer and the proposed network in [39] share a similar complexity in entropy and context modeling, yet the Contextformer achieves an additional 10% BD-Rate [169] performance gain on the Kodak dataset.

Comparatively, the Channel-first Order (CfO) configuration of the Contextformer demonstrates superior performance over the Spatial-first Order (SfO) model, highlighting the critical role of cross-channel dependencies. As shown in Fig. 4.12a, the CfO variant allocates more than 70% of the informational content to the first two channel segments, in contrast



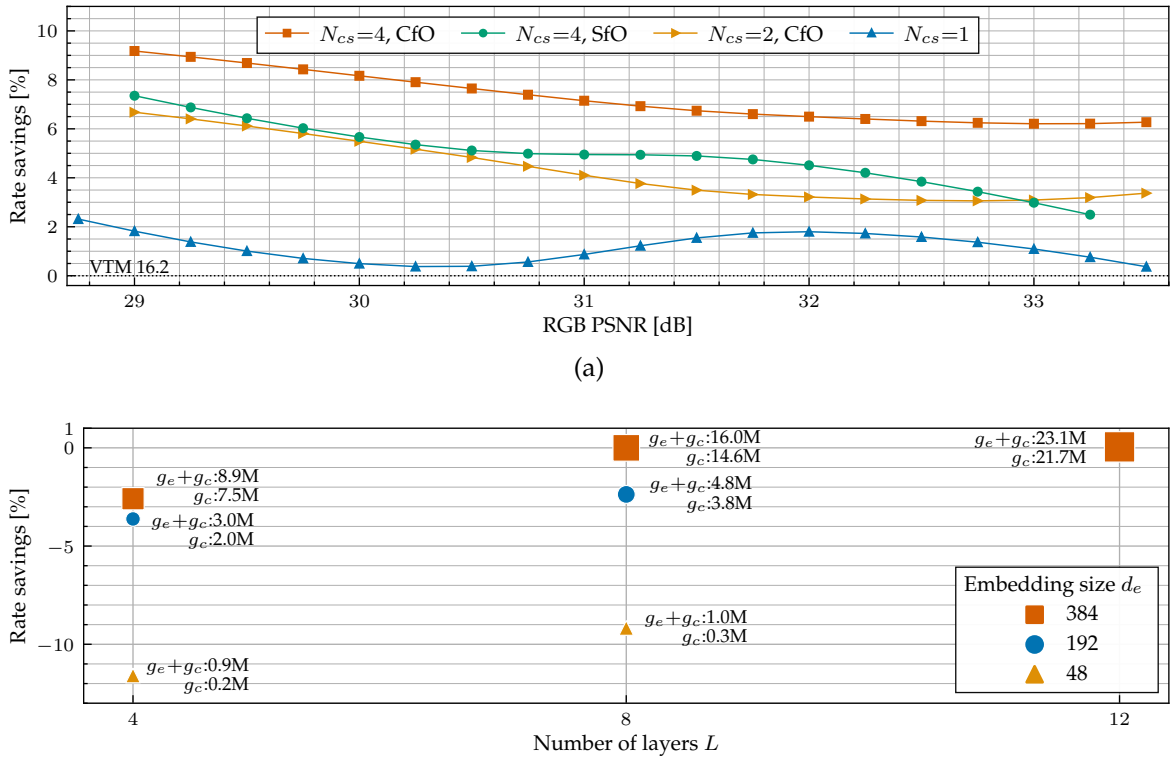
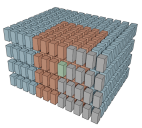


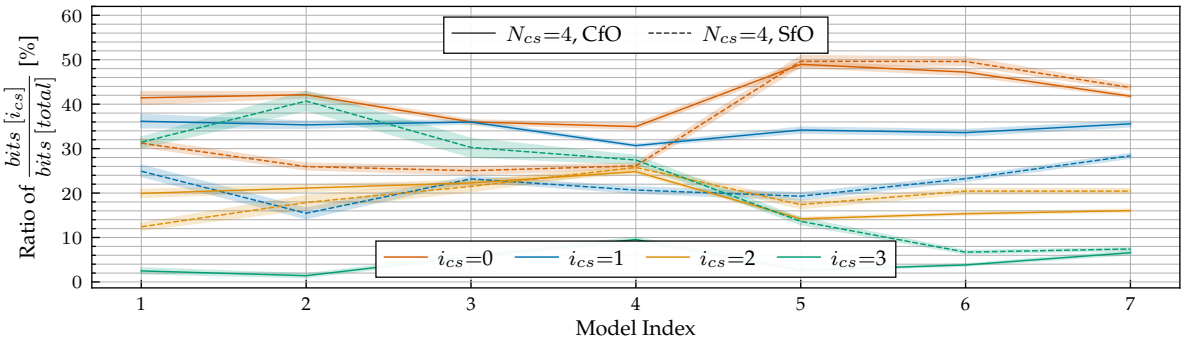
Figure 4.11: Ablation studies on various Contextformer models using the Kodak [138] dataset, illustrating the effects of hyperparameters on rate savings. In (a), models maintain the base configuration while varying the number of channel segments (N_{cs}) and coding strategies ($\mathcal{S}_c$). The performance is compared to to VTM 16.2 [178]. In (b), the influence of model size is examined by adjusting the embedding size (d_e) and the number of layers (L), while performance is compared to the base configuration. The size of each point represents the combined number of parameters in the entropy model parameter network (g_{ep}) and the context model (g_c), with the total number of parameters annotated.

to the SfO model's relatively uniform distribution. For example, at low to mid-range bitrates, the last channel segment in the SfO model predominantly contains more information. However, this information cannot be used for coding elements from the preceding three channel segments due to the coding order, potentially resulting in inefficient modeling. Fig. 4.12b shows the proportion of images from the Kodak dataset according to the coding order, with lower rate-distortion loss (see Eq. (2.23)). The figure indicates that spatial-first coding marginally provides benefits at lower bitrates, i.e., models with $\lambda_{<4}$—corresponding to Bpp < 0.3 on average—particularly in uniformly textured images, denoting an elevated significance of spatial dependencies in such smoother images.

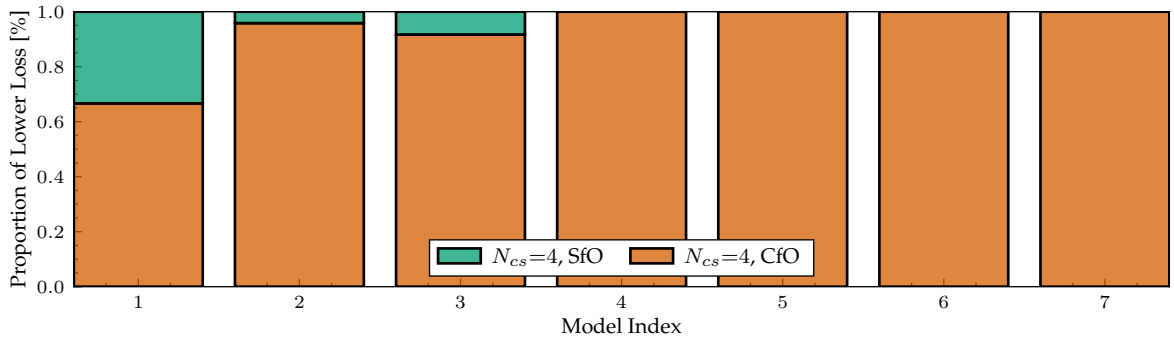
4.5.2.4 Visual Results

To evaluate qualitative performance, reconstructed images from the Contextformer models (optimized separately for MSE and MS-SSIM) are compared with those from traditional codecs such as BPG [91] and VTM 16.2 [178]. Fig. 4.13 illustrates the reconstructions of *kodim07* (Kodak [138] dataset), along with several cropped sections of the images. Each image was reconstructed at approximately 0.1 Bpp. Similarly, Fig. 4.14 depicts reconstructions





(a)



(b)

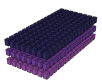
Figure 4.12: Ablation studies on Contextformer models with different coding strategies ($\mathcal{S}_c$) using the Kodak [138] dataset. In (a), the percentile bit distribution across channel segments is illustrated for models utilizing different $\mathcal{S}_c$, where each model index corresponds to a model trained for a specific λ. Notably, the CfO variant allocates more bits to the first two segments across all model indices, while last channel segment of the SfO dominates other segments in the low bitrate models. (b) displays the percentage of Kodak [138] images where one $\mathcal{S}_c$ surpasses another in terms of RD loss according to Eq. (2.23).

of *kodim23* at approximately 0.06 Bpp.

From both figures, it is evident that traditional codecs exhibit noticeable artifacts, such as blocking, aliasing, and smearing, particularly in regions with high-frequency details such as edges and textures. In contrast, both the MSE- and MS-SSIM-optimized versions of the Contextformer preserve finer details, resulting in sharper contours and fewer visual distortions. This leads to improved perceptual quality, as reflected by the higher PSNR and MS-SSIM metrics achieved at lower bitrates in comparison to the classical codecs.

The distortions introduced by the Contextformer are influenced by the loss functions used during optimization. The MSE-optimized model produces less noisy reconstruction with more vivid colors, though they may appear slightly blurred, particularly in areas requiring fine detail. Meanwhile, the MS-SSIM-optimized model excels in maintaining structural and textural consistency, making it visually more appealing in terms of preserving the overall appearance of objects.

Objectively, as shown Fig. 4.13, the MSE-optimized Contextformer shows a PSNR of 30.834 at 0.098 Bpp compared to VTM 16.22 [178] (30.230 dB at 0.108 Bpp) and BPG [91] (28.791 dB at 0.103 Bpp). The MS-SSIM-optimized version demonstrates its capacity to en-



hance perceptual quality with an MS-SSIM score of 0.971 at 0.086 Bpp. In Fig. 4.14, the MSE-optimized Contextformer reconstruction achieves a PSNR of 32.268 dB at 0.059 Bpp, surpassing both VTM 16.22 [178] (31.617 dB at 0.063 Bpp) and BPG [91] (30.175 dB at 0.060 Bpp). Similarly, the MS-SSIM-optimized model achieves an MS-SSIM of 0.967 at 0.065 Bpp, outperforming the other codecs.

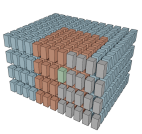
Overall, the results indicate that the Contextformer provides superior RD performance and better preserves the visual integrity of the images compared to traditional codecs, even at lower bitrates.

4.6 Chapter Summary

This chapter provided an in-depth investigation of state-of-the-art LIC frameworks, with a particular focus on various context models. It identified key limitations of existing methods, such as their inability to fully exploit spatial and channel-wise dependencies in the latent space, and their limited adaptability to content variations in context modeling. To address these challenges, the chapter introduced the Contextformer, a transformer-based context model that advances beyond spatial attention mechanisms to a novel spatio-channel attention approach, effectively capturing both spatial and cross-channel dependencies in a dynamic manner.

The chapter extensively explored different configurations of the spatio-channel attention mechanism, along with various architectural choices and hyperparameters. Ablation studies provided valuable insights into how these design choices influenced compression performance and computational complexity, leading to the refinement of Contextformer's final architecture. Experimental results demonstrated significant improvements in rate-distortion performance, with Contextformer achieving up to an 11% reduction in BD-Rate [169] for PSNR compared to VTM 16.2 [178] on the test datasets. Additionally, Contextformer surpassed existing LIC frameworks in terms of PSNR and MS-SSIM across a range of context model architectures. Visual examples further highlighted Contextformer's ability to deliver superior perceptual quality at lower bitrates.

Furthermore, the chapter introduced several architectural and algorithmic optimizations that significantly improved Contextformer's computational efficiency while maintaining its compression performance. However, despite these improvements, Contextformer remains resource-intensive, though it demonstrates considerable promise for scenarios requiring high compression performance, such as cloud storage. The chapter's findings lay the groundwork for the next chapter, which focuses on achieving more computationally efficient innovations.



	Bpp	PSNR	MS-SSIM
BPG	0.103	28.791	0.946
VTM 16.2	0.108	30.230	0.960
Contextformer (MSE)	0.098	30.834	0.968
Contextformer (MS-SSIM)	0.086	27.542	0.971

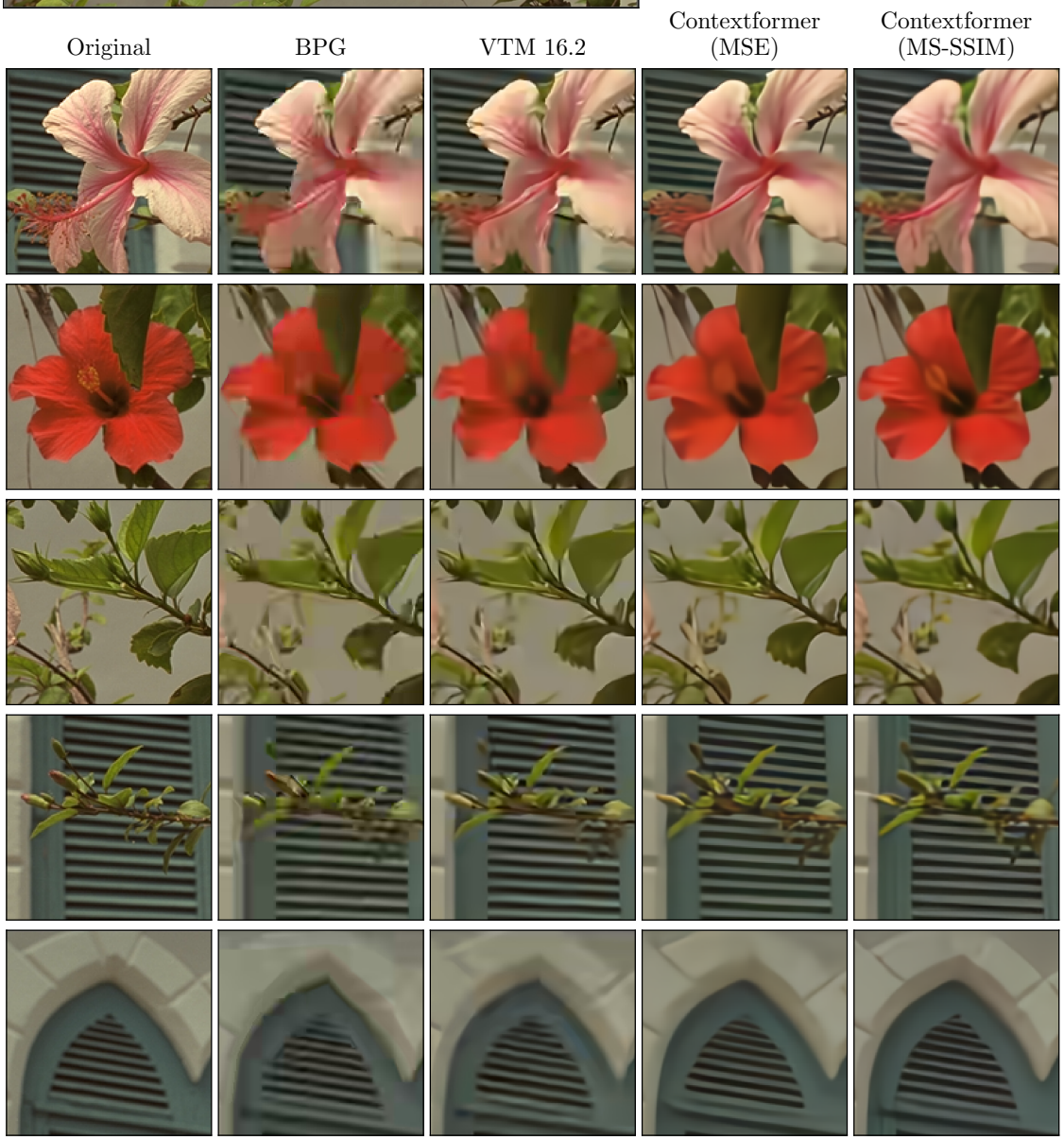
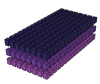
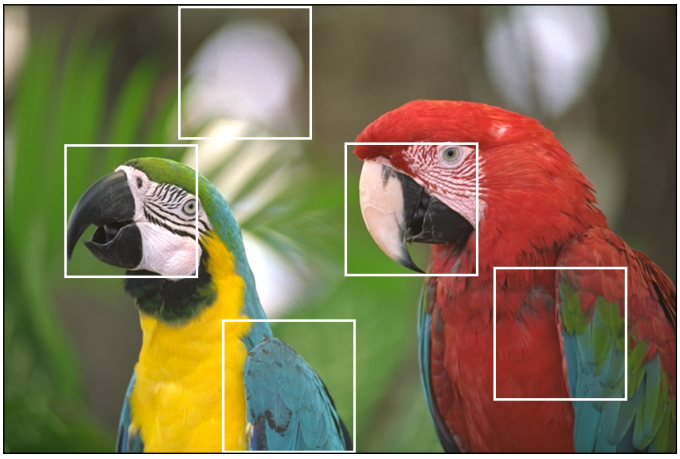


Figure 4.13: The *kodim23* image from the Kodak [138] dataset, and several crops from the images compressed by BPG [91], VTM 16.2 [178], Contextformer (optimized for MSE or MS-SSIM), where each algorithm requires about 0.1 Bpp.





	Bpp	PSNR	MS-SSIM
BPG	0.060	30.175	0.922
VTM 16.2	0.063	31.617	0.941
Contextformer (MSE)	0.059	32.268	0.952
Contextformer (MS-SSIM)	0.065	29.694	0.967

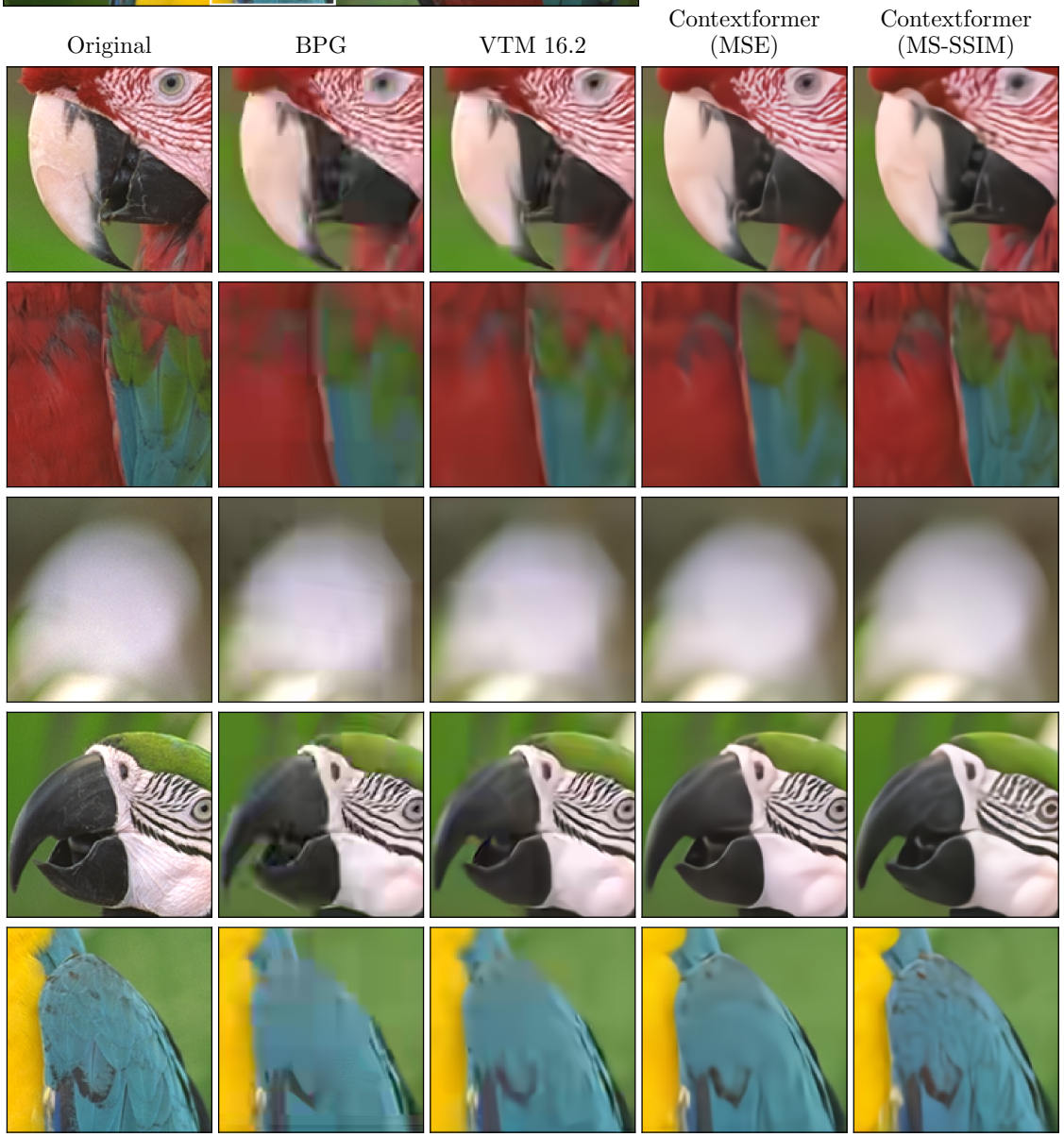
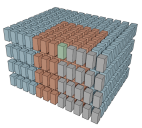


Figure 4.14: The *kodim07* image from the Kodak [138] dataset, and several crops from the images compressed by BPG [91], VTM 16.2 [178], Contextformer (optimized for MSE or MS-SSIM), where each algorithm requires about 0.06 Bpp.



Algorithm 1: Algorithmic optimizations for Contextformer

Input: Seq. Latent Tensor $\hat{\boldsymbol{s}}$, Contextformer g_c, Coding Strategy $\mathcal{S}_c$, Optimization Method *opt*

Output: Contextformer Output $\boldsymbol{\Omega}$

$Q_{ctx}, Q_{crt} \leftarrow \{\}$ `// initialize priority queues`
$\boldsymbol{\Omega} \leftarrow \mathbf{0}$ `// initialize Context Model Output`
$H, W \leftarrow get\_original\_shape(\hat{\boldsymbol{s}})$
$N_{cs}, \mathcal{S}_c, K \leftarrow get\_parameters(g_c)$`// hyperparameters of Contextformer`

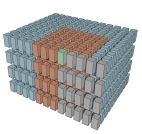
`//loop over sliding windows & store indices`
foreach *coord. (i,j,k)* ***in*** *3DMeshCube(H,W,*N_{cs}*)* **do**
    `//get indices residing in current window`
    $indices \leftarrow get\_indices(i, j, k)$
    `//compute priority described in Algorithm 2`
    $priority \leftarrow get\_priority((i, j, k), indices, (H, W, N_{cs}), \mathcal{S}_c, K, opt)$
    `//store current latent index`
    $Q_{crt}[priority].append((i, j, k))$
    `//store latent indices used as context. If "SCS" method is utilized, store only the context for the last segment`
    if *"SCS"* ***in*** *opt* ***and*** $k < N_{cs} - 1$ **then**
        continue
    else
        $Q_{ctx}[priority].append(indices)$
    end
end

`//loop over coding priorities & run context model`
foreach *priority* ***in*** Q_{crr} **do**
    `//restore latent indices`
    $i_{crt} \leftarrow Q_{crt}[priority]$
    $i_{ctx} \leftarrow Q_{ctx}[priority]$
    `//run context model for the current latents`
    $\boldsymbol{\Omega}[i_{crt}] \leftarrow g_c(\hat{\boldsymbol{s}}[i_{ctx}])$
end

Algorithm 2: Helper function computing the priority of each sliding window ($get\_priority$)

Input: Current Coordinates (i, j, k), Indices within Current Window $indices$, Latent Tensor Size (H, W, N_{cs}), Coding Strategy $\mathcal{S}_c$, Spatial Kernel Size K, Optimization Method opt
Output: Priority $priority$

if opt ***is*** *"None"* **then**
  `//if no optimization is in use, the priority is set to the linear index according to the coding order`
  if $\mathcal{S}_c$ ***is*** *"SfO"* **then**
    $priority = H \cdot W \cdot k + W \cdot i + j$
  end
  if $\mathcal{S}_c$ ***is*** *"CfO"* **then**
    $priority = W \cdot N_{cs} \cdot i + N_{cs} \cdot j + k$
  end
end
if opt ***is*** *"BDS"* **then**
  `//if BDS is in use, the priority is set to length of the number of elements within the window`
  $priority = length(indices)$
end
if opt ***is*** *"3D-WPP"* **then**
  `//if 3D-WPP is in use, the priority is set to wavefront index according to the coding order`
  if $\mathcal{S}_c$ ***is*** *"SfO"* **then**
    $priority = (K \cdot (H - 1) + W) \cdot k + K \cdot i + j$
  end
  if $\mathcal{S}_c$ ***is*** *"CfO"* **then**
    $priority = K \cdot N_{cs} \cdot i + N_{cs} \cdot j + k$
  end
end



Chapter 5

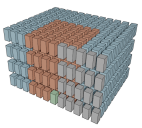
High-efficiency Learned Image Compression Framework

This chapter explores several architectural designs for parallel context modeling and introduces a computationally efficient transformer-based autoregressive context model, aptly named the *Efficient Contextformer (eContextformer)*, building upon the previous work, Contextformer, introduced in Chapter 4. The eContextformer incorporates recent advancements in efficient transformer models and fast context model architectures, fusing patch-wise, checkered, and channel-wise grouping techniques through the introduction of a novel shifted window spatio-channel attention mechanism. Additional complexity optimizations are proposed, which can dynamically scale the attention span and cache previous attention computations, significantly reducing runtime and model complexities. These enhancements enable the use of online rate-distortion algorithms (oRDO), described in Chapter 2, to further improve compression performance. In this chapter, the performance and complexity of the eContextformer are comprehensively analyzed based on empirical evaluations conducted on commonly used test datasets. Furthermore, ablation studies are presented to demonstrate the effects of model design, training strategies, and hyperparameters.

