DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Question Generation and Answering in the Electrical Power System Components Domain

Aarav Malik

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Question Generation and Answering in the Electrical Power System Components Domain

Erstellung und Beantwortung von Fragen im Bereich der Komponenten elektrischer Energiesysteme

Author:	Aarav Malik
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Dr. Felix Dietrich (TUM) and Dr. Rakebul Hasan (Siemens)
Submission Date:	16/08/2022

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

[illegible]

Munich, 16/08/2022 Aarav Malik

Acknowledgments

I would like to thank Prof. Dr. Hans-Joachim Bungartz for giving me this opportunity to work on my thesis under his supervision.

I am very grateful to Dr. Felix Dietrich for being a great guide and an advisor. His support throughout the this work has extremely helped me structure it.

Without Dr. Rakebul Hasan this work would not have come to fruition. He helped me build this work by introducing me to various ideas and research approaches. He guided me whenever I was stuck and supported me to explore ideas.

Abstract

Obtaining training data for the Question Answering (QA) task is time-consuming and resource-intensive. While there are some domains for which such datasets exist, there are no such datasets for Electric Power System Components domain. Siemens has use cases where they can use QA models to extract relevant information from the manuals of such components. In this work, we explore the possibility of synthetically generating Question and Answer pairs using an unsupervised NMT model in a low resource setting. We approach this by building a paragraph corpus in Electric Power Systems Components domain. We use the UNMT model to generate context, question, and answer triples that make up our synthetic training dataset. UNMT model does so by randomly sampling paragraphs and then randomly sampling named entities or noun phrases as answers. It then masks the answers and turns them into "fill-in-the-blank" cloze questions, and finally, it translates them into a natural question. Then we fine-tune three state-of-the-art pre-trained transformer-based models on this synthetic training data for the downstream question answering task. We also curate a ground truth dataset of manually labeled question and answer pairs to evaluate our approach. QA models trained on synthetic training data answer natural questions quite well. With this approach, all three models achieved between 83.5 F1 and 85 F1 scores.

Contents

1 Introduction

Question Answering (QA) is a discipline in computer science concerned with building systems to answer questions posed by humans in a natural language. In the Information Retrieval discipline, a QA system uses a combination of techniques from computational linguistics, information retrieval, and knowledge representation to find answers. In Natural Language Processing (NLP) discipline, a QA system is built using models, such as BERT, which are then trained on the downstream task of question answering. In this work, we focus on extractive question answering; extractive QA focuses on extracting spans of answers in a given context. Extractive question answering systems are supervised systems, and a question answering system for a specific domain needs labeled training data for that specific domain.

Question-answering research has gained increasing momentum recently due to the emergence of transformer-based models such as BERT[Dev+18]. There has been a major upward momentum in their performance since September 2018, when the early transformer-based QA systems were published. A *transformer* is a deep learning model that adopts the attention mechanism. These models are first trained to learn the language (language models) and then trained on specific downstream tasks such as question answering, paragraph summarization, and more for a specific application.

Although these models are accessible, there are still challenges to their successful application. These models need labeled training data to train on the downstream question-answering task. It is expensive to annotate data with human annotators, and human annotators are required to have high domain knowledge.

A recent paper *Unsupervised Question Answering by Cloze Translation* [LDR19] demonstrates that training data can be generated in an unsupervised manner for Extractive QA. Our goal is to investigate unsupervised question generation method to generate labeled data for a specific domain. We subsequently produce pre-trained extractive question answering model for that domain which could be used for specific use cases.

2 State of the Art

In this chapter, we will review the previous research done on the problem of unsupervised question answering. We will also examine contemporary Language Models (LMs) that we use in our work.

2.1 Transformer

The transformer is a model architecture proposed by [Vas+17] that does away with recurrences and entirely relies on the attention mechanism to draw global dependencies between inputs and outputs. Before transformer models, RNNs were a popular model architecture choice for NLP tasks like machine translation. Recurrence-based architectures struggle with storing long-term dependencies in a sequence of input. The hidden state at every step depends on (usually) the most recent word in the sequence in the encoder. When the decoder accesses the last hidden state of the encoder, it loses most of the relevant information from the words at the start of the sequence. The attention mechanism helps deal with this challenge.

In the attention mechanism, at each step of the decoder, hidden state of all previous encoder steps is assigned a weighed sum; this allows the decoder to assign greater weight to certain elements of inputs for every output. Although attention is helpful, it is still computationally inefficient as both encoder and decoder have to wait for sequential computations at every step and cannot leverage parallelization. The transformer only uses a self-attention mechanism and extracts features for every word using other words in the sentence that are important to encode the word.

Figure 2.1 shows the model architecture of transformer. It has an encoder-decoder architecture. The encoder encodes the input sequence of symbol representation to continuous representation, and the decoder decodes this continuous representation to a symbol representation, one element at a time.

1. **Encoder-decoder stacks:** The model comprises an encoder and a decoder with six identical layers. Each layer is divided into two sub-layers, multi-head self-attention and a position-wise fully connected feed-forward network. The decoder

has an additional third sub-layer that performs multi-head attention over the output of the encoder stack.

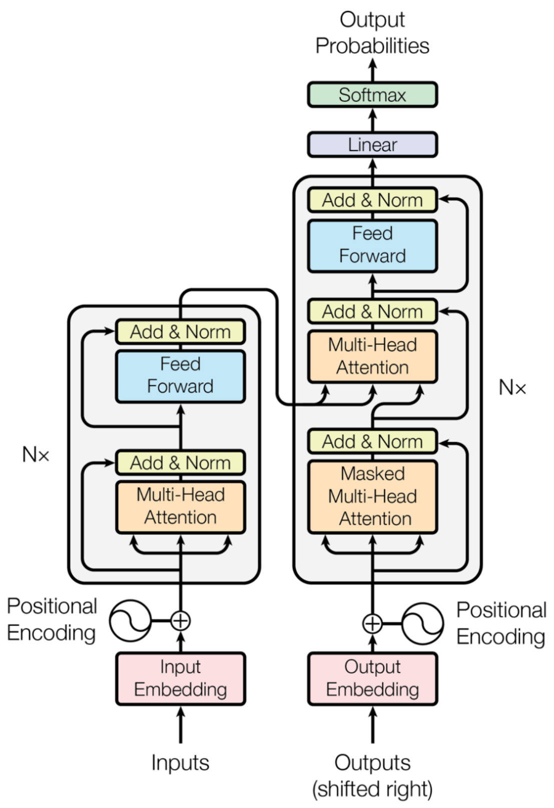


Figure 2.1: Transformer architecture. Encoder on the left and decoder on the right. Blocks of attention and feed forward networks N-times.

2. **Attention:** Attention is a mapping between query Q, key K, and value V to an output. Output is calculated as the weighed sum of values where the weight for a value is computed using the compatibility function on a query and its corresponding key.

   - **Scalar dot-product attention:** Q is the matrix of query vectors, K is the matrix of key vectors, and V is the matrix of value vectors. Attention for queries is calculated using the following equation. Queries and Keys have dimension d_k and values has dimension d_v.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

- **Multi-head attention:** At different positions, the model attends to information from different representation sub-spaces using multi-head attention.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.2}$$

and

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

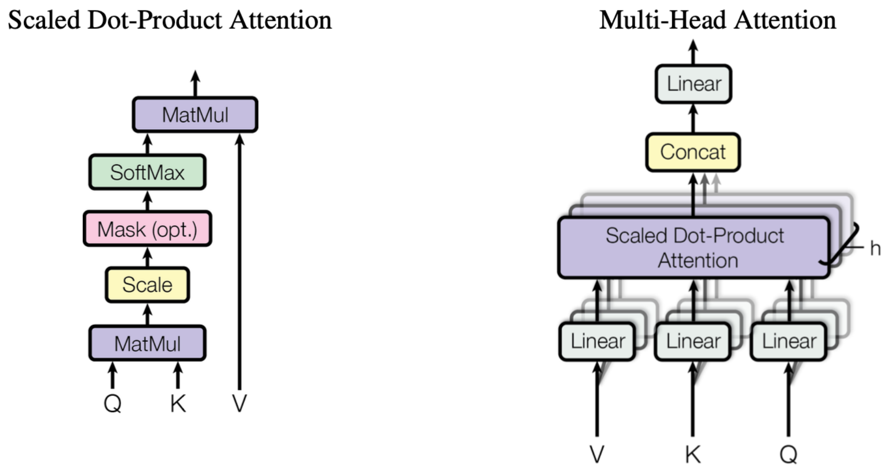


Figure 2.2: Scalar dot-product attention on the left. Multi-head attention on the right.

3. **Position-wise feed-forward network:** A feed-forward network (FNN) is applied to every position separately and identically, using two linear transformations and ReLU activation in between, in both encoder and decoder.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.4}$$

4. **Positional encoding:** Transformer only uses attention, so in order to store relation between sequence of tokens the authors of [Vas+17] use positional encodings.

2.2 BERT

BERT [Dev+18] stands for Bi-directional Encoder Representations from Transformer. BERT was developed by *Google AI Language* in 2018 with the idea to leverage the pre-trained language representation for downstream tasks like question answering, sentence classification, named entity recognition, and more. Then state-of-the-art, Generative Pre-trained Transformer (OpenAI GPT) [Rad+18] is made by stacking decoder from the transformer [Vas+17] and pre-training on the task to predict the next word in the sequence. It would take a sequence of words in the English language that makes it unidirectional (left to right).

BERT used a similar idea of pre-training like OpenAI GPT, but instead of training for the next word prediction task in a sequence of words, it masked the tokens and trained on the task of predicting masked tokens. This allows the transformer to leverage both directions (left to right and right to left) to understand the context and predict the masked tokens. This is referred to as Masked Language Modeling (MLM). As BERT only encodes the language, it only uses the encoder part of the original transformer and encodes the language by training for MLM task. Figure 2.3 shows the pre-training architectures for BERT and OpenAI GPT. BERT, besides training for MLM objective, is also trained on the task of predicting whether two sentences are consecutive or not (Next Sentence Prediction).

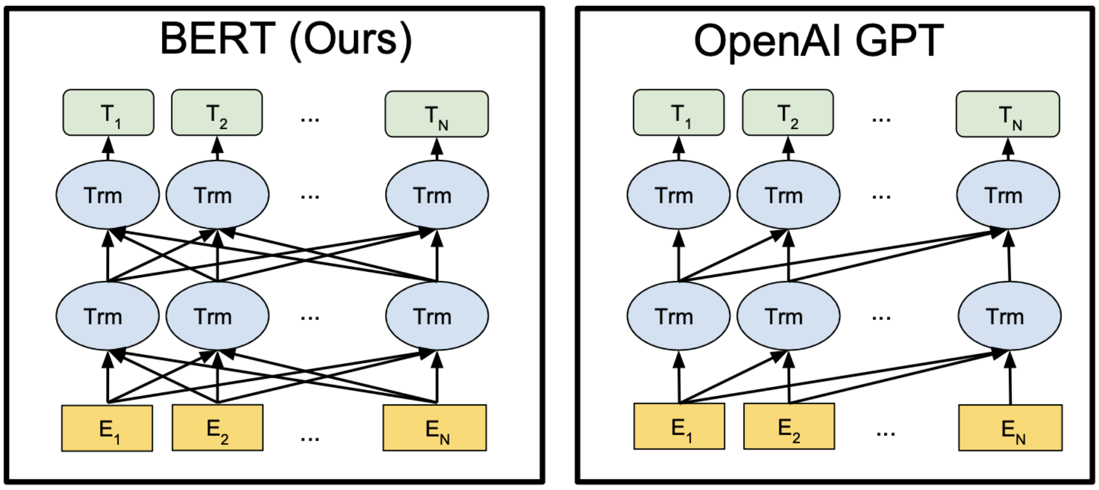


Figure 2.3: Pre-training model architectures for BERT and OpenAI GPT. BERT uses bi-directional transformer and OpenAi GPT uses left to right transformer.

Pre-training tasks are not very useful for both BERT and GPT. Their utility lies in fine-tuning them to the downstream tasks in NLP like question-answering, named entity recognition, and more. GPT, because of its unidirectional nature, is not able to utilize pre-trained embeddings. Training GPT for a specific downstream task requires updating pre-trained embeddings specific to the downstream task. On the other hand, BERT can be trained for a specific downstream task by just training the final layer. Figure 2.4 shows the training and fine-tuning procedure for BERT.

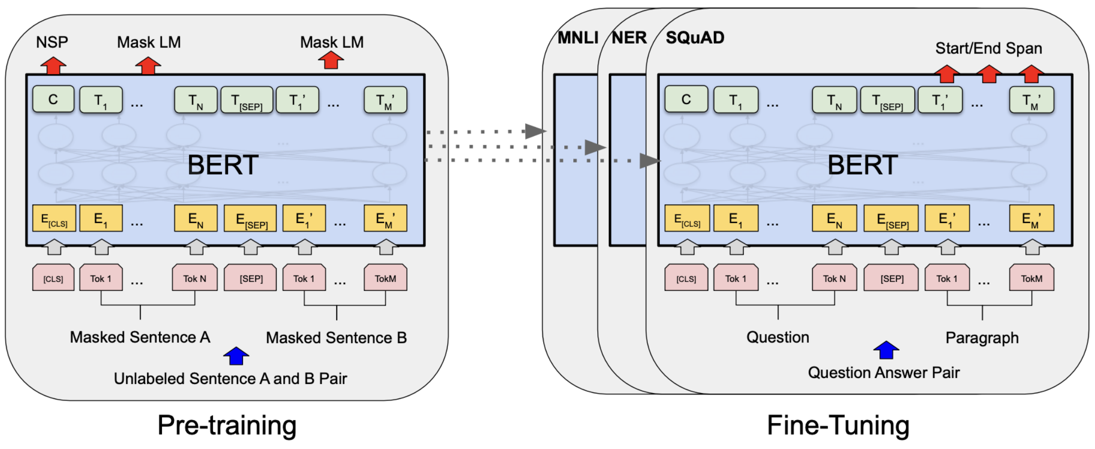


Figure 2.4: Bert pre-trainig and fine-tuning procedure. For both procedures same architecture is used, only the output layer is different.

2.3 RoBERTa

RoBERTa [Liu+19] stands for: A Robustly Optimized BERT Pre-training Approach. *Facebook AI* published it in 2019. Authors found that BERT [Dev+18] is significantly under-trained, so they proposed various updates to BERT's training procedure and called it RoBERTa. RoBERTa either matches or exceeds the performance of other post-BERT approaches.

Following are the five modifications in RoBERTa's approach:

1. **Data:** BERT is known to perform better when trained on larger datasets. [Bae+19] reported that using more training data can improve the performance on downstream NLP tasks. RoBERTa uses 160 GB of uncompressed data for training. RoBERTa uses the following four corpora in the English language of varying sizes and domains.
   a) **BookCorpus** [Zhu+15] and English **Wikipedia** (16 GB). This is the same dataset used for training BERT.
   b) **CC-News** (76 GB). This is English portion of *Common Crawl* data-set [Nag]. It has 63 M news articles crawled between September 2016 and February 2019.
   c) **OpenWebText** (38 GB). This is the data extracted from Reddit URLs with more than three upvotes.
   d) **Stories** (31 GB). It is a subset of CommonCrawl data after filtering to match the story-like style of Winograd schemas.

Table 2.1: Comparison between static and dynamic masking for $BERT_{BASE}$. F1 scores reported by authors for **SQuAD 2.0**, **MNLI-m** and **SST-2**. $RoBERTa_{DYNAMIC}$ performs comparable or slightly better compared to $BERT_{BASE}$.

Masking	**SQuAD 2.0**	**MNLI-m**	**SST-2**
$BERT_{BASE}$	76.3	84.3	92.8
$RoBERTa_{STATIC}$	78.3	84.3	92.5
$RoBERTa_{DYNAMIC}$	78.7	84.0	92.9

2. **Masking:** In BERT's MLM pre-training objective, tokens are masked randomly, and the model predicts the missing tokes. Masking is done only once in pre-processing. This results in *static masking* as the same mask is seen in every training

epoch. Authors of [Liu+19] applied ten different masks to the dataset, and each sentence was seen four times, i.e., the model was trained for 40 epochs. They also implemented *dynamic masking* by generating a random mask for the sentence before feeding it to the model. As shown in table 2.1, *dynamic masking* performs better, so RoBERTa uses dynamic masking.

3. **Next Sentence Prediction (NSP):** BERT model is trained on the objective of predicting whether two document segments belong to the same document (NSP Objective). During training, the model sees two contiguous document segments, either sampled from the same or different documents with equal probabilities. [Dev+18] emphasized that this training objective improved the performance on downstream NLP task while some research [CL19] [Yan+19] [Jos+20] points out that NSP objective is not necessary.

