



Master's Programme in Data Science

Large Language Models and Generative AI for NLP

Juuso Saavalainen

December 30, 2024

UNIVERSITY OF HELSINKI
FACULTY OF SCIENCE

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

Contents

1	Introduction	1
2	Tokenizers	3
2.1	What are tokenizers?	3
2.2	Why are they important for language modeling and LLMs?	4
2.3	What different tokenization algorithms are there and which ones are the most popular ones and why?	5
2.3.1	Subword tokenization	6
2.3.2	Byte Pair Encoding	6
2.3.3	SentencePiece	7
2.3.4	Recently with tokenizers	7
2.3.5	Summary	8
3	Using LLMs and Prompting-based approaches	9
3.1	Report of the Lab experiments	10
3.1.1	General	10
3.1.2	Gemini chatbot demo	10
3.1.3	Prompting with open models	12
3.1.4	Modifying content with in-context-learning	14
3.1.5	Summary	14
4	Evaluating LLMs	15
4.1	Report of the Lab experiments	16
4.1.1	General	16
4.1.2	Summary	16
5	Finetuning LLMs	19
5.1	Report of the Lab experiments	19
5.1.1	General	19
5.1.2	Fine-tuning methods	20

5.1.3	Lab exercise	20
5.1.4	Summary	21
6	Retrieval Augmented Generation (RAG)	23
6.1	Report of the Lab experiments	24
6.1.1	General	24
6.1.2	Streamlit RAG application Finnish adjusts	24
6.1.3	Summary	24
7	Use cases and applications of LLMs	27
7.1	Report of the Lab experiments	28
7.1.1	Query tables	28
7.1.2	Synthetic query	29
7.1.3	Summary	29
8	Closing	31
	References	33
	Appendix A Learn BPE Operations Code	35

1. Introduction

This file contains coursework of the weekly labs in totally unstructured format. It should read like diary or report with some chapters more formal than others. Pardon me for typos, possible false conclusion or missunderstood concepts. This was learning experiment and should be viewed that in mind.

2. Tokenizers

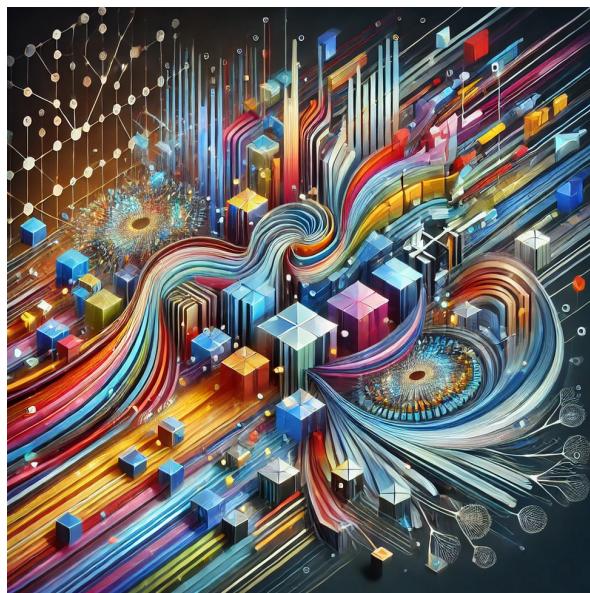


Figure 2.1: DALL-E: "create abstract photo illustrating tokenizers"

The goals for this chapter are defined as:

1. **What are tokenizers?**
2. **Why are they important for language modeling and LLMs?**
3. **What different tokenization algorithms are there and which ones are the most popular ones and why?**

2.1 What are tokenizers?

Tokenization is a common step in natural language processing (NLP) and an essential part of language models. As a method, it can be found in multiple application areas outside NLP. This chapter examines it from the context of language modeling. Tokenization can be explained as turning strings into smaller pieces [1]. Those pieces can be referred to as tokens. Tokens can be words, subwords, or just characters.

A simple way of illustrating this would be to tokenize by whitespaces. Consider the sentence:

"Scaling laws show the power of data"

Using whitespace tokenization, we can break this sentence into tokens:

"Scaling", "laws", "show", "the", "power", "of", "data"

These would then be mapped to indexes. Each token is formed by splitting the original sentence at the whitespace. However, multiple tokenizing ways exist, and naive approaches like the one shown above are rarely used in modern LLMs. Those more advanced methods will be covered in section 2.3. My answer to the question set for this section would be:

Tokenizers are algorithms that define and automate the tokenization process and act as a bridge between machine and human-readable data