Parts of the work presented in this chapter have been published in [1]–[3].

5.1 Motivation

As explained in Chapter 2 and Section 4.1, compression performance is strongly influenced by the precision of entropy modeling, where context modeling plays a fundamental role by analyzing dependencies between latent elements and using this information to estimate the conditional probabilities of the values. This process facilitates more efficient encoding and reduces redundancy. To improve estimation accuracy, state-of-the-art approaches [37]–[42], [44]–[46], including the Contextformer introduced in Chapter 4, implement a serial autoregressive context model, handling each individual latent element sequentially. While effective in performance, such approaches are inherently limited by their sequential nature, leading to significant computational overhead. Each prediction step depends on the completion of its predecessors, introducing latency and limiting encoding and decoding speed. Moreover, these models often fail to fully leverage modern parallel computing architectures, such as



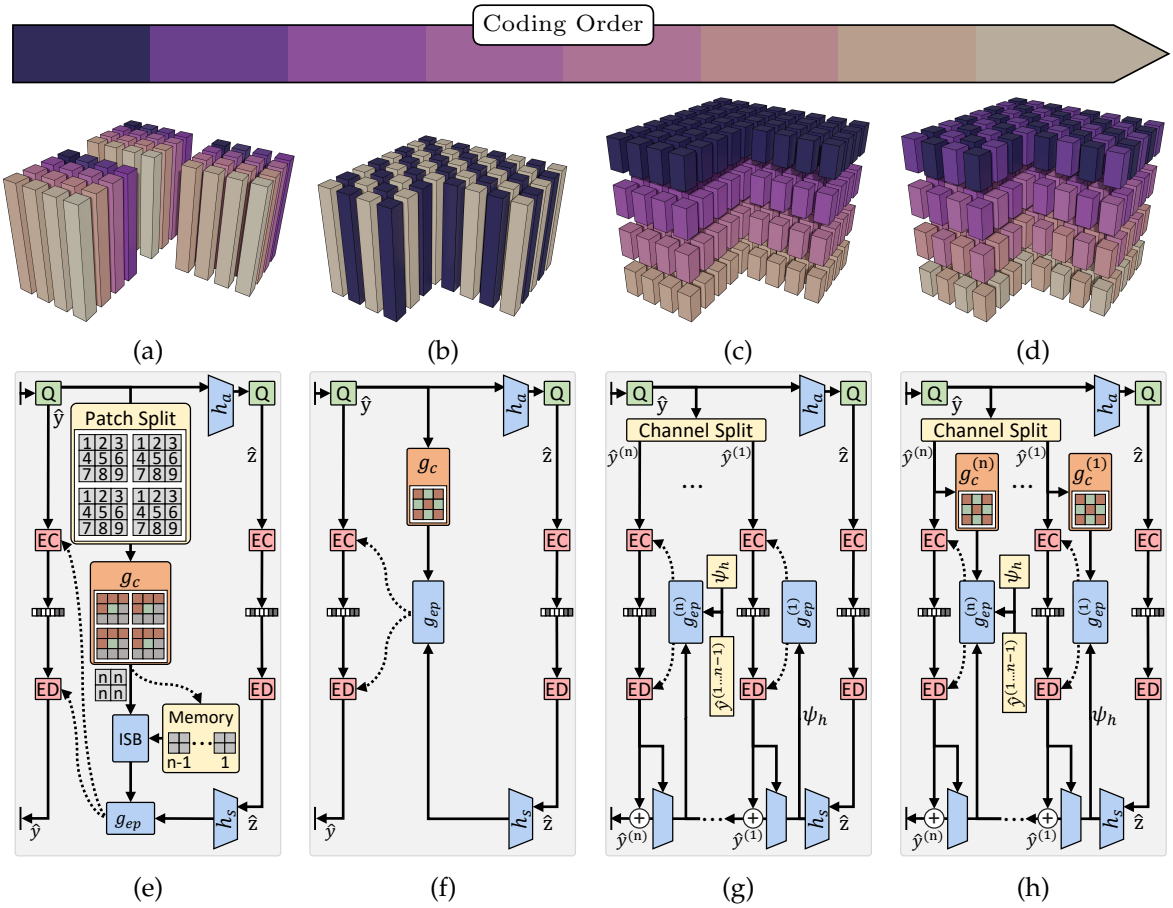


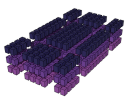
Figure 5.1: Illustration of various latent grouping techniques used for parallelized context modeling: (a) patch-wise grouping [3], (b) checkered grouping [46], [52], (c) channel-wise grouping [35], and (d) a combination of checkered and channel-wise grouping [56], [59]. Latent elements are coded according to Coding Order, where elements within the same group (colored identically) are coded simultaneously. The context model can utilize information from previously coded groups. (e-h) provide simplified illustrations of the corresponding entropy models for (a-d), respectively. In (e), Koyuncu *et al.* [3] employs multi-scale 2D context modeling with an Information Sharing Block (ISB), which aggregates inter-patch information using a recurrent neural network. For simplicity, only one kernel of the multi-scale model is depicted. In (f), He *et al.* [52] uses checkered masks for convolution. In (g), Minnen&Singh [35] code each channel segment separately, applying multiple CNNs to channel-wise concatenated previously coded groups. Finally, He *et al.* [56] combines checkered and channel-wise grouping techniques, as illustrated in (h).

NPUs and GPUs, which are designed to efficiently handle multiple operations concurrently. The inability to parallelize tasks within the context modeling phase represents a critical inefficiency in these LIC frameworks.

The inherent limitations of serial autoregressive models have driven the development of parallel approaches for enhancing the computational efficiency of context modeling. These optimizations can be grouped into two categories:

1. Algorithmic optimizations [2], [51], [12]

2. Architectural optimizations [3], [35], [46], [52], [56], [59], [60], [63], [64]

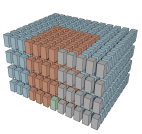


The first group includes optimizations such as wavefront coding [51], [12], and SCS, BDS, and 3D-WPP introduced in Section 4.3. These algorithms exploit the properties of existing context models by determining which latent elements can be processed simultaneously. Since they use the existing context model architecture, they maintain the LIC's compression performance. However, they achieve only marginal improvements in reducing complexity due to the limited number of independent latent elements—the conditional entropy of each latent element is still iteratively dependent on all preceding elements.

The second group focuses on limiting the conditional entropy computation by dividing the latent tensor into independent coding groups that can be processed simultaneously. Studies [3], [35], [46], [52], [56], [59], [60], [63], [64] have introduced several parallel context model architectures by exploring various heuristics to group latent elements (see Fig. 5.1). Inspired by block-based processing in classical coding frameworks [17], [24], [25], [192], Koyuncu *et al.* [3] split the latent tensor into spatially non-overlapping patches. A multi-scale context model [37]–[39] simultaneously exploits local relations inside each patch. However, straightforward parallelization significantly lowers compression performance since information within each patch is not available to other patches. To address this, a recurrent neural network (RNN)-based module, called the Information Sharing Block (ISB), was proposed to exploit inter-patch relations by aggregating information from co-located tensor elements across patches. Later, Lin *et al.* [64] explored different coding orders within each patch to avoid the need for additional modules like ISB.

In [52], He *et al.* investigated correlations in the latent space by training a context model conditioned on randomly generated masks. Their experiments revealed that the highest correlations lie in a cross-shaped pattern—spatially adjacent neighbors are strongly correlated, while diagonal neighbors share less mutual information. Based on this analysis, they proposed splitting latent elements into two groups using a checkered pattern. The first group's entropy is estimated solely by the hyperprior network, while a context model with a checkered mask is used for the second group. As illustrated in [64], the checkered grouping can be seen as a special case of patch-wise grouping, where the patch size is 2×2. Moreover, Qian *et al.* [46], explained in Section 4.1, introduced a parallel version of their context model architecture, adopting the checkered grouping technique in their transformer-based context model with spatial attention. Additionally, Wang *et al.* [60] developed a hybrid model incorporating checkered patterns across both spatial and channel dimensions, simplifying model complexity but sacrificing some RD performance.

The ChARM architecture [35], described in Section 4.1, is another example of a parallel context model. It uses a channel-wise grouping strategy, splitting the latent tensor into channel segments while discarding spatial relations within each segment. Each segment is concatenated with its preceding segments after decoding, and the concatenated tensor is processed by additional CNN-based modules aggregating information between channel segments. The models in [56], [63] achieve higher RD performance by utilizing combinations of checkered and channel-wise groupings. He *et al.* [56] extends the ChARM [35] architecture with checkered masked convolutions, while Lu *et al.* [63] uses a similar framework but splits the latent tensor into four groups using a checkered pattern. Liu *et al.* [59] enhances the

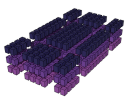


framework of [56] by incorporating transformer-based modules with spatial attention.

The architectural optimizations proposed in the literature are not limited to context modeling. For instance, multiple studies [57], [58], [60] introduced asymmetrical compression frameworks, shifting the complexity from the decoder to the encoder. In these approaches, the synthesis transform has orders of magnitude lower complexity than the analysis transform. Wang *et al.* [60] presents a low-complexity decoder with real-time coding speed, while Yang and Mandt [57] achieves low complexity by avoiding the implementation of a context model. Fu *et al.* [58] provides a lower complexity decoder compared to their baseline, though their framework is still slow due to the implementation of a serial context model. In general, asymmetric frameworks achieve ultra-low complexities compared to symmetrical ones, yet their RD performance is not competitive with more advanced and high-complexity frameworks [2], [3], [35], [46], [52], [56], [59], [60], [63], [64]. Therefore, the results of asymmetrical models are reported in this thesis for completeness; however, deeper investigation of those methods is beyond the scope of this thesis.

While advancements in parallel context modeling significantly improve efficiency—reducing the runtime complexity of entropy modeling by two to three orders of magnitude—each method has its own limitations. Approaches using patch-wise and checkered groupings [3], [46], [52], [64] outperform their serial baselines at high bitrates in terms of RD performance but underperform at low bitrates, experiencing an average drop of up to 3% in BD-Rate [169]. Additionally, Koyuncu *et al.* [3] suffers from increased model complexity due to the ISB. Similarly, in order to process each channel segment, ChARM-based architectures [35], [56], [59], [63] aggregate information from all preceding channel segments using separate NN-based modules. This increases model complexity even more than the ISB, reaching two to three orders of magnitude higher complexity compared to non-ChARM-based approaches. He *et al.* [56] showed that combining ChARM with checkered grouping significantly narrows the performance gap between serial and parallel context model approaches, although the parallel model still shows a 1.5% drop in BD-Rate [169] compared to a serial baseline using ChARM with a 2D masked convolution. To close this gap, both He *et al.* [56] and Liu *et al.* [59] presented more complex analysis and synthesis transforms to compensate for the loss introduced by parallelization. For instance, He *et al.* [56] used multiple stacked residual blocks instead of GDN/IGDN in the analysis and synthesis transforms, while Liu *et al.* [59] replaced GDN/IGDN with CNN and transformer-based modules.

Moreover, these methods suffer from the limitations discussed in Section 4.1. For instance, context models with patch-wise and checkered groupings [3], [46], [52], [64] focus only on spatial dependencies in the latent space and do not fully employ content-adaptive modeling for spatio-channel elements. The ChARM [35] lacks the ability to exploit spatial dependencies. He *et al.* [56] and Lu *et al.* [63] do not use content-adaptive mechanisms, and Liu *et al.* [59] introduces content-adaptive context models through their transformer-based design but only implements a classical MHA, limiting the attention mechanism for the channel dimension, as explained in Section 4.2.2.



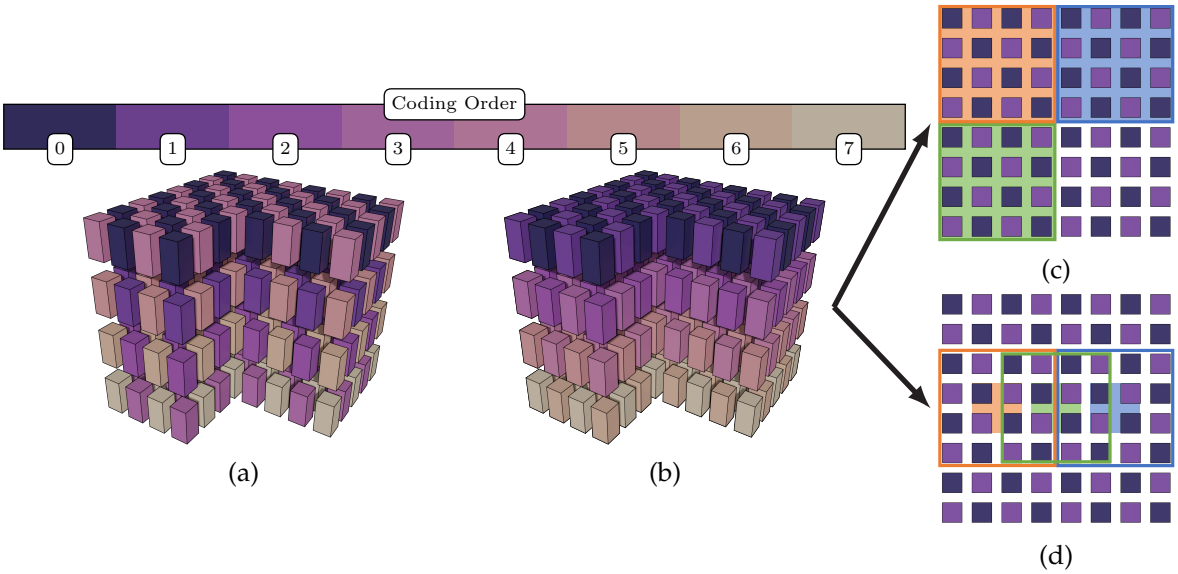


Figure 5.2: Illustration of context modeling in the experimental pContextformer for different coding strategies ($\mathcal{S}_c$): (a) Spatial-first-Order (SfO) and (b) Channel-first-Order (CfO). Latent elements are encoded based on the Coding Order, with elements in the same group (colored similarly) being processed simultaneously. The context model uses information from groups that have already been encoded. (c-d) demonstrate the application of a 4×4 attention kernel on the latent groups in (b). In (c), attention is applied to non-overlapping patches with a stride of 4, while in (d), attention is applied to overlapping patches with a stride of 2. For simplicity, (c-d) show a top-down view of the latent tensor.

5.2 Investigations on Parallelization of Contexformer

Although the optimization techniques presented in Section 4.3 improve the efficiency of Contextformer, it still exhibits high complexity compared to prior methods, as shown in Section 4.5.2.2. Further investigation into the model's design reveals several interrelated areas for enhancement:

1. High computational complexity within each sliding-window
2. Discrepancy in model behavior between training and testing phases
3. Excessive number of required autoregressive steps

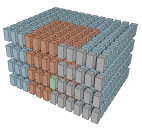
To recall and refine Eq. (4.5) and Algorithm 2, the complexity, with a stride of $s{=}1$ describing the spatial traverse of the sliding window, is reformulated as:

$$\mathbb{C}_{\text{ctx}} \sim N N_s \mathbb{C}_{\text{trans}}, \tag{5.1a}$$

$$\mathbb{C}_{\text{trans}} = (2N_w^2 d_e + 4N_w {d_e}^2) + (2N_w d_e d_{mlp}), \tag{5.1b}$$

$$N = \frac{HWN_{cs}}{256s^2}, \quad N_w = K^2 N_{cs}, \tag{5.1c}$$

where N represents the number of spatial positions of sliding-window, and N_s stands for the number of sequential steps or number of groups of elements which is coded sequentially within each window. For instance, $N_s{=}N_{cs}$ in Contextformer, since N_{cs} channel segments



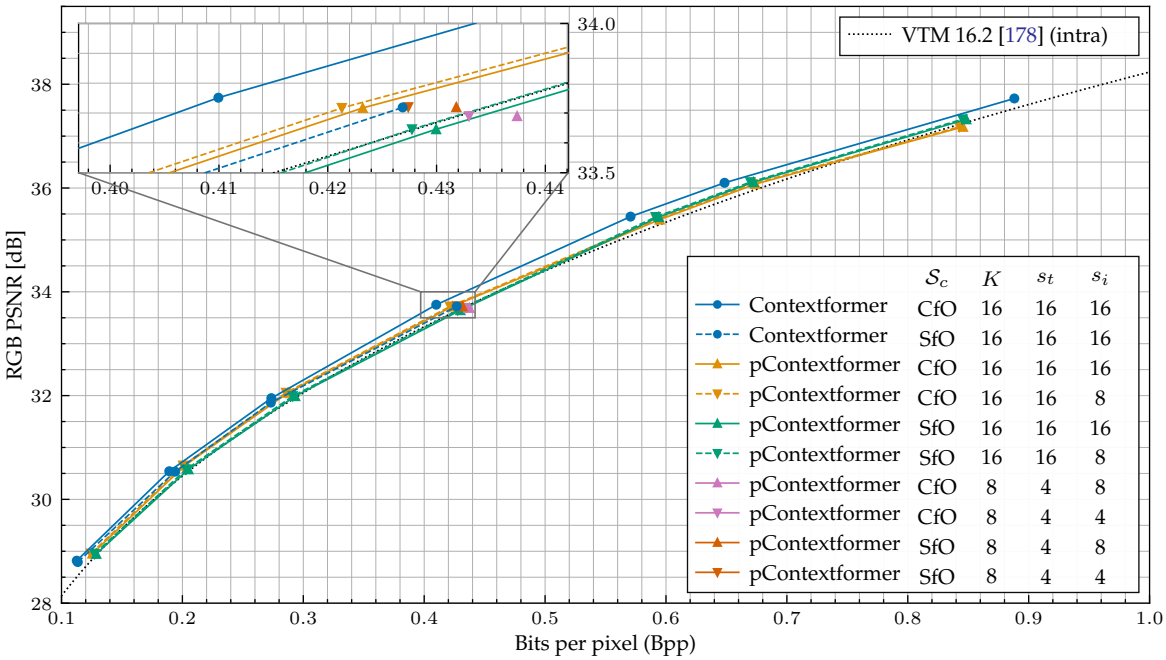


Figure 5.3: Ablation study on the experimental pContextformer model using the Kodak [138] dataset, illustrating the impact of various coding strategies ($\mathcal{S}c$), attention kernel sizes (K), and the strides used during training (s_T) and inference (s_I).

should be processed for each spatial coordinate independently. The total number of autoregressive steps for Contextformer is equivalent to $N{\cdot}N_s$. The complexity of the embedding layer is omitted for simplicity, as it is applied to all latent elements at once, prior to modeling.

In the Contextformer, a large spatial kernel size of $K{=}16$ was used during training to capture spatial dependencies in the latent space. This choice contributed to the first two issues: total complexity $\mathbb{C}_{\text{ctx}}$ grows biquadratically with kernel size K and quadratically with input resolution. To manage this complexity during training, Contextformer was trained on $256{\times}256$ image crops, effectively disabling sliding-window computation (since the spatial latent space resolution was $16{\times}16$). Although sliding-window attention reduces latency compared to global attention during inference, the Contextformer did not incorporate this behavior during training, leading to discrepancies between training and inference performances. To rectify this, training could be adjusted to incorporate sliding-window attention by either increasing the crop size or reducing the kernel size while keeping the crop dimensions constant. The first option increases training complexity without affecting inference complexity. The second option greatly reduces the computational complexity of each window while increasing the parallelization capability.

Even with a reduced kernel size, Contextformer still requires excessive autoregressive steps, as the sliding-window computation cannot be optimized at large scales due to $s{=}1$. To address this, increasing the stride reduces window interdependencies and quadratically reduces total complexity $\mathbb{C}_{\text{ctx}}$. In the extreme case of $s{=}K$, all window positions can be computed simultaneously, similar to the patch-wise grouping technique [3]. This reduces the number of autoregressive steps to N_w and the window positions to $N{=}\frac{HW}{256K^2}$. However, within each window, context modeling remains serial ($N_s{=}K^2N_{cs}$) without improving the

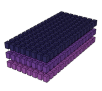


Table 5.1: BD-Rate [169] performance over VTM 16.2 [178] (intra) and complexity of various pContextformers with K=16 compared to Contextformer, showing the effect of coding strategy ($\mathcal{S}_c$) and using overlapped windows during inference.

Method	Coding Strategy	Overlapped Window @Inference	kMACpp	BD-Rate [%]
Contextformer	CfO	✓	$30 \cdot 10^3$	−6.9
pContextformer	CfO	✗	320	−3.1
	CfO	✓	1072	−3.5
	SfO	✗	320	−1.2
	SfO	✓	1072	−1.9

complexity. Moreover, patch-wise grouping can introduce discontinuities in context modeling, necessitating additional modules like ISB [3].

To address both issues, an experimental model, *parallelized Contextformer (pContextformer)* is introduced, which combines patch-wise [3] and checkered [52] grouping techniques. Fig. 5.2 illustrates the parallel context modeling strategy of the pContextformer. Integration of checkered grouping decouples the number of autoregressive steps from the window size, resulting in N_s=$2N_{cs}$. Moreover, it allows pContextformer to operate with arbitrary strides (s), enabling the use of overlapping or non-overlapping windows due to reduced window interdependencies (see Figs. 5.2c and 5.2d). Additionally, pContextformer inherently supports a channel-wise grouping strategy [35], leveraging the spatio-channel attention mechanism. However, unlike ChARM [35], it uses the same architecture to aggregate channel information. Overall, the model requires eight autoregressive steps for N_{cs}=4.

To evaluate performance and complexity, a set of pContextformer models was trained on 256×256 image crops, using the same strategies described in Section 4.5.1, for approximately 65% of the total training iterations. The coding strategy $\mathcal{S}_c$ (SfO or CfO) and kernel size K is varied during the experiments. Additionally, the effect of overlapping or non-overlapping windows is explored by selecting different stride parameters for both training (s_T) and inference (s_I). Overlapping windows are achieved by setting the stride (s_T or s_I) to $\frac{K}{2}$. When K=16, the model learns only global attention, but this could transition to local attention by varying the stride during inference. Performance and complexity are measured using the same methodologies as described in Section 4.5.1.2. For simplification of the experiments, complexity evaluation focuses on a single pass on the encoder side, processing the entire latent tensor $\hat{\boldsymbol{y}}$ simultaneously under a predefined causality mask.

The outcomes of these evaluations are detailed in Fig. 5.3 and Tables 5.1 and 5.2, leading to the following key observations:

- In case of $s_T = K = 16$: The pContextformer learns a global attention mechanism. The SfO strategy exhibits superior performance over CfO coding at high bitrates, yet it underperforms at lower bitrates. The reduced efficiency at low bitrates is due to

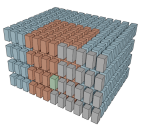


Table 5.2: Performance and complexity analysis of various pContextformer models with respect to attention kernel size (K), coding strategy ($\mathcal{S}_c$), and the use of overlapping windows during both training and inference. PSNR remains approximately the same across models, with lower ΔBpp values compared to the model in the first row indicating higher performance.

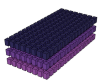
Coding		Overlapped Window			
Strategy	K	@Training	@Inference	kMACpp	ΔBpp
CfO	8	✓	✗	195	0
CfO	8	✓	✓	710	$-4 \cdot 10^{-3}$
SfO	8	✓	✗	195	$-6 \cdot 10^{-3}$
SfO	8	✓	✓	710	$-10 \cdot 10^{-3}$
CfO	16	✗	✗	320	$-14 \cdot 10^{-3}$
CfO	16	✗	✓	1072	$-16 \cdot 10^{-3}$

limited spatial dependency utilization, exacerbated by sparse latent space and non-overlapping windows during inference. The SfO strategy benefits more from overlapping window attention during inference ($s_I < K$).