   Authors of [Liu+19] compared the following approaches and found that the NSP training objective is unnecessary for NLP downstream tasks with their implementation.

   a) SEGMENT-PAIR+NSP: This is the input format used in BERT [Dev+18]. Input consists of two segments, either from the same document or different documents. Each segment can have multiple full sentences. Input is bound in length by 512 tokens.
   b) SENTENCE-PAIR+NSP: Input is a pair of two sentences, either from a contiguous part of the same document or from a different document. This input is obviously less than 512 tokens (input length for BERT), so batch size is increased to match the input length of SEGMENT-PAIR+NSP. Model is trained with NSP objective.
   c) FULL-SENTENCES: Full sentences are sampled contiguously from one or more documents as input. At the end of the document, if there are less than 512 tokens, sentences are sampled from the next document. Training is done without the NSP objective.
   d) DOC-SENTENCES: Full sentences are sampled only from the same document. Batch size is adjusted dynamically to match the tokens used in FULL-SENTENCES. Training is done without the NSP objective.

   Table 2.2 shows the performances reported by authors of [Liu+19] in various settings. Authors find that between SEGMENT-PAIR+NSP and SENTENCE-PAIR+NSP, the model performs worse in individual sentence setting. This is due to the reason that model is unable to learn the long-term dependencies. Between FULL-SENTENCES and DOC-SENTENCES setting, performance on the downstream task is better for inputs where sentences do not cross the document boundaries. Removing the NSP objective in pre-training improves performance.

Table 2.2: Models' performance on various Data-sets. F1 scores reported for SQuAD 1.1/2.0 [Raj+16][RJL18] and accuracy scores for MNLI-m, SST-2[Soc+13] and RACE [Lai+17]. Results are medians over five random initializations (seeds).

Model	**SQuAD 1.1/2.0**	**MNLI-m**	**SST-2**	**RACE**
$RoBERTa_{SEGMENT-PAIR(withNSP)}$	90.4/78.7	84.0	92.9	64.2
$RoBERTa_{SENTENCE-PAIR(withNSP)}$	88.7/76.2	82.9	92.1	63.0
$RoBERTa_{FULL-SENTENCES(withoutNSP)}$	90.4/79.1	84.7	92.5	64.8
$RoBERTa_{DOC-SENTENCES(withoutNSP)}$	90.6/79.7	84.7	92.7	65.6
$BERT_{BASE}$	88.5/76.3	84.3	92.8	64.3

4. **Large Batch Size:** Work of [Ott+18] in neural machine translation has demonstrated that training with a large batch size can improve optimization speed and downstream task performance. Work of [You+19] has shown that BERT is responsive to batch sizes. When batch size increases, the training steps decrease. BERT was trained for 1M steps with a batch size of 256. As batch size increases, the number of steps required for training decreases. Suppose batch size is increased from 256 to 2k, then the steps decrease from 1M to 125k and for a batch size of 8k steps further reduce to 31. Large batch sizes are easy to parallelize via distributed data parallel training. In table 2.3 we see the comparison between perplexity and downstream task performance as reported by the authors for $BERT_{BASE}$.

Table 2.3: $BERT_{BASE}$ accuracy performance for various batch sizes (bsz). Learning rate is fine-tuned for every batch size.

bsz	**steps**	**lr**	**MNLI-m**	**SST-2**
256	1M	1e-4	84.7	92.7
2k	125K	7e-4	**85.2**	**92.9**
8k	31	1e-3	84.6	92.8

5. **Tokenization:** RoBERTa uses byte-level BPE (Byte Pair Encoding) vocabulary containing 50K sub-word units, whereas BERT uses a character-level BPE vocabulary of size 30K.

RoBERTa is trained with dynamic masking, large batch sizes, full sentences without NSP objective, larger training data, and uses byte-level BPE with a larger vocabulary. Table 2.4 shows the development set results reported by the authors in various settings.

Table 2.4: Devevelopment set results for RoBERTa. Training is done over more data ($16GB \rightarrow 160GB$) and done for longer time ($steps 100k \rightarrow 300k \rightarrow 500k$).

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
BERT$_{LARGE}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
XLNet$_{LARGE}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Compared to the other models, it is clear that training choices made by authors of [Liu+19] have a clear positive impact on the model's performance.

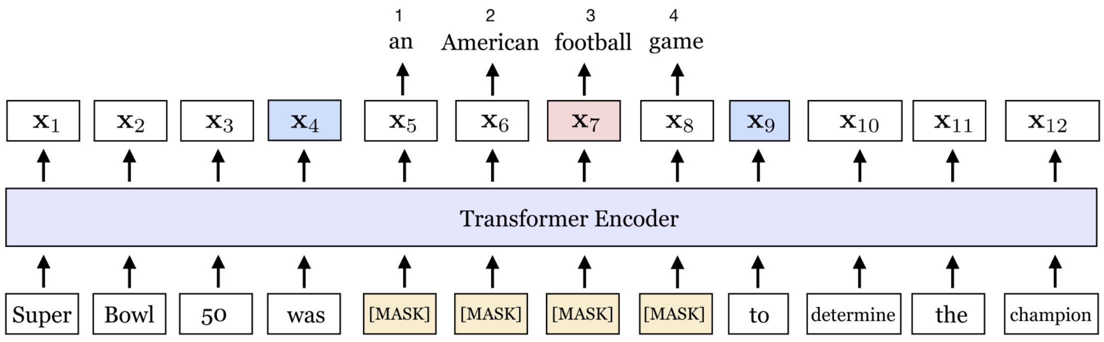
2.4 SpanBERT

SpanBERT [Jos+20] is a variant of BERT [Dev+18] with a pre-training method that better represents and predicts text spans. NLP downstream tasks like question answering require a better understanding of the relationship between one or more spans of words. To answer the question *"Which country has the highest GDP?"*, the model needs to understand the relationship that the answer, *"United States of America"*, is a country. It is easier to predict *America* when the previous known words are *United States of*.

Compared to BERT, SpanBERT uses a different masking scheme and training objectives. BERT masks random individual tokens, whereas, SpanBERT masks random contiguous spans of tokens. Also, SpanBERT uses a new training objective, *span boundary objective* (SBO), that is not used in training BERT. In SBO, the model learns to predict whole spans by just observing the boundary words. Figure 2.5 demonstrates this approach. SpanBERT is trained on only single segments instead of two segments with the NSP objective.

Figure 2.5: Illustration of training spanBERT. Model learns to predict the masked span *an American footbal game* by using x_4 and x_9 tokens (SBO loss). The equation shows the MLM and SBO loss to predict the term *football*.

$$\begin{aligned}\mathcal{L}(\text{football}) &= \mathcal{L}_{\text{MLM}}(\text{football}) + \mathcal{L}_{\text{SBO}}(\text{football})\\ &= -\log P(\text{football} \mid \mathbf{x}_7) - \log P(\text{football} \mid \mathbf{x}_4, \mathbf{x}_9, \mathbf{p}_3)\end{aligned}$$



- **Span Masking:** SpanBERT iteratively samples masked tokens till it fills its masking budget of 15%. At every iteration, span length is sampled from a geometric distribution skewed towards smaller span lengths, and then a random position is selected from the text for starting position of the mask. Like BERT, SpanBERT also replaces 80% of masked tokens with [MASK], 10% with random tokens, and 10% with original tokens. SpanBERT does this at span level, so all tokens in span are either replaced with [MASK] or random tokens.

- **Span Boundary Objective:** Span selection models usually create fixed length spans using boundary tokens [Lee+16][He+18]. Authors approach this issue by storing the maximum content of internal span in the boundary tokens. This is done using span boundary objective where each token of the masked span is predicted using boundary token.
- **Single-Sequence Training:** Authors find that single sequence input compared to BERT's two sequences with NSP objective performs better, so they train SpanBERT with single sequences from the same document without NSP objective.

Table 2.5: Test results of SpanBERT and other baselines on SQuAD1.1 and SQuAD2.0

	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
Human Perf.	82.3	91.2	86.8	89.4
Google BERT	84.3	91.3	80.0	83.3
Our BERT	86.5	92.6	82.8	85.9
Our BERT-1seq	87.5	93.3	83.8	86.6
SpanBERT	**88.8**	**94.6**	**85.7**	**88.7**

As seen in table 2.5, SpanBERT beats all the baseline models on extractive question answering task on both SQuAD 1.1 and SQuAD 2.0. Authors also found that their training approach resulted in similar gains on other benchmark datasets like NewsQA, TriviaQA, SearchQA, HotpotQA, and Natural Questions.

2.5 MiniLM

Microsoft Research published a model, MiniLM [Wan+20] in 2020, that works by mimicking the self-attention module of a large transformer-based Language Model (LM), also referred to as teacher model. LMs have shown noteworthy success in various NLP downstream tasks [HR18] [Rad+18] [Dev+18] [Jos+20] like question answering, text summarization etc. Pre-trained LMs, like BERT, learn the contextualized representations by predicting missing tokens; next, they are fine-tuned for a downstream task with a task-specific layer. The size of these models is large such that they contain hundreds of millions of parameters, making fine-tuning them challenging. This makes it difficult to deploy these models online for real-life applications due to latency and capacity constraints.

Authors of [Wan+20] adopt an approach of distillation where a small student model is trained to compress a large pre-trained LM (teacher model) [Agu+20] [Sun+19a] [MA20]. This results in a smaller number of parameters with performance still being competitive on the downstream NLP tasks. First, the LM is trained on a specific downstream task, and then the student model is trained through distillation. This is task-specific distillation. This is still expensive because it requires fine-tuning a large LM on a specific task.

Table 2.6: MiniLM comparison with other task agnostic distillation approaches.

Approach	Teacher Model	Distilled Knowledge	Layer-to-Layer Distillation	Requirements on the number of layers of students	Requirements on the hidden size of students
DistilBERT	$\text{BERT}_{\text{BASE}}$	Soft target probabilities Embedding outputs			✓
TinyBERT	$\text{BERT}_{\text{BASE}}$	Embedding outputs Hidden states Self-Attention distributions	✓		
MOBILEBERT	$\text{IB-BERT}_{\text{LARGE}}$	Soft target probabilities Hidden states Self-Attention distributions	✓	✓	✓
MINILM	$\text{BERT}_{\text{BASE}}$	Self-Attention distributions Self-Attention value relation			

Authors of [Wan+20] take a different approach. They distill the LM itself and then fine-tune the student model on a specific downstream task. This is task agnostic distillation. In similar approaches (DistilBERT [MA20], TinyBERT [Jia+19] and *MOBILE*BERT [Sun+19b]), as shown in table 2.6 student model do not have flexibility in terms of architecture or number of layers. Instead, MiniLM is trained to deeply mimic the self-attention modules of the teacher model, specifically the self-attention modules of the last transformer layer. The challenge of doing layer-to-layer mapping between the student and teacher model is resolved by mimicking only the last transformer layer. This also allows the student model to have a flexible number of layers. In addition to attention distribution, MiniLM uses scaled dot-product between values in the self-attention module as the new deep self-attention knowledge. MiniLM also shows that using a teacher assistant model helps with distillation when the size difference between student and teacher is large.

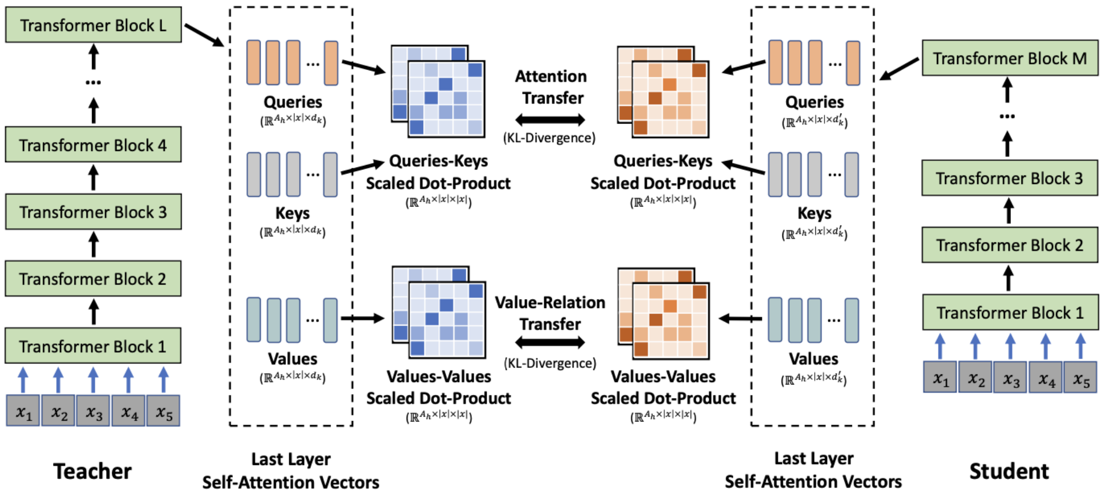

Transformer Block L
...
Transformer Block 4
Transformer Block 3
Transformer Block 2
Transformer Block 1
x_1 x_2 x_3 x_4 x_5
Teacher
Queries ($\mathbb{R}^{A_h \times |x| \times d_k}$)
Keys ($\mathbb{R}^{A_h \times |x| \times d_k}$)
Values ($\mathbb{R}^{A_h \times |x| \times d_k}$)
Last Layer Self-Attention Vectors
Queries-Keys Scaled Dot-Product ($\mathbb{R}^{A_h \times |x| \times |x|}$)
Values-Values Scaled Dot-Product ($\mathbb{R}^{A_h \times |x| \times |x|}$)
Attention Transfer (KL-Divergence)
Value-Relation Transfer (KL-Divergence)
Queries-Keys Scaled Dot-Product ($\mathbb{R}^{A_h \times |x| \times |x|}$)
Values-Values Scaled Dot-Product ($\mathbb{R}^{A_h \times |x| \times |x|}$)
Queries ($\mathbb{R}^{A_h \times |x| \times d'_k}$)
Keys ($\mathbb{R}^{A_h \times |x| \times d'_k}$)
Values ($\mathbb{R}^{A_h \times |x| \times d'_k}$)
Last Layer Self-Attention Vectors
Transformer Block M
...
Transformer Block 3
Transformer Block 2
Transformer Block 1
x_1 x_2 x_3 x_4 x_5
Student

Figure 2.6: Deep self-attention distillation. MiniLM is trained to mimic self-attention module of last transformer layer of teacher model. Authors also use self-attention value-relation transfer to achieve deeper mimickry.

Figure 2.6 shows the overview of deep self-attention distillation. Authors introduce three key ideas:

1. **Self-Attention Distribution Transfer:** Self-attention module on the last layer of the teacher transformer model is an important component. [JSS19] shows that the last layers of BERT encode semantic features and capture long-distance dependency knowledge that is important for downstream NLP tasks. Like [Jia+19] and [Sun+19b], MiniLM uses self-attention distributions in student training. It minimizes the KL-divergence between self-attention distributions of student and

teacher.

$$\mathcal{L}_{AT} = \frac{1}{A_h|x|} \sum_{a=1}^{A_h} \sum_{t=1}^{|x|} D_{KL}(\mathbf{A}_{L,a,t}^{T} || \mathbf{A}_{M,a,t}^{S}) \tag{2.5}$$

$|x|$ represents the sequence length, A_h is the number of attention heads, L represents the number of teacher layers, M is the number of student layers, $\mathbf{A}_L^T$ and $\mathbf{A}_M^S$ are attention distributions of last transformer layers of teacher and student model respectively. They are computed by a scaled dot product of queries and keys.

Distilling just the last transformer layer allows flexibility in terms of layer mapping and the number of layers.

2. **Self-Attention Value-Relation Transfer:** Queries, keys, and values are three vital vectors that comprise the self-attention module. Queries and keys are transferred through attention distributions, but authors achieve deeper mimicry by transferring value relation. Value relation is computed by a multi-head scaled dot product between values. KL-divergence between the teacher and student model uses value relation the training objective.