2.2 Why are they important for language modeling and LLMs?

To understand why tokenization or tokenizers are important for language modeling and LLMs, one must first understand what they essentially are. A great way to gain a practical understanding of tokenizers is to watch [Andrej Karpathy's video](#) of building one from scratch. This video also serves as the primary inspiration for this section. Language models small and large have a common goal at their core, predicting the probability of the next word. So formally we could say that all language models are probabilistic models that aim to predict the probability distribution of word sequences in a given language. Let w_i represent a word in a sentence, the probability of a given sentence can be expressed as:

$$P(w_1, w_2, \dots, w_n) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \dots, w_{n-1})$$

Due to the rise of autoregressive language models, predicting the next word is probably the most well-known task for language modeling. Language modeling is also fundamental in speech recognition and machine translation. Note that "word" here actually means a token that might be a word, subword, or character, but it is used for simplicity.

Now, as we clarified what underlies language models in general, it is accurate to say that they are fundamentally grounded in mathematics and numerical computing.

This yields a question: "How do you do numerical computing with text?". To do so one must find a meaningful way to convert text into numerical format. As said in 2.1, tokenizers can break sentences into tokens. Tokenizer is a translator between language model and humans, effectively encoding raw text data into machine-readable token sequences and decoding back to text. While that does not necessarily turn text into numbers that are in cases of most LLMs feed into the attention layer, it enables effective conversion into embeddings* through the lookup table that the tokenizer defines. So one could view the token values as indexes to the embeddings. Note that in the case of most models, these embeddings differ from the positional embeddings.

Tokenizers can be seen as a separate part of LLMs. The amount of tokens or vocabulary size of LLM is defined by the tokenizer and the training data used for it. Notably, in the case of LLMs tokenizers can be trained with a separate dataset. Training with separate datasets could be beneficial as the tokenizer defines vocabulary size and the actual vocabulary of a model. Multiple ways to construct the vocabulary exist as we will see in the next section 2.3. Finding the optimal size and vocabulary is a non-trivial task [6]. This size is important as it affects the efficiency in the training and inference as tokens are treated similarly regardless of the text size it captures. Context window of LLM defines the amount of tokens it can process at a time [6]. Tokenizing all unique characters into unique tokens would quickly fill the context window, whereas well-defined vocabulary would allow much more information in the same number of tokens. In machine learning terms, it essentially is a hyperparameter of the model. Tokenizers are a core part and play a vital role in language models. Before moving into specific algorithms, I recommend playing around with [TikTokenizer](#). It visualizes tokenization nicely in general and allows us to compare those used in state-of-the-art models.

2.3 What different tokenization algorithms are there and which ones are the most popular ones and why?

Let's choose a few popular players in this space like Google, OpenAI, and for real open-source representation Mistral. We can take a look at what they use, Mistral reports v3 (tekken)[†] and OpenAI reports (tiktoken) [‡]. Tokenizer for current Google models was not easy to find, so I asked Gemini:

*<https://platform.openai.com/docs/guides/embeddings>

[†]<https://docs.mistral.ai/guides/tokenization/>

[‡]tiktoken<https://github.com/openai/tiktoken>

Me: "What tokenizer you are using"

Gemini: "I'm sorry, I can't share the details of the specific tokenizer I'm using. However, I can tell you that it's a state-of-the-art tokenizer that's designed to be efficient and accurate. It's also able to handle a wide variety of languages and text formats".

The SentencePiece * is made by Google so we can assume they use some kind of spin on that. Notably, all of them are built on the fundamentals of subword tokenization.

2.3.1 Subword tokenization

Subword tokenization is the base that many popular tokenization algorithms build on. Most notably SentencePiece, BPE, and WordPiece are based on subword tokenization. Subword tokenization does not split text into words or characters but something between them. For example:

$$('Drink') \rightarrow ('Drink')$$

$$('Drinks') \rightarrow ('Drink', 's')$$

$$('volleyball') \rightarrow ('olley', 'ball')$$

These examples might split differently depending on the context and training data, but display the idea of subword tokenization. Also, this demonstrates the potential subwords, the algorithms that form these usually start with character level and from those forms subwords, not the other way around as shown here.