- In case of $s_T < K$: The pContextformer learns an overlapping window attention. The SfO strategy demonstrates better performance than CfO strategy. Furthermore, using overlapping windows during inference particularly helps models with smaller kernel sizes ($K=8$) to achieve similar performance to those using larger kernels ($K=16$).
- Overall: The optimized pContextformer architectures reduce computational complexity exponentially—by over 100-fold—with a marginal performance degradation of 3%. More efficient models could potentially be developed by implementing overlapping window attention during both training and inference phases. However, the rudimentary deployment of such overlapping window attention, although theoretically advantageous, remains suboptimal as it quadruples the complexity.

5.3 Efficient Contextformer

The experiments conducted in Section 5.2 indicate that the pContextformer supports both checkered and patch-wise parallelization techniques, yet it necessitates a more sophisticated mechanism for managing overlapping window attention during training and inference. Based on the advancements in efficient transformers explained in Section 2.3.4, this can be achieved by substituting the initial ViT [85]-based architecture with a SwinT [88]-based alternative, complemented by the addition of spatio-channel attention. Fig. 5.4 illustrates the compression framework with the modified architecture, namely the Efficient Contextformer (eContextformer).



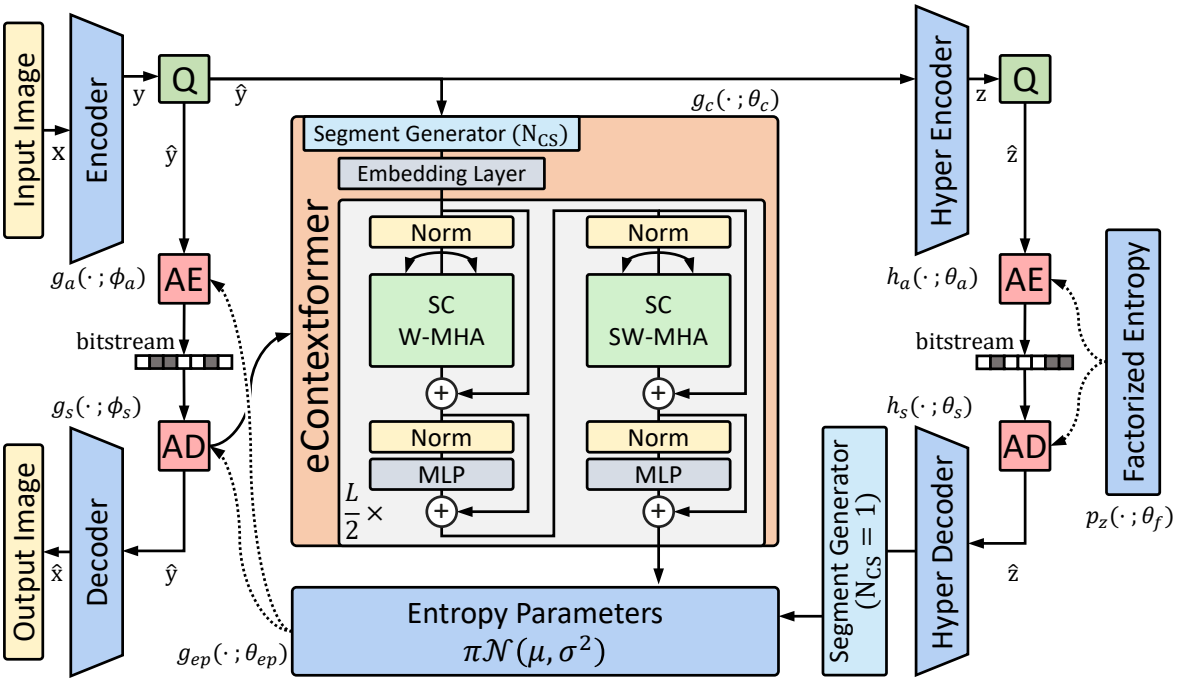


Figure 5.4: Illustration of the proposed LIC framework utilizing eContextformer with spatio-channel window and shifted-window multi-head attention (SC-W-MHA and SC-SW-MHA).

Fig. 5.5 shows the segment generator of the revised architecture. The segment generator rearranges the latent variable $\hat{\boldsymbol{y}} \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}\times C}$ into coding groups with a segmented spatio-channel format $\hat{\boldsymbol{s}} \in \mathbb{R}^{N_{cs}\times\frac{H}{16}\times\frac{W}{16}\times p_c}$, where the segmentation granularity is defined by $N_{cs}{=}\frac{C}{p_c}$. To avoid violating the autoregressive process, a zero-valued start token is prepended to the segmented tensor $\hat{\boldsymbol{s}}$, and its values are systematically shifted according to the coding group order, resulting in the padded and shifted representation $\hat{\boldsymbol{u}}$. Let $M(i,j)$ be the function representing the checkered indexing, formulated as:

$$M(i,j) = (i+j) \bmod 2, \tag{5.2}$$

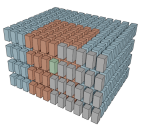
where (i,j) corresponds to the spatial coordinates of a tensor. Here, $M(i,j){=}0$ denotes the first checkered group, and $M(i,j){=}1$ denotes the second one.

The padded and shifted representation $\hat{\boldsymbol{u}}$ is mathematically defined as:

$$\hat{\boldsymbol{u}}_{n_{cs},i_u,j_u} = \begin{cases} 0, & \text{if } n_{cs}=0 \text{ and } M(i_r,j_r)=0, \\ \hat{\boldsymbol{s}}_{n_{cs},i_s,j_s}, & \text{if } n_{cs}\geq 0 \text{ and } M(i_u,j_u)=1 \text{ and } M(i_s,j_s)=0, \\ \hat{\boldsymbol{s}}_{n_{cs}-1,i_s,j_s}, & \text{if } n_{cs}>0 \text{ and } M(i_u,j_u)=0 \text{ and } M(i_s,j_s)=1, \end{cases} \tag{5.3}$$

where $n_{cs} \in [0, N_{cs}-1]$ represents the index of the channel segments, (i_u,j_u) and (i_s,j_s) stand for the spatial coordinates of each tensor. For simplicity, the last dimension is omitted.

Next, the rearranged data is transformed by a linear layer into a tensor embedding $\boldsymbol{e} \in \mathbb{R}^{N_{cs}\times\frac{H}{16}\times\frac{W}{16}\times d_e}$, expanding the last dimension. In each autoregressive step $n \in [0, 2N_{cs}{-}1]$, a part of the embedding representation $\boldsymbol{e}_{0\ldots\lfloor\frac{n}{2}\rfloor}$ traverses through L layers of a SwinT [88]. Here, only the first index of $\boldsymbol{e}$ is addressed for simplicity. The spatio-channel window and



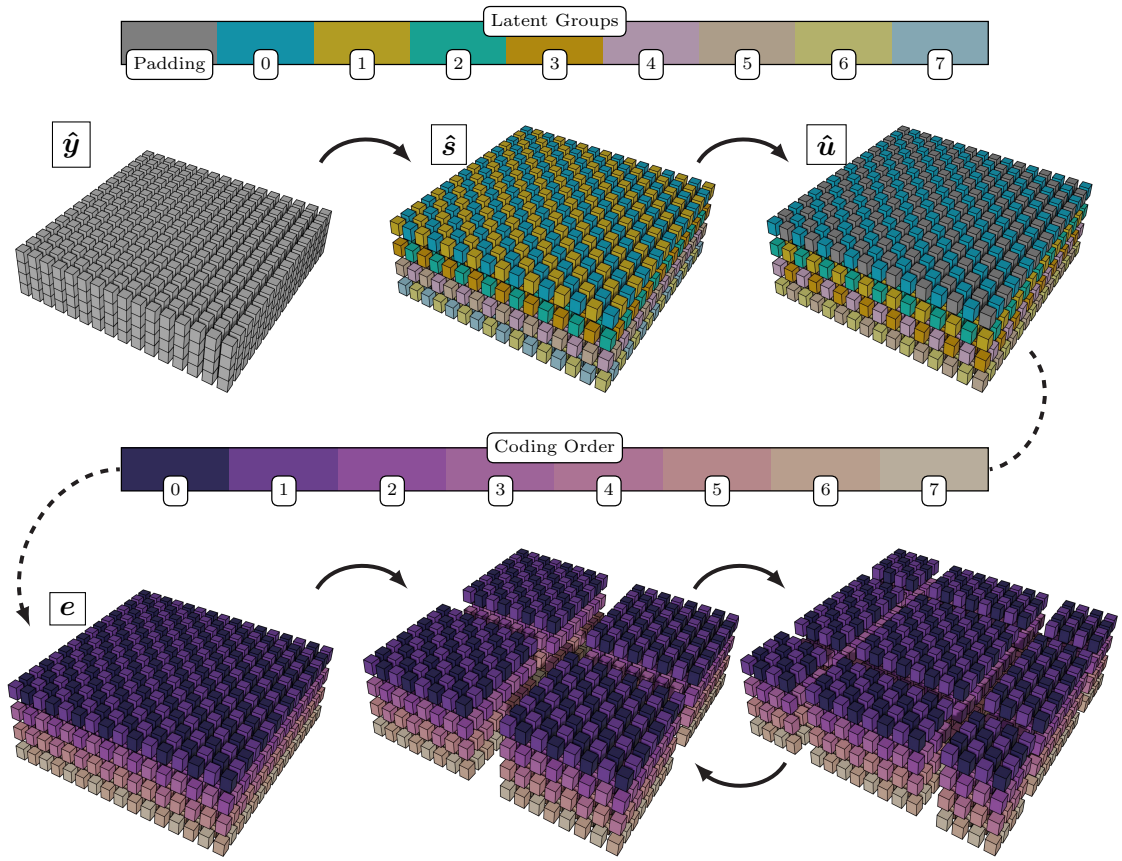
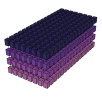


Figure 5.5: Illustration of (top row) the segment generation creating the Latent Groups, and (bottom row) the context modeling process in eContextformer, where latent elements are encoded based on the Coding Order. Following the arrows, the segment generator first splits the latent tensor $\hat{\boldsymbol{y}}$ into channel segments, generating $\hat{\boldsymbol{s}}$. It prepends zero-valued start token (shown in dark gray) while shifting the values of each group, obtaining $\hat{\boldsymbol{u}}$. Next, $\hat{\boldsymbol{u}}$ is mapped into embedding representation $\boldsymbol{e}$, which is fit to the transformer layers in eContextformer. In each coding step n, $\boldsymbol{e}_{0\ldots\lfloor\frac{n}{2}\rfloor}$ is processed by eContextformer, applying SC-W-MHA and SC-SW-MHA mechanisms alternately across its layers, in order to model the entropy of the $\hat{\boldsymbol{s}}_{\lfloor\frac{n}{2}\rfloor,i,j}$ with spatial coordinates, where $M(i,j)=0$ or $M(i,j)=1$ for even- and odd-numbered steps, respectively.

shifted-window multi-head attention mechanisms (SC-W-MHA and SC-SW-MHA) are alternately applied to $\boldsymbol{e}_{0\ldots\lfloor\frac{n}{2}\rfloor}$, as illustrated in Fig. 5.5. Prior to the window attention, each intermediate tensor is rearranged into $\mathbb{R}^{N_w\times(\lfloor\frac{n}{2}\rfloor K^2)\times d_e}$. Here, the first dimension, $N_w{=}\frac{HW}{256K^2}$, indicates the number of windows processed simultaneously, while the second dimension corresponds to the sequential data within each spatio-channel window with the size of $(\lfloor\frac{n}{2}\rfloor+1)\times K\times K$.

Moreover, SC-(S)W-MHA is provided with a checkered mask to ensure that each group integrates data from all previously coded groups without violating coding causality. For instance, for each even-numbered coding step, the elements with spatial coordinates (i,j) where $M(i,j){=}1$, and channel index $\lfloor\frac{n}{2}\rfloor$, are masked during attention computation.

In alignment with suggestions from [88], absolute positional encodings are substituted with relative ones to enhance permutation invariance and efficiency. Moreover, the imple-



mentation of MHA with h heads improves parallel processing capabilities and establishes distinct connections among the various segments within the spatio-channel framework.

5.4 Complexity Optimizations

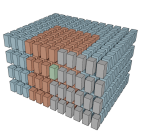
5.4.1 Architectural Optimizations

Using a start token is advantageous for the Contextformer, as it enables the model to utilize different values for each sliding-window position, incorporating elements from preceding window positions. However, in the eContextformer, the start token has a higher-dimensional representation compared to the original Contextformer, where the attention modules always compute the same value for the first coding group, regardless of the input content. To eliminate this inefficiency, the entropy modeling for the first coding group skips context modeling altogether, assuming it as zero, and instead relies solely on the hyperprior. This optimization, termed *Skipping First coding Group* (SFG), reduces the total number of autoregressive steps in the eContextformer to $2N_{cs} - 1$, which corresponds to a theoretical complexity reduction of 13% for N_{cs}=4.

Another inefficiency arises from how the eContextformer applies SC-(S)W-MHA mechanisms iteratively, as illustrated in Fig. 5.5. The generation of shifted and non-shifted windows requires that the input be in a *planar* form, while attention is calculated on sequential data. Thus, the transformations between $\mathbb{R}^{\lfloor \frac{n}{2} \rfloor \times \frac{H}{16} \times \frac{W}{16} \times d_e} \leftrightarrow \mathbb{R}^{N_w \times (\lfloor \frac{n}{2} \rfloor K^2) \times d_e}$ partition and recombine the intermediate outputs of the SwinT [88] layers before and after applying SC-(S)W-MHA. When the eContextformer is configured in the Spatial-first Order (SfO), its spatio-channel kernel expands iteratively over the channel segments, applying two autoregressive steps per segment to model the spatial context. The first step employs a checkered mask to block information from subsequent parts, while the second step processes the entire segment. This results in half of the computation in each even step—around 10% of the total computation—being unnecessary. To address this, the segment generation process is reformulated as $\hat{\boldsymbol{y}} \rightarrow \hat{\boldsymbol{r}} \in \mathbb{R}^{2N_{cs} \times \frac{H}{16} \times \frac{W}{32} \times p_{cs}}$, where all the elements processed during even-numbered steps are rearranged into the first dimension, with the first dimension dedicated to coding groups. To ensure equal receptive field sizes for both spatial dimensions, the window size is set to $K \times \frac{K}{2}$. This optimization is referred to as *Efficient Coding Group Rearrangement* (EGR). Fig. 5.6 illustrates the optimized application of SC-(S)W-MHA using SFG and EGR[†].

The integration of EGR reveals a unique characteristic inherent to the proposed SC-(S)W-MHA. In each autoregressive step n for coding $\hat{\boldsymbol{r}}_n$, the eContextformer leverages the previously coded spatio-channel groups $\hat{\boldsymbol{r}}_{<n}$ without requiring a coding mask. Essentially, SC-(S)W-MHA mimics the original shifted window mechanism of SwinT [88], applied to $\hat{\boldsymbol{r}}_n$. However, the receptive field of attention dynamically expands with each sequential coding step, encompassing both spatial and channel dimensions. Therefore, the combined implementation of EGR and SC-(S)W-MHA is also referred to as *expanding SC-(S)W-MHA*.

[†] The optimized processing of eContextformer is available as a flipbook animation in pp. 2–146, bottom left.



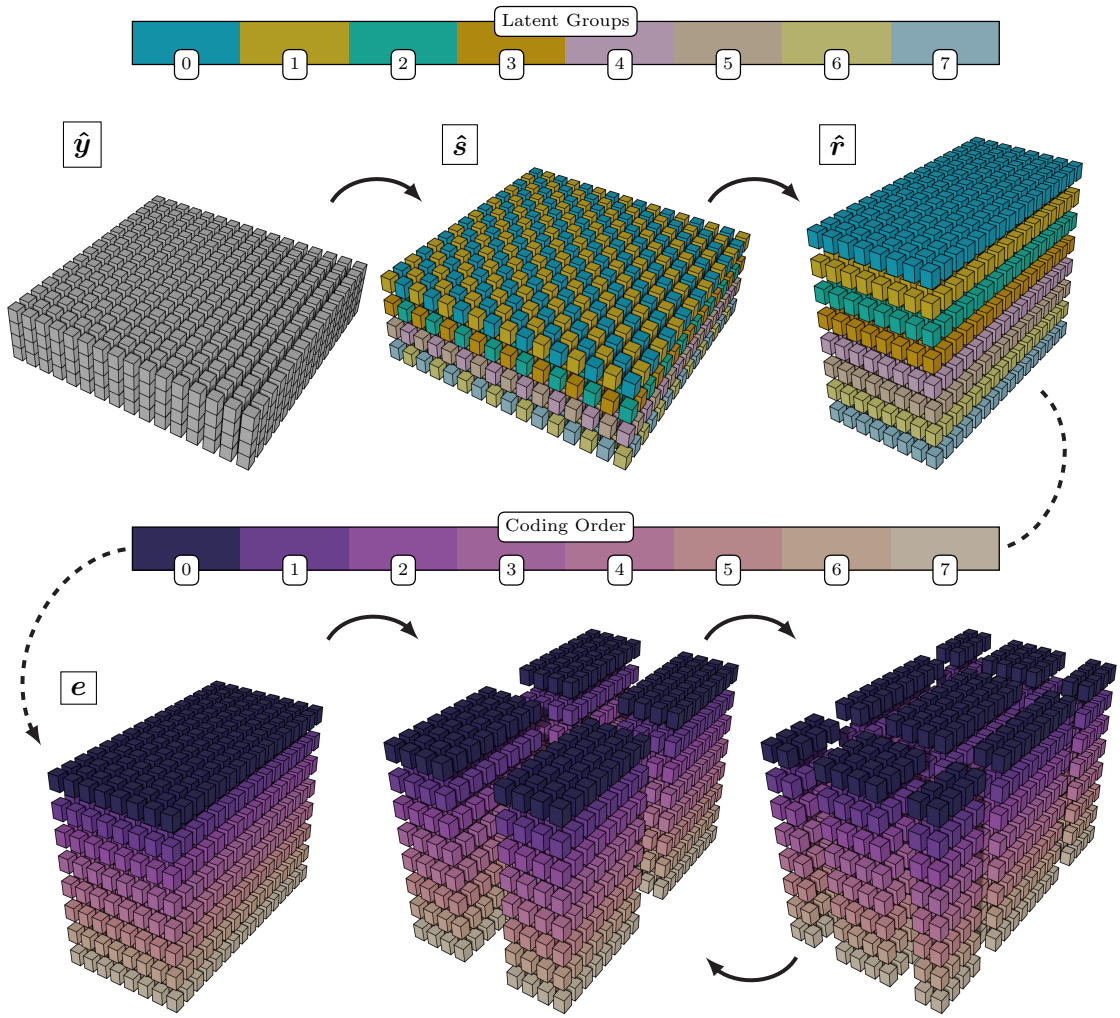
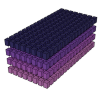


Figure 5.6: Illustration of (top row) the optimized segment generation creating the Latent Groups according to proposed EGR technique, and (bottom row) the context modeling process in eContextformer, where latent elements are encoded based on the Coding Order. Following the arrows, the segment generator first splits the latent tensor $\hat{\boldsymbol{y}}$ into channel segments and reorders them to $\hat{\boldsymbol{r}}$ using the proposed EGR technique. Next, $\hat{\boldsymbol{r}}$ is mapped into embedding representation $\boldsymbol{e}$, which is fit to the transformer layers in eContextformer. In each coding step n, $\boldsymbol{e}_{0\ldots n-1}$ is processed by eContextformer for entropy modeling of the $\hat{\boldsymbol{r}}_n$, by applying SC-W-MHA and SC-SW-MHA mechanisms alternately across eContextformer's layers. Notably, $\hat{\boldsymbol{r}}_0$ is coded only by using the hyperprior without the context modeling according to the proposed SFG technique.

5.4.2 Algorithmic Optimizations

Let $\mathrm{Attention}(\boldsymbol{Q}_{0\ldots n}, \boldsymbol{K}_{0\ldots n}, \boldsymbol{V}_{0\ldots n})$ represent the attention computed up to coding step n, where $\boldsymbol{Q}_{0\ldots n}$, $\boldsymbol{K}_{0\ldots n}$, and $\boldsymbol{V}_{0\ldots n}$ contain all the queries, keys, and values up to step n, respectively. Recalling Eq. (2.18), $\mathrm{Attention}(\boldsymbol{Q}_{0\ldots n}, \boldsymbol{K}_{0\ldots n}, \boldsymbol{V}_{0\ldots n})$ computes attention between every query and key-value pair in a single pass, simultaneously for all coding steps $0 \leq n$. This approach is beneficial for encoder-side operations.

However, during the decoding phase of the autoregressive process, only the attention between the current query and all preceding key-value pairs, $\mathrm{Attention}(\boldsymbol{Q}_n, \boldsymbol{K}_{0\ldots n}, \boldsymbol{V}_{0\ldots n})$,



is required for each step n. To improve efficiency on the decoder side, a more optimal implementation can be achieved using *key-value caching*, which stores the preceding key-value pairs in memory for reuse in upcoming coding steps. The key-value caching method is detailed in the white paper [193] and the published work [194]. This method balances the computational complexity of expanding SC-(S)W-MHA, reducing it from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

5.5 Online Rate-Distortion Optimization

The integration of the proposed optimization methods significantly reduces the encoder complexity of eContextformer, enabling the use of an online rate-distortion optimization (oRDO) technique (see Section 2.3.6) to further improve compression performance. For this purpose, an oRDO algorithm based on Campos *et al.* [53] has been adapted, replacing its original noisy quantization method with the straight-through estimator (STE), as described by Bengio *et al.* [121]. Algorithm 3 presents the pseudo-code for the modified oRDO method, which refines the quantized latent tensor elements $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ on the encoder side.

To simplify the oRDO process and achieve faster convergence, an exponential decay mechanism [195] is applied to the learning rate schedule. The learning rate at the i-th iteration is calculated as follows:

$$\alpha_i = \alpha_0 \gamma^i, \tag{5.4a}$$

$$N_{oRDO} = \lceil \log \left(\frac{2{\cdot}10^{-7}}{\alpha_0} \right) / \log(\gamma) \rceil, \tag{5.4b}$$

where γ represents the decay rate, α_0 is the initial learning rate, and N_{oRDO} is the number of oRDO steps. If the immediate learning rate becomes too low, performance improvements saturate, and therefore, the oRDO process halts when $\alpha_i \leq 2{\cdot}10^{-7}$. Throughout the iterations, the oRDO algorithm updates only the latent space tensor tuple $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$, storing the best-performing version, and returns it for encoding into the bitstream.

5.6 Comparison to the Prior Art Context Models

Table 5.3 summarizes the improvements introduced by the eContextformer over the Contextformer and pContextformer. Compared to the Contextformer, the most significant improvements are evident in the required number of autoregressive steps and attention window positions—the computation of both is decoupled in the eContextformer and pContextformer. For instance, given a 4K image with a resolution of $3840{\times}2160$, the Contextformer requires about 130,000 serial operations over the latent tensor, while the eContextformer and pContextformer require only 8 or even 7 steps when SFG is applied. Moreover, the overlapping window strategy of the pContextformer provides lower efficiency compared to the proposed SC-(S)W-MHA of the eContextformer due to two aspects. First, achieving higher RD performance in the pContextformer necessitates larger spatial kernels; in contrast, the eContextformer can achieve a similar "effective" kernel size by using window and shifted-windows alternately. Second, the overlapping window strategy overburdens the implementation of

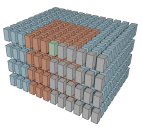


Table 5.3: Improvements of eContextformer over Contexformer and pContextformer.

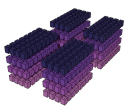
	Contextformer	pContextformer	eContextformer
Model architecture	ViT [85] inspired	ViT [85] inspired	SwinT [88] inspired
Attention mechanism	spatio-channel sliding window	spatio-channel overlapping window	expanding spatio-channel shifted window
Attention span			
spatial	16×16	16×16	8×8
channel-wise	4	4	4
Processing type	SCM	PCM by fusing [3], [35], [52]	PCM by fusing [3], [35], [52]
# autoregressive steps	$\frac{HW}{64}$	7	7
# window positions (N)	$\frac{HW}{64}$	$\frac{HW}{4096}$	$\frac{HW}{4096}$
Additional Optimizations			
Decoding runtime	3D-WPP (43% reduction)	EGR, SFG (30% reduction)	EGR, SFG, Caching (64% reduction)
Complexity	SCS (75% reduction)*	EGR, SFG (26% reduction)	EGR, SFG, Caching (84% reduction)
Supports oRDO	✗	✗	✓

* SCS reduces only the encoder side complexity since it can be implemented only on the encoder side due to its non-causal process. The other methods (EGR, SFG and Caching) reduce both complexity of both encoder and decoder side implementations.

key-value caching, demanding additional cache for storing multiple copies of the same key-value pairs at the overlapping parts.