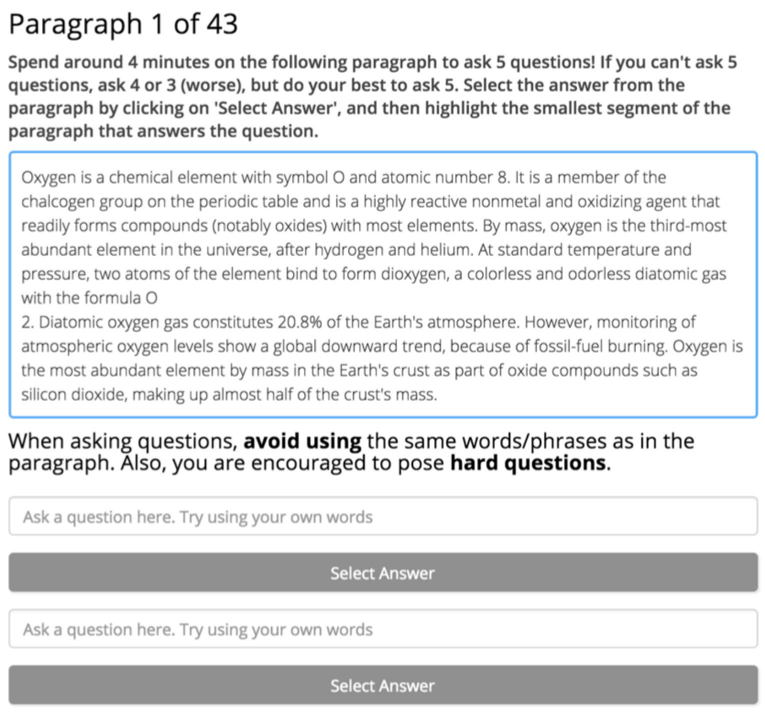
$$\mathbf{VR}_{L,a}^{T} = softmax(\frac{\mathbf{V}_{L,a}^{T} \mathbf{V}_{L,a}^{T\mathbf{T}}}{\sqrt{d_k}}), \mathbf{VR}_{M,a}^{S} = softmax(\frac{\mathbf{V}_{M,a}^{S} \mathbf{V}_{M,a}^{S\mathbf{T}}}{\sqrt{d_k'}}) \tag{2.6}$$

$$\mathcal{L}_{AT} = \frac{1}{A_h|x|} \sum_{a=1}^{A_h} \sum_{t=1}^{|x|} D_{KL}(\mathbf{VR}_{L,a,t}^{T} || \mathbf{VR}_{M,a,t}^{S}) \tag{2.7}$$

3. **Teacher Assistant:** Authors use a teacher assistant [Mir+20]. It acts as a bridge between the teacher and the final student. Teacher with L layers and d_h hidden size is first used to train a student model with M layers and d_h' hidden size ($M <= \frac{1}{2}L$ and $d_h' <= \frac{1}{2}d_h$). Then a final student model is trained using this student model as teacher.

2.6 SQuAD

SQuAD [Raj+16] stands for Stanford Question Answering Dataset. It is a collection of 107,785 question and answer pairs from 536 Wikipedia articles that have been crowd-sourced. SQuAD has been developed for reading comprehension task, other datasets for same task fall short either due to their size or the quality of their questions [RBR13] [Ber+14a] [Ber+14b] [Hil+15]. Unlike other datasets, SQuAD does not provide answer choices for the question. A system learns to find the correct span from all possible spans in the context, thus increasing the complexity of the dataset. Although choosing answer spans restricts the type of questions that can be answered, SQuAD still shows a wide variety of question and answer types.


Paragraph 1 of 43

Spend around 4 minutes on the following paragraph to ask 5 questions! If you can't ask 5 questions, ask 4 or 3 (worse), but do your best to ask 5. Select the answer from the paragraph by clicking on 'Select Answer', and then highlight the smallest segment of the paragraph that answers the question.

Oxygen is a chemical element with symbol O and atomic number 8. It is a member of the chalcogen group on the periodic table and is a highly reactive nonmetal and oxidizing agent that readily forms compounds (notably oxides) with most elements. By mass, oxygen is the third-most abundant element in the universe, after hydrogen and helium. At standard temperature and pressure, two atoms of the element bind to form dioxygen, a colorless and odorless diatomic gas with the formula O
2. Diatomic oxygen gas constitutes 20.8% of the Earth's atmosphere. However, monitoring of atmospheric oxygen levels show a global downward trend, because of fossil-fuel burning. Oxygen is the most abundant element by mass in the Earth's crust as part of oxide compounds such as silicon dioxide, making up almost half of the crust's mass.

When asking questions, **avoid using** the same words/phrases as in the paragraph. Also, you are encouraged to pose **hard questions**.

Ask a question here. Try using your own words
Select Answer
Ask a question here. Try using your own words
Select Answer

Figure 2.7: Crowd facing web-interface to collect questions and answers. Workers are encouraged to make question in their own words.

SQuAD was curated in three stages: passage curation, question and answer collection and additional answer collection:

1. **Passage Curation:** Passages were collected from English Wikipedia. 536 articles were chosen at random from the top 1000 articles. Articles varied in the topics from musical celebrities to abstract concepts. 22,315 paragraphs were extracted and paragraphs under 500 characters were rejected. Paragraphs are split in 80-10-10 ratio as training, development, and test set, respectively.

2. **Question and Answer Collection:** Crowdworkers only from Canada and USA were selected with a 97% HIT acceptance rate and at least 1000 HITs. Corwdworkers were tasked to spend 4 minutes on every paragraph and create 5 questions for every paragraph (at least 3 questions in case it was hard to form questions). Figure 2.7 shows the the crowd facing web-interface. Workers had a sample paragraph with a set of good and bad questions and the reason for their categorization for guidance. They were encouraged to frame questions in their own words and enter them in the text field, and answers were highlighted in the paragraph.

3. **Additional Answer Collection:** Two additional answers were generated for questions from the development test and test set paragraphs. Workers were shown the question alongside paragraphs and were asked to mark the shortest span from the paragraph. They were recommended 2 minutes for 5 questions. 2.6% of the questions were marked unanswerable by at least one worker. This also gave a benchmark of human performance on the data.

Authors analyzed three criteria to analyze SQuAD, (i) diversity of answer types, (ii) the difficulty of questions in terms of the type of reasoning required to answer them, and (iii) the degree of syntactic divergence between the question and answer sentences.

1. **Diversity of answer types:** Answers are automatically categorized. First numeric and non-numeric answers are split, then non-numeric answers are tagged using POS tags using Stanford CoreNLP. Finally, nouns are tagged using NER (Named Entity Recognition) tags. Table 2.7 shoes the distribution of answer types.

2. **Reasoning required to answer questions:** Authors sampled 4 questions from all the 48 articles in the development set and manually categorized the questions and answers based on syntactic or lexical divergence as shown in table 2.8.

3. **Stratification of syntactic divergence:** Authors use another automatic method to measure the difficulty of questions and use it to stratify the dataset. They find the anchor word that is present in both question and answer. They pick two unlexicalized paths, one from anchor to *wh** word and another from anchor to answer, and then calculate the edit distance between the two paths. Syntactic

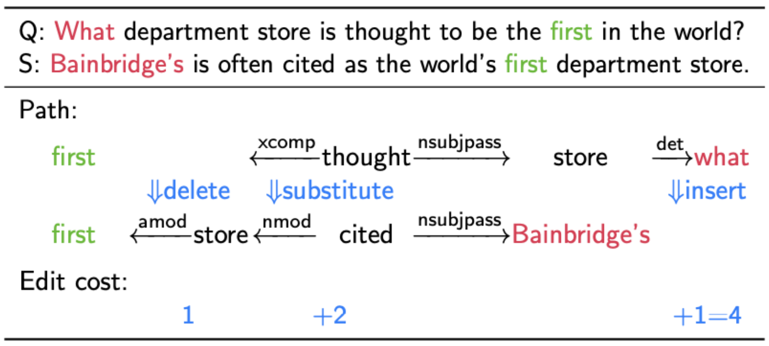
Table 2.7: Distribution of answer types in SQuAD dataset where answers are categorized automatically using POS and NER tags.

Answer type	Percentage	Example
Date	8.9%	19 October 1512
Other Numeric	10.9%	12
Person	12.9%	Thomas Coke
Location	4.4%	Germany
Other Entity	15.3%	ABC Sports
Common Noun Phrase	31.8%	property damage
Adjective Phrase	3.9%	second-largest
Verb Phrase	5.5%	returned to Earth
Clause	3.7%	to avoid trivialization
Other	2.7%	quietly

Table 2.8: Manually categorized development set SQuAD questions. Crowd sourced answers are underligned and words relevant to reason type are bolded.

Reasoning	Description	Example	Percentage
Lexical variation (synonymy)	Major correspondences between the question and the answer sentence are synonyms.	Q: What is the Rankine cycle sometimes **called**? Sentence: The Rankine cycle is sometimes **referred** to as a practical Carnot cycle.	33.3%
Lexical variation (world knowledge)	Major correspondences between the question and the answer sentence require world knowledge to resolve.	Q: Which **governing bodies** have veto power? Sen.: **The European Parliament and the Council of the European Union** have powers of amendment and veto during the legislative process.	9.1%
Syntactic variation	After the question is paraphrased into declarative form, its syntactic dependency structure does not match that of the answer sentence even after local modifications.	Q: What Shakespeare scholar **is currently on the faculty**? Sen.: **Current faculty include** the anthropologist Marshall Sahlins, ..., Shakespeare scholar David Bevington.	64.1%
Multiple sentence reasoning	There is anaphora, or higher-level fusion of multiple sentences is required.	Q: What collection does **the V&A Theatre & Performance galleries** hold? Sen.: **The V&A Theatre & Performance galleries** opened in March 2009. ... **They** hold the UK's biggest national collection of material about live performance.	13.6%
Ambiguous	We don't agree with the crowd-workers' answer, or the question does not have a unique answer.	Q: What is the main goal of criminal punishment? Sen.: **Achieving crime control via incapacitation and deterrence** is a major goal of criminal punishment.	6.1%

divergence is defined as minimum edit distance over all anchors. Figure 2.8 shows an example of how the edit distance is calculated for an anchor.


Q: What department store is thought to be the first in the world?
S: Bainbridge's is often cited as the world's first department store.

Path:
first $\xleftarrow{\text{xcomp}}$ thought $\xrightarrow{\text{nsubjpass}}$ store $\xrightarrow{\text{det}}$ what
⇓delete ⇓substitute ⇓insert
first $\xleftarrow{\text{amod}}$ store $\xleftarrow{\text{nmod}}$ cited $\xrightarrow{\text{nsubjpass}}$ Bainbridge's
Edit cost:
1 +2 +1=4

Figure 2.8: Example of computing edit distance between question and answer for syntactic divergence.

2.7 Unsupervised Question Answering by Cloze Translation

Extractive question answering is a task to answer a question assuming that the answer can be found in the given context paragraph. There has been substantial progress in this task in NLP (Natural Language Processing). BERT [Dev+18] based ensemble models already beat the human performance on benchmark data-sets like SQuAD [Raj+16]. Training these models requires large training data, only available in a few domains or languages. Authors of *Unsupervised Question Answering by Cloze Translation*[LDR19] came up with an unsupervised approach to train question answering model using the methods of cloze translation and techniques used in language translation models - like French to English. They leverage the existing models, their architectures, and pre-training routines, making this approach flexible and scalable for various use cases.

Figure 2.9 represents the schematic of unsupervised question answering by cloze translation approach. The approach works in four steps.

1. Step 1 - Randomly sample a context paragraph from a certain domain.
2. Step 2 - Randomly sample candidate answers from the selected paragraph in the previous step using pre-trained systems like NER (Named Entity Recognition) or non-chunkers.
3. Step 3 - Extract the cloze question (cloze question is nothing but a statement with the answer masked).
4. Step 4 - Translate the cloze question to a natural language question using the cloze-to-natural language translator model that has been trained in an unsupervised way.

Translating a cloze question to a natural question is the most important and difficult task of this approach. Rule-based approaches exist in the English language to convert statements to questions. Authors tested the rule-based approach of [HS10] and reported a weaker performance compared to their approach. Also, building such a system requires considerable engineering effort. Supervised systems like [DSC17],[DC18] and [HR19] for this task exist, but they require parallel question and answer corpus for training, and such data is not available in an unsupervised learning setting. Authors borrow the ideas from [Lam+18], [CL19] and [ALA18] to convert cloze questions to natural questions. It uses translation techniques similar to the one used for translating one language to another, like French to English.

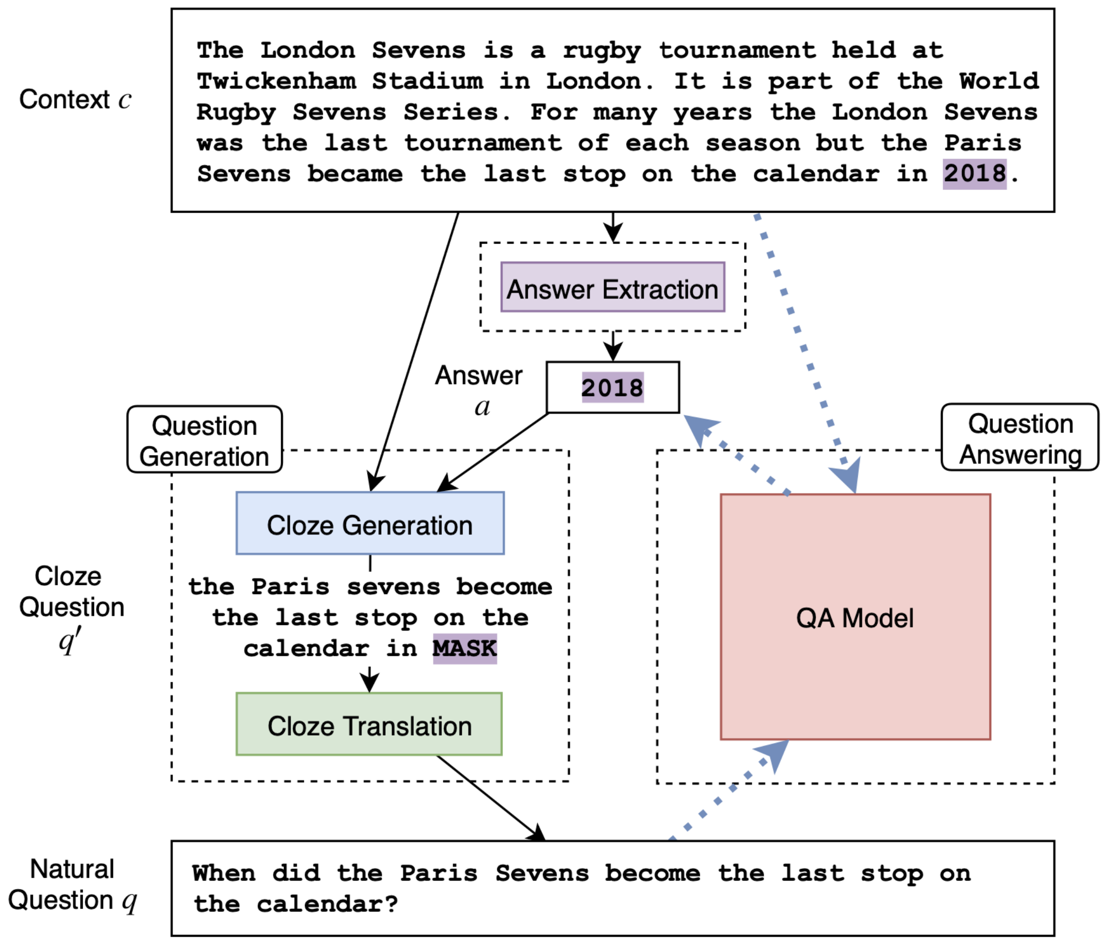

Context c
The London Sevens is a rugby tournament held at Twickenham Stadium in London. It is part of the World Rugby Sevens Series. For many years the London Sevens was the last tournament of each season but the Paris Sevens became the last stop on the calendar in 2018.
Answer Extraction
Answer a
2018
Question Generation
Question Answering
Cloze Generation
Cloze Question q'
the Paris sevens become the last stop on the calendar in MASK
Cloze Translation
QA Model
Natural Question q
When did the Paris Sevens become the last stop on the calendar?

Figure 2.9: Schematic of Unsupervised Question answering using cloze translation. Right side of the image with dotted arrows represents traditional extractive question answering. Left side of the image with solid arrows represents the process of generating unsupervised training data, which is used to train the extractive question answering model.

Extractive question answering models are expected to find answer span $a = (b, e)$ given the question q and context c (here b and e represent the start and end position of answer in the context c). Authors of [LDR19] model the problem of unsupervised extractive question answering in two stages. The first stage is the generative model $p(q, a, c)$ that generates the question-answer pairs, and the second stage is training a discriminative model $p(a|q, c)$ using the training data generated by p. Generator $p(q, a, c) = p(c)p(a|c)p(q|a, c)$ generates the training data in reverse direction. Context is sampled using $p(c)$, then answer is sampled using the model $p(a|c)$ and finally a natural question is generated using the model $p(q|a, c)$.

2.7.1 Context and Answer Generation

Model (context generator) $p(c)$ samples a context paragraph uniformly from the given corpus of documents from a specific domain, and model (answer generator) $p(a|c)$ samples answers from the sampled paragraph. Model $p(a|c)$ incorporates any prior belief of what makes up a good answer. The answer can be sampled in the following ways:

- **Noun Phrase (NP):** Model extracts all the noun phrases as possible answers from the sampled paragraph c and samples answer a from the extracted list of possible answers.
- **Named Entities (NE):** NER (Named Entity Recognition) system selects the possible answer candidates, and then answer a is sampled uniformly from the possible answer candidates.

2.7.2 Question Generation

Model (question generator) $p(q|a, c)$ generates the question from sampled context c and answer a. It can be modeled as $p(q|a, c) = p(q|q')$ where $q' = cloze(a, c)$.