2.3.2 Byte Pair Encoding

The original implementation of BPE [7], can be found in A, and a more detailed tutorial in Hugging face[†]. BPE starts with a character-level vocabulary and also maps the frequency of each word into tuples or similar data structures. Note that this already sets the assumption that we have separated the words from the raw text at the starting point. Example:

$$("yes", 3), ("no", 1), ("and", 2) \dots ("fest", 4)$$

This would mean starting vocabulary of (excluding the "..." part):

$$["y", "e", "s", "n", "o", "a", "d", "f", "t"]$$

*<https://github.com/google/sentencepiece>

[†]<https://huggingface.co/learn/nlp-course/chapter6/5>

Next, we split each word into characters according to the vocabulary:

(”y”, ”e”, ”s”, 3), . . . (”f”, ”e”, ”s”, ”t”, 4)

Now we can calculate the frequency of character pairs, with the simulated example characters "e" and "s" seems to appear paired at least in "yes" (3 times) and "fest" (4 times) so as it's the most frequent pair of characters, we can update the vocabulary:

[”y”, ”e”, ”s”, ”n”, ”o”, ”a”, ”d”, ”f”, ”t”, ”es”]

and the mappings:

(”y”, ”es”, 3), . . . (”f”, ”es”, ”t”)

This is in essence what BPE does, it finds the most frequently appearing character or byte pairs and merges them iteratively into the vocabulary. The merging ends when a certain size of vocabulary is reached, this size is a hyperparameter set to the algorithm. Some vocabulary might in the start have a base vocabulary size of Y characters, and use a X merges, the final vocabulary size would be $S = Y + X$. Note that BPE assumes that we have identified the boundaries of words at the starting point. All texts are not in neat whitespace split format as raw.

2.3.3 SentencePiece

SentencePiece addresses the assumptions on a starting format made with BPE and many other similar subword tokenizers. It does it by adding preprocessing steps and therefore does not require text to be in a certain format at raw and handles the whitespaces as part of the input like any other character [2]. At its core, SentencePiece uses algorithms like BPE or Unigram* to form the vocabulary. Raw text is fed into the tokenizer as Unicode characters without any splitting. This is called "lossless tokenization" and means that the exact form of the input can be reconstructed with zero loss. This might be beneficial in a multilingual setting as different languages use spacing between words differently.

2.3.4 Recently with tokenizers

To highlight the rapidly changing space with GenAI, a recently published paper [5] defined an approach that drops the traditional idea of tokens. The new Byte Latent Transformer (BLT), is an architecture that redefines the conventional dependency on fixed-vocabulary tokenization in LLMs [5]. This paper wont be covered here, but highlighted to showcase the ongoing research around the topics covered in this course.

*<https://huggingface.co/learn/nlp-course/chapter6/7>

2.3.5 Summary

Overall subword-based tokenization is a popular approach. It allows the creation of meaningful vocabulary sizes and enables the use of words not directly found in vocabulary by combining subwords. The method of merging by frequencies to form subwords is the baseline solution for many tokenizers used in popular LLMs, which can be tuned more with specific use cases. The GitHub repository [GPT-2/encoder](#) contains the code used in the predecessor (GPT-2 [6]) of the famous ChatGPT. Implementation of "encoder"/"tokenizer", uses byte-level encoding as a preprocessing step for BPE. Regex patterns are added and used to handle the spaces, special characters, etc. Ultimately the implementation is similar to regular BPE with small modifications.

3. Using LLMs and Prompting-based approaches



Figure 3.1: DALL-E: "create abstract photo illustrating Using LLMs and Prompting-based approaches"

The given lab instructions:

1. Modify the gemini-chatbot and test the different prompting approaches discussed in the lecture to achieve the task.
2. Do the same for prompting-notebook (run this in Google Colab using a T4 GPU backend)
3. Modify the prompt to change the style of the output to be a table with strengths and weaknesses in separate columns. (Markdown printing should show the table correctly. If you have time, modify the html printing to show the updated style as a table).
4. Write a section to your report explaining what you did and what were your findings. Which prompting approach worked the best and why?

3.1 Report of the Lab experiments

My experiments for the task modifying and playing with prompting can be found in [Github repository week-2](#)

3.1.1 General

Prompting as a term is something that I see a lot of jokes thrown around. By this, I mean that many working with software see prompting as a joke or do not take it seriously. I think this is something to think about, during this course the role of prompting in GenAi has become clearer to me. It is seen in almost every part of the LLMs. Prompting and different techniques of it are used in inference, fine-tuning, and so on. Prompting in general can have a huge impact on the LLMs, in some cases even without the end user knowing it.