Similar to the analysis of the Contextformer provided in Section 4.4, the eContextformer can also be parameterized to achieve an improved version of different state-of-the-art context models explained in Section 5.1, defining how dependencies in the latent space are handled. Notably, Figs. 5.1 and 5.6 provide a visual aid for the following comparisons:

- In case of $N_{cs} = 1$: The attention mechanism is limited to spatial dimensions, and the parallelization technique is simplified to the checkered grouping. Despite these limitations, the eContextformer requires even fewer autoregressive steps, resulting in a much lower model and runtime complexity compared to its variants with $N_{cs} > 1$. It can still



outperform certain prior models in terms of context modeling capability. For instance, compared to [3], [52], [64], the eContextformer has much larger support for spatial dependencies and can achieve better content adaptivity due to the attention mechanism. Unlike Koyuncu *et al.* [3], it can aggregate inter-window information without the need for modules like ISB [3]. Compared to [46], the eContextformer ($N_{cs} = 1$) provides a better complexity trade-off with respect to increasing image resolution—the complexity of SC-(S)W-MHA scales linearly with the resolution, whereas the complexity of Qian *et al.* [46] increases quadratically due to its global attention mechanism.

- In case of $C > N_{cs} > 1$: The attention mechanism can handle both spatial and cross-channel dependencies. The model's ability to leverage channel-wise information scales directly with the number of channel segments, though this scalability is inversely related to the model and runtime complexity. The operational characteristics of the eContextformer mirror state-of-the-art models that combine checkered and channel-wise grouping techniques, such as [56], [59], [63]. However, contrary to those approaches, the eContextformer achieves a more adaptive context model due to the spatio-channel attention, whereas He *et al.* [56] and Lu *et al.* [63] lack attention-based mechanisms for context modeling, and content-adaptive modeling in Liu *et al.* [59] is limited to spatial dimensions. Moreover, all of those models require additional neural network modules to aggregate information from previously coded channel segments, which drastically increases the model complexity. In contrast, the eContextformer's design allows the same model to be used for each autoregressive step, aggregating the information with the attention mechanism directly.

- In the extreme case $N_{cs} = C$: The eContextformer maximizes its use of spatio-channel attention by treating each channel as a coding group, resulting in $2C$ total autoregressive steps. This represents the most comprehensive use of the model's capabilities, with other versions acting as scaled-down adaptations to optimize the balance between efficiency and computational complexity. Under these conditions, the eContextformer acts theoretically as an advanced form of the 3D context models [40]–[42] with patch-wise [3] and checkered [52] grouping and an attention mechanism. Although this sophisticated version might boost the performance of the eContextformer, in practice, the resulting model might not be feasible for real-world applications due to its high computational demands.

5.7 Experiments

5.7.1 Experimental Setup

5.7.1.1 Training

The eContextformer is parameterized with the following default configuration:

$$\{C_N = 192,\ C_M = 192,\ L = 8,\ d_e = \frac{8C_M}{N_{cs}},\ d_{mlp} = 4d_e,\ h = 12,\ N_{cs} = 4\,,\ K = 8\,,\ \mathcal{S}_c = \text{SfO}\}. \quad (5.5)$$

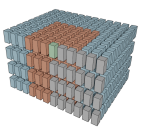


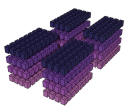
Table 5.4: The architecture of the proposed model using the eContextformer. Each row depicts one layer or component of the model. "Conv: $K_c \times K_c \times C$ s2" stands for a convolutional layer with kernel size of $K \times K$, number of C output channels, and stride "s" of 2. Similarly, "Deconv" is an abbreviation for transposed convolutions. The fully-connected "Dense" layers are specified by their output dimension, whereas $D_1 = 2C_M + d_e$, $D_2 = (4k_m C_M)/N_{cs}$. ResBlock and sRAB are the residual blocks and simplified residual attention mechanisms, shown in Fig. 4.2

Encoder (g_a)	Decoder (g_s)	Hyperencoder (h_a)	Hyperdecoder (h_s)
Conv: $3\times3\times C_N$ s2	2×ResBlock: $3\times3\times C_M$	Conv: $5\times5\times C_N$ s1	Conv: $5\times5\times\times C_M$ s1
GDN [120]	Deconv: $3\times3\times C_N$ s2	Leaky ReLU [132]	Deconv: $5\times5\times C_M$ s2
sRAB [39], [47], [133]	IGDN [120]	Conv: $5\times5\times C_N$ s1	Leaky ReLU [132]
Conv: $3\times3\times C_N$ s2	Deconv: $3\times3\times C_N$ s2	Conv: $5\times5\times C_N$ s2	Conv: $5\times5\times C_M$ s1
GDN [120]	IGDN [120]	Leaky ReLU [132]	Deconv: $5\times5\times\frac{3}{2}M$ s2
Conv: $3\times3\times C_N$ s2	Deconv: $3\times3\times C_N$ s2	Conv: $5\times5\times C_N$ s1	Leaky ReLU [132]
GDN [120]	sRAB [39], [47], [133]	Conv: $5\times5\times C_N$ s2	Deconv: $5\times5\times2M$ s2
Conv: $3\times3\times C_N$ s2	IGDN [120]		Leaky ReLU [132]
Conv: $1\times1\times C_M$ s2	Deconv: $3\times3\times3$ s2		

Context Model (g_c)	Entropy Parameters (g_e)
eContextformer:	Dense: $(2D_1+D_2)/3$
$\{L, d_e, d_{mlp}, h, N_{cs}, K, \mathcal{S}_c\}$	GELU [129]
	Dense: $(D_1+2D_2)/3$
	GELU [129]
	Dense: D_2

Most of these parameters are shared with the Contextformer (see Eq. (4.8)), ensuring a fair comparison of the parallelization efforts. The eContextformer utilizes only the SfO coding method, as the combination of CfO with parallelization was found to deteriorate performance, as observed in Section 5.2. The spatial kernel size of expanding SC-(S)W-MHA is defined as $K \times \frac{K}{2}$, as explained in Section 5.4.1. A detailed description of the compression framework using the eContextformer is provided in Table 5.4. Notably, the hyperprior employs 5×5 convolutions, in contrast to the 3×3 ones used in Contextformer, to better align with other state-of-the-art parallel architectures [35], [56]. Table 5.4 provides a detailed description of the compression framework utilizing the eContextformer.

Similar to the Contextformer (see Section 4.5.1.1), the eContextformer was trained for 120 epochs (around 1.2 million iterations) on random 256×256 image crops from the Vimeo-90K [144] dataset. A batch size of 16 was used, and the ADAM optimizer [190] was employed with an initial learning rate of 10^{-4}. The models were trained across different bitrates by setting $\lambda_{MSE} \in \{0.002, 0.004, 0.007, 0,014, 0.026, 0.034, 0.058\}$, using MSE as the



distortion metric $\mathbf{D}(\cdot)$. To assess perceptual quality using MS-SSIM [160], the models were fine-tuned for approximately 500,000 iterations, using the distortion function $1-$MS-SSIM and $\lambda_{\text{MS-SSIM}}=1300\lambda_{\text{MSE}}$. For models targeting higher bitrates (λ_5, λ_6, and λ_7 for both MSE and MS-SSIM [160]-optimized models), the bottleneck size C_M was increased to 312, following common practice [34], [35], [47]. It was empirically determined that using fewer heads ($h = 6$) yielded better results for the highest bitrate model (λ_7).

To explore the effects of training dataset and crop size (see Section 5.2 and Section 5.7.2.2), the models were also fine-tuned using image crops of 256×256 and 384×384 from the COCO [146] dataset.

5.7.1.2 Evaluation

The empirical evaluation of the proposed compression framework using eContextformer was conducted on the Kodak [138], Tecnick [139], and CLIC-P/-M [140] datasets, with comprehensive dataset details provided in Section 3.1.

Comparative analyses were performed to benchmark the eContextformer's performance against a wide range of state-of-the-art LIC frameworks with serial and parallel context models, including traditional 2D and 3D models as well as advanced transformer-based models, summarized Table 5.5. The models from the previous chapter described in Table 4.2 are also included to the comparison. The SCM and PCM variants from Qian *et al.* [46] were tested, along with the *small* (S) and *large* (L) models of Liu *et al.* [59], which differ in their number of channels in their transforms. The evaluation also covered low-complexity asymmetric frameworks developed by Wang *et al.* [60], Fu *et al.* [58], and Yang&Mandt [57]. Specifically, the model (LL) from Wang *et al.* [60] was selected due to its superior RD performance, which is characterized by its *large* synthesis and *large* analysis transforms.

Due to the proposed optimizations in Section 5.4, eContextformer achieves significantly reduced complexity, enabling iterative oRDO explained in Chapter 2. Additional experiments using oRDO with eContextformer were conducted and are detailed in Section 5.7.2.4. For comparison, results from Yang and Mandt [57]'s framework, which integrates Stochastic Gumbel Annealing (SGA) [54] to perform oRDO for 3000 steps, were also included. Traditional image coding standards such as BPG [91] and VTM 16.2 [178] served as benchmarks in this analysis. The methodologies used to evaluate all state-of-the-art approaches are described in detail in Section 3.3.

The number of model parameters was determined using summary functions from either PyTorch [171] or Tensorflow [172], depending on the implementation details of each framework. Following the guidelines from the JPEG AI standardization effort [79], model complexity was measured in kilo MACpp (kMACpp) using the ptflops package and reverified through DeepSpeed [173]. For models without explicit hooks for precise complexity measurement of the attention mechanism, these were implemented based on the official SwinT [88] repository*.

Runtime measurements were also performed using DeepSpeed [173], deliberately excluding the time spent on arithmetic coding to ensure a consistent basis for comparison

* https://github.com/microsoft/Swin-Transformer

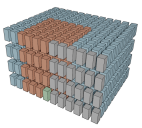
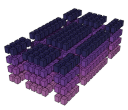
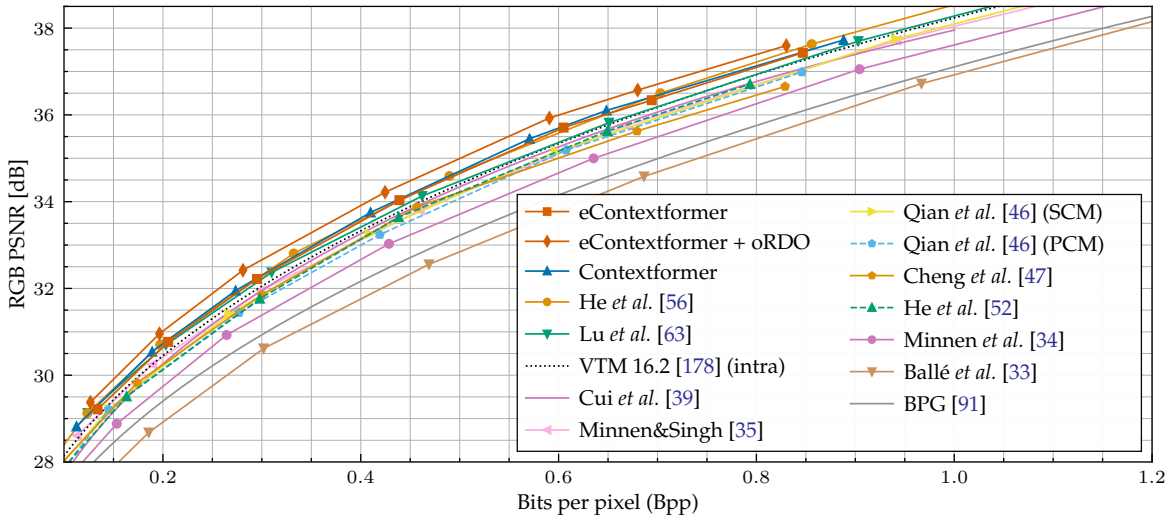


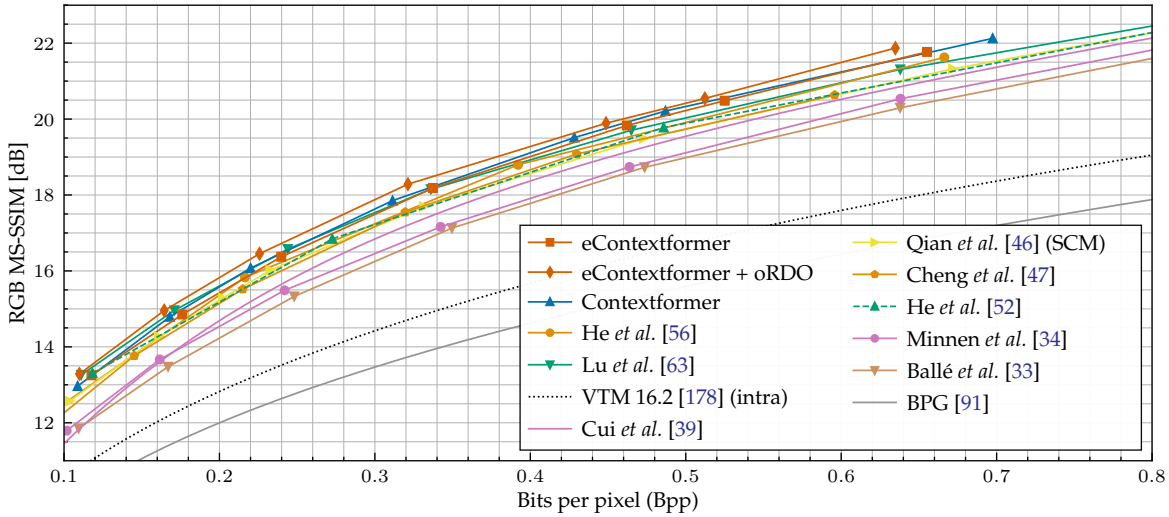
Table 5.5: Key properties of the state-of-the-art LIC frameworks compared to the eContextformer.

	Model Details	
Model	Encoder (g_a) and Decoder (g_s)	Entropy Model
He *et al.* [52]	Uses the same framework as Cheng *et al.* [47] (see Table 4.2).	The entropy model has the same architecture as Cheng *et al.* [47], but the SCM using 2D masked convolutions is extended with checkered grouping to achieve a PCM.
Qian *et al.* [46] (PCM)	Uses the same framework as Qian *et al.* [46] (SCM) (see Table 4.2).	The entropy model has the same architecture as Cheng *et al.* [47], but the transformer-based SCM with spatial attention is extended with checkered grouping to achieve PCM.
He *et al.* [56]	A similar framework to Minnen&Singh [35] (see Table 4.2) is used, but every activation function is replaced by stacked CNN layers with residual connections and ReLU activation, which are slightly more complex than ResBlocks. It contains four attention modules similar to sRAB and has higher channel capacity ($C_N = 192$ and $C_M = 320$).	A similar hyperprior network as Cheng *et al.* [47] is used (see Table 4.2). The entropy model has a similar architecture to Minnen&Singh [35] but achieves PCM by extending ChARM with 2D masked convolutions and checkered grouping. The model uses 5 channel groups instead of 10. Moreover, a Gaussian distribution is used for probability modeling of the residual $p(\hat{\boldsymbol{r}} \mid \hat{\boldsymbol{z}})$, where $\hat{\boldsymbol{r}} = Q(\boldsymbol{y} - \boldsymbol{\mu}) + \boldsymbol{\mu}$, achieving similar performance as GMM.
Liu *et al.* [59] (L)	Uses a similar framework as Cheng *et al.* [47] (see Table 4.2), but all activation functions are replaced by multiple layers combining CNN and Swin Transformer (SwinT) layers with residual connections. The model has much higher channel capacity ($C_N = 256$ and $C_M = 320$).	The entropy model has a similar architecture to Minnen&Singh [35], where all CNNs are replaced by multiple layers combining convolutional and SwinT layers, applying attention to the spatial dimensions. Similar to He *et al.* [56], a Gaussian distribution is used for probability modeling of the residual.
Liu *et al.* [59] (S)	Uses the same framework as Liu *et al.* [59] (L) but with lower channel capacity ($C_N = 128$).	The entropy model is the same as in Liu *et al.* [59] (L), but the number of some neural network layers is reduced.
Lu *et al.* [63]	Uses a similar framework as Minnen *et al.* [34] (see Table 4.2), but all activation functions are replaced with ViT-based transformer layers with residual connections, and the model has higher channel capacity ($C_M = 320$).	The entropy model has the same architecture as He *et al.* [56], where the activation functions are replaced with ViT-based transformer layers similar to its analysis and synthesis transforms. Moreover, it has the ChARM architecture combined with checkered grouping, where the 2-stage checkered grouping of He *et al.* [52] is extended to 4-stage grouping for the spatial dimensions. The Gaussian distribution is used for probability modeling of the residual $p(\hat{\boldsymbol{r}} \mid \hat{\boldsymbol{z}})$, similar to He *et al.* [56].
Wang *et al.* [60] (LL)	Uses a similar framework as Cheng *et al.* [47] (see Table 4.2), but it does not contain attention layers, and CNN layers without stride are replaced with depth-wise convolutions for lower complexity. Several versions of the model are available with varying complexity; the Large Encoder and Large Decoder (LL) model is chosen due to its higher performance, which has channel capacities of $C_N = 192$ and $C_M = 192$.	Uses a similar hyperprior as Minnen *et al.* [34], where all convolutional layers are replaced with depth-wise convolutions to achieve a low-complexity model. The entropy model contains a PCM similar to He *et al.* [52]. The checkered grouping is applied in both spatial and channel dimensions, while the channel groups with different checkered patterns are merged into a single group to keep the total modeling steps as two.
Fu *et al.* [58]	Uses a similar framework as Cheng *et al.* [47] (see Table 4.2), where the residual convolutional layers are replaced with residual modules containing multiple branches and grouped convolutions to reduce complexity. Moreover, fewer residual modules are used on the decoder side to achieve an asymmetrical compression framework.	The entropy model is similar to the one in the Contextformer (see Table 4.1), but the model has an SCM with 2D masked convolutions from Minnen *et al.* [34].
Tang *et al.* [43]	The architecture of the framework is based on Liu *et al.* [40] but introduces asymmetry to each layer by spatially factorizing $K \times K$ convolutional kernels into $1 \times K$ and $K \times 1$ asymmetrical CNNs. Additionally, attention layers are replaced by graph attention networks with reduced complexity.	The hyperprior implements a similar architecture to the analysis and synthesis transforms with graph attention networks and residual layers. The context model uses the same SCM with 3D masked convolutions as in Chen *et al.* [48].
Yang&Mandt [57]	The architecture of the framework is highly asymmetrical, where the analysis transforms are the same as in He *et al.* [56], and the synthesis transform has only CNNs with 2 layers.	The entropy model is similar to the one in Minnen *et al.* [34] but without any context model.





(a)



(b)

Figure 5.7: RD performance on the Kodak [138] dataset, comparing various compression frameworks to eContextformer trained on Vimeo-90K [144] in terms of: (a) PSNR, and (b) MS-SSIM [160].

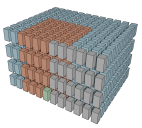
across different codec implementations. All evaluations, including runtime tests, were carried out on a system equipped with an NVIDIA Titan RTX GPU and an Intel Core i9-10980XE CPU, using PyTorch 1.10.2 [171] and CUDA Toolkit 11.4 [170].

More details on the experimental setup and how results of the state-of-the-art approaches were obtained are described in Section 3.3.

5.7.2 Experimental Results

5.7.2.1 Model Performance

Fig. 5.7a presents the RD performance of the eContextformer on the Kodak [138] dataset, when trained using 256×256 image crops from the Vimeo90K [144] dataset. In terms of



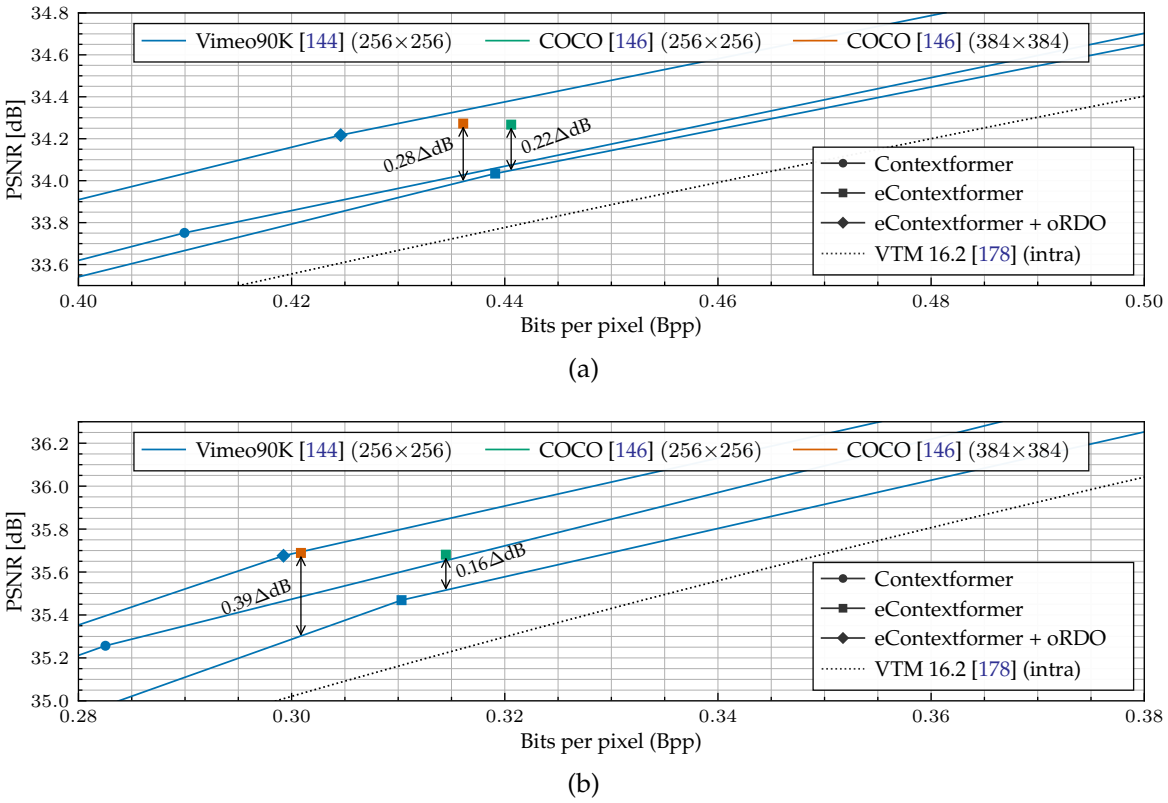
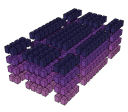


Figure 5.8: Experimental study of the effects of different training datasets and crop sizes on the performance, showing the RD performance on (a) Kodak [138] and (b) Tecnick [139] datasets.

PSNR, this model consistently outperforms VTM 16.2 [178] across all evaluated bitrates, achieving a 5.3% reduction in bitrate, as detailed in Table 5.7. Compared to Cui *et al.* [39] and the Contextformer, the key differences in the proposed LIC framework lie in the context modeling and a slightly larger hyperprior. Relative to the implementation by Cui *et al.* [39], the eContextformer achieves an additional 8.5% bitrate reduction. However, it exhibits 1.6% lower performance compared to the Contextformer. This performance gap, introduced by the parallelization of the Contextformer, is consistent with observations from other parallel context modeling studies [3], [52], [56] (see Section 5.1).

The eContextformer also competes closely with models that utilize the ChARM architecture, which typically involve a higher number of parameters, such as those developed by Minnen&Singh[35] and He *et al.* [56]. Specifically, compared to VTM 16.2 [178], Minnen and Singh [35] demonstrates a 1.6% lower BD-Rate [169] performance, whereas the model from He *et al.* [56] achieves a 6.7% gain.

The perceptually optimized models display similar performance characteristics, as assessed in Fig. 5.7b. In terms of the MS-SSIM metric [160], the eContextformer achieves an average bitrate saving of 48.3% compared to VTM 16.2 [178], performing 0.8% better than He *et al.* [56] but 1.1% worse than the Contextformer. This comparative evaluation highlights the trade-offs between computational complexity and perceptual quality inherent in the design of advanced image compression algorithms.