Cloze Generation

When we mask the answer in a statement, it represents a cloze. To generate a cloze question from a paragraph, $q' = cloze(a, c)$, first step is to reduce the scope of context. In extractive question answering, answers are spans of a few words. By reducing scope, the idea is to match the detail of context to the detail of the question that needs to be generated. Sentence boundary represents a natural choice for this. The scope can be further reduced to a sub-clause around the answer as well. Figure 2.10 and figure 2.11 demonstrate cloze generation for the example from figure 2.9.

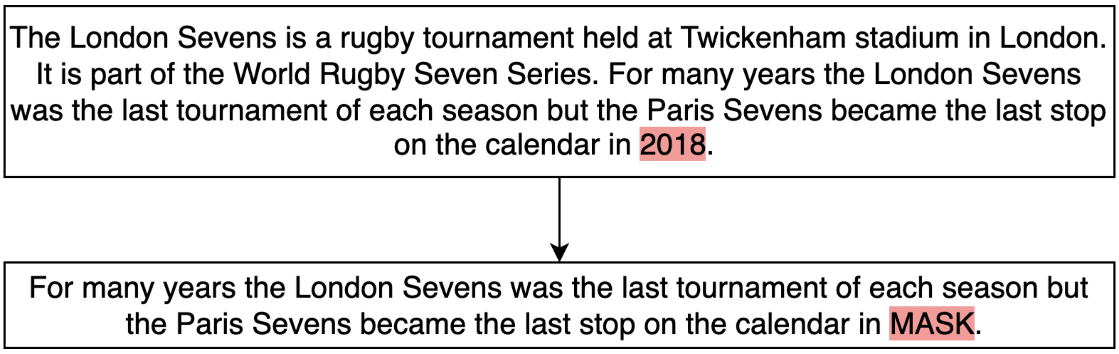

The London Sevens is a rugby tournament held at Twickenham stadium in London. It is part of the World Rugby Seven Series. For many years the London Sevens was the last tournament of each season but the Paris Sevens became the last stop on the calendar in 2018.
For many years the London Sevens was the last tournament of each season but the Paris Sevens became the last stop on the calendar in MASK.

Figure 2.10: Cloze generation with sentence boundary. Scope of the context is reduced to the whole sentence around the answer.

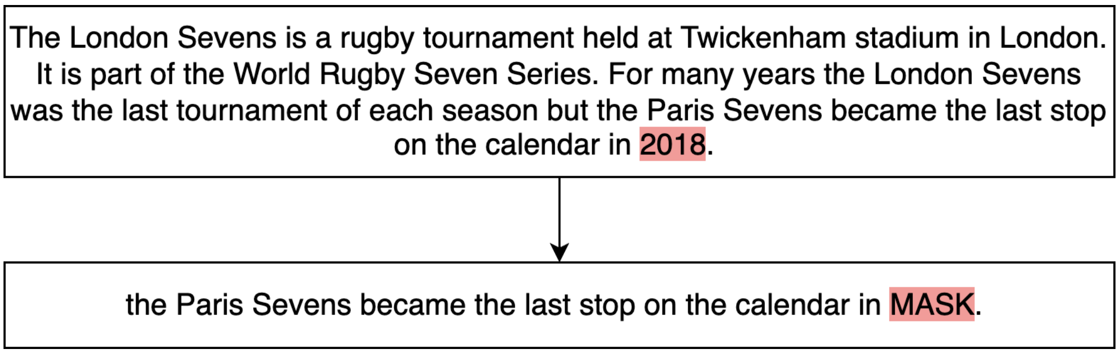

The London Sevens is a rugby tournament held at Twickenham stadium in London. It is part of the World Rugby Seven Series. For many years the London Sevens was the last tournament of each season but the Paris Sevens became the last stop on the calendar in 2018.
the Paris Sevens became the last stop on the calendar in MASK.

Figure 2.11: Cloze generation with sub-clause boundary. Scope of the context is reduced to the sub-clause around the answer

Cloze Translation

To translate cloze question q' to a natural question q, authors of [LDR19] explored four approaches.

1. **Identity Mapping:** The idea here is that the cloze question itself provides some signal to learn the question answer behavior. The masked token (place holder for answer) is replaced with a *wh** word. The *wh** word can either be randomly chosen, or it can be chosen with heuristics.

2. **Noisy Cloze:** The relationship between the cloze question q' and the natural

question *q* can be seen as some form of perturbation. Noisy cloze is generated in four steps:

a) Remove the mask token from the cloze.

b) Apply a noise function, like one in [Lam+18], to the cloze. The noise function includes word dropout or word-reorder, or both.

c) Add a *wh** word in the front to match the natural question syntax as most natural questions begin with a *wh** word.

d) Add a '?' at the end as in the English language, all questions must end with a '?'.

As evident, this approach does not produce natural sounding questions.

3. **Rule Based:** In the rule-based approach, a system uses a set of rules designed for a specific language to generate a list of questions and order them by their ranks to select the best candidate. Rules comprise syntactic transformations, choosing fitting *wh** word and its movement within the cloze. Authors used statement-to-question generator from [HS10] for their work. This approach requires lots of engineering effort as one needs to build a system that adheres to rules specific to a language.

4. **Seq2Seq:** This approach can be implemented by training a seq2seq model in an unsupervised way. While this approach does not produce perfect questions, it is much cheaper as it does not require significant engineering effort like the rule-based approach, and it produces much better natural questions compared to *noisy cloze* and *identity mapping*.

2.7.3 Unsupervised Cloze Translation

Unsupervised cloze translation is based on the recent work of Unsupervised Neural Machine Translation in the field of NLP. A translation model is trained between non-parallel corpora of source and target language sentences [Lam+18]. In [LDR19], authors use cloze corpus as source language sentence and question corpus as target language sentence and train a model to learn a mapping between the two corpora.

- **Cloze Corpus**

  The cloze corpus C was prepared by randomly sampling Wikipedia paragraphs and using the cloze generation approach in section 2.7.2. The authors chose

Named Entities (NE) and Noun Phrases (NP) as answer spans. Clozes were either generated using the Sentence (SE) boundary or Sub-clause (SC) boundary. Also, type-specific mask tokens were used to mask the answer spans *(PERSON/ORG/NORP, THING, TEMPORAL, NUMERIC, PLACE)* when named entity mentions were chosen as answers. Cloze corpus of 5M was generated this way.

- **Question Corpus**

  To generate question corpus questions were collected from *Common Crawl* with the following selection criteria:

  1. Questions must start with one of the *wh** word, *how much/many, what, when, where* and *who*.
  2. Question must end in a single '?'.
  3. Question must be less than 20 tokens.

  100M deduplicated questions were collected this way and 5M question were randomly chosen to curate the question corpus Q.

Similar to [Lam+18], authors train two models $p_{s\to t}(q|q')$ and $p_{t\to s}(q'|q)$ to translate cloze to natural question and natural question to cloze question, respectively. This is done by in-domain and cross-domain training. In-domain training is done using denoising-autoencoding, and cross-domain training is done using online-back-translation. The natural questions are chosen using $argmax_q p_{s\to t}(q|q')$ at inference.

Wh* Heuristics

To assign appropriate *wh** word a mapping of MASK TOKEN to *wh** word is used. *PERSON/ORG/NORP* is mapped to *who*, *TEMPORAL* to *when*, *NUMERIC* to *how much/many*, *THING* to *what* and *PLACE* to *where*.

Table 2.9 are some examples of close translations generated using Unsupervised Neural Machine Translation (UNMT) model. The model generates well-formed questions. Authors reported that 68% of translations generated by the UNMT model were classified as well-formed; this number rose to 7.5% when pre-training was used for the language model. This performance is significant compared to 75.6% classified as well-formed questions for rule-based question generator and 92.3% of SQuAD questions

Table 2.9: Examples of cloze translations generated using Unsupervised Neural Machine Translated (UNMT) model. These sample are generated using *wh** heuristic and sub-clause cloze generation.

#	Cloze Question	Answer	Generated Question
1	they joined with PERSON/NORP/ORG to defeat him	Rom	Who did they join with to defeat him?
2	the NUMERIC on Orchard Street remained open until 2009	second	How much longer did Orchard Street remain open until 2009?
3	making it the third largest football ground in PLACE	Portugal	Where is it making the third football ground?
4	he speaks THING, English, and German	Spanish	What are we , English , and German?
5	Arriving in the colony early in TEMPORAL	1883	When are you in the colony early?
6	The average household size was NUMERIC	2.30	How much does a Environmental Engineering Technician II in Suffolk , CA make?
7	WALA would be sold to the Des Moines-based PERSON/NORP/ORG for $86 million	Meredith Corp	Who would buy the WALA Des Moines-based for $86 million?

classified as well-formed. This signals that the synthetic data generated is of good quality. We use the UNMT model in our work to generate the synthetic data.

[LDR19] approach achieves a 56.4% F1 on SQuAD and the performance increases to 64.5% F1 when it is calculated on a subset of SQuAD where the answer is a named entity mention.

2.8 Summary

In this chapter, we reviewed the transformer architecture and attention mechanism. We learned about architecture and training approaches for various state-of-the-art transformer-based models. We discussed unsupervised question answering using cloze translation. We learned how unsupervised question answering addresses the lack of training data by generating synthetic data to train transformer-based models for the question answering task. While there is still scope for improving the quality of generated questions for future work, we notice that the quality of synthetic training data is good enough to teach transformer models the required question-answering behavior.

3 Question Generation and Answering in the Electrical Power System Components Domain

The goal of our task here is, given a question and context in the Electrical Power System Components (EPSC) domain, to retrieve the correct answer. As our task description is the same as extractive question answering (EQA) with an additional constraint on context and question to belong from a specific domain, one might think to replicate [LDR19]'s UNMT model approach for electrical power system component domain. Although the UNMT model is trained in an unsupervised way, i.e., no aligned question and answer pairs were used for training, it still requires unaligned cloze corpus and question corpus. While cloze corpus is straightforward to build as described in section 2.7.2, building a question corpus is resource intensive task. There are no publicly available datasets in this domain that could be leveraged for the question-answering task.

In this chapter, we will first discuss the specific use cases at Siemens, where the work of this thesis was performed. We will then discuss the choice of data and its processing. Further, we elaborate on our experiments and methodologies. Next, we will discuss a dataset we manually prepared as ground truth data for the final evaluation of our trained models. Finally, we will discuss the results and future work.

3.1 Use Case

Answering questions from users in an enterprise domain is a challenging task. Businesses rely increasingly on automated systems for customer, or technical support [Rou19]. Historically, the task of answering questions has been performed by building complex information retrieval systems. The establishment cost of these systems is high, and a user is expected to have some IT knowledge to run and interact with such systems. Also, the quality of answers retrieved depends on the quality of query user inputs in the system; users cannot interact and ask questions in their natural language.

When the context paragraphs are available and the question is given, users can read the context and look for an answer. This approach only works if the given context paragraph is small. Even though this works for some use cases, this approach is slow and requires manual effort. It does not scale when trying to find answers to more than a few questions.

We can distinguish the type of question-answering problems based on the characteristics of both questions and answers. Short questions (up to a dozen words long) differ from the long form of questions, with word lengths ranging from 10 to 50 or more words. We can also categorize based on answer lengths. Answers can be one to a few words long contiguous spans from the provided context. This is factoid-based question answering. Answers can also be long, non-contiguous, and spanning over multiple paragraphs. In our work, we focus on factoid-based question answering. We can see examples of factoid-based question answering in Figure 3.1. We see that the posed questions here are short and can be answered with a contiguous span of a few words from the provided context.

In natural language processing (NLP), transformer-based models like BERT [Dev+18] allow users to ask questions in their natural language. They can be used to extract information from a large corpus of documents like technical and scientific manuals. These state-of-the-art models are very powerful and easy to use. These models are readily available as pre-trained models from open source libraries like Hugging Face [22] that can be fine-tuned for the downstream task of question answering by training them on a few thousand question and answer pairs. We will later describe in this chapter how simple it is to fine-tune and use these models. These models may be simple to use, but they still require training data that is expensive and resource-intensive to gather. The publicly available datasets like SQuAD cannot be used because of domain mismatch. These datasets are built using data from sources like Wikipedia and can not be used for specific domains like EPSC. Building our own training data is one option; it is a time-consuming and resource-intensive task, as we will discuss later in the section 3.3.6 in detail.

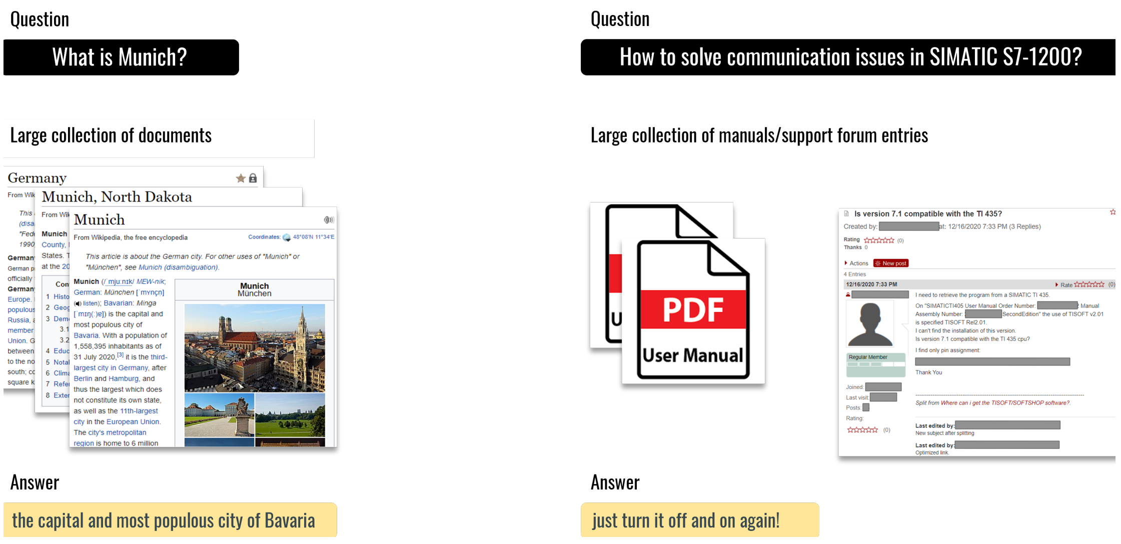

Question
What is Munich?
Large collection of documents
Answer
the capital and most populous city of Bavaria
Question
How to solve communication issues in SIMATIC S7-1200?
Large collection of manuals/support forum entries
PDF
User Manual
Answer
just turn it off and on again!

Figure 3.1: Image illustrating question answering. (Left) Question answering on Wikipedia articles. (Right) Question answering on manuals and support forums.

Siemens engages in the production and supply of systems for power generation, power transmission, and medical diagnosis. These power systems contain electrical components. A user has to go through the technical manual for each component, which is 10s of pages long. There are 100s of such components in a large electrical power system. One of the use cases at Siemens is to look up specific information about these components. As we discussed earlier, the manual way is time-consuming that does not scale, and building an information retrieval system is complex and resource intensive. Also, it would be required to build multiple such information retrieval systems for every large electrical power system.

We use pre-trained question-answering systems to tackle the challenges. The idea behind our approach is to use the pre-trained question answering models like RoBERTa [Liu+19] and fine-tune them on questions and answers from a specific domain, for example, electrical power system components. This allows the question answering system to learn and adapt to that specific domain. We will see in further sections that fine-tuning these question-answering systems is straightforward; we will also discuss the number of training samples required for training. One of the challenges that still remains is finding quality training data to fine-tune the models.

Another use case at Siemens involves building question answering systems for various clients for client-specific use cases. Clients usually do not provide training samples for the training question answering system. It requires them to dedicate subject matter experts to generate question-answer pairs, and that is expensive. They still expect good performance for the models without providing quality training samples. In our approach, we circumvent the challenges associated with generating training data by using state-of-the-art unsupervised machine translation (UNMT) approach as discussed in section 2.7.

We use [LDR19]'s UNMT model to generate question and answer pairs. We use the generated question-answer pairs as synthetic training data for fine-tuning the pre-trained question-answering models. We evaluate the performance of the trained models on a test set from synthetic training data. This evaluation tells us how well the fine-tuned models generalize to the synthetic test set. We also evaluate our fine-tuned models on a hand-curated EPSC dataset. This dataset serves as the ground truth, and we do our final model evaluation on this dataset. We argue that if models trained on synthetic data perform well on ground truth, we can use this approach for training question-answering models for the above-mentioned use cases where a labeled dataset is not available. We can fine-tune the models and test their performance on synthetic data and use that as the indicator for their performance on use-case-specific data.