5.7.2.2 Optimization of the Training Setup

Figs. 5.8a and 5.8b show the comparative performance of the eContextformer on the Kodak [138] and Tecnick [139] datasets when trained with 256×256 image crops from the Vimeo90K [144] dataset. The eContextformer closely matches the performance of the Contextformer on the Kodak [138] dataset but exhibits lower performance on higher-resolution test datasets such as Tecnick [139], which might be degraded due to several issues.

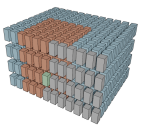
The first issue is that the initial 256×256 image crops result in a 16×16 latent space during training. The combination of a $K=8$ attention window size with the relatively low resolution of latent variables restricts the model's capacity for learning efficient context modeling. For instance, SC-W-MHA layers learn a 2×2 grid of patches in the latent space, whereas SC-SW-MHA layers can explore only a single patch for the central area of the latent space. Therefore, a larger latent space size is required during training to improve learning efficiency. However, this is not possible with the Vimeo90K [144] dataset, which has an original resolution of 448×256—training with the original image size might result in a bias towards horizontal image orientations.

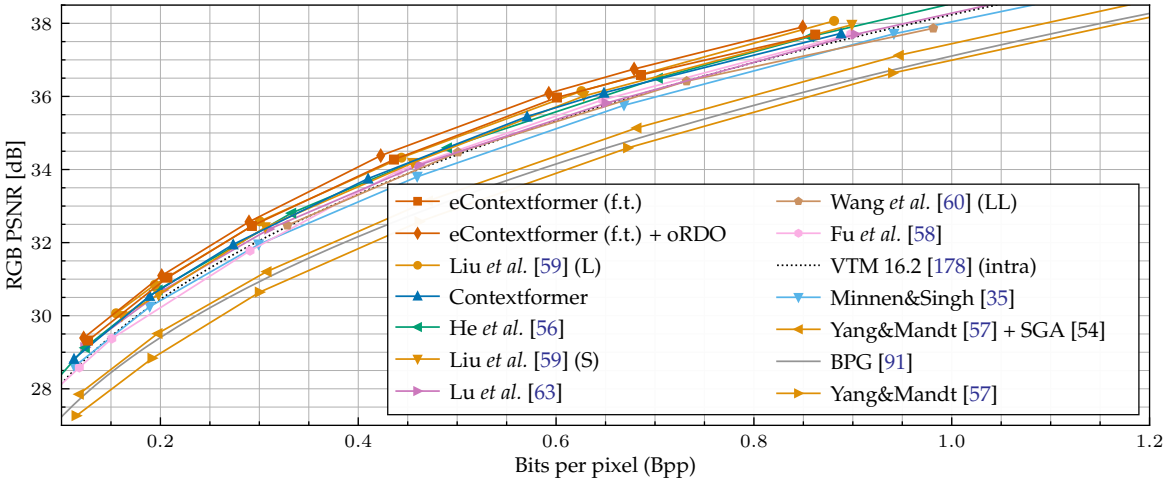
Another issue is that the dataset complexity assessments from Section 3.1.3 show that the Vimeo90K [144] dataset contains images with limited variety in terms of content complexity compared to all test datasets, especially to Tecnick [139] and CLIC-P/-M [140]. In contrast, the COCO [146] dataset displays a closer match to those test datasets according to multiple complexity metrics. Additionally, COCO [146] contains images with a large variety of resolutions, allowing training models with larger crop sizes.

Similar issues have been observed in other studies [3], [35], [46], [56], [58]. To address these challenges, these studies experimented with training on higher-resolution datasets, such as ImageNet [196], COCO [146], and DIV2K [197], and utilized larger crop sizes, up to 512×512, which resulted in improved RD performance.

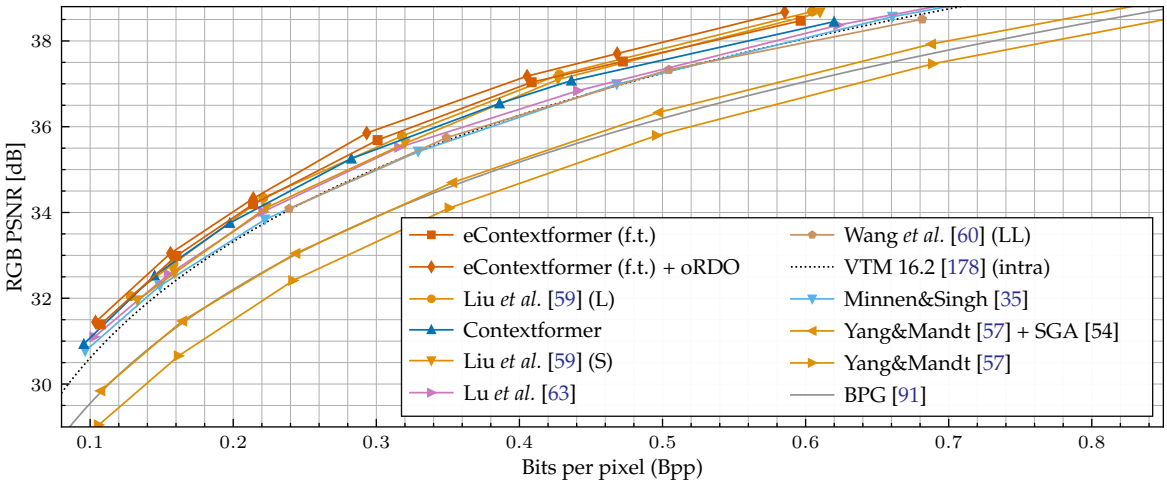
Following the prior-art studies and the analysis in Section 3.1.3, the eContextformer was fine-tuned using 256×256 and 384×384 image crops from the COCO [146] dataset to explore both the effect of training with a more complex dataset and larger crop sizes. As illustrated in Figs. 5.8a and 5.8b, both adjustments provide significant improvements in performance on both test datasets, as reflected in the complexity assessments in Section 3.1.3. Training with larger crops provides additional performance gain—the benefit is more pronounced on the Tecnick [139] dataset—suggesting that inefficient initial training of SC-(S)W-MHA mechanism. Based on these observations, the eContextformer fine-tuned with 384×384 image crops from COCO [146] is used for further analysis.

Fig. 5.9 demonstrates the RD performance of the fine-tuned (f.t.) eContextformer compared to state-of-the-art compression frameworks on the Kodak [138], Tecnick [139], and CLIC-P [140]. Table 5.7 summarizes the BD-Rate [169] performance of each model compared to VTM 16.2 [178]. The eContextformer (f.t.) matches the performance of leading LIC frameworks, showing more than 5% bitrate savings from the initial configuration and achieving average bitrate savings of 10.9%, 13.2%, 12.3%, and 6.9% over VTM 16.2 [178] across those datasets. Notably, the eContextformer (f.t.) surpasses Liu *et al.* [59] with a 1% bitrate saving on higher-resolution datasets, such as Tecnick [139].

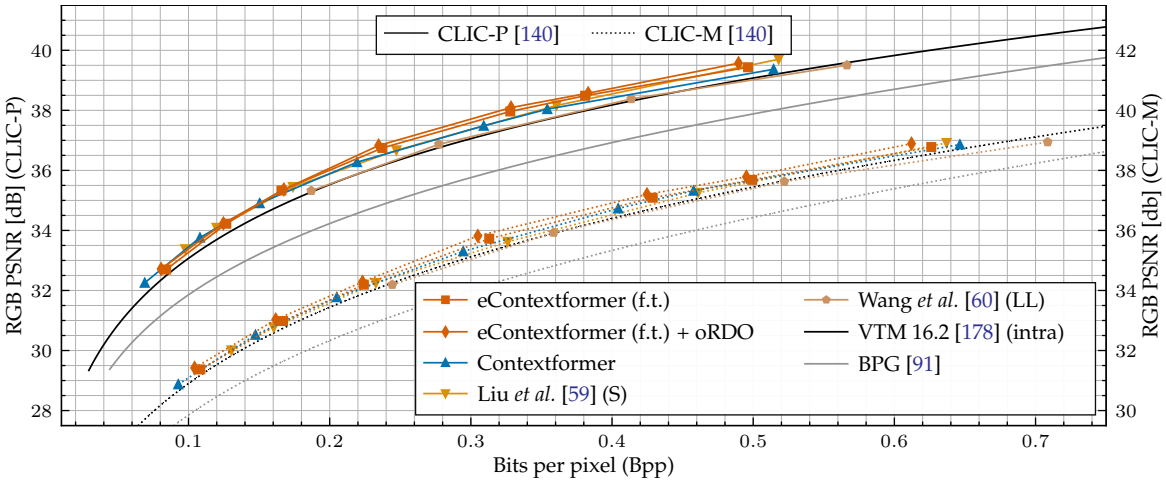




(a)

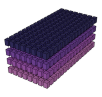


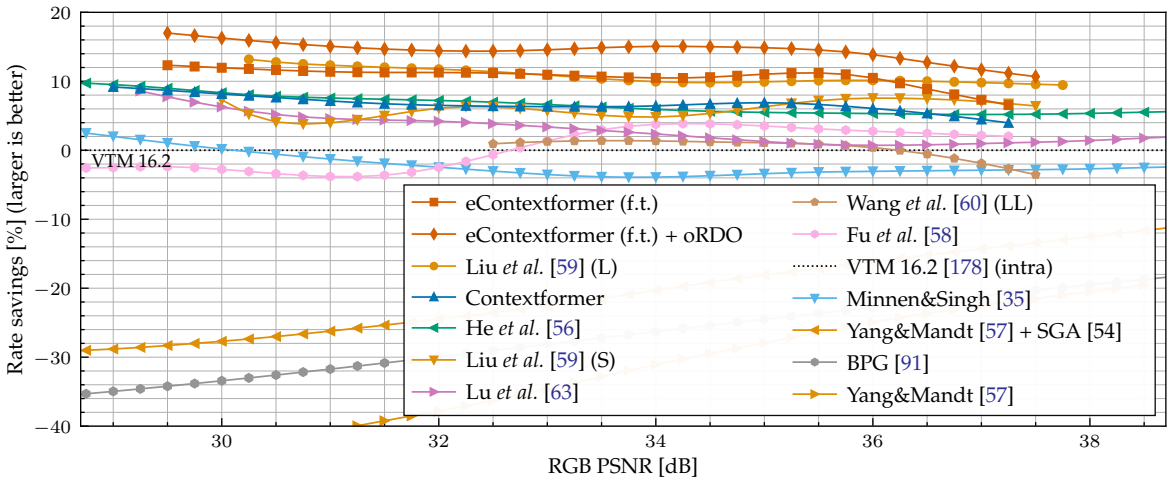
(b)



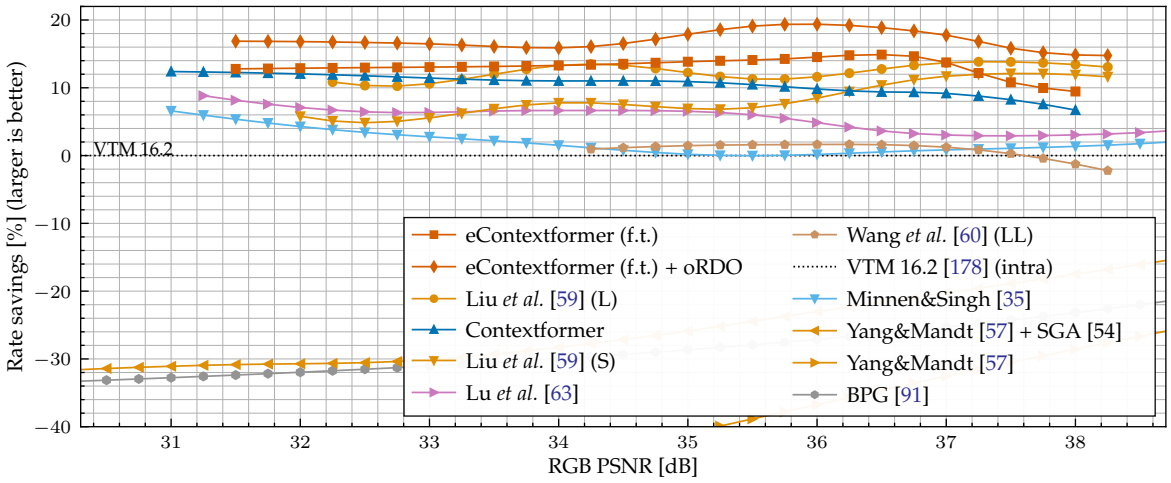
(c)

Figure 5.9: RD performance in terms of PSNR, comparing various frameworks to eContextformer finetuned with COCO [146] dataset across: (a) Kodak [138], (b) Tecnick [139], and (c) CLIC-P/-M [140] datasets.

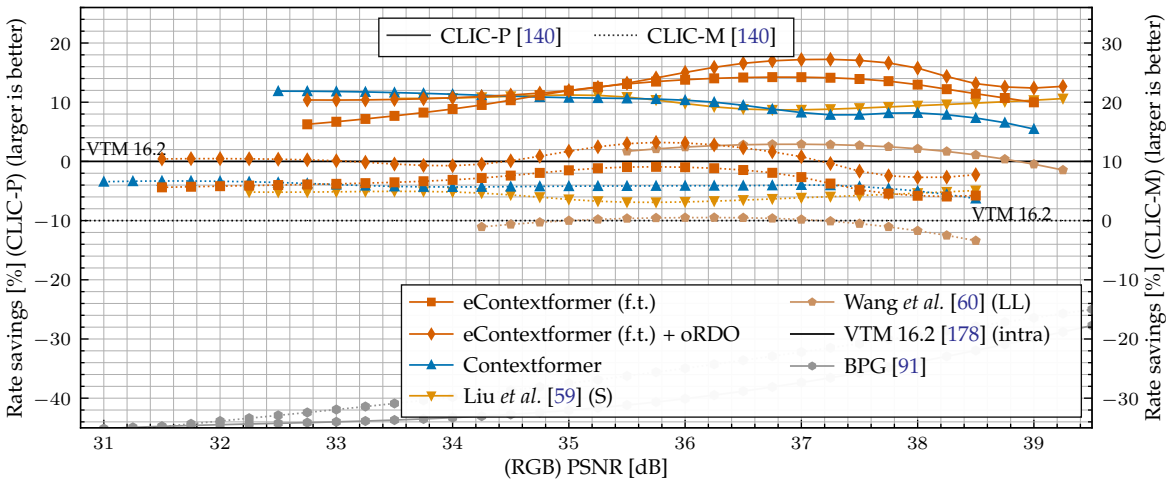




(a)



(b)



(c)

Figure 5.10: Rate savings relative to VTM 16.2 [178] as a function of PSNR, comparing various frameworks to eContextformer finetuned with COCO [146] dataset across: (a) Kodak [138], (b) Tecnick [139], and (c) CLIC-P/-M [140] datasets.

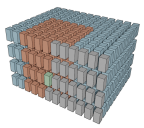


Table 5.7: BD-Rate [169] performance of eContextformer compared to various compression frameworks accross Kodak [138], Tecnick [139], and CLIC-P/-M [140] datasets.

	BD-Rate [%] over VTM 16.2 [178] (intra)			
Method	Kodak [138]	Tecnick [139]	CLIC-P [140]	CLIC-M [140]
eContextformer (f.t.)	−10.9	−13.2	−12.3	− 6.9
+ oRDO	−14.7	−17.0	−14.1	−10.4
Contextformer	− 6.9	−10.6	− 8.8	− 5.8
Liu *et al.* [59] (L)	−11.0	−12.2	–	
He *et al.* [56]	− 6.7	–	–	
Liu *et al.* [59] (S)	− 6.2	− 8.0	− 9.8	− 4.1
Guo *et al.* [45]	− 3.7	–	–	
Lu *et al.* [63]	− 3.6	− 5.9	–	
Wang *et al.* [60] (LL)	− 0.6	− 0.7	− 1.6	0.4
Fu *et al.* [58]	− 0.3	–	–	
Lu *et al.* [62]	0.4	–	–	
Minnen&Singh [35]	1.9	− 2.2	–	
Cui *et al.* [39]	3.2	–	–	
Qian *et al.* [46] (SCM)	3.4	–*	–*	
Cheng *et al.* [47]	4.2	4.8	5.9	9.1
He *et al.* [52]	4.5	1.5	–	
Qian *et al.* [46] (PCM)	5.6	–*	–*	
Tang *et al.* [43]	5.7	3.5	–	
Qian *et al.* [44]	7.9	–	–	
Kim *et al.* [61]	8.5	–	–	
Chen *et al.* [48]	10.6	–	–	
Minnen *et al.* [34]	14.7	15.4	11.6	13.9
Yang&Mandt [57]	36.5	44.7	–	
+ SGA [54]	22.4	26.7	–	
BPG [91]	30.3	29.5	36.2	27.8
Ballé *et al.* [33]	38.7	34.2	38.3	40.7
VTM 16.2 [178] (intra)	0.0	0.0	0.0	0.0

* Can not be computed due to OOM

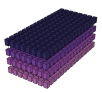


Table 5.8: Ablation study of the proposed optimization methods for eContextformer with respect to the effects on the context model complexity.

EGR	SFG	Single Pass	Caching	kMACpp
✗	✗	✗	✗	1200
✓	✗	✗	✗	1075
✓	✓	✗	✗	829
✓	✓	✓	✗	212
✓	✓	✗	✓	204

Fig. 5.10 highlights the eContextformer's performance on the generalized BD-Rate [35] metric, as described in Chapter 3. On the Kodak [138] dataset, the eContextformer demonstrates 6.5% bit savings at the highest quality point, while achieving up to 12.3% bitrate savings at the lowest quality point compared to VTM 16.2 [178]. Even larger performance gains on the generalized BD-Rate metric are observed on the other datasets, underscoring the model's generalization capability across different image types. The eContextformer provides up to 14.9%, 14.2%, and 9.1% relative bitrate savings over VTM 16.2 on the Tecnick [139], CLIC-P [140], and CLIC-M [140] datasets, respectively.

5.7.2.3 Model and Runtime Complexity

Table 5.8 highlights the impact of the proposed complexity optimizations for the eContextformer, as discussed in Section 5.4. The implementation of both EGR and SFG methodologies simplifies the computation of context modeling, with each method reducing the complexity by approximately 10-13% during encoding and decoding. Furthermore, when combined with key-value caching, the model achieves an overall complexity reduction of 84%. Caching is particularly more efficient than the traditional single-pass approach, where the entire latent variable $\hat{\boldsymbol{y}}$ is processed simultaneously because it selectively computes only the necessary attention map components.

Table 5.9 offers a comparative analysis of the eContextformer against existing LIC frameworks, focusing on the number of parameters and the computational demands of the entropy model, including components such as g_c, h_a, h_s, and g_{ep}. The eContextformer's SC-(S)W-MHA exhibits lower computational requirements than both the sliding window attention of the Contextformer and the global attention used in [46]. With the optimizations enabled, the eContextformer demonstrates a complexity that is 145x lower than that of the Contextformer for an equivalent parameter count. The expanding SC-(S)W-MHA efficiently gathers channel segment information without requiring additional NN modules, resulting in a reduced and less complex entropy parameter network relative to [39], and significantly fewer parameters in the entropy model compared to ChARM-based approaches [35], [56], [59].

Table 5.10 outlines the encoding and decoding runtime complexities for the eContextformer, comparing it against various prior LIC frameworks and VTM 16.2 [178]. The op-

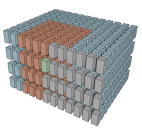


Table 5.9: Number of parameters of different modules and the entropy model complexity of various LIC frameworks during encoding compared to proposed model with eContextformer

	Number of Parameters $[\times 10^6]$		Complexity [kMACpp]*
Method	Analysis&Synthesis (g_a+g_s)	Entropy Model (h_a+h_s+g_c+g_{ep})	Entropy Model (h_a+h_s+g_c+g_{ep})
eContextformer (opt.)**	17.5	26.9	253
Contextformer (opt.)†	17.5	20.1	$30 \cdot 10^3$
Liu et al. [59] (L)‡	28.5	47.4	125
Liu et al. [59] (S)	8.8	36.2	116
He et al. [56]	14.7	44.8	266
Cui et al. [39]	17.5	21.8	56
Qian et al. [46] ‡	7.6	29.0	868
Minnen&Singh [35]	8.4	113.5	426
Yang&Mandt [57]	8.6	15.2	22
Wang et al [60] (LL)	6.6	10.8	37
Minnen et al. [34]	5.6	8.3	24

* The complexity is measured on 4K images with 3840x2160 resolution.
** The combination of the proposed EGR, SFG and Caching optimizations are applied.
† The combination of the proposed BDS and SCS are applied.
‡ The complexity is approximated by using images with 1024x1024 resolution due to OOM.

timizations integrated into the eContextformer have enhanced encoding and decoding efficiencies by a factor of three, yielding a 210-fold improvement in performance over the optimized Contextformer configuration (described in Section 4.3). Additionally, the runtime scalability of the eContextformer proves superior to its non-optimized counterpart. For instance, although 4K images contain 21x the pixel count of Kodak images [138], the relative increase in encoding and decoding runtime for the optimized model on these high-resolution images is only 14-fold compared to the Kodak dataset. Furthermore, the optimized eContextformer demonstrates competitive runtime performance relative to Minnen and Singh [35] and both the small and large configurations of Liu *et al.* [59] (S/L).

There were issues with the memory usage of some of the models. For example, the global attention mechanism in Qian *et al.* [46] significantly increases memory consumption with increasing image resolution. Similarly, the models of Liu *et al.* [59], which employ residual transformer-based layers in their synthesis and analysis transforms and incorporate ChARM-based context models, also impose considerable memory demands. As a result, testing the large configuration model Liu *et al.* [59] (L) and the models with SCM and PCM in Qian *et al.* [46] on 4K images is not feasible using a GPU with 24 GB of memory.

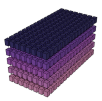


Table 5.10: Encoding and decoding time of different compression frameworks compared to the framework with eContextformer using proposed optimization methods.

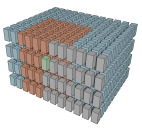
	Encoding Time [s]		Decoding Time [s]	
Method	Kodak	4K*	Kodak	4K*
eContextformer (opt.)**	0.16	2.24	0.17	2.41
eContextformer	0.33	6.55	0.35	6.75
Contextformer	56	1240	62	1440
Contextformer (opt.)†	8.72	120	44	820
Yang&Mandt [57]	0.09	0.81	0.02	0.43
Wang et al. [60] (LL)	0.04	0.56	0.04	0.57
Minnen&Singh [35]	0.16	1.99	0.12	1.37
Liu et al. [59] (S)	0.18	1.74	0.17	1.47
Liu et al. [59] (L)	0.25	–‡	0.23	–‡
Qian et al. [46] (PCM)	1.57	–‡	2.59	–‡
Cheng et al. [47]	2.22	54	5.81	140
Chen et al. [40]	4.11	28	316	7486
VTM 16.2 [178] (intra)	420	950	0.84	2.53

* The time is measured on 4K images with 3840x2160 resolution.
** The combination of the proposed EGR, SFG and Caching optimizations are applied. Less than 0.1% change in BD-Rate performance is observed due to the complexity optimizations.
† Complexity optimization techniques proposed in Section 4.3 are applied.
‡ Can not be computed due to OOM.

5.7.2.4 Online Rate-Distortion Optimization

Different combinations of hyperparameters (α_0, γ) in Eq. (5.4) result in varied learning behaviors, affecting the performance of the oRDO process. Fig. 5.11a shows the immediate learning rate α_i during the oRDO process for different combinations of (α_0, γ). The optimal hyperparameters for (α_0, γ) were explored through a search using the Tree-Structured Parzen Estimator (TPE) [198]. This search process uses Algorithm 3, guided by the same objective function employed during training (see Eq. (2.23)), with constraints $\alpha_0 \in [0.02, 0.08]$ and $\gamma \in [0.5, 0.7]$. To mitigate overfitting, the parameter search was conducted using a set of 20 randomly selected full-resolution images from the COCO [146] dataset.