3.2 Data

In section 3.1 we learned about the two use cases of question answering systems at Siemens. We discussed the challenges associated with getting quality training data for fine-tuning question-answering models. In our approach, we use the UNMT model to generate synthetic question and answer pairs, but we need to gather and process the raw data before we can do that. This section will discuss the steps and factors involved in the data pipeline. We will use a Wikipedia page as a running example to understand the evolution of data at every step. Figure 3.2 shows all the steps involved in the data processing pipeline.

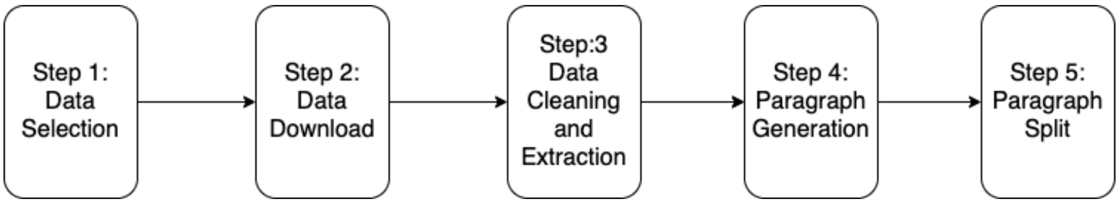
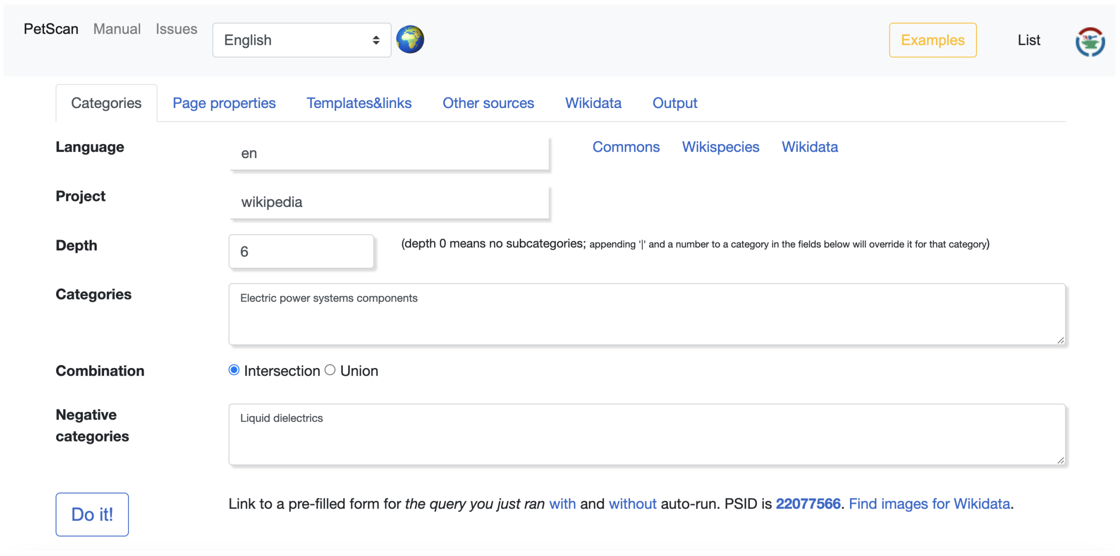


Figure 3.2: Data Processing Pipeline. Raw data goes through all the steps in sequence.

3.2.1 Data Selection

In this thesis, we choose to work with Wikipedia data. We chose Wikipedia data over data specific to previously discussed use cases at Siemens because the specific data is hard to acquire. At Siemens, due to its large organization size, processes and approvals are involved for accessing company data. Technical manuals for the electrical power systems components are owned by teams at a subsidiary of Siemens that are different from the subsidiary where the work of this thesis is being performed. Getting access to this data would have been time-consuming and out of the time scope of this thesis.

We used the tool PetScan [Met21] to collect the data from Wikipedia. PetScan is a tool for querying Wiki-media databases and editing Wiki data. It allows users to apply various filters like selecting all articles belonging to a certain category, all items with a certain property, etc. Figure 3.3 depicts the query we use. We collect all the pages that belong to the "Electric power systems components" category and all its subcategories. We do not collect pages that belong to the "Liquid dielectrics" category. The depth field specifies how deep the tool recursively looks for pages in the subcategories. The depth of 6 returned the maximum number of results.



Language	en
Project	wikipedia
Depth	6 (depth 0 means no subcategories; appending '|' and a number to a category in the fields below will override it for that category)
Categories	Electric power systems components
Combination	[✓ Intersection] Union
Negative categories	Liquid dielectrics

Link to a pre-filled form for *the query you just ran* with and without auto-run. PSID is **22077566**. Find images for Wikidata.

Figure 3.3: PetScan Query Tool to filter Wikipedia data. Our query sets language to 'en' (English), depth to 6 (maximum depth we found for our requirement), categories to 'Electric power system components' and negative categories to 'Liquid dielectric'. Combination field is not significant to use as we only have one category.

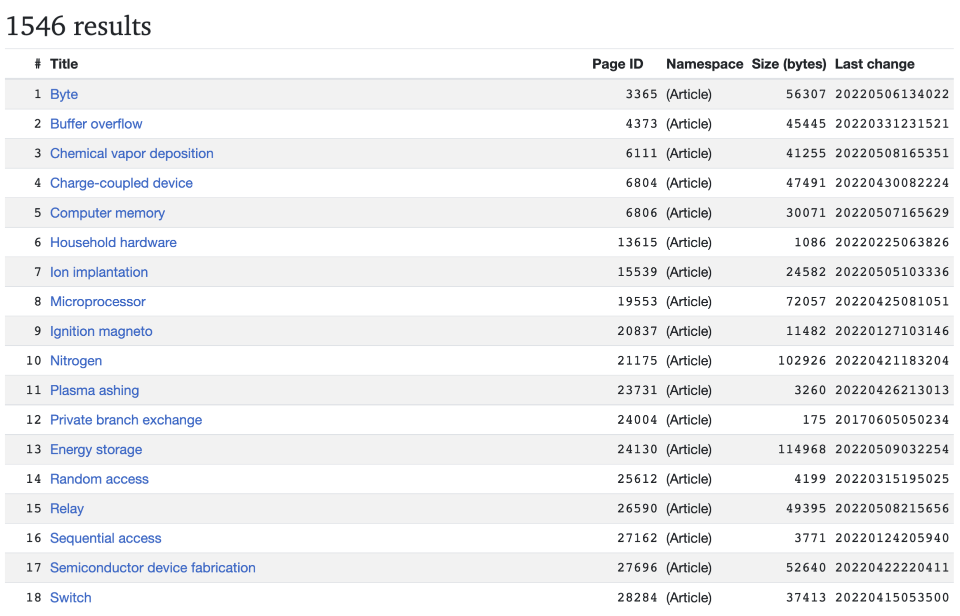
We retrieved 1586[1] results for our specified query. We can see the snapshot of the results in Figure 3.4. All the listed pages belong to "Electric power systems components" or one of its subcategories, but they do not belong to "Liquid dielectrics" or its subcategories.

3.2.2 Data Download

We use Wikipedia's special export [2] feature to download the retrieved pages. The special export wraps the set of retrieved pages in XML and exports it as an XML dump. It also allows users to select the revision of the articles required. We only select the current revisions of the retrieved articles for this work.

[1]Results mentioned here, and the results shown in the image do not match because Wikipedia data is continuously being updated. This image was taken a few months after the data was extracted for this project.

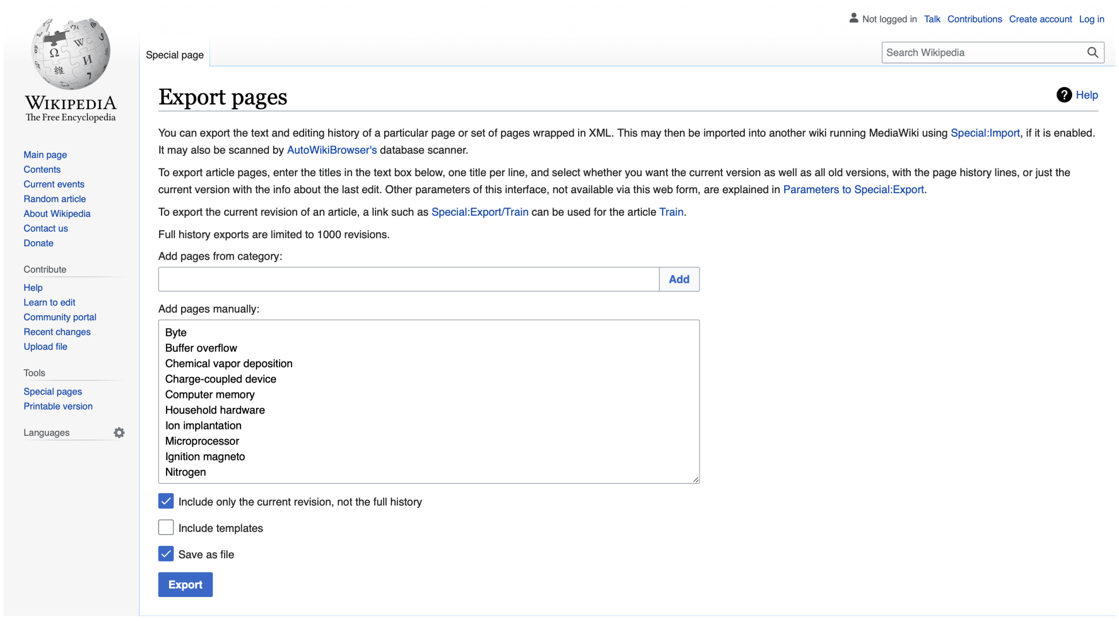
[2]https://en.wikipedia.org/wiki/Wikipedia:Special:Export



1546 results

#	Title	Page ID	Namespace	Size (bytes)	Last change
1	Byte	3365	(Article)	56307	20220506134022
2	Buffer overflow	4373	(Article)	45445	20220331231521
3	Chemical vapor deposition	6111	(Article)	41255	20220508165351
4	Charge-coupled device	6804	(Article)	47491	20220430082224
5	Computer memory	6806	(Article)	30071	20220507165629
6	Household hardware	13615	(Article)	1086	20220225063826
7	Ion implantation	15539	(Article)	24582	20220505103336
8	Microprocessor	19553	(Article)	72057	20220425081051
9	Ignition magneto	20837	(Article)	11482	20220127103146
10	Nitrogen	21175	(Article)	102926	20220421183204
11	Plasma ashing	23731	(Article)	3260	20220426213013
12	Private branch exchange	24004	(Article)	175	20170605050234
13	Energy storage	24130	(Article)	114968	20220509032254
14	Random access	25612	(Article)	4199	20220315195025
15	Relay	26590	(Article)	49395	20220508215656
16	Sequential access	27162	(Article)	3771	20220124205940
17	Semiconductor device fabrication	27696	(Article)	52640	20220422220411
18	Switch	28284	(Article)	37413	20220415053500

Figure 3.4: List of articles (top 18) returned for PetScan query in figure 3.3.



Export pages

You can export the text and editing history of a particular page or set of pages wrapped in XML. This may then be imported into another wiki running MediaWiki using Special:Import, if it is enabled. It may also be scanned by AutoWikiBrowser's database scanner.

To export article pages, enter the titles in the text box below, one title per line, and select whether you want the current version as well as all old versions, with the page history lines, or just the current version with the info about the last edit. Other parameters of this interface, not available via this web form, are explained in Parameters to Special:Export.

To export the current revision of an article, a link such as Special:Export/Train can be used for the article Train.

Full history exports are limited to 1000 revisions.

Add pages from category:

[ ] Add

Add pages manually:

```
Byte
Buffer overflow
Chemical vapor deposition
Charge-coupled device
Computer memory
Household hardware
Ion implantation
Microprocessor
Ignition magneto
Nitrogen
```

☑ Include only the current revision, not the full history

☐ Include templates

☑ Save as file

Export

Figure 3.5: Wikipedia Special Export. User can add page titles and export them as Wikipedia dump.

3.2.3 Data Cleaning and Extraction

As we are primarily interested in the text data of Wikipedia articles, we need to further extract and clean the text from the downloaded Wikipedia dump. Figure 3.6 is a sample example depicting the structure of a Wikipedia dump file. Each of the `<page> ... </page>` tags represents a Wikipedia article. The text data of articles resides within the `<text> ... </text>` tags. We are interested in the text data, and in this subsection, we elaborate on the steps involved in extraction and cleaning.

```
<mediawiki xml:lang="en">
   <page>
     <title>Page title</title>
     <restrictions>edit=sysop:move=sysop</restrictions>
     <revision>
       <timestamp>2001–01–15T13:15:00Z</timestamp>
       <contributor><username>Foobar</username></contributor>
       <comment>I have just one thing to say!</comment>
       <text>A bunch of [[Special:MyLanguage/text | text]] here.</text>
     </revision>
     <revision>
       <timestamp>2001–01–15T13:10:27Z</timestamp>
       <contributor><ip>10.0.0.2</ip></contributor>
       <comment>new!</comment>
       <text>An earlier [[Special:MyLanguage/revision | revision]].</text>
     </revision>
   </page>

   <page>
     <title>Talk:Page title</title>
     <revision>
       <timestamp>2001–01–15T14:03:00Z</timestamp>
       <contributor><ip>10.0.0.2</ip></contributor>
       <comment>hey</comment>
       <text>Sample article text!!!</text>
     </revision>
   </page>
 </mediawiki>
```

Figure 3.6: An example showing structure of a wikipedia dump file. Each Wikipedia page in the dump is contained in `<page>...</page> tags.`

There are python modules like `wiki_dump_reader` or open source tools like *wikiextractor* that can be used to extract text data from Wikipedia dumps. We used a python module called `wikipedia_dump_reader` for our work because of its ease of use. Figure 3.7 is an example code snippet for using this module to extract text from the Wikipedia dump file downloaded in the data download step.

```python
from wiki_dump_reader import Cleaner, iterate

cleaner = Cleaner()
for title, text in iterate('wiki-dump.xml'):
    text = cleaner.clean_text(text)
    cleaned_text, links = cleaner.build_links(text)
```

Figure 3.7: Illustration showing the use of `wiki_dump_reader` in data cleaning and extraction.

`iterate` allows to traverse through XML element tree one page at a time and extract the raw text within the `<text> ... </text>` tags. `Cleaner` removes all the markups, image links, reference links, etc., and returns the plain text. Figure 3.8 illustrates the Wikipedia dump of one Wikipedia article titled *Common control* belonging to *Electric Power System Components* domain. Figure 3.9 shows the text after cleaning and extraction.

For further processing, we remove the headers and all the text below *See Also* header that contains links to other Wikipedia articles and references. It does not make up for meaningful text that models can learn from, so we remove it. We also removed the line breaks and returned the whole article as one text blob. Figure 3.10 shows the text after data cleaning and extraction step in the data processing pipeline.

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.10/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
      ↪ http://www.mediawiki.org/xml/export-0.10/␣http://www.mediawiki.org/xml/export-0.10.xsd" version="0.10" xml:lang="en">
  <page>
    <title>Common control</title>
    <ns>0</ns>
    <id>40911</id>
    <revision>
      <id>1023606745</id>
      <parentid>981816460</parentid>
      <timestamp>2021-05-17T09:32:18Z</timestamp>
      <contributor>
        <username>John of Reading</username>
        <id>11308236</id>
      </contributor>
      <minor/>
      <comment>Typo/[[WP:AWB/GF|general]] fixes, replaced: DIrector  Director</comment>
      <model>wikitext</model>
      <format>text/x-wiki</format>
      <text bytes="2884" xml:space="preserve">{{more citations needed|date=January 2013}}
In [[telecommunications]], '''common control''' is a principle of switching [[telephone call]]s in an automatic [[telephone exchange]] that employs shared
      ↪ control equipment which is attached to the circuit of a call only for the duration of establishing or otherwise controlling the call.&lt;ref&gt;{{Cite
      ↪ book
|url=http://archive.org/details/bellsystem_No5CrossbarVol1EquipmentApplicationsMay63
|title=No. 5 Crossbar Volume 1: Equipment Applications May 63|last=Western Electric Co.
|date=1963-05-01
}}&lt;/ref&gt; Thus, such control equipment need only be provided in as few units to satisfy overall exchange traffic, rather than being duplicated for every
      ↪ subscriber line.

In contrast, ''direct control'' systems have subsystems for call control that are an integral part of the switching network. [[Strowger exchange]]s are usually
      ↪ direct control systems, whereas crossbar, and electronic exchanges (including all [[stored program control]] systems) are common control
      ↪ systems. Common control is also known as indirect control or register control.

== History ==
Early semi-mechanical installations with common control components existed, for example [[rotary system|rotary]] systems in Sweden and France in 1915,
      ↪  and the first [[panel switch]]es in Newark, New Jersey, also in 1915. The first large-scale, fully automatic, common control switching system
      ↪ deployed in commercial production service was the "ATlantic" central office in Omaha, Nebraska, a panel system cut over on December 10, 1921.
      ↪  Other panel offices for Kansas City and New York CIty ("PENnsylvania") were in planning at the same time and opened shortly after.

In 1922, common control was introduced in [[Strowger switch|Strowger-type]] step-by-step systems,&lt;ref&gt;Automatic Electric Company, "The
      ↪ Automatic Director in Strowger Metropolitan Telephone Systems", in "Automatic Telephone", Volume 10(11-12), November 1922, p.116
&lt;/ref&gt; resulting in the first installations of [[Director telephone system|Director]] systems in Havanna, Cuba in 1924, and in London, England in 1927.

By the mid-1920s, common control ideas had extended to include [[Marker (telecommunications)|marker]] systems for testing for idle [[Trunking#
      ↪ Telecommunications|trunks]].{{citation needed|date=February 2019}}

During the 1960s, common control exchanges became [[stored program control exchange|stored program control]] exchanges,&lt;ref name=bosse2007&gt;J.
      ↪ G. Van Bosse, F.U. Devetak, "Signaling in Telecommunication Networks", 2nd edition (2007), p.111&lt;/ref&gt; and by the 1970s they used [[
      ↪ common-channel signaling]] in which the channels that are used for [[signaling (telecommunication)|signaling]] are not used for [[message]]
      ↪ traffic (out of band signaling).&lt;ref name=bosse2007/&gt;

== References ==
{{reflist}}
{{FS1037C MS188}}

[[Category:Telephone exchange equipment]]
{{telephony-stub}}</text>
      <sha1>0bixwk0cfaedagn2ktjxe51gtjblmj7</sha1>
    </revision>
  </page>
</mediawiki>
```

Figure 3.8: Wikipedia dump with one page titled "Common control". Text before cleaning consists of markups, links, images etc. that is not useful for training models.

```
In telecommunications, common control is a principle of switching telephone calls in an automatic telephone
        ↪ exchange that employs shared control equipment which is attached to the circuit of a call only for the
        ↪ duration of establishing or otherwise controlling the call. Thus, such control equipment need only be
        ↪ provided in as few units to satisfy overall exchange traffic, rather than being duplicated for every
        ↪ subscriber line.
In contrast, direct control systems have subsystems for call control that are an integral part of the switching network.
        ↪  Strowger exchanges are usually direct control systems, whereas crossbar, and electronic exchanges (
        ↪ including all stored program control systems) are common control systems. Common control is also known
        ↪  as indirect control or register control.
== History ==
Early semi−mechanical installations with common control components existed, for example rotary systems in
        ↪ Sweden and France in 1915, and the first panel switches in Newark, New Jersey, also in 1915. The first large
        ↪ −scale, fully automatic, common control switching system deployed in commercial production service was
        ↪ the ATlantic central office in Omaha, Nebraska, a panel system cut over on December 10, 1921. Other panel
        ↪ offices for Kansas City and New York CIty (PENnsylvania) were in planning at the same time and opened
        ↪ shortly after.
In 1922, common control was introduced in Strowger−type step−by−step systems, resulting in the first installations
        ↪ of Director systems in Havanna, Cuba in 1924, and in London, England in 1927.
By the mid−1920s, common control ideas had extended to include marker systems for testing for idle trunks.
During the 1960s, common control exchanges became stored program control exchanges, and by the 1970s they used
        ↪  common−channel signaling in which the channels that are used for signaling are not used for message
        ↪ traffic (out of band signaling).
== References ==
Telephone exchange equipment
```

Figure 3.9: Wikipedia dump with one page titled "Common control" after cleaning and extraction.

```
In telecommunications, common control is a principle of switching telephone calls in an automatic telephone
        ↪ exchange that employs shared control equipment which is attached to the circuit of a call only for the
        ↪ duration of establishing or otherwise controlling the call. Thus, such control equipment need only be
        ↪ provided in as few units to satisfy overall exchange traffic, rather than being duplicated for every
        ↪ subscriber line. In contrast, direct control systems have subsystems for call control that are an integral part
        ↪ of the switching network. Strowger exchanges are usually direct control systems, whereas crossbar, and
        ↪ electronic exchanges (including all stored program control systems) are common control systems. Common
        ↪  control is also known as indirect control or register control. Early semi−mechanical installations with
        ↪ common control components existed, for example rotary systems in Sweden and France in 1915, and the
        ↪ first panel switches in Newark, New Jersey, also in 1915. The first large−scale, fully automatic, common
        ↪ control switching system deployed in commercial production service was the ATlantic central office in
        ↪ Omaha, Nebraska, a panel system cut over on December 10, 1921. Other panel offices for Kansas City and
        ↪ New York CIty (PENnsylvania) were in planning at the same time and opened shortly after. In 1922,
        ↪ common control was introduced in Strowger−type step−by−step systems, resulting in the first installations
        ↪ of Director systems in Havanna, Cuba in 1924, and in London, England in 1927. By the mid−1920s,
        ↪ common control ideas had extended to include marker systems for testing for idle trunks. During the 1960s,
        ↪  common control exchanges became stored program control exchanges, and by the 1970s they used
        ↪ common−channel signaling in which the channels that are used for signaling are not used for message
        ↪ traffic (out of band signaling). Telephone exchange equipment.
```

Figure 3.10: Final text blob extracted for "Common control" page by removing headers and new lines.

3.2.4 Data Processing

Our aim at this step is to build the paragraph corpus from the cleaned Wikipedia text. We split the Wikipedia text blob obtained after data cleaning and extraction into paragraphs of appropriate word length for it to be consumed by the UNMT model to generate synthetic training data. As we focus on factoid-based extractive question answering, the answers are expected to be a few words long at the most. On average, a paragraph word length is 200 words, we chose to set paragraph length to 150 words as our expected answers are not more than a few words, and 150 words are enough to provide context for the extracted answers.

We used `PreProcessor` module by Haystack to split the Wikipedia text blob into paragraphs. Haystack is an open source framework built to bridge the gap between research and industry in the field of NLP. The `PreProcessor` module provides various options for cleaning and splitting the text. It takes one single document as input and returns paragraphs as a list of documents. In the Figure 3.11 we see the available parameters for `PreProcessor` and our implementation of it.

```
from haystack.preprocessor.preprocessor import PreProcessor
preprocessor = PreProcessor(
                    clean_empty_lines=True,
                    clean_whitespace=True,
                    clean_header_footer=False,
                    split_by="word",
                    split_overlap=50
                    split_length=150,
                    split_respect_sentence_boundary=True
               )
```

Figure 3.11: Illustration of how we use haystack `PreProcessor` module. We split by 'word', with a split overlap of 50 words and create paragraphs of 150 words long while respecting the sentence boundary.

`split_respect_sentence_boundary` parameter when set to True, it does not split a paragraph mid-sentence even if it gets longer than the specified split length. When `word_overlap` is set to 0, then the two adjacent output paragraphs have no overlap. If it is set to a positive number, then the two adjacent paragraphs overlap by that amount in word count. In figure 3.12 we can see the final paragraph corpus generated for our running example. Our paragraph corpus is ready at this stage, and the UNMT model can utilize the prepared paragraphs to generate synthetic question-answer pairs.

```
{"text": "In␣telecommunications,␣common␣control␣is␣a␣principle␣of␣switching␣telephone␣calls␣in␣an␣automatic␣telephone␣exchange␣
    ↪ that␣employs␣shared␣control␣equipment␣which␣is␣attached␣to␣the␣circuit␣of␣a␣call␣only␣for␣the␣duration␣of␣establishing␣
    ↪ or␣otherwise␣controlling␣the␣call.␣Thus,␣such␣control␣equipment␣need␣only␣be␣provided␣in␣as␣few␣units␣to␣satisfy␣overall␣
    ↪ exchange␣traffic,␣rather␣than␣being␣duplicated␣for␣every␣subscriber␣line.␣In␣contrast,␣direct␣control␣systems␣have␣
    ↪ subsystems␣for␣call␣control␣that␣are␣an␣integral␣part␣of␣the␣switching␣network.␣Strowger␣exchanges␣are␣usually␣direct␣
    ↪ control␣systems,␣whereas␣crossbar,␣and␣electronic␣exchanges␣(including␣all␣stored␣program␣control␣systems)␣are␣common␣
    ↪ control␣systems.␣Common␣control␣is␣also␣known␣as␣indirect␣control␣or␣register␣control.␣Early␣semi-mechanical␣
    ↪ installations␣with␣common␣control␣components␣existed,␣for␣example␣rotary␣systems␣in␣Sweden␣and␣France␣in␣1915,␣and␣the␣
    ↪ first␣panel␣switches␣in␣Newark,␣New␣Jersey,␣also␣in␣1915.", "paragraph_id": "576fe293-cf36-437a-acf9-53d54b855fb8"}
{"text": "Strowger␣exchanges␣are␣usually␣direct␣control␣systems,␣whereas␣crossbar,␣and␣electronic␣exchanges␣(including␣all␣stored␣
    ↪ program␣control␣systems)␣are␣common␣control␣systems.␣Common␣control␣is␣also␣known␣as␣indirect␣control␣or␣register␣
    ↪ control.␣Early␣semi-mechanical␣installations␣with␣common␣control␣components␣existed,␣for␣example␣rotary␣systems␣in␣
    ↪ Sweden␣and␣France␣in␣1915,␣and␣the␣first␣panel␣switches␣in␣Newark,␣New␣Jersey,␣also␣in␣1915.␣The␣first␣large-scale,␣
    ↪ fully␣automatic,␣common␣control␣switching␣system␣deployed␣in␣commercial␣production␣service␣was␣the␣ATlantic␣central␣
    ↪ office␣in␣Omaha,␣Nebraska,␣a␣panel␣system␣cut␣over␣on␣December␣10,␣1921.␣Other␣panel␣offices␣for␣Kansas␣City␣and␣New␣
    ↪ York␣CIty␣(PENnsylvania)␣were␣in␣planning␣at␣the␣same␣time␣and␣opened␣shortly␣after.␣In␣1922,␣common␣control␣was␣
    ↪ introduced␣in␣Strowger-type␣step-by-step␣systems,␣resulting␣in␣the␣first␣installations␣of␣Director␣systems␣in␣Havanna,␣
    ↪ Cuba␣in␣1924,␣and␣in␣London,␣England␣in␣1927.", "paragraph_id": "7a36fb04-b7a9-46b7-8259-5a69b8225eca"}
{"text": "Other␣panel␣offices␣for␣Kansas␣City␣and␣New␣York␣CIty␣(PENnsylvania)␣were␣in␣planning␣at␣the␣same␣time␣and␣opened␣
    ↪ shortly␣after.␣In␣1922,␣common␣control␣was␣introduced␣in␣Strowger-type␣step-by-step␣systems,␣resulting␣in␣the␣first␣
    ↪ installations␣of␣Director␣systems␣in␣Havanna,␣Cuba␣in␣1924,␣and␣in␣London,␣England␣in␣1927.␣By␣the␣mid-1920s,␣common␣
    ↪ control␣ideas␣had␣extended␣to␣include␣marker␣systems␣for␣testing␣for␣idle␣trunks.␣During␣the␣1960s,␣common␣control␣
    ↪ exchanges␣became␣stored␣program␣control␣exchanges,␣and␣by␣the␣1970s␣they␣used␣common-channel␣signaling␣in␣which␣the␣
    ↪ channels␣that␣are␣used␣for␣signaling␣are␣not␣used␣for␣message␣traffic␣(out␣of␣band␣signaling).␣Telephone␣exchange␣
    ↪ equipment", "paragraph_id": "d781d42b-08b6-47d9-933d-304b5a5392b7"}
```

Figure 3.12: Paragraphs generated using split overlap from "Common control" page.

We experimented to see the impact of split overlap on models' performance. Details of the experiment are detailed in the next section. Although the results in both cases were comparable, we chose to split with overlap for two reasons. First, the number of paragraphs generated is significantly more than the number of paragraphs generated with no overlap. Second, intuitively models get a better signal to learn due to overlap. In figure 3.13 we can see the paragraphs generated when `split_overlap=0`. Here the number of paragraphs is less, meaning less synthetic training data would be generated.