Figs. 5.11b and 5.11c display the findings from the TPE for models trained for λ_1 and λ_4. It was observed that models designed for higher bitrates ($\lambda_{>3}$) tend to perform slightly better with a higher initial learning rate compared to those targeting lower bitrates ($\lambda_{<4}$). This indicates that optimizing denser latent spaces benefits from larger initial learning rate increments. Consequently, the configuration (α_0, γ)=$(0.032, 0.62)$ is selected for low bitrate



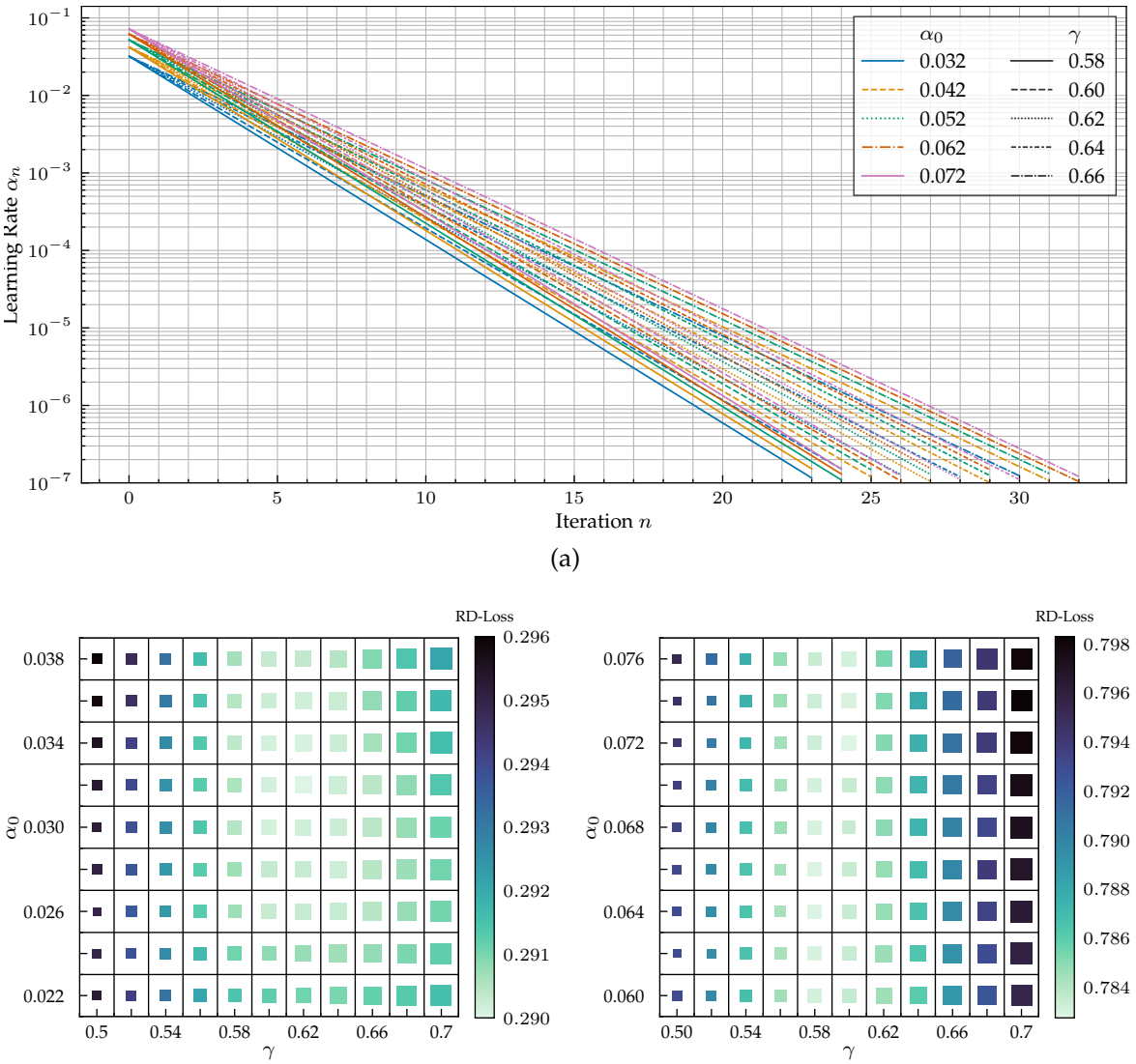


Figure 5.11: Illustration of (a) the learning rate decay for oRDO across optimization iterations, shown for different initial learning rates (α_0) and decay rates (γ). Results of the TPE [198] for eContextformer are displayed in (b) for λ_1 and (c) for λ_4, with each square size representing the number of required oRDO iteration steps.

models ($\lambda_{<4}$), while higher bitrate models ($\lambda_{>3}$) use (α_0, γ)=$(0.072, 0.60)$, leading to a uniform 26 iteration steps across all models.

Figs. 5.7, 5.9 and 5.10 and Table 5.7 demonstrate that the proposed oRDO enhances the RD performance of the eContextformer by up to 4%, achieving average bitrate savings of 14.7%, 17.0%, 14.1%, and 10.4% over VTM 16.2 [178] Kodak [138], Tecnick [139], CLIC-P [140], and CLIC-M [140] datasets, respectively. Notably, the initial models without fine-tuning benefit more from oRDO, achieving up to 7% additional gain. This suggests the potential suboptimalities of the initial training and underscores the importance of the fine-tuning step with larger crop images.

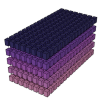


Table 5.11: Ablation study for eContextformer on the Kodak dataset [138], illustrating the impact of different configurations on BD-Rate [169] performance relative to VTM 16.2 [178] (intra) and the complexity of the entropy model during both encoding and decoding.

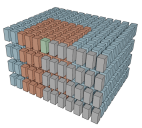
	AGMM				kMACpp**		
Context Model	Type*	k_m	Finetuned	oRDO	Encoder	Decoder	BD-Rate [%]
Multi-scale 2D [39]	SCM	1	✗	✗	56	47	3.2
Contextformer	SCM	1	✗	✗			− 5.0
	SCM	3	✗	✗	$\approx 30 \cdot 10^3$	$\approx 120 \cdot 10^3$	− 6.9
	SCM	3	✓	✗			−10.2
pContextformer (Section 5.2)	PCM	3	✗	✗	1121	1112	− 3.5
eContextformer (optimized) †	PCM	1	✗	✗	242	233	− 3.6
	PCM	3	✗	✗	253	244	− 5.2
	PCM	3	✗	✓	$\approx 20 \cdot 10^3$	244	−12.6
	PCM	3	✓	✗	253	244	−10.9
	PCM	3	✓	✓	$\approx 20 \cdot 10^3$	244	−14.7

* SCM and PCM stand for parallel and serial context models.
** The entropy model complexity during encoding and decoding.
† The combination of the proposed EGR, SFG and Caching optimizations are applied.

5.7.3 Ablation Studies

Table 5.11 presents the findings from ablation studies conducted on the Kodak dataset [138] using models such as Cui *et al.* [39], Contextformer, pContextformer, and eContextformer. All models employed the same synthesis and analysis transformations but diverged primarily in their approach to entropy modeling. Increasing the number of mixtures k_m of AGMM, as shown in [47], was found to enhance the performance of both the Contextformer and eContextformer by 1.6-1.9%, whereas incorporating spatio-channel attention within the context model yields a significant performance improvement of over 6%.

In contrast to the straightforward parallelization approach used in pContextformer, the integration of the proposed SC-(S)W-MHA mechanism, alongside complexity optimization techniques, allowed the eContextformer to achieve a more favorable performance-to-complexity balance. Fine-tuning with higher resolution images also proved beneficial for both Contextformer and eContextformer. However, sliding window attention exhibited less stability during training with higher-resolution images, which may have contributed to the modest performance gains observed with the Contextformer. Furthermore, fine-tuning reduced discrepancies between training and testing at different image resolutions, thereby diminishing the relative performance gains provided by oRDO in fine-tuned models compared



to those without fine-tuning.

Notably, oRDO does not affect the entropy model's complexity on the decoder side, as it is implemented solely during the encoding phase. Each oRDO iteration involves one forward pass and one backward pass through the entropy model (see Algorithm 3), with the backward pass carrying approximately twice the computational load of the forward pass, as described in [173], [199]. Consequently, the complexity of the encoder side entropy model, with n oRDO iterations, is projected to be three times that of the encoder without oRDO. Additionally, across all models, the decoder side entropy model complexity remains marginally lower (approximately 9 kMACpp) due to the non-utilization of the hyperprior's analysis transform (h_a) during the decoding process.

5.7.3.1 Visual Results

The visual quality comparison in Figs. 5.12 and 5.13 highlights the effectiveness of the eContextformer models with oRDO compared to traditional codecs, BPG [91] and VTM 16.2 [178]. The two uncompressed images from Kodak [138] dataset, *kodim07* and *kodim23*, and enlarged crops compressed by BPG [91] and VTM 16.2 [178], are presented to demonstrate the results. For a fair comparison, images were compressed using each algorithm, targeting similar bitrates.

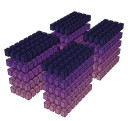
Across the two images, traditional codecs exhibit typical artifacts, including smear and aliasing, particularly noticeable in regions with intricate textures and sharp edges. The eContextformer models with oRDO produce sharper edges, more vivid colors and excel in preserving finer structure and texture details, offering better perceptual quality at similar bitrates.

Objectively, the MSE-optimized eContextformer achieves superior PSNR (e.g., 35.065 for *kodim07*, compared to 34.069 by VTM 16.2 [178]). In contrast, the MS-SSIM-optimized model excels in preserving perceptual quality, maintaining texture grain and structural integrity. This advantage is reflected in its higher MS-SSIM scores, particularly for *kodim07*, where it reaches 0.991, surpassing both VTM 16.2 [178] and BPG [91].

Similar trends are observed for *kodim23* at lower bitrates. The MSE-optimized model achieves a PSNR of 32.892 at 0.065 Bpp, outperforming both VTM 16.2 [178] (32.150 at 0.071 Bpp) and BPG [91] (31.200 at 0.077 Bpp), delivering clearer and less noisy reconstructions. Meanwhile, the MS-SSIM-optimized model, with an MS-SSIM of 0.972, emphasizes structural fidelity over pixel fidelity, trading slight reductions in PSNR for improved perceptual quality.

5.8 Chapter Summary

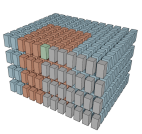
This chapter provided an in-depth investigation of state-of-the-art LIC frameworks with parallel context models, particularly focusing on grouped latent coding techniques. It identified key limitations of existing methods regarding context modeling capability (as discussed in Chapter 4), performance degradation due to parallelization, and the increased model complexity associated with parallel approaches. To address these challenges, the chapter ex-



plored techniques building upon the Contextformer framework, ultimately leading to the development of the Efficient Contextformer (eContextformer), a transformer-based architecture that combines multiple grouping techniques and introduces a novel spatio-channel shifted window attention mechanism.

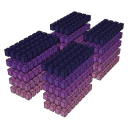
Comprehensive experiments and ablation studies explored design choices, including attention mechanism configurations, and training strategies based on findings from Chapter 3. These experiments offered valuable insights into balancing performance and complexity, resulting in significant optimizations that improved both speed and efficiency. In terms of PSNR, eContextformer showed substantial improvements, achieving up to 13% bitrate savings over VTM 16.2 [178] and outperforming state-of-the-art LIC frameworks on test datasets.

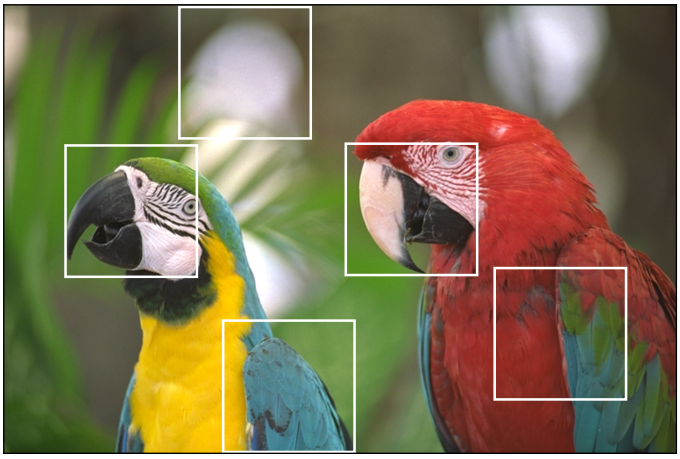
The chapter also introduced several architectural and algorithmic optimizations that reduced model complexity and runtime, including dynamic attention scaling and computation caching. These advancements led to a 145x reduction in model complexity and a 210x increase in decoding speed compared to Contextformer. Crucially, these optimizations enabled support for per-image optimization strategies such as oRDO, allowing eContextformer to achieve up to 17% bitrate savings over VTM 16.2 [178]. Visual examples further demonstrated eContextformer's ability to deliver higher perceptual quality at lower bitrates compared to state-of-the-art approaches.



	Bpp	PSNR	MS-SSIM
BPG	0.227	32.515	0.977
VTM 16.2	0.224	34.069	0.983
eContextformer (MSE)	0.205	35.065	0.985
eContextformer (MS-SSIM)	0.210	32.290	0.991

Figure 5.12: The *kodim07* image from the Kodak [138] dataset, and several crops from the images compressed by BPG [91], VTM 16.2 [178], eContextformer using oRDO (optimized for MSE or MS-SSIM), where each algorithm requires about 0.22Bpp.





	Bpp	PSNR	MS-SSIM
BPG	0.077	31.200	0.937
VTM 16.2	0.071	32.150	0.946
eContextformer (MSE)	0.065	32.892	0.952
eContextformer (MS-SSIM)	0.072	30.196	0.972

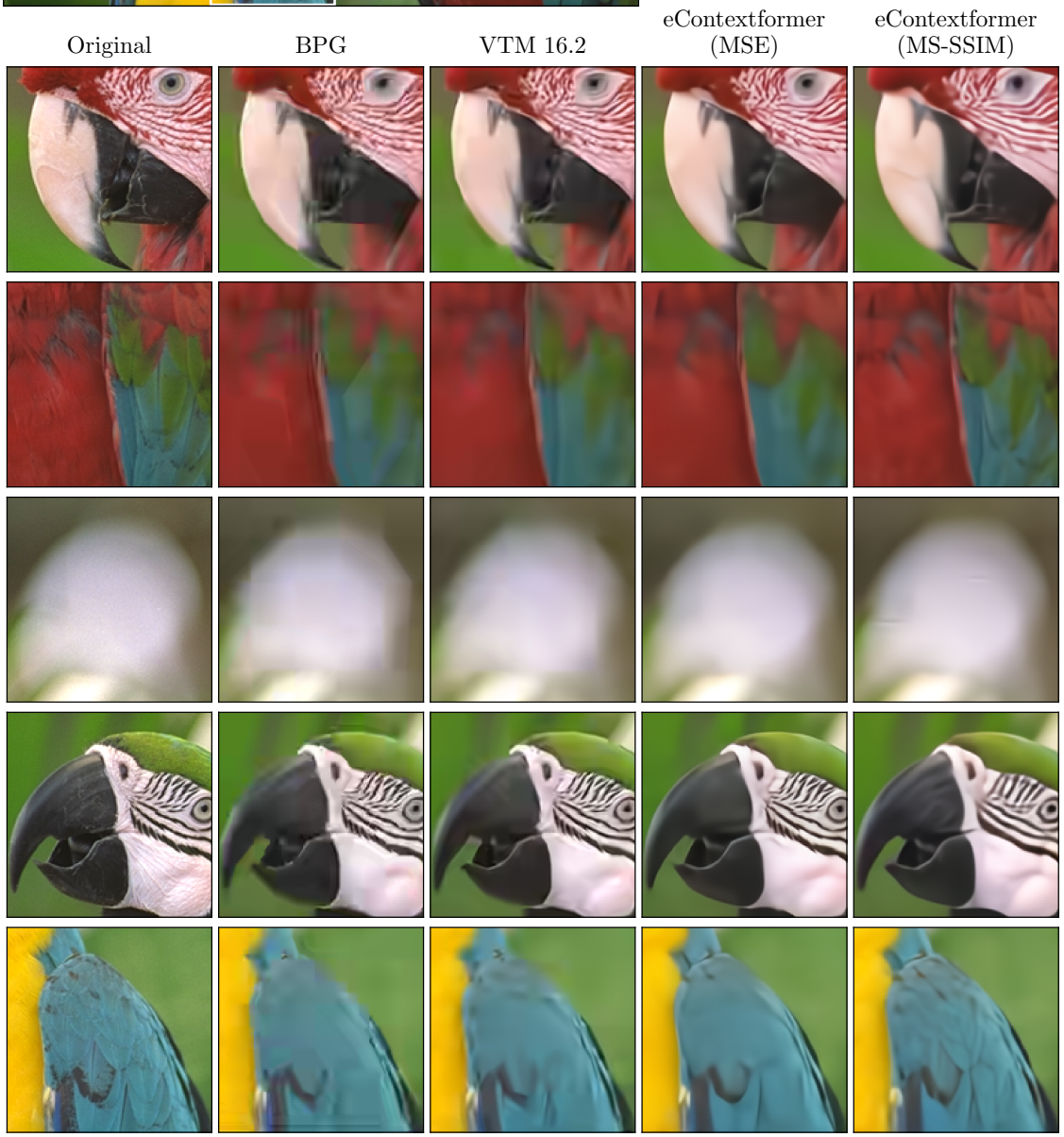
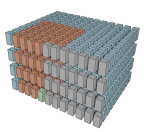


Figure 5.13: The *kodim23* image from the Kodak [138] dataset, and several crops from the images compressed by BPG [91], VTM 16.2 [178], eContextformer using oRDO (optimized for MSE or MS-SSIM), where each algorithm requires about 0.07Bpp.



Algorithm 3: Proposed oRDO method for refining latent tensors

Input: Original image $\boldsymbol{x}$, LIC Modules $\{g_a, g_s, h_a, h_s, g_{ep}, f_e, g_c\}$, Lagrangian Multiplier λ, Optimizer $optimizer$, Scheduler Parameters $\{\alpha_0, \gamma\}$, Number of oRDO Steps N_{oRDO}

Output: Refined Latent Tensors $\{\boldsymbol{y}_{rdo}, \boldsymbol{z}_{rdo}\}$

// Compression according to Eq. (2.22) to find the initial parameters

$\hat{\boldsymbol{x}}_0, \hat{\boldsymbol{y}}_0, \hat{\boldsymbol{z}}_0, \boldsymbol{y}_0, \boldsymbol{z}_0, p_{\hat{\boldsymbol{z}}_0}, p_{\hat{\boldsymbol{y}}_0} \leftarrow compress\_decompress(\boldsymbol{x}, \{g_a, g_s, h_a, h_s, g_{ep}, f_e, g_c\})$

// Find the initial loss according to Eq. (2.23)

$\mathcal{L}_0 \leftarrow \mathbb{E}[\log_2(p_{\hat{\boldsymbol{y}}_0})] + \mathbb{E}[\log_2(p_{\hat{\boldsymbol{z}}_0})] + \lambda \cdot \mathbf{D}(\boldsymbol{x}, \hat{\boldsymbol{x}}_0)$

// Initialize oRDO parameters

$N_{steps} \leftarrow \lceil \log(\frac{2e-7}{\alpha_0}) / \log(\gamma) \rceil$

$\boldsymbol{y}_{rdo}, \boldsymbol{z}_{rdo}, \hat{\boldsymbol{y}}_{best}, \hat{\boldsymbol{z}}_{best}, \mathcal{L}_{best}, \alpha_i \leftarrow \boldsymbol{y}_0, \boldsymbol{z}_0, \hat{\boldsymbol{y}}_0, \hat{\boldsymbol{z}}_0, \mathcal{L}_0, \alpha_0$

foreach *i in [1,...,N_{oRDO}]* **do**

  // Compress according to Eq. (2.22) without computing $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{z}}_{rdo}$

  $\hat{\boldsymbol{y}}_{rdo}, \hat{\boldsymbol{z}}_{rdo} \leftarrow Q(\boldsymbol{y}_{rdo}), Q(\boldsymbol{z}_{rdo})$

  $\hat{\boldsymbol{x}}_{rdo} \leftarrow g_s(\hat{\boldsymbol{y}}_{rdo})$

  $p_{\hat{\boldsymbol{y}}_{rdo}}, p_{\hat{\boldsymbol{z}}_{rdo}} \leftarrow g_e(g_c(\hat{\boldsymbol{y}}_{rdo}), h_s(\hat{\boldsymbol{z}}_{rdo})), f_e(\hat{\boldsymbol{z}}_{rdo})$

  // Find the current loss according to Eq. (2.23)

  $\mathcal{L}_{rdo} = \mathbb{E}[\log_2(p_{\hat{\boldsymbol{y}}_0})] + \mathbb{E}[\log_2(p_{\hat{\boldsymbol{z}}_0})] + \lambda \cdot \mathbf{D}(\boldsymbol{x}, \hat{\boldsymbol{x}}_0)$

  // Store best performing latent tensors

  if $\mathcal{L}_{rdo} < \mathcal{L}_{best}$ **then**

    $\hat{\boldsymbol{y}}_{best}, \hat{\boldsymbol{z}}_{best}, \mathcal{L}_{best} \leftarrow \hat{\boldsymbol{y}}_{rdo}, \hat{\boldsymbol{z}}_{rdo}, \mathcal{L}_{rdo}$

  end

  // Compute gradients according to $\mathcal{L}_{rdo}$

  $\mathcal{L}_{rdo}.backward()$

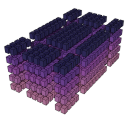
  // Update latent space parameters with α_i

  $\boldsymbol{y}_{rdo}, \boldsymbol{z}_{rdo} \leftarrow optimizer.update(\boldsymbol{y}_{rdo}, \boldsymbol{z}_{rdo}, \alpha_i)$

  // Update α_i according to Eq. (5.4)

  $\alpha_i \leftarrow \alpha_0 \gamma^i$

end



Chapter 6

Conclusion

The rapid growth of online media sharing, storage, and the increasing demand for higher quality visual content have created unprecedented pressure on storage and transmission systems. As camera and rendering technologies evolve, and as the volume of visual content continues to surge, the need for more advanced, scalable, and efficient compression techniques becomes critical. Traditional, hand-engineered codecs have made significant strides but are increasingly limited in their ability to handle the rising complexity and variability of modern visual data.

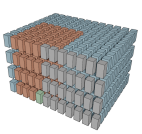
In contrast, deep learning approaches have transformed the field of computer vision, offering greater flexibility and efficiency in handling complex data structures. Among these innovations, learned image compression (LIC) has emerged as a promising alternative, leveraging deep neural networks to generate powerful latent representations of images and using learned entropy models for efficient compression. Learned entropy modeling, particularly context modeling, plays a crucial role in reducing bitrate while maintaining high image quality. However, as this thesis has identified, existing context models in LIC have not yet reached their full potential in balancing compression performance with computational efficiency.

Attention-based neural networks, such as transformers, have shown a unique ability to capture complex, long-range dependencies in data, making them ideal candidates for advancing context modeling within LIC frameworks. This thesis builds upon this motivation, proposing novel transformer-based context models to address the current limitations and push the boundaries of context modeling in LIC.

6.1 Conclusion

This thesis introduces two innovations in the field of learned image compression, namely, Contextformer and Efficient Contextformer (eContextformer), aimed at enhancing both compression performance and computational efficiency. These transformer-based context models address key limitations identified in state-of-the-art approaches, particularly in their ability to fully exploit spatial and channel-wise dependencies, while reducing computational overhead.

Contextformer introduced a dynamic and content-adaptive context model, utilizing a



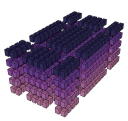
novel Spatio-Channel Multi-head Attention (SC-MHA) mechanism. This approach allowed the model to capture both spatial and cross-channel dependencies, improving the accuracy of the context model and leading to significant rate-distortion performance gains. Extensive experiments demonstrated that Contextformer achieved up to an 11% reduction in BD-Rate [169] for PSNR compared to traditional coding standard VTM 16.2 [178], while also outperforming existing LIC frameworks on test datasets, including Kodak [138], CLIC2020 [140] and Tecnick [139] datasets.

While the Contextformer with SC-MHA delivers noticeable compression gain, it also increases the runtime complexity of context modeling. To optimize the encoder-side and decoder-side implementations, the architecture of the SC-MHA was improved by introducing sliding-window processing for efficient coding of high resolution images, alongside with proposed algorithmic optimizations such as Batched Dynamic Sequence processing (BDS), Skipped intermediate Channel-Segments processing (SCS), and 3D Wavefront Coding (3D-WC). These algorithmic optimizations determine the independent sliding-window positions prior to context modeling to parallelize their processing during actual context modeling. The combination of these optimizations led up to 10x faster encoding and 2x faster decoding compared to baseline implementation of the Contextformer, while keeping its architecture unchanged.

Despite its high compression performance, Contextformer remained computationally intensive, due to its inherit serial autoregressive context modeling approach, where each latent element is modeled sequentially. To overcome this challenge, Efficient Contextformer (eContextformer) was developed as a more computationally efficient alternative. By combining several existing parallelization techniques, such as patch-wise and checkered grouping, with a novel Spatio-Channel Shifted Window Multi-head Attention (SC-SW-MHA) mechanism, e-Contextformer drastically reduced both model complexity and runtime. Comprehensive experiments on test datasets, including Kodak [138], CLIC2020 [140] and Tecnick [139] datasets, showed that eContextformer achieved up to 13% BD-Rate [169] over VTM 16.2 [178] while maintaining high image quality.