```
{"text": "In␣telecommunications,␣common␣control␣is␣a␣principle␣of␣switching␣telephone␣calls␣in␣an␣automatic␣telephone␣exchange␣
    ↪ that␣employs␣shared␣control␣equipment␣which␣is␣attached␣to␣the␣circuit␣of␣a␣call␣only␣for␣the␣duration␣of␣establishing␣
    ↪ or␣otherwise␣controlling␣the␣call.␣Thus,␣such␣control␣equipment␣need␣only␣be␣provided␣in␣as␣few␣units␣to␣satisfy␣overall␣
    ↪ exchange␣traffic,␣rather␣than␣being␣duplicated␣for␣every␣subscriber␣line.␣In␣contrast,␣direct␣control␣systems␣have␣
    ↪ subsystems␣for␣call␣control␣that␣are␣an␣integral␣part␣of␣the␣switching␣network.␣Strowger␣exchanges␣are␣usually␣direct␣
    ↪ control␣systems,␣whereas␣crossbar,␣and␣electronic␣exchanges␣(including␣all␣stored␣program␣control␣systems)␣are␣common␣
    ↪ control␣systems.␣Common␣control␣is␣also␣known␣as␣indirect␣control␣or␣register␣control.␣Early␣semi-mechanical␣
    ↪ installations␣with␣common␣control␣components␣existed,␣for␣example␣rotary␣systems␣in␣Sweden␣and␣France␣in␣1915,␣and␣the␣
    ↪ first␣panel␣switches␣in␣Newark,␣New␣Jersey,␣also␣in␣1915.", "paragraph_id": "27a114ba-3077-489e-831c-b51690142beb"}
{"text": "The␣first␣large-scale,␣fully␣automatic,␣common␣control␣switching␣system␣deployed␣in␣commercial␣production␣service␣was␣
    ↪ the␣ATlantic␣central␣office␣in␣Omaha,␣Nebraska,␣a␣panel␣system␣cut␣over␣on␣December␣10,␣1921.␣Other␣panel␣offices␣for␣
    ↪ Kansas␣City␣and␣New␣York␣CIty␣(PENnsylvania)␣were␣in␣planning␣at␣the␣same␣time␣and␣opened␣shortly␣after.␣In␣1922,␣common␣
    ↪ control␣was␣introduced␣in␣Strowger-type␣step-by-step␣systems,␣resulting␣in␣the␣first␣installations␣of␣Director␣systems␣
    ↪ in␣Havanna,␣Cuba␣in␣1924,␣and␣in␣London,␣England␣in␣1927.␣By␣the␣mid-1920s,␣common␣control␣ideas␣had␣extended␣to␣include␣
    ↪ marker␣systems␣for␣testing␣for␣idle␣trunks.␣During␣the␣1960s,␣common␣control␣exchanges␣became␣stored␣program␣control␣
    ↪ exchanges,␣and␣by␣the␣1970s␣they␣used␣common-channel␣signaling␣in␣which␣the␣channels␣that␣are␣used␣for␣signaling␣are␣not␣
    ↪ used␣for␣message␣traffic␣(out␣of␣band␣signaling).␣Telephone␣exchange␣equipment", "paragraph_id":
    ↪ "702e419e-8ef4-4c09-91e5-0615b244d7bb"}
```

Figure 3.13: Paragraphs generated from "Common control" page dump without using split overlap.

3.2.5 Data Split for Ground Truth

As there are not any benchmark question answering data-sets available in *electric power system components* domain, we created a data-set of 453 manually annotated samples to evaluate our models. We describe our process of ground truth annotation in section 3.3.6. We obtained a corpus of 13845 clean and processed paragraphs, and out of that, we held out 8% of the paragraphs for manual annotation. Manual annotation is time intensive task; only a limited number of samples could be annotated in the time and scope of this thesis. We tried to maximize the paragraphs available for synthetic data generation, and at the same time, we annotated as samples as we could for ground truth generation.

3.3 Methodology

The question answering (QA) system must find spans of answers for a given question from the provided context in factoid-based extractive question answering. Traditionally, QA systems are trained in a supervised fashion using datasets like SQuAD [Raj+16]. SQuAD has 100,000+ question-answer pairs from 500+ articles. These questions are posed by crowd workers on a set of Wikipedia articles. The size and resources required to build such a dataset make it a resource-intensive task.

SQuAD like datasets have enabled researchers to propel the research in general question answering task, even beating human accuracy. While this is impressive, these datasets cannot be used for training QA models in a specialized domain like *Electric Power System Components*. One approach is to build domain specific question-answering dataset and use the new dataset to train and test the models. This is what researchers are doing in the domain of medicine and law. Texts in such domains have a distinctly different language and vocabulary. Institutions are putting in lots of resources to carry forward research in those domains. For a low-resource setting like ours, this approach is not ideal.

In our approach, we use [LDR19]'s UNMT model to generate synthetic training data. Our idea is that if the UNMT approach produces quality training data, then using this data, we can train state-of-the-art transformer models on the downstream task of question answering. Apart from pre-trained QA models in *Electric Power System Components* domain, we also curated a manually annotated dataset to evaluate models trained with our approach.

3.3.1 Experiment Setup

Figure 3.14 shows the schematic representation of our experiment setup.

1. **Step 1 - Data processing:** We discussed this step in detail in the section 3.2 of this chapter. In this step, we download and sanitize the data. We also prepare a paragraph corpus that can be consumed by the UNMT model.
2. **Step2 - Ground truth data spilt:** We split our paragraph corpus into two parts, one part is used for generating synthetic data for training, and another part is used for creating ground truth data.
3. **Step 3 - Synthetic data generation:** We use the UNMT model made available by authors of [LDR19] to generate synthetic data.
4. **Step 4 - Synthetic Data Split:** We split the synthetic data into three parts; Train set, Dev set, and Synthetic Test set, in a random 60-20-20 split, respectively.
5. **Step 5 - Training QA models:** We train three state-of-the-art transformer models, RoBERTa, MiniLM, and SpanBERT, on synthetic training data on the downstream task of question answering. We fine-tune the models till we see good performance on the dev set.
6. **Step 6 - Ground Truth Generation:** We use the UNMT model to generate question-answer pair from held-out paragraphs from step 2. Then we manually annotate question-answer pairs, and that represents our ground truth.
7. **Step 7 - Evaluation:** We evaluate the best performing models on the synthetic test set.
8. **Step 8 - Ground Truth Evaluation:** Lastly, we evaluate the models on the ground truth and report their performance.

3.3.2 Synthetic Data Generation

The authors of [LDR19] have taken one of the first steps in NLP towards unsupervised question answering. They developed an approach that achieves a 64.5% F1 score when an answer is a named entity mention without using the SQuAD dataset. We use their UNMT model to generate the synthetic data from the paragraph corpus we curated. Figure 3.15 shows the bash script we use generate synthetic data [3]. UNMT

[3]Further details can be found on their GitHub page.

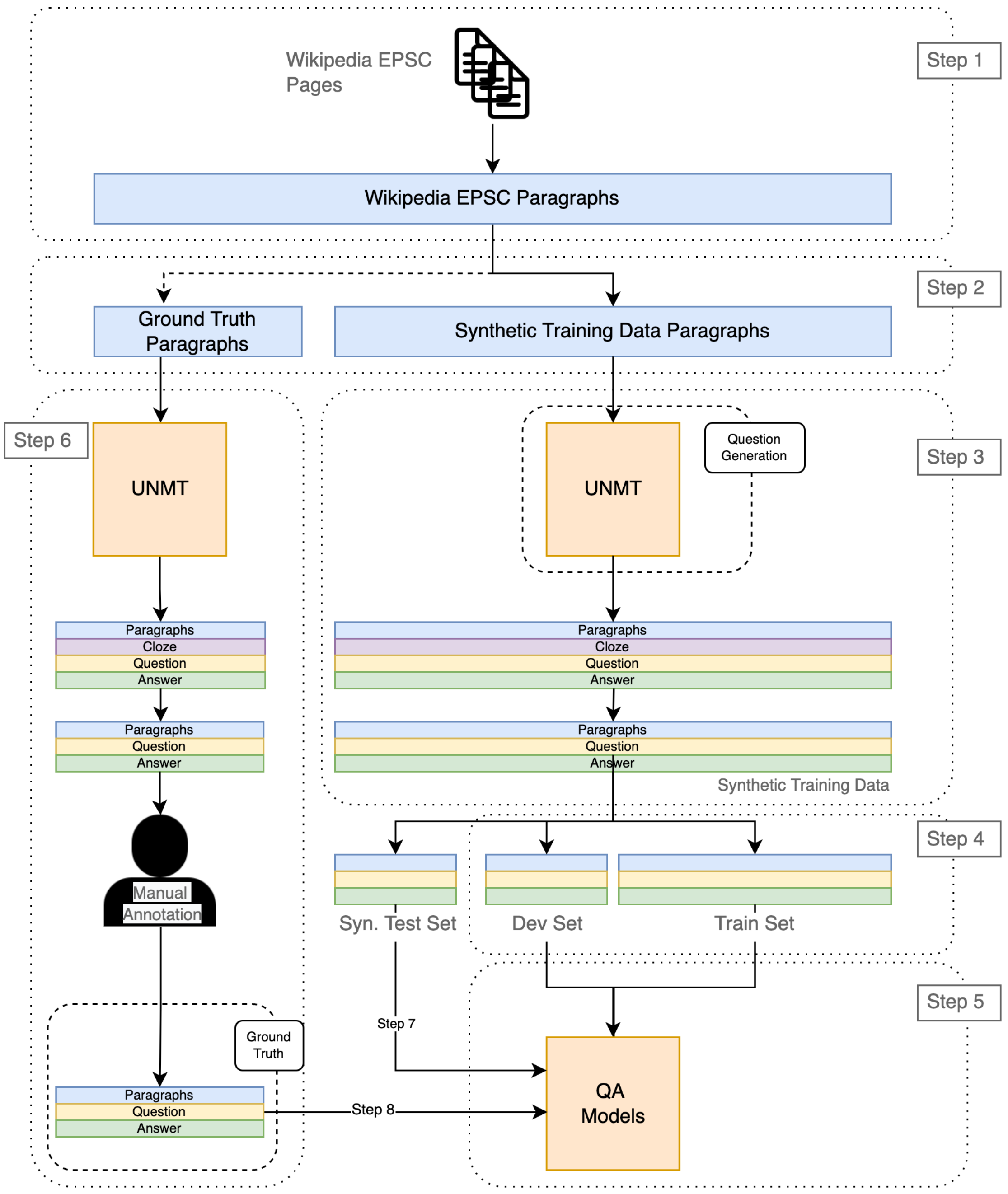


Figure 3.14: Schematic of our experiment setup. The indicated steps show the flow of data from its raw form to synthetic (on right branch) question-answer pairs and ground truth (on left branch).

model outputs question and answer pairs for the context paragraphs that we provide as input in the SQuAD format. We chose the UNMT model with the parameters use_named_entity_clozes (NE), use_subclause_clozes (SC), and use_wh_heuristic (Wh*) because the authors demonstrated that this model had the best performance on the downstream tasks.

```
python −m unsupervisedqa.generate_synthetic_qa_data input.jsonl output \
−−input_file_format "jsonl" \
−−output_file_format "squad" \
−−translation_method unmt \
−−use_named_entity_clozes \
−−use_subclause_clozes \
−−use_wh_heuristic
```

Figure 3.15: Script used to generate question-answer pairs using UNMT model.

While we pick the model with the best performance on the downstream task, i.e., the model with NE, SC, and Wh* parameters set to true, we also wanted to maximize the number of synthetic training samples that we generate. If we generate a paragraph corpus without any split overlap, we will generate less number of paragraphs, and fewer paragraphs mean the UNMT model generates less synthetic training data. As we can see in the 3.1 that Config Id number 4 generates almost 5000 more samples than Config Id 1, we choose the former of the two configurations to generate synthetic training data to train our models.

Table 3.1: Synthetic Data Generation Configurations.

Config Id	Paragraph Generation Approach	Named Entity Cloze (NE)	Sub-clause (SC)	Wh* Heuristics (WH)	Time (mins.)	Number of Training Samples
1	FL	T	T	T	58	6304
2	FL	T	T	F	47	8970
3	FL	T	F	F	74	10261
4	**SW**	**T**	**T**	**T**	100	**11187**
5	SW	T	T	F	77	15493
6	SW	T	F	F	128	17506

One might question that what the impact of synthetic data generated from the six configurations listed in Table 3.1. We did a study to investigate that, and we report our findings in section 3.4.

3.3.3 Training

For Training, we use Haystack by Deepset [4], an open-source framework for building search systems that work intelligently over large document collection. It implements the Framework for Adapting Representation Models (FARM) that makes transfer learning with transformer-based models fast and enterprise ready. One of its core features is the easy finetuning of language models to one's task and domain language. *FarmReader* module provides a straightforward way to download pre-trained models from Hugging Face and finetune them for our task. Figure 3.16 shows simple three line code to download and fine-tune pre-trained models.

```
# load a pre−trained model
from haystack.reader.farm import FARMReader
reader = FARMReader(model_name_or_path=model, use_gpu=True)
reader.train(train_data, train_filename, save_dir=save_dir, n_epochs=2, dev_split=.25, test_filename=test_filename)
```

Figure 3.16: Illustration to show usage of FARMReader module by Haystack. We use pre-trained language models from HuggingFace and train them for 2 epochs.

We trained three models (MiniLM, RoBERTa, and SpanBERT) on the synthetic question-answer pairs generated by the UNMT model. We split the synthetic data in 60-20-20 split where 60% is the training set, 20% is the development set, and 20% is the synthetic test set. We train all three models for two epochs on the training set and evaluate their performances on the synthetic dev set after training for every 300 samples. After the training, we do two evaluations of the fine-tuned models. The first evaluation is on the synthetic test set, and the second evaluation is on manually annotated ground truth data. We perform training on Google Colab with Nvidia SMI GPUs.

[4]https://haystack.deepset.ai/overview/intro

3.3.4 Evaluation

We perform two evaluations of the trained models. We evaluate models on the synthetic test set and the manually annotated ground truth. Good performance on the synthetic test set indicates that the model is able to learn the question-answering behavior and generalize well on the synthetic data. But through this work, we want to evaluate the quality of synthetic data generated by the UNMT model. To achieve that, we evaluate models on manually annotated ground truth data. Our idea is that if the model performs well on real-world ground truth data, it would indicate that the quality of synthetic data used for training is good and matches real data. For use cases at Siemens, it can be very helpful in low resource settings where we do not have ground truth data available, and we can only train and evaluate models only on synthetic data. If the model generalizes well to synthetic data and if the quality of synthetic data is comparable to real-world data, as we establish in this work, then we can reliably use the UNMT approach to generate training data in low-resource settings.

3.3.5 Baseline Experiment

In order to establish baselines to analyze the model's performance with respect to UNMT synthetic data generation method, we conducted two experiments where we generated synthetic data by identity mapping and noisy cloze translation method and trained the QA models on that data. Figure 3.17 is a schematic representation for experiment setup for the baseline experiments. Identity mapping considers that the cloze questions themselves provide a signal to learn question-answering behavior. The noisy cloze method characterizes the difference between the cloze question and the natural question in the form of perturbation. We have provided more details about both the approaches in section 2.7.2.

3.3.6 Ground Truth Generation

Annotation is a very expensive and resource-intensive task. We had one annotator to read and prepare question answers for ground truth data. We took an approach where we used the output of the UNMT model and refined those samples. After preparing the paragraph corpus, we split it into two parts. We use around 8% of the paragraphs and generate synthetic question-answer pairs using the UNMT model and manually refine them. We take this approach because synthetic questions already provide a good starting point for the annotator. Starting from scratch would have required a lot more

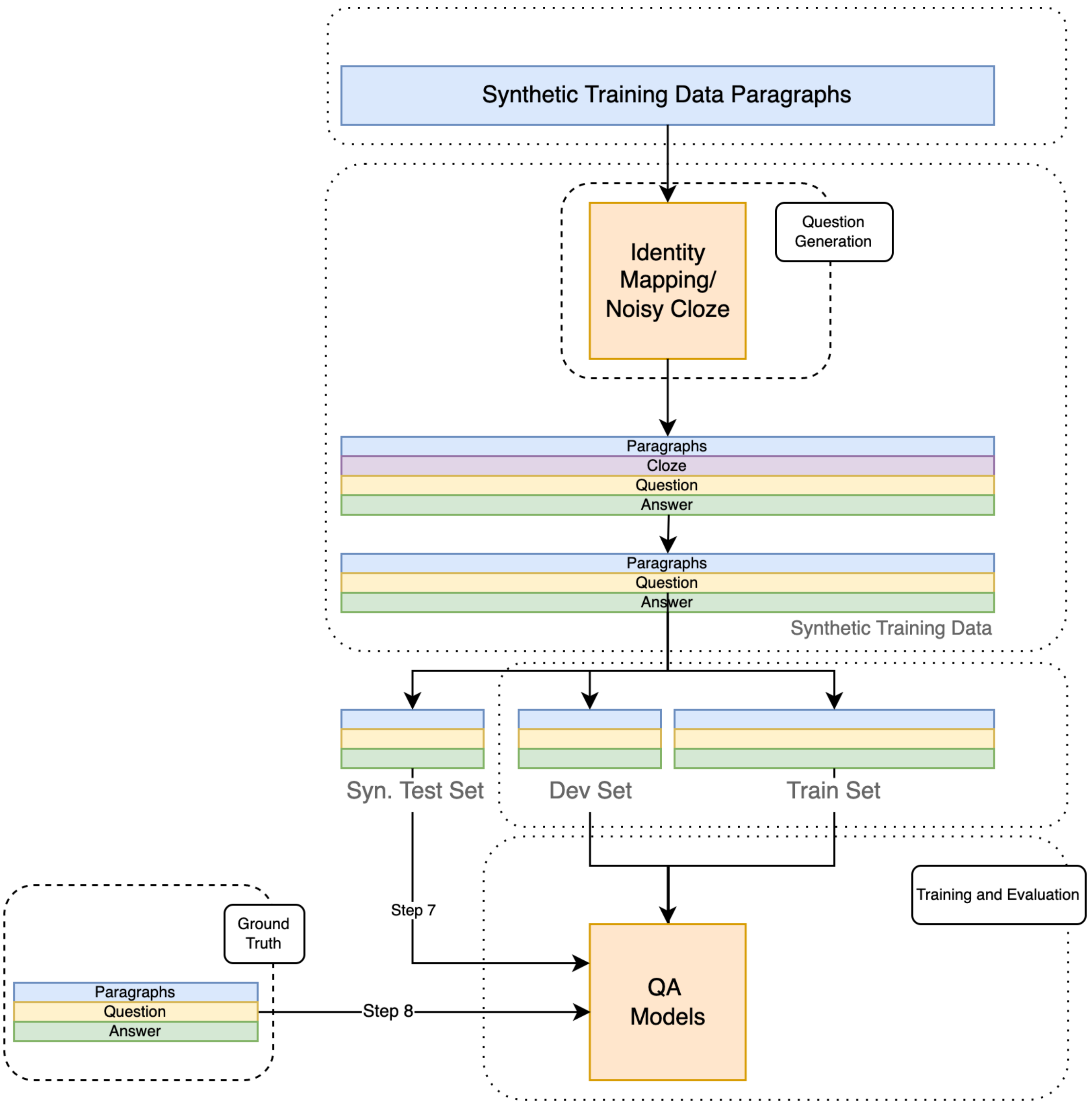


Figure 3.17: Our baseline experiment setup. We use Identity Mapping model and Noisy cloze model to generate synthetic training data.

time to prepare ground truth data, so that put it out of the time scope of this thesis. Table 3.2 summarizes the manual effort required to process 500 samples for annotating factoid-based question answers while using synthetic questions and answers generated by the model as a starting point.

Table 3.2: Summary of our effort to generate ground truth dataset.

Amount of Time	75 hr (approx.)
Number of Samples	500
Number of Annotations	453

We spent 8-9 minutes per sample on average for annotation. We used the open source annotation tool, Haystack Annotation Tool, to perform the annotations. We completed about 26 annotations per day, spending around 4 hours in one sitting. Annotation of 500 samples took 23-25 days spread over 1.5 months. We created 1 question-answer pair per paragraph. In cases where no valid question-answer pair could be found, the sample was removed.

We only create questions starting with *Who, What When, Where* and *How much/many* and ending with a question mark. We specifically chose these question words because synthetic training data generated by the UNMT model only contains the mentioned question words. Adding other question words would guarantee a bad performance on those test examples. In the figure 3.18 and figure 3.19 we can see the distribution of question types before and after annotation.

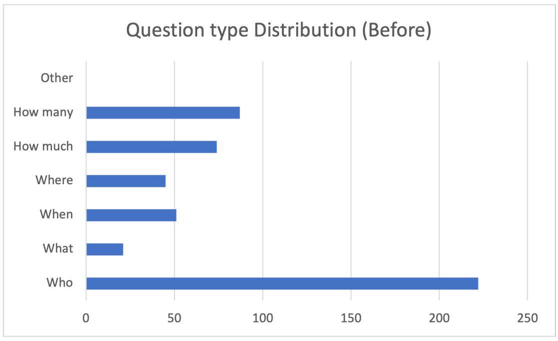


Figure 3.18: Distribution of type of questions in synthetic data before annotation.

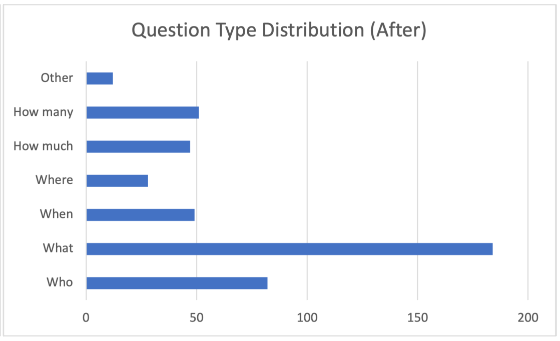


Figure 3.19: Distribution of question types in ground truth data after annotation.

We see a huge change in the number of questions for *What* and *Who* question types. Most new questions created pertained to *THINGS*, and *What* question type was most appropriate compared to *Who* question type. This shift in numbers makes sense because,

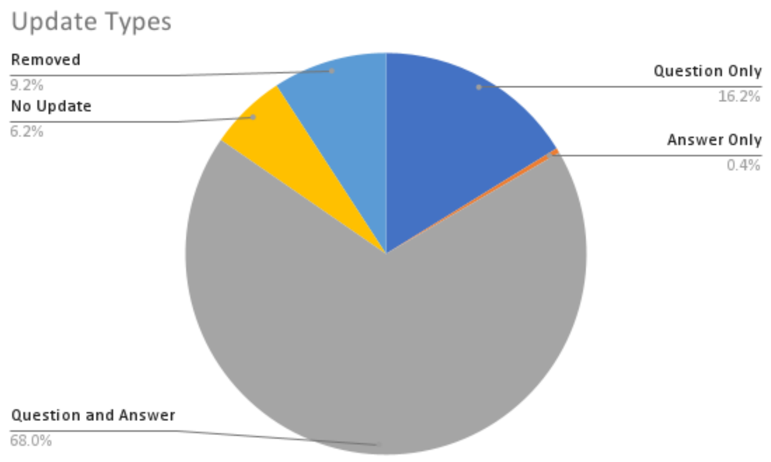


Figure 3.20: Pie chart showing type of updates done during annotation. More than $2/3^{rd}$ of samples needed both question and answer update.

in the context of the electrical power system components domain, it is more likely that a what question type fits compared to Who question type. Fewer person entities are to show up in the electrical power system components context where *Who* question type would fit.

There were five types of updates made while generating ground truth using synthetic data. No update is the category for question-answer pairs that were selected without any changes to either question or answer. Only 6% samples were such that required no update. The major category is Question and Answers; we updated both the question and the answer for that sample, and in most cases, that meant creating new questions and answers. For these samples, synthetic questions and answers did not prove to be useful. Figure 3.20 shows the types of updates and their percentages.

3.4 Result and Discussion

3.4.1 Impact of Data Generation Approaches

We study the impact of various data generation configurations on the training of RoBERTa and MiniLM. We fine-tune these models on the data generated with each of the configurations in table 3.1. In table 3.3 we report the F1, EM and Top_4 accuracy scores of our fine-tuned models on synthetic test data generated from respective configuration. We report the scores before fine-tuning and after fine-tuning. We see that both the models achieve a very high score on synthetic test data for all configurations, meaning they generalize well on synthetic data. As there is no significant impact on the performance of models due to the various configurations of paragraph generation, Named Entity (NE), Sub-clause (SC), or Wh heuristics used to generate synthetic training data, we optimize for the size of the training data that can be generated, and we pick the configuration of [LDR19]'s best-performing model.

Table 3.3: Synthetic test set performance with respect to various data generation configurations from table 3.1. UNMT model is used to generate question and answer pairs in all these configuration

Config Id	Model	Before Finetuning			After finetuning		
		EM	**F1**	**Top-4**	**EM**	**F1**	**Top-4**
1	MiniLM	31.5	37.7	59.6	93.5	96.3	97.6
	RoBERTa	34.2	42.8	62.7	94.6	97.0	98.1
2	MiniLM	19.4	25.8	58.9	93.6	96.2	97.6
	RoBERTa	21.7	29.8	59.9	95.3	97.3	98.5
3	MiniLM	20.4	28.8	60.9	96.9	98.0	98.6
	RoBERTa	22.0	32.3	59.5	97.3	98.3	98.8
4	MiniLM	31.6	38.0	60.1	95.8	97.6	98.7
	RoBERTa	34.4	42.9	62.6	97.2	98.3	99.1
5	MiniLM	34.2	42.8	62.7	94.6	97.0	98.1
	RoBERTa	19.0	25.6	58.4	95.5	97.4	98.5
6	MiniLM	-	-	-	-	-	-
	RoBERTa	-	-	-	-	-	-

3.4.2 Synthetic Test Set Performance

We evaluate our fine-tuned models on the synthetic test set. We report the performance of models of pre-trained models from HuggingFace before fine-tuning in table 3.4, table 3.5 and table 3.6 for UNMT, Identity Mapping and Noisy Cloze approaches, respectively. We see that out-of-the-box pre-trained models have low F1 scores for UNMT and the noisy cloze approach, which is expected because models have not fine-tuned to the question-answering task. Performance for the Identity Mapping approach is higher, and we reason that in Identity Mapping question answering task is reduced to finding the fill-in-the-blank masked token. Pre-trained models are trained on Masked Language modeling objective, and predicting a masked token is an easy task for them.

Table 3.4: Model performance on synthetic test set for UNMT approach.

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	unmt	31.7	38.3	60.4
RoBERTa	unmt	34.7	43.1	62.4
SpanBERT	unmt	27.7	34.9	61.9

Table 3.5: Model performance on synthetic test set for Identity Mapping approach.

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	ident	71.7	81.7	93.0
RoBERTa	ident	69.5	77.1	92.6
SpanBERT	ident	61.4	73.2	88.2

We report the performance of our fine-tuned models on synthetic test data in table 3.7, table 3.8 and table 3.9 for UNMT, Identity Mapping and Noisy Cloze approaches,

Table 3.6: Model performance on synthetic test set for Noisy Cloze approach.

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	noisy cloze	12.3	15.9	59.9
RoBERTa	noisy cloze	9.0	11.5	60.4
SpanBERT	noisy cloze	7.3	9.5	58.3

respectively. All three models achieve a high F1 score for all three approaches. Models are able to generalize well on synthetic data and learn the question-answering behavior. We observe that the models trained with the UNMT approach beat our baseline Noisy Cloze approach by around 6% F1. However, the UNMT approach lags little behind the Identity Mapping approach.

Table 3.7: Performance of our fine-tuned models on synthetic test set for UNMT approach

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	unmt	94.6	96.9	98.2
RoBERTa	unmt	96.5	97.9	98.7
SpanBERT	unmt	95.7	97.4	98.6

Table 3.8: Performance of our fine-tuned models on synthetic test set for Identity Mapping approach.

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	ident	98.7	99.1	99.4
RoBERTa	ident	99.0	99.3	99.5
SpanBERT	ident	99.0	99.3	99.5

Table 3.9: Performance of our fine-tuned models on synthetic test set for Noisy Cloze approach.

Model	Syn. Data Generation Approach	Exact Match (EM)	F1 Score	Top_4
MiniLM	noisy	84.2	89.2	92.2
RoBERTa	noisy	88.4	91.8	93.8
SpanBERT	noisy	86.8	90.6	93.3

3.4.3 Ground Truth Evaluation

We evaluate our fine-tuned models on the manually annotated dataset of 453 samples. We report our results in table 3.10, table 3.11 and table 3.12 for UNMT, Identity Mapping and Noisy Cloze approaches, respectively. We note that all three models for all three approaches achieve more than 80% F1 score. All the models are able to generalize well on our ground truth (real world) data. This is a positive result in the argument for synthetic data generation in a low-resource setting.

We compare the models trained with the UNMT approach and models trained with our baseline noisy cloze approach. We find that the UNMT approach exceeds in performance by 1%-3% in the F1 score. We argue that the UNMT approach does show promise of application in use-cases similar to ours. We do not find much difference comparing UNMT, and Identity Mapping approaches. While UNMT is slightly ahead in performance for MiniLM and SpanBERT, it lags behind by 0.3% F1 score for RoBERTa. Although, the UNMT approach gives a better Exact Match score for all three models.

Table 3.10: Performance of our fine-tuned models on ground truth dataset for UNMT approach

Model	Syn. Data Generation Approach	Exact Match (EM)	F1 Score	Top_4
MiniLM	unmt	61.5	84.9	92.8
RoBERTa	unmt	61.5	83.7	92.2
SpanBERT	unmt	62.4	84.2	92.8

Table 3.11: Performance of our fine-tuned models on ground truth dataset for Identity Mapping approach.

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	ident	56.8	84.2	94.2
RoBERTa	ident	55.3	84.0	94.0
SpanBERT	ident	57.1	83.9	93.7

Table 3.12: Performance of our fine-tuned models on ground truth dataset for Noisy Cloze approach.

Model	**Syn. Data Generation Approach**	**Exact Match (EM)**	**F1 Score**	**Top_4**
MiniLM	noisy	59.5	83.9	92.0
RoBERTa	noisy	57.1	81.7	92.6
SpanBERT	noisy	57.1	81.0	90.8

4 Conclusion

Question answering task in an enterprise setting poses a tough challenge due to the lack of annotated training data due to domain mismatch. In this problem setting, unsupervised data generation for training question answering models is a good approach compared to the resource-intensive process of annotating training data. We explored Unsupervised Question Answering using Cloze Generation and used its UNMT model to generate synthetic training data for question-answering models. We also explored three state-of-the-art transformer-based models to understand their training approaches. We chose RoBERTa, SpanBERT, and MiniLM to pick different training approaches and model sizes from the best available performing models to evaluate their performance on our approach.

From the results we have seen so far, we can say that the unsupervised approach to generate synthetic data for question-answering tasks shows promise. It is significantly useful in use cases at Siemens where domain-specific training data is unavailable. We propose a simple approach to train a question-answering system specific to Electric Power System Component Domain. Our approach can also work with any domain in the English language by using a different source of raw data.

We also contributed a manually labeled small dataset of 453 samples in the Electric Power System Component Domain. We reiterate that effort required to generate quality data is resource intensive and not feasible in an enterprise setting due to high costs.

4.1 Future Work

While we see a promise in unsupervised approach to generate data for training question answering system, we propose further work in building a larger dataset using annotators with expert domain knowledge. Another experiment that can be undertaken will be to try the approach on other domains and validate the applicability of unsupervised question answering in low resource setting. We also recommend experiment with few shot learning and study the impact on performance when question answering systems are trained on synthetic dataset and additionally trained on a small real world.

List of Figures

List of Tables

Bibliography

[22] *Hugging Face – The AI community building the future.* May 3, 2022. URL: `https://huggingface.co/` (visited on 05/03/2022).

[Agu+20] G. Aguilar, Y. Ling, Y. Zhang, B. Yao, X. Fan, and C. Guo. "Knowledge distillation from internal representations." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05. 2020, pp. 7350–7357.

[ALA18] M. Artetxe, G. Labaka, and E. Agirre. "Unsupervised statistical machine translation." In: *arXiv preprint arXiv:1809.01272* (2018).

[Bae+19] A. Baevski, S. Edunov, Y. Liu, L. Zettlemoyer, and M. Auli. "Cloze-driven pretraining of self-attention networks." In: *arXiv preprint arXiv:1903.07785* (2019).

[Ber+14a] J. Berant, V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and C. D. Manning. "Modeling biological processes for reading comprehension." In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1499–1510.

[Ber+14b] J. Berant, V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and C. D. Manning. "Modeling biological processes for reading comprehension." In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1499–1510.

[CL19] A. Conneau and G. Lample. "Cross-lingual language model pretraining." In: *Advances in neural information processing systems* 32 (2019).

[DC18] X. Du and C. Cardie. "Harvesting paragraph-level question-answer pairs from wikipedia." In: *arXiv preprint arXiv:1805.05942* (2018).

[Dev+18] J. Devlin, M. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *CoRR* abs/1810.04805 (2018). arXiv: `1810.04805`.

[DSC17] X. Du, J. Shao, and C. Cardie. "Learning to ask: Neural question generation for reading comprehension." In: *arXiv preprint arXiv:1705.00106* (2017).

[He+18] L. He, K. Lee, O. Levy, and L. Zettlemoyer. "Jointly predicting predicates and arguments in neural semantic role labeling." In: *arXiv preprint arXiv:1805.04787* (2018).

[Hil+15] F. Hill, A. Bordes, S. Chopra, and J. Weston. "The goldilocks principle: Reading children's books with explicit memory representations." In: *arXiv preprint arXiv:1511.02301* (2015).

[HR18] J. Howard and S. Ruder. "Universal language model fine-tuning for text classification." In: *arXiv preprint arXiv:1801.06146* (2018).

[HR19] T. Hosking and S. Riedel. "Evaluating rewards for question generation models." In: *arXiv preprint arXiv:1902.11049* (2019).

[HS10] M. Heilman and N. A. Smith. "Good question! statistical ranking for question generation." In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. 2010, pp. 609–617.

[Jia+19] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. "Tinybert: Distilling bert for natural language understanding." In: *arXiv preprint arXiv:1909.10351* (2019).

[Jos+20] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. "Spanbert: Improving pre-training by representing and predicting spans." In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 64–77.

[JSS19] G. Jawahar, B. Sagot, and D. Seddah. "What does BERT learn about the structure of language?" In: *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*. 2019.

[Lai+17] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. "Race: Large-scale reading comprehension dataset from examinations." In: *arXiv preprint arXiv:1704.04683* (2017).

[Lam+18] G. Lample, M. Ott, A. Conneau, L. Denoyer, and M. Ranzato. "Phrase-based & neural unsupervised machine translation." In: *arXiv preprint arXiv:1804.07755* (2018).

[LDR19] P. Lewis, L. Denoyer, and S. Riedel. "Unsupervised question answering by cloze translation." In: *arXiv preprint arXiv:1906.04980* (2019).

[Lee+16] K. Lee, S. Salant, T. Kwiatkowski, A. Parikh, D. Das, and J. Berant. "Learning recurrent span representations for extractive question answering." In: *arXiv preprint arXiv:1611.01436* (2016).

[Liu+19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. "Roberta: A robustly optimized bert pretraining approach." In: *arXiv preprint arXiv:1907.11692* (2019).

[MA20] S. Mukherjee and A. Awadallah. "XtremeDistil: Multi-stage distillation for massive multilingual models." In: *arXiv preprint arXiv:2004.05686* (2020).

[Met21] Meta. *PetScan/en — Meta, discussion about Wikimedia projects*. 2021. URL: `https://meta.wikimedia.org/w/index.php?title=PetScan/en&oldid=21883648` (visited on 03/01/2010).

[Mir+20] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh. "Improved Knowledge Distillation via Teacher Assistant." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020), pp. 5191–5198. DOI: `10.1609/aaai.v34i04.5963`.

[Nag] S. Nagel. *News Dataset Available – Common Crawl*. `https://commoncrawl.org/2016/10/news-dataset-available/`. (Accessed on 08/09/2022).

[Ott+18] M. Ott, S. Edunov, D. Grangier, and M. Auli. "Scaling Neural Machine Translation." In: *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1–9. DOI: `10.18653/v1/W18-6301`.

[Rad+18] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. "Improving language understanding by generative pre-training." In: (2018).

[Raj+16] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. "SQuAD: 100, 000+ Questions for Machine Comprehension of Text." In: *CoRR* abs/1606.05250 (2016). arXiv: `1606.05250`.

[RBR13] M. Richardson, C. J. Burges, and E. Renshaw. "Mctest: A challenge dataset for the open-domain machine comprehension of text." In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 193–203.

[RJL18] P. Rajpurkar, R. Jia, and P. Liang. "Know What You Don't Know: Unanswerable Questions for SQuAD." In: *Association for Computational Linguistics (ACL)*. 2018.

[Rou19] S. Roukos. *Question Answering for Enterprise Use Cases | by Salim Roukos | Towards Data Science*. Nov. 4, 2019. URL: `https://towardsdatascience.com/question-answering-for-enterprise-use-cases-70ed39b74296` (visited on 05/03/2022).

[Soc+13] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. “Recursive deep models for semantic compositionality over a sentiment treebank.” In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[Sun+19a] S. Sun, Y. Cheng, Z. Gan, and J. Liu. “Patient knowledge distillation for bert model compression.” In: *arXiv preprint arXiv:1908.09355* (2019).

[Sun+19b] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. “Mobilebert: Task-agnostic compression of bert by progressive knowledge transfer, 2019.” In: *URL https://openreview. net/pdf* (2019).

[Vas+17] 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* 30 (2017).

[Wan+20] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. “MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers.” In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 5776–5788.

[Yan+19] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. “Xlnet: Generalized autoregressive pretraining for language understanding.” In: *Advances in neural information processing systems* 32 (2019).

[You+19] Y. You, J. Li, J. Hseu, X. Song, J. Demmel, and C.-J. Hsieh. “Reducing BERT pre-training time from 3 days to 76 minutes.” In: *arXiv preprint arXiv:1904.00962* (2019).

[Zhu+15] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.” In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27.