Additionally, the eContextformer architecture was further refined with the introduction of architectural optimizations, including the proposed Efficient Coding Group Rearrangement (EGR) technique. The EGR eliminates unnecessary attention computations in latent parts, allowing the attention kernel to expand dynamically during coding. The refined architecture enabled seamless integration of an algorithmic optimization called key-value caching, reducing the computation of the attention mechanism even more. With the proposed optimizations, eContextformer achieved a 145x reduction in model complexity and a 210x increase in decoding speed compared to Contextformer, making it suitable for real-time and resource-constrained applications.

The proposed complexity optimizations enabled the introduction of online Rate-Distortion Optimization (oRDO), which further enhanced eContextformer's flexibility, enabling it to dynamically adapt its compression strategy to the specific characteristics of each image. This adaptability allowed eContextformer to achieve up to 17% BD-Rate [169] reduction over VTM 16.2 [178] while maintaining high perceptual quality across a wide range of test



datasets, including Kodak [138], CLIC2020 [140] and Tecnick [139] datasets.

Moreover, the thesis presented comprehensive ablation studies and performance evaluations, which provided valuable insights into the impact of various design choices on both compression performance and computational efficiency. These studies validated the effectiveness of the proposed innovations and laid the groundwork for future research in optimizing context models for LIC.

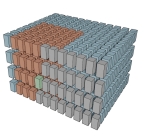
6.2 Future Work

While the contributions of this thesis significantly advance the state-of-the-art in LIC, several potential directions for future work arise from the innovations and findings presented. These directions explore how the proposed Contextformer and eContextformer models can be extended and improved, and they highlight challenges that remain in the broader field of LIC. Below is the author's proposal for future research topics of key importance:

1. **Investigating Alternative Loss Functions:** This thesis primarily used MSE and complementary MS-SSIM [160] (1−MS-SSIM) loss functions to maximize performance for PSNR and MS-SSIM [160] metrics. More perceptually optimized metrics, such as NLPD [164] and VMAF [166], could be explored both for training and testing. For instance, the JPEG AI standardization activity [79] evaluates performance across nine different metrics (see Section 3.2) but still trains its models using a combination of MSE and complementary 1−MS-SSIM. The challenge here lies in balancing different metric requirements. Optimizing for perceptual metrics may lead to models with high subjective quality but lower pixel fidelity, resulting in visually appealing images that differ significantly from the original content. Additionally, using different loss functions makes it harder to compare models directly, as each may prioritize different quality aspects, potentially masking the true effects of the proposed methods.

2. **Extension to More Complex Media Types:** The proposed transformer-based (e)Contextformer models could be extended to handle more complex media data types, such as 3D images and video. These models already extend typical spatial attention to spatio-channel attention, so incorporating an additional temporal dimension would be a natural evolution. For example, the Joint Video Exploration Team (JVET) [200] has shown interest in deep learning for video compression, where these advancements could be particularly useful. However, this extension would increase complexity, necessitating further research into how to efficiently scale these models for larger data volumes and dependencies in video streams.

3. **Complexity Reduction for Mobile Devices:** When extending the proposed methods to video compression or higher-resolution images, complexity becomes even more critical, especially for mobile devices with limited processing power. This thesis focused on efficient attention mechanisms, adhering to the traditional definition of attention as described in Section 2.3.4. Future work could explore more computationally efficient methods, such as low-rank matrix approximations [201], [202], sparse attention [203],

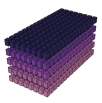


[204], or even linearized [205], [206] attention mechanisms. However, these simplifications often involve performance trade-offs. The key challenge is minimizing the degradation in performance while gaining computational efficiency.

4. **Investigating Advanced Latent Grouping Techniques:** The SC-(S)W-MHA mechanism proposed in this thesis is a fusion of widely adopted latent grouping techniques, such as patch-wise, channel-wise, and checkered grouping. There may be undiscovered ways of grouping latents to optimize computation further, allowing the context model to focus only on the necessary parts of the tensor on the decoder side. A promising approach in this direction, published recently in Mentzer *et al.* [207], uses pseudo-random sequences to minimize mutual information within coding groups. However, further research into heuristic or learned grouping methods tailored to the dynamics of the underlying transform coding could improve performance. The challenge here lies in learning sparse, discrete groupings with neural networks, which remains a difficult problem.

5. **Adapting (e)Contextformers to the Transform Coding Stage:** The focus of this thesis was on optimizing learned entropy modeling, particularly context modeling, while maintaining the rest of the compression framework for fair comparisons. Future work could explore the application of attention mechanisms like SC-MHA and SC-SW-MHA to the transform coding stage, enabling better exploitation of dependencies between intermediate transform outputs. However, introducing autoregressive attention in the transform coding stage may reduce performance and throughput, as all tensor elements are already available simultaneously. Methods that apply attention computation by clustering elements, such as those presented in [208]–[210], could be adapted to selectively mask unnecessary computations during this stage to maintain efficiency.

Each of these research directions presents its own challenges but offers exciting opportunities to push the boundaries of learned compression, enabling more efficient, scalable, and adaptive solutions for the growing demands of modern multimedia applications.



Bibliography

Publications by the author

Journal publications

[1] A. B. Koyuncu, P. Jia, A. Boev, E. Alshina, and E. Steinbach, "Efficient Contextformer: Spatio-Channel Window Attention for Fast Context Modeling in Learned Image Compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 8, pp. 7498–7511, 2024 (Cited on pages 2, 3, 5, 27, 81).

Conference publications

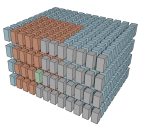
[2] A. B. Koyuncu, H. Gao, A. Boev, G. Gaikov, E. Alshina, and E. Steinbach, "Contextformer: A Transformer with Spatio-Channel Attention for Context Modeling in Learned Image Compression," in *European Conference on Computer Vision (ECCV)*, Springer, 2022, pp. 447–463 (Cited on pages 2, 3, 5, 24, 27, 49, 81, 82, 84).

[3] A. B. Koyuncu, K. Cui, A. Boev, and E. Steinbach, "Parallelized Context Modeling for Faster Image Coding," in *2021 International Conference on Visual Communications and Image Processing (VCIP)*, IEEE, 2021, pp. 1–5 (Cited on pages 2–5, 49–52, 57, 59, 62, 81–84, 86, 87, 94, 95, 100, 101).

[4] P. Jia, A. B. Koyuncu, G. Gaikov, A. Karabutov, E. Alshina, and A. Kaup, "Learning-based Conditional Image Coder Using Color Separation," in *2022 Picture Coding Symposium (PCS)*, IEEE, 2022, pp. 49–53 (Cited on pages 2, 5).

[5] H. B. Dogaroglu, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "Adapting Learned Image Codecs to Screen Content via Adjustable Transformations," in *2024 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2024, pp. 1774–1780 (Cited on page 5).

[6] P. Jia, A. B. Koyuncu, J. Mao, Z. Cui, Y. Ma, T. Guo, T. Solovyev, A. Karabutov, Y. Zhao, J. Wang, *et al.*, "Bit Rate Matching Algorithm Optimization in JPEG-AI Verification Model," in *2024 Picture Coding Symposium (PCS)*, IEEE, 2024, pp. 1–5 (Cited on page 5).



[7] P. Jia, J. Mao, E. Koyuncu, A. B. Koyuncu, T. Solovyev, A. Karabutov, Y. Zhao, E. Alshina, and A. Kaup, "Bit Distribution Study and Implementation of Spatial Quality Map in the JPEG-AI Standardization," in *2024 International Conference on Visual Communications and Image Processing (VCIP)*, IEEE, 2024, pp. 1–5 (Cited on page 5).

[8] N. Giuliani, H. You, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "CALC-VFS: Content-Adaptive Low-Complexity Video Frame Synthesis," in *25th IEEE International Symposium on Multimedia*, IEEE, 2023, pp. 57–61 (Cited on page 5).

[9] K. Cui, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "Quality-Blind Compressed Color Image Enhancement with Convolutional Neural Networks," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5 (Cited on page 6).

[10] K. Cui, A. B. Koyuncu, A. Boev, E. Alshina, and E. Steinbach, "Convolutional Neural Network-based Post-Filtering for Compressed YUV420 Images and Video," in *2021 Picture Coding Symposium (PCS)*, IEEE, 2021, pp. 1–5 (Cited on page 6).

[11] A. B. Koyuncu, E. Erçelik, E. Comulada-Simpson, J. Venrooij, M. Kaboli, and A. Knoll, "A Novel Approach to Neural Network-based Motion Cueing Algorithm for a Driving Simulator," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 2118–2125 (Cited on page 6).

Standardization-relevant reports

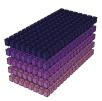
[12] E. Alshina, A. Boev, P. Jia, A. B. Koyuncu, E. Koyuncu, A. Karabutov, T. Solovyev, J. Sauer, M. Sychev, G. Gaikov, T. Guo, Y. Feng, Z. Cui, J. Mao, S. Qian, J. Wang, D. Yu, Y. Zhao, and M. Li, "Presentation of the Huawei Response to the JPEG AI Call for Proposals: Device Agnostic Learnable Image Coding Using Primary Component Extraction and Conditional Coding," Huawei Technologies, Tech. Rep. wg1m96016-REQ, 2022. [Online]. Available: https://sd.iso.org/documents/ui/#!/doc/4fedde90-6234-4aa4-9ecb-331f08e90245 (Cited on pages 61, 82, 83).

General publications

[13] M. Broz, *How Many Photos Are There? (Statistics & Trends in 2024)*, Sep. 2024. [Online]. Available: https://photutorial.com/photos-statistics/#images-shared-over-social-media (Cited on page 1).

[14] D. Duce, *Portable Network Graphics (PNG) Specification (Third Edition)*, World Wide Web Consortium, Recommendation, Jul. 2024 (Cited on pages 1, 8).

[15] ITU, "Image Parameter Values for High Dynamic Range Television for Use in Production and International Programme Exchange," *International Telecommunication Union Recommendation BT.2100*, 2018 (Cited on pages 1, 9).



[16] G. K. Wallace, "The JPEG Still Picture Compression Standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992 (Cited on pages 1, 2, 8, 10, 13, 14, 28, 41, 44).

[17] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 Still Image Compression Standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001 (Cited on pages 1, 8, 13, 14, 83).

[18] Google, *WebP: Compression Techniques*, https://developers.google.com/speed/webp/docs/compression (accessed Dec. 01, 2022), 2010 (Cited on pages 1, 8, 42).

[19] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952 (Cited on pages 1, 12, 14, 18, 19).

[20] R. A. Finkel and J. L. Bentley, "Quad Trees a Data Structure for Retrieval on Composite Keys," *Acta Informatica*, vol. 4, pp. 1–9, 1974 (Cited on pages 2, 13).

[21] X. Li *et al.*, "Multi-Type-Tree, Joint Video Exploration Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/Sc 29/Wg 11, JVET-D0117r1," in *4th Meeting: Chengdu, CN*, 2016, pp. 1–3 (Cited on pages 2, 13).

[22] J. Lainema, F. Bossen, W.-J. Han, J. Min, and K. Ugur, "Intra Coding of the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1792–1801, 2012 (Cited on pages 2, 14).

[23] J. Pfaff, A. Filippov, S. Liu, X. Zhao, J. Chen, S. De-Luxán-Hernández, T. Wiegand, V. Rufitskiy, A. K. Ramasubramonian, and G. Van der Auwera, "Intra Prediction and Mode Coding in VVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3834–3847, 2021 (Cited on pages 2, 14).

[24] V. Sze, M. Budagavi, and G. J. Sullivan, "High Efficiency Video Coding (HEVC)," in *Integrated Circuit and Systems, Algorithms and Architectures*, vol. 39, Springer, 2014, p. 40 (Cited on pages 2, 8, 10, 13, 14, 20, 28, 41, 44, 46, 61, 83).

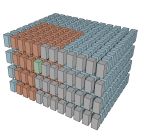
[25] "Versatile Video Coding," Rec. ITU-T H.266 and ISO/IEC 23090-3, Standard, Aug. 2020 (Cited on pages 2, 10, 13, 14, 20, 28, 41, 44, 46, 83).

[26] M. M. Hannuksela, E. Aksu, V. Vadakital, and J. Lainema, "Overview of the High Efficiency Image File Format," *JCTVC-V0072*, 2015 (Cited on pages 2, 8).

[27] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A Survey on Deep Learning: Algorithms, Techniques, and Applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018 (Cited on page 2).

[28] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A Survey of Deep Learning Techniques for Autonomous Driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020 (Cited on page 2).

[29] S. Dong, P. Wang, and K. Abbas, "A Survey on Deep Learning and Its Applications," *Computer Science Review*, vol. 40, p. 100 379, 2021 (Cited on page 2).



[30] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, 2023 (Cited on page 2).

[31] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-End Optimization of Nonlinear Transform Codes for Perceptual Quality," in *2016 Picture Coding Symposium (PCS)*, IEEE, 2016, pp. 1–5 (Cited on pages 2, 17, 18).

[32] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-End Optimized Image Compression," in *5th International Conference on Learning Representations, ICLR 2017*, 2017 (Cited on pages 2, 17–19, 25, 26, 28, 45, 48, 53).

[33] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational Image Compression with a Scale Hyperprior," in *International Conference on Learning Representations*, 2018 (Cited on pages 2, 17–19, 25, 26, 28, 45, 48, 53, 66–69, 99, 104).

[34] D. Minnen, J. Ballé, and G. Toderici, "Joint Autoregressive and Hierarchical Priors for Learned Image Compression," in *NeurIPS*, 2018 (Cited on pages 2, 3, 18, 20, 25–28, 45, 48–51, 64, 66–70, 97–99, 104, 106).

[35] D. Minnen and S. Singh, "Channel-Wise Autoregressive Entropy Models for Learned Image Compression," in *2020 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2020, pp. 3339–3343 (Cited on pages 2, 3, 27, 43–45, 48, 50–52, 59, 62, 64–68, 70, 82–84, 87, 94, 96–107).

[36] J. Ballé, P. A. Chou, D. Minnen, S. Singh, N. Johnston, E. Agustsson, S. J. Hwang, and G. Toderici, "Nonlinear Transform Coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 2, pp. 339–353, 2020 (Cited on pages 2, 16, 21).

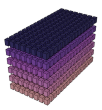
[37] J. Zhou, S. Wen, A. Nakagawa, K. Kazui, and Z. Tan, "Multi-Scale and Context-Adaptive Entropy Model for Image Compression," *arXiv preprint arXiv:1910.07844*, 2019 (Cited on pages 2, 3, 50–52, 62, 81, 83).

[38] T. Guo, J. Wang, Z. Cui, Y. Feng, Y. Ge, and B. Bai, "Variable Rate Image Compression with Content Adaptive Optimization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2020 (Cited on pages 2, 3, 26, 27, 50–53, 62, 81, 83).

[39] Z. Cui, J. Wang, S. Gao, T. Guo, Y. Feng, and B. Bai, "Asymmetric Gained Deep Image Compression with Continuous Rate Adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 532–10 541 (Cited on pages 2, 3, 27, 45, 48, 50–53, 62, 63, 65–70, 72, 81, 83, 96, 99, 100, 104–106, 109).

[40] H. Liu, T. Chen, P. Guo, Q. Shen, X. Cao, Y. Wang, and Z. Ma, "Non-Local Attention Optimized Deep Image Compression," *arXiv preprint arXiv:1904.09757*, 2019 (Cited on pages 2, 3, 45, 50, 52, 53, 62, 70, 71, 81, 95, 98, 107).

[41] H. Liu, T. Chen, Q. Shen, and Z. Ma, "Practical Stacked Non-Local Attention Modules for Image Compression.," in *CVPR Workshops*, 2019 (Cited on pages 2, 26, 27, 50, 52, 62, 81, 95).



[42] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Conditional Probability Models for Deep Image Compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4394–4402 (Cited on pages 2, 3, 17, 50, 52, 62, 81, 95).

[43] Z. Tang, H. Wang, X. Yi, Y. Zhang, S. Kwong, and C.-C. J. Kuo, "Joint Graph Attention and Asymmetric Convolutional Neural Network for Deep Image Compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 1, pp. 421–433, 2022 (Cited on pages 2, 45, 48, 50, 98, 104).

[44] Y. Qian, Z. Tan, X. Sun, M. Lin, D. Li, Z. Sun, L. Hao, and R. Jin, "Learning Accurate Entropy Model with Global Reference for Image Compression," in *International Conference on Learning Representations*, 2020 (Cited on pages 2, 3, 45, 48, 50–52, 62, 66–69, 81, 104).

[45] Z. Guo, Z. Zhang, R. Feng, and Z. Chen, "Causal Contextual Prediction for Learned Image Compression," *IEEE Transactions on Circuits and Systems for Video Technology*, 2021 (Cited on pages 2, 3, 26, 27, 45, 48, 50–52, 62, 64, 66–68, 81, 104).

[46] Y. Qian, X. Sun, M. Lin, Z. Tan, and R. Jin, "Entroformer: A Transformer-based Entropy Model for Learned Image Compression," in *International Conference on Learning Representations*, 2021 (Cited on pages 2, 3, 27, 45, 48, 50–52, 62, 64, 66–71, 81–84, 95, 97–99, 101, 104–107).

[47] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7939–7948 (Cited on pages 2, 3, 26, 27, 45, 48, 49, 53, 63–69, 71, 96–99, 104, 107, 109).

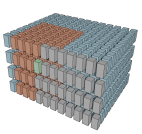
[48] T. Chen, H. Liu, Z. Ma, Q. Shen, X. Cao, and Y. Wang, "End-to-End Learnt Image Compression via Non-Local Attention Optimization and Improved Context Modeling," *IEEE Transactions on Image Processing*, vol. 30, pp. 3179–3191, 2021. DOI: 10.1109/TIP.2021.3058615 (Cited on pages 2, 26, 27, 48, 50, 52, 53, 66–69, 71, 98, 104).

[49] J. Lee, S. Cho, and S.-K. Beack, "Context-Adaptive Entropy Model for End-to-End Optimized Image Compression," in *6th International Conference on Learning Representations, ICLR 2018*, 2018 (Cited on pages 2, 3, 20, 49, 57).

[50] L. Zhou, Z. Sun, X. Wu, and J. Wu, "End-to-End Optimized Image Compression with Attention Mechanism.," in *CVPR Workshops*, 2019 (Cited on page 2).

[51] M. Li, K. Ma, J. You, D. Zhang, and W. Zuo, "Efficient and Effective Context-based Convolutional Entropy Modeling for Image Compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 5900–5911, 2020. DOI: 10.1109/TIP.2020.2985225 (Cited on pages 2, 61, 82, 83).

[52] D. He, Y. Zheng, B. Sun, Y. Wang, and H. Qin, "Checkerboard Context Model for Efficient Learned Image Compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 771–14 780 (Cited on pages 2, 3, 45, 48, 59, 82–84, 87, 94, 95, 98–100, 104).



[53] J. Campos, S. Meierhans, A. Djelouah, and C. Schroers, "Content Adaptive Optimization for Neural Image Compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019 (Cited on pages 2, 28, 93).

[54] Y. Yang, R. Bamler, and S. Mandt, "Improving Inference for Neural Image Compression," *Advances in Neural Information Processing Systems*, vol. 33, pp. 573–584, 2020 (Cited on pages 2, 28, 97, 104).

[55] C. Gao, T. Xu, D. He, Y. Wang, and H. Qin, "Flexible Neural Image Compression via Code Editing," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 184–12 196, 2022 (Cited on pages 2, 28).

[56] D. He, Z. Yang, W. Peng, R. Ma, H. Qin, and Y. Wang, "Elic: Efficient Learned Image Compression with Unevenly Grouped Space-Channel Contextual Adaptive Coding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5718–5727 (Cited on pages 2, 26, 27, 45, 48, 82–84, 95, 96, 98–106).

[57] Y. Yang and S. Mandt, "Computationally-Efficient Neural Image Compression with Shallow Decoders," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 530–540 (Cited on pages 2, 33, 45, 48, 84, 97, 98, 102–104, 106, 107).

[58] H. Fu, F. Liang, J. Liang, B. Li, G. Zhang, and J. Han, "Asymmetric Learned Image Compression with Multi-Scale Residual Block, Importance Scaling, and Post-Quantization Filtering," *IEEE Transactions on Circuits and Systems for Video Technology*, 2023 (Cited on pages 2, 45, 48, 84, 97, 98, 101–104).

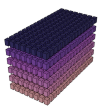
[59] J. Liu, H. Sun, and J. Katto, "Learned Image Compression with Mixed Transformer-CNN Architectures," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14 388–14 397 (Cited on pages 2, 3, 26, 27, 45, 48, 82–84, 95, 97, 98, 101–107).

[60] G.-H. Wang, J. Li, B. Li, and Y. Lu, "Evc: Towards Real-Time Neural Image Compression with Mask Decay," *arXiv preprint arXiv:2302.05071*, 2023 (Cited on pages 2, 45, 48, 82–84, 97, 98, 102–104, 106, 107).

[61] J.-H. Kim, B. Heo, and J.-S. Lee, "Joint Global and Local Hierarchical Priors for Learned Image Compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5992–6001 (Cited on pages 2, 3, 27, 45, 48, 50, 52, 62, 66–69, 104).

[62] M. Lu, P. Guo, H. Shi, C. Cao, and Z. Ma, "Transformer-based Image Compression," in *2022 Data Compression Conference (DCC)*, IEEE, 2022, pp. 469–469 (Cited on pages 2, 3, 26, 27, 45, 48, 66–68, 104).

[63] M. Lu, F. Chen, S. Pu, and Z. Ma, "High-Efficiency Lossy Image Coding Through Adaptive Neighborhood Information Aggregation," *arXiv preprint arXiv:2204.11448*, 2022 (Cited on pages 2, 3, 26, 27, 45, 48, 82–84, 95, 98, 99, 102–104).



[64] F. Lin, H. Sun, J. Liu, and J. Katto, "Multistage Spatial Context Models for Learned Image Compression," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023, pp. 1–5 (Cited on pages 2, 82–84, 95).

[65] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. Chinen, S. J. Hwang, J. Shor, and G. Toderici, "Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4385–4393 (Cited on pages 2, 42).

[66] Z. Duan, M. Lu, Z. Ma, and F. Zhu, "Lossy Image Compression with Quantized Hierarchical Vaes," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 198–207 (Cited on pages 2, 33).

[67] L. Helminger, A. Djelouah, M. Gross, and C. Schroers, "Lossy Image Compression with Normalizing Flows," in *Neural Compression: From Information Theory to Applications–Workshop@ ICLR 2021*, 2021 (Cited on pages 2, 33).

[68] Y.-H. Chen, Y.-C. Weng, C.-H. Kao, C. Chien, W.-C. Chiu, and W.-H. Peng, "Transtic: Transferring Transformer-based Image Compression From Human Perception to Machine Perception," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 23 297–23 307 (Cited on pages 2, 26, 27, 33).

[69] L. D. Chamain, F. Racapé, J. Bégaint, A. Pushparaja, and S. Feltman, "End-to-End Optimized Image Compression for Machines, a Study," in *2021 Data Compression Conference (DCC)*, IEEE, 2021, pp. 163–172 (Cited on pages 2, 38).

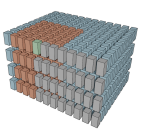
[70] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. V. Gool, "Soft-to-Hard Vector Quantization for End-to-End Learning Compressible Representations," *Advances in Neural Information Processing Systems*, vol. 30, 2017 (Cited on pages 2, 17, 18).

[71] N. Zou, H. Zhang, F. Cricri, H. R. Tavakoli, J. Lainema, M. Hannuksela, E. Aksu, and E. Rahtu, "L 2 C–Learning to Learn to Compress," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, IEEE, 2020, pp. 1–6 (Cited on pages 2, 26, 27).

[72] Z. Guo, Z. Zhang, R. Feng, and Z. Chen, "Soft Then Hard: Rethinking the Quantization in Neural Image Compression," in *International Conference on Machine Learning*, PMLR, 2021, pp. 3920–3929 (Cited on pages 2, 27).

[73] Y. Hu, W. Yang, and J. Liu, "Coarse-to-Fine Hyper-Prior Modeling for Learned Image Compression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 11 013–11 020 (Cited on pages 2, 27).

[74] H. Fu and F. Liang, "Learned Image Compression with Generalized Octave Convolution and Cross-Resolution Parameter Estimation," *Signal Processing*, vol. 202, p. 108 778, 2023 (Cited on pages 2, 27).



[75] L. Zhou, C. Cai, Y. Gao, S. Su, and J. Wu, "Variational Autoencoder for Low Bit-rate Image Compression," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 2617–2620 (Cited on pages 2, 26, 27).

[76] H. Fu, F. Liang, J. Lin, B. Li, M. Akbari, J. Liang, G. Zhang, D. Liu, C. Tu, and J. Han, "Learned Image Compression with Gaussian-Laplacian-Logistic Mixture Model and Concatenated Residual Modules," *IEEE Transactions on Image Processing*, vol. 32, pp. 2063–2076, 2023 (Cited on pages 2, 27).

[77] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wang, "Image and Video Compression with Neural Networks: A Review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 6, pp. 1683–1698, 2019 (Cited on page 2).

[78] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, pp. 353–374, 2023 (Cited on pages 2, 16).

[79] J. Ascenso, E. Alshina, and T. Ebrahimi, "The JPEG AI Standard: Providing Efficient Human and Machine Visual Data Consumption," *IEEE MultiMedia*, vol. 30, no. 1, pp. 100–111, 2023 (Cited on pages 2, 41, 42, 45, 65, 97, 117).

[80] S. Liu, E. Alshina, J. Pfaff, M. Wien, P. Wu, and Y. Ye, "JVET AHG Report: Neural-Network-based Video Coding," *Joint Video Experts Team of ITU-T SG*, vol. 16, 2020 (Cited on page 2).

[81] Z. Niu, G. Zhong, and H. Yu, "A Review on the Attention Mechanism of Deep Learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.03.091. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122100477X (Cited on page 3).

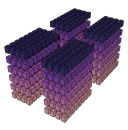
[82] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008 (Cited on pages 3, 21–23, 59).

[83] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image Transformer," in *International Conference on Machine Learning*, PMLR, 2018, pp. 4055–4064 (Cited on pages 3, 24, 58).

[84] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," in *European Conference on Computer Vision*, Springer, 2020, pp. 213–229 (Cited on pages 3, 23).

[85] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, 2020 (Cited on pages 3, 23, 24, 53–55, 88, 94).

[86] P. Esser, R. Rombach, and B. Ommer, "Taming Transformers for High-Resolution Image Synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 873–12 883 (Cited on pages 3, 23, 24, 55, 58, 59).



[87] Y. Jiang, S. Chang, and Z. Wang, "Transgan: Two Transformers Can Make One Strong Gan," *arXiv preprint arXiv:2102.07074*, 2021 (Cited on pages 3, 23).

[88] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022 (Cited on pages 3, 23, 24, 88–91, 94, 97).

[89] M. M. Naseer, K. Ranasinghe, S. H. Khan, M. Hayat, F. Shahbaz Khan, and M.-H. Yang, "Intriguing Properties of Vision Transformers," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23 296–23 308, 2021 (Cited on pages 3, 50, 52).

[90] I. CompuServe, *Graphics Interchange Format — Version 89a*, 1990 (Cited on page 8).

[91] F. Bellard, *BPG Image Format*, https://bellard.org/bpg (accessed Dec. 01, 2022), 2015 (Cited on pages 8, 41, 44, 46, 47, 64, 67–69, 73–77, 97, 99, 102–104, 110, 112, 113).

[92] B. A. Wandell, "Foundations of Vision," in Sinauer Associates, 1995 (Cited on pages 9, 11).

[93] ITU, "Studio Encoding Parameters of Digital Television for Standard 4: 3 and Widescreen 16: 9 Aspect Ratios," *International Telecommunication Union Recommendation BT.601*, 2011 (Cited on page 9).

[94] ITU, "Parameter Values for the HDTV Standards for Production and International Programme Exchange," *International Telecommunication Union Recommendation BT.709*, 2015 (Cited on pages 9, 34, 46, 47).

[95] R. C. Gonzales and P. Wintz, *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., 1987 (Cited on page 9).

[96] D. Salomon, *Data Compression*. Springer, 2002 (Cited on page 9).

[97] D. Salomon and G. Motta, *Handbook of Data Compression*. Springer Science & Business Media, 2010 (Cited on pages 9, 12).

[98] A. C. Bovik, *Handbook of Image and Video Processing*. Academic press, 2010 (Cited on page 9).

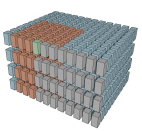
[99] V. K. Goyal, "Theoretical Foundations of Transform Coding," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 9–21, 2001 (Cited on page 9).

[100] A. C. Bovik, "The Essential Guide to Image Processing," in Academic Press, 2009, pp. 429–432 (Cited on page 11).

[101] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948 (Cited on page 11).

[102] J. Rissanen and G. G. Langdon, "Arithmetic Coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979 (Cited on pages 12, 18, 19).

[103] R. C. Pasco, "Source Coding Algorithms for Fast Data Compression," Ph.D. dissertation, Stanford University CA, 1976 (Cited on pages 12, 18, 19).



[104] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987 (Cited on page 12).

[105] M. Bun, *Lecture Notes 6: Information Theory: Entropy, Mutual Information*, Lecture notes for CAS CS 591 B: Communication Complexity, Boston University, Available from: https://cs-people.bu.edu/mbun/courses/591_F19/notes/lec6.pdf, 2019 (Cited on page 12).

[106] Ahmed, Nasir and Natarajan, T_ and Rao, Kamisetty R, "Discrete Cosine Transform," *IEEE Transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974 (Cited on pages 13, 16, 18).

[107] M. Unser and T. Blu, "Mathematical Properties of the JPEG2000 Wavelet Filters," *IEEE Transactions on Image Processing*, vol. 12, no. 9, pp. 1080–1090, 2003 (Cited on page 13).

[108] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, "Block Partitioning Structure in the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1697–1706, 2012 (Cited on page 13).

[109] X. Zhao, S.-H. Kim, Y. Zhao, H. E. Egilmez, M. Koo, S. Liu, J. Lainema, and M. Karczewicz, "Transform Coding in the VVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3878–3890, 2021 (Cited on page 13).

[110] T. R. Fischer, M. W. Marcellin, and M. Wang, "Trellis-coded Vector Quantization," *IEEE Transactions on Information Theory*, vol. 37, no. 6, pp. 1551–1566, 1991 (Cited on page 14).

[111] I. S. ISO, I. JTC, *et al.*, "Information Technology-JPEG 2000 Image Coding System-Part 1: Core Coding System," *ISO/IEC 15444-1*, 2001 (Cited on page 14).

[112] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based Adaptive Binary Arithmetic Coding in the H. 264/Avc Video Compression Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, 2003 (Cited on page 14).

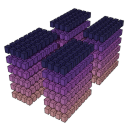
[113] G. Sullivan, "Rate-Distortion Optimization for Video Compression," *IEEE Signal Processing Magazine*, vol. 84, 1998 (Cited on page 14).

[114] D. Bull and F. Zhang, *Intelligent Image and Video Compression: Communicating Pictures*. Academic Press, 2021 (Cited on page 15).

[115] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989 (Cited on page 16).

[116] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993 (Cited on page 16).

[117] H. Lin and S. Jegelka, "Resnet with One-Neuron Hidden Layers Is a Universal Approximator," *Advances in Neural Information Processing Systems*, vol. 31, 2018 (Cited on page 16).



[118] D.-X. Zhou, "Universality of Deep Convolutional Neural Networks," *Applied and Computational Harmonic Analysis*, vol. 48, no. 2, pp. 787–794, 2020 (Cited on page 16).

[119] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation, Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Ed. De Rumelhart and J. McClelland. Vol. 1. 1986," *Biometrika*, vol. 71, no. 599-607, p. 6, 1986 (Cited on page 16).

[120] J. Ballé, V. Laparra, and E. P. Simoncelli, "Density Modeling of Images Using a Generalized Normalization Transformation," in *4th International Conference on Learning Representations, ICLR 2016*, 2016 (Cited on pages 16, 26, 53, 63, 96).

[121] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *arXiv preprint arXiv: 1308.3432*, 2013 (Cited on pages 17, 18, 93).

[122] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," in *International Conference on Learning Representations*, 2022 (Cited on page 17).

[123] A. Van Den Oord, O. Vinyals, *et al.*, "Neural Discrete Representation Learning," *Advances in Neural Information Processing Systems*, vol. 30, 2017 (Cited on pages 17, 18).

[124] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional Image Generation with Pixelcnn Decoders," *Advances in Neural Information Processing Systems*, vol. 29, 2016 (Cited on page 20).

[125] R. Baeza-Yates, B. Ribeiro-Neto, *et al.*, *Modern Information Retrieval*. ACM press New York, 1999, vol. 463 (Cited on page 21).

[126] P. Wadler, "Theorems for Free!" In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, 1989, pp. 347–359 (Cited on page 22).

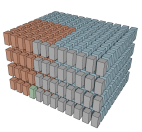
[127] R. A. Horn, "The Hadamard Product," in *Proc. Symp. Appl. Math*, vol. 40, 1990, pp. 87–169 (Cited on pages 22, 53).

[128] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814 (Cited on pages 23, 26, 53).

[129] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," *arXiv preprint arXiv:1606.08415*, 2016 (Cited on pages 23, 26, 63, 96).

[130] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778 (Cited on pages 23, 26).

[131] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer Normalization*, 2016. arXiv: 1607.06450 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1607.06450 (Cited on page 23).



[132] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *Proc. icml*, Atlanta, GA, vol. 30, 2013, p. 3 (Cited on pages 26, 63, 96).

[133] Y. Zhang, K. Li, K. Li, B. Zhong, and Y. Fu, "Residual Non-Local Attention Networks for Image Restoration," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HkeGhoA5FX (Cited on pages 26, 53, 63, 96).

[134] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-Local Neural Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803 (Cited on page 26).

[135] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Practical Full Resolution Learned Lossless Image Compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 629–10 638 (Cited on page 27).

[136] C. Cremer, X. Li, and D. Duvenaud, "Inference Suboptimality in Variational Autoencoders," in *International conference on machine learning*, PMLR, 2018, pp. 1078–1086 (Cited on page 27).

[137] J. Zhao, B. Li, J. Li, R. Xiong, and Y. Lu, "A Universal Encoder Rate Distortion Optimization Framework for Learned Compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1880–1884 (Cited on page 28).

[138] R. Franzen, "Kodak Lossless True Color Image Suite," 1999 (Cited on pages 31, 35–38, 47, 64, 65, 67–70, 73, 74, 76, 77, 86, 97, 99–106, 108–110, 112, 113, 116, 117).

[139] N. Asuni and A. Giachetti, "TESTIMAGES: A Large-Scale Archive for Testing Visual Devices and Basic Image Processing Algorithms.," in *STAG*, 2014, pp. 63–70 (Cited on pages 32, 35–40, 47, 64, 65, 67–69, 97, 100–105, 108, 116, 117).

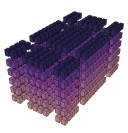
[140] G. Toderici, L. Theis, N. Johnston, E. Agustsson, F. Mentzer, J. Ballé, W. Shi, and R. Timofte, *Workshop and Challenge on Learned Image Compression (CLIC2020)*, CVPR, 2020. [Online]. Available: http://www.compression.cc (Cited on pages 32, 35–40, 47, 64, 65, 67–69, 97, 101–105, 108, 116, 117).

[141] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015 (Cited on page 32).

[142] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org (Cited on page 32).

[143] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," *SN Computer Science*, vol. 2, no. 6, p. 420, 2021 (Cited on page 32).

[144] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, "Video Enhancement with Task-Oriented Flow," *International Journal of Computer Vision (IJCV)*, vol. 127, no. 8, pp. 1106–1125, 2019 (Cited on pages 33, 38–40, 47, 64, 96, 99–101).



[145] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "CompressAI: A PyTorch Library and Evaluation Platform for End-to-End Compression Research," *arXiv preprint arXiv:2011.03029*, 2020 (Cited on pages 33, 42, 64).

[146] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755 (Cited on pages 33, 38–40, 47, 97, 100–103, 107).

[147] S. Winkler, "Analysis of Public Image and Video Databases for Quality Assessment," *IEEE Journal of Selected Topics in Signal Processing*, vol. 6, no. 6, pp. 616–625, 2012 (Cited on page 33).

[148] X. Han, I. R. Khan, and S. Rahardja, "High Dynamic Range Image Tone Mapping: Literature Review and Performance Benchmark," *Digital Signal Processing*, vol. 137, p. 104 015, 2023 (Cited on page 33).

[149] A. T. 801.03, "Digital Transport of One-way Video Signals-Parameters for Objective Performance Assessment," *American National Standards Institute Washington, DC, USA*, 1996 (Cited on pages 33, 34, 36, 39).

[150] H. Yu and S. Winkler, "Image Complexity and Spatial Information," in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, 2013, pp. 12–17. DOI: 10.1109/QoMEX.2013.6603194 (Cited on pages 33, 34, 36, 39).

[151] D. Hasler and S. E. Suesstrunk, "Measuring Colorfulness in Natural Images," in *Human vision and electronic imaging VIII*, SPIE, vol. 5007, 2003, pp. 87–95 (Cited on page 33).

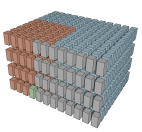
[152] B. Zhou, S. Xu, and X.-x. Yang, "Computing the Color Complexity of Images," in *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, IEEE, 2015, pp. 1898–1902 (Cited on pages 33, 35, 38).

[153] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural Features for Image Classification," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 6, pp. 610–621, 1973 (Cited on pages 33–35, 38).

[154] B. Sebastian V, A. Unnikrishnan, and K. Balakrishnan, "Gray Level Co-Occurrence Matrices: Generalisation and Some New Features," *arXiv preprint arXiv:1205.4831*, 2012 (Cited on pages 33–35, 38).

[155] L. Mahon and T. Lukasiewicz, "Minimum Description Length Clustering to Measure Meaningful Image Complexity," *Pattern Recognition*, vol. 145, p. 109 889, 2024, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2023.109889. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320323005873 (Cited on pages 33, 34, 36, 39).

[156] M. Hall-Beyer, "GLCM Texture: A Tutorial," *National Council on Geographic Information and Analysis Remote Sensing Core Curriculum*, vol. 3, no. 1, p. 75, 2000 (Cited on page 34).



[157] I. Sobel and G. Feldman, "A 3×3 Isotropic Gradient Operator for Image Processing," *Pattern Classification and Scene Analysis*, pp. 271–272, Jan. 1973 (Cited on page 34).

[158] L. Duan, J. Liu, W. Yang, T. Huang, and W. Gao, "Video Coding for Machines: A Paradigm of Collaborative Compression and Intelligent Analytics," *IEEE Transactions on Image Processing*, vol. 29, pp. 8680–8695, 2020 (Cited on page 38).

[159] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004 (Cited on pages 41, 42).

[160] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale Structural Similarity for Image Quality Assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, IEEE, vol. 2, 2003, pp. 1398–1402 (Cited on pages 41–43, 64, 65, 69, 97, 99, 100, 117).

[161] J. Brandenburg, A. Wieckowski, T. Hinz, and B. Bross, "VVenC Fraunhofer Versatile Video Encoder," *Cit. On*, p. 3, 2020 (Cited on pages 41, 44).

[162] H. R. Sheikh and A. C. Bovik, "Image Information and Visual Quality," *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 430–444, 2006 (Cited on page 41).

[163] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "FSIM: A Feature Similarity Index for Image Quality Assessment," *IEEE Transactions on Image Processing*, vol. 20, no. 8, pp. 2378–2386, 2011. DOI: 10.1109/TIP.2011.2109730 (Cited on page 41).

[164] V. Laparra, A. Berardino, J. Ballé, and E. Simoncelli, "Perceptually Optimized Image Rendering.," *Journal of the Optical Society of America. A, Optics, Image Science, and Vision*, vol. 34, no. 9, pp. 1511–1525, 2017 (Cited on pages 41, 117).

[165] Z. Wang and Q. Li, "Information Content Weighting for Perceptual Image Quality Assessment," *IEEE Transactions on Image Processing*, vol. 20, no. 5, pp. 1185–1198, 2010 (Cited on page 41).

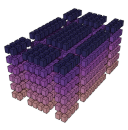
[166] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, M. Manohara, *et al.*, "Toward a Practical Perceptual Video Quality Metric," *The Netflix Tech Blog*, vol. 6, no. 2, p. 2, 2016 (Cited on pages 41, 117).

[167] P. Gupta, P. Srivastava, S. Bhardwaj, and V. Bhateja, "A Modified PSNR Metric Based on HVS for Quality Assessment of Color Images," in *2011 International Conference on Communication and Industrial Application*, IEEE, 2011, pp. 1–4 (Cited on page 41).

[168] F. Gongfan, *PyTorch MS-SSIM*, https://github.com/VainF/pytorch-msssim, 2019 (Cited on page 42).

[169] G. Bjontegaard, "Calculation of Average PSNR Differences Between RD-curves," *ITU SG16 Doc. VCEG-M33*, 2001 (Cited on pages 43, 44, 65, 72, 75, 84, 87, 100, 101, 104, 109, 116).

[170] NVIDIA, P. Vingelmann, and F. H. Fitzek, *CUDA, Release: 11.4*, 2021. [Online]. Available: https://developer.nvidia.com/cuda-toolkit (Cited on pages 44, 45, 65, 71, 99).



[171] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (Cited on pages 44, 45, 64, 65, 97, 99).

[172] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/ (Cited on pages 44, 45, 64, 97).

[173] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System Optimizations Enable Training Deep Learning Models with over 100 Billion Parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506 (Cited on pages 45, 65, 97, 110).

[174] V. Sovrasov. "Ptflops: A Flops Counting Tool for Neural Networks in Pytorch Framework." (2018), [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch (Cited on pages 45, 65).

[175] T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray, *et al.*, "Scaling Laws for Autoregressive Generative Modeling," *arXiv preprint arXiv:2010.14701*, 2020 (Cited on page 45).

[176] F. Marin, A. Rohatgi, and S. Charlot, "WebPlotDigitizer, a Polyvalent and Free Software to Extract Spectra From Old Astronomical Publications: Application to Ultraviolet Spectropolarimetry," in *SF2A-2017: Proceedings of the Annual meeting of the French Society of Astronomy and Astrophysics*, 2017, p. Di (Cited on page 46).

[177] A. Rohatgi, *WebPlotDigitizer: Version 4*, 2015. [Online]. Available: https://automeris.io/WebPlotDigitizer (Cited on page 46).

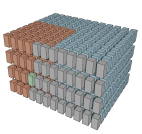
[178] J. V. E. Team, *Versatile Video Coding (VVC) Reference Software: VVC Test Model (Vtm)*, https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM (accessed Dec. 01, 2022), 2022 (Cited on pages 46, 47, 64, 65, 67–71, 73–77, 86, 87, 97, 99–105, 107–113, 116).

[179] S. Tomar, "Converting Video Formats with FFmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006 (Cited on pages 46, 47).

[180] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional Image Generation with PixelCNN Decoders," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 4797–4805 (Cited on page 49).

[181] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic Convolution: Attention over Convolution Kernels," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 030–11 039 (Cited on page 50).

[182] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. John Wiley & Sons, 2009 (Cited on page 50).



[183] P. Wang, X. Wang, F. Wang, M. Lin, S. Chang, H. Li, and R. Jin, "Kvt: K-Nn Attention for Boosting Vision Transformers," in *European conference on computer vision*, Springer, 2022, pp. 285–302 (Cited on page 52).

[184] Y. Wang, Y. Zhang, F. Zhang, S. Wang, M. Lin, Y. Zhang, and X. Sun, "Ada-Nets: Face Clustering via Adaptive Neighbour Discovery in the Structure Space," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=QJWVP4CTmW4 (Cited on page 52).

[185] J. Ziv, "On Universal Quantization," *IEEE Transactions on Information Theory*, vol. 31, no. 3, pp. 344–347, 1985 (Cited on page 53).

[186] T. Kato, S. Omachi, and H. Aso, "Asymmetric Gaussian and Its Application to Pattern Recognition," in *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*, Springer, 2002, pp. 405–413 (Cited on page 53).

[187] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. [Online]. Available: https://aclanthology.org/N19-1423 (Cited on page 59).

[188] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1827–1838, 2012 (Cited on page 61).

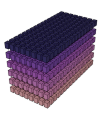
[189] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1648–1667, Dec. 2012 (Cited on page 61).

[190] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014 (Cited on pages 64, 96).

[191] D.-W. Kim, J. Ryun Chung, and S.-W. Jung, "Grdn: Grouped Residual Dense Network for Real Image Denoising and Gan-based Real-World Noise Modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019 (Cited on page 64).

[192] B. Bross, J. Chen, J.-R. Ohm, G. J. Sullivan, and Y.-K. Wang, "Developments in International Video Coding Standardization After AVC, with an Overview of Versatile Video Coding (VVC)," *Proceedings of the IEEE*, vol. 109, no. 9, pp. 1463–1493, 2021 (Cited on page 83).

[193] A. Matton and A. Lam, *Making Pytorch Transformer Twice as Fast on Sequence Generation*. https://scale.com/blog/pytorch-improvements (accessed Dec. 01, 2022), 2020 (Cited on page 93).



[194] Y. Yan, F. Hu, J. Chen, N. Bhendawade, T. Ye, Y. Gong, N. Duan, D. Cui, B. Chi, and R. Zhang, "FastSeq: Make Sequence Generation Faster," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, 2021, pp. 218–226 (Cited on page 93).

[195] Z. Li and S. Arora, "An Exponential Learning Rate Schedule for Deep Learning," in *8th International Conference on Learning Representations, ICLR 2020*, 2020 (Cited on page 93).

[196] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2009, pp. 248–255 (Cited on page 101).

[197] E. Agustsson and R. Timofte, "Ntire 2017 Challenge on Single Image Super-Resolution: Dataset and Study," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 126–135 (Cited on page 101).

[198] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," *Advances in Neural Information Processing Systems*, vol. 24, 2011 (Cited on pages 107, 108).

[199] J. Kaplan, "Notes on Contemporary Machine Learning for Physicists," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:121204748 (Cited on page 110).

[200] J. Boyce, K. Suehring, X. Li, and V. Seregin, *JVET-J1010: JVET Common Test Conditions and Software Reference Configurations*, Jul. 2018 (Cited on page 117).

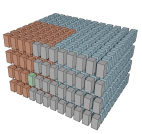
[201] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-Attention with Linear Complexity," *arXiv preprint arXiv:2006.04768*, 2020 (Cited on page 117).

[202] G. I. Winata, S. Cahyawijaya, Z. Lin, Z. Liu, and P. Fung, "Lightweight and Efficient End-to-End Speech Recognition using Low-Rank Transformer," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 6144–6148 (Cited on page 117).

[203] G. M. Correia, V. Niculae, and A. F. Martins, "Adaptively Sparse Transformers," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 2174–2184 (Cited on page 117).

[204] S. Jaszczur, A. Chowdhery, A. Mohiuddin, L. Kaiser, W. Gajewski, H. Michalewski, and J. Kanerva, "Sparse is Enough in Scaling Transformers," *Advances in Neural Information Processing Systems*, vol. 34, pp. 9895–9907, 2021 (Cited on page 117).

[205] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention," in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 5156–5165. [Online]. Available: https://proceedings.mlr.press/v119/katharopoulos20a.html (Cited on page 118).



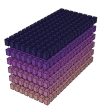
[206] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, *et al.*, "Rethinking Attention with Performers," in *International Conference on Learning Representations*, 2020 (Cited on page 118).

[207] F. Mentzer, E. Agustson, and M. Tschannen, "M2T: Masking Transformers Twice for Faster Decoding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 5340–5349 (Cited on page 118).

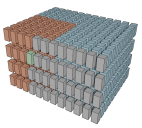
[208] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The Efficient Transformer," in *International Conference on Learning Representations*, 2019 (Cited on page 118).

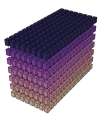
[209] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient Content-based Sparse Attention with Routing Transformers," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 53–68, 2021 (Cited on page 118).

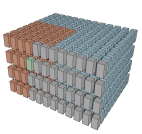
[210] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan, "Sparse Sinkhorn Attention," in *International Conference on Machine Learning*, PMLR, 2020, pp. 9438–9447 (Cited on page 118).

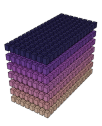


List of Figures

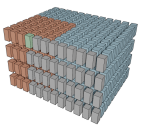


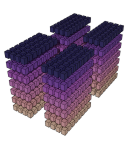




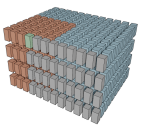


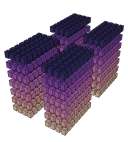
List of Tables





List of Algorithms





List of Commands