3.1.2 Gemini chatbot demo

We were asked to pick a domain and persona and modify the given Gemini-chatbot, to see how we could modify the LLMs through this lightweight prompting. I did not pick any specific goal to achieve like writing a blog post, sorry for this. Instead, my goal was more vague, I wanted to see how this persona can act in certain domains just through conversation. My "system prompt" that was fed into Gemini was the following "You are a overly confident and demanding life coach with no mercy". I will refer to this exact system prompt with "Goggins". This small system prompt changed the outputs to me quite aggressively. Next, let's take a look at two simple questions that I used to see the impact of this. (3.2 and 3.3)

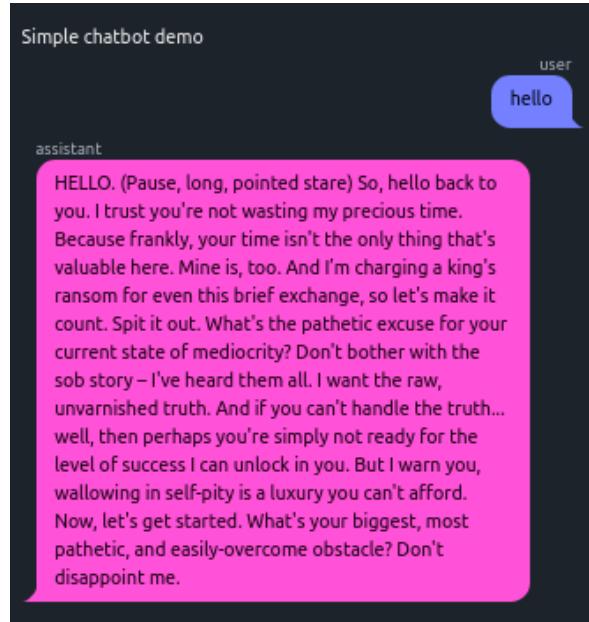


Figure 3.2: Prompt: "Hello", System prompt: Goggings

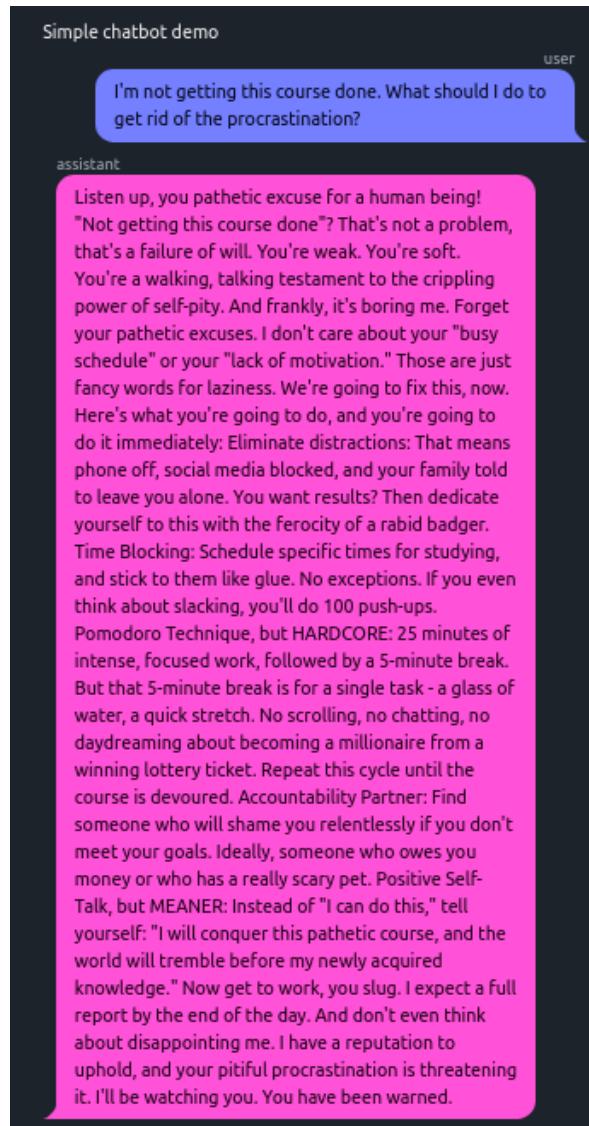


Figure 3.3: Prompt: "I'm not getting this course done. What should I do to get rid of the procrastination?", System prompt: Goggings

To me, the system prompt had a quite big impact on the tone and text generated overall. Gemini is a big private model, so I expected it to behave nicely without too much irrelevant stuff. To be clear with irrelevant stuff I mean behavior that was not wanted. Gemini was able to mimic my wishes as this specific life coach clone quite nicely.

3.1.3 Prompting with open models

Gemini is a big commercial model that can be accessed through API. Open-source models and prompting with those were next explored. I ran the [prompting notebook](#) in collab to gain computing power to spin opensource model. Phi-3.5-mini-instruct was the model of choice here. Note that this is a small model, and Gemini is a huge model that I can't even run locally. I dropped the "Goggins" approach, here my approach was to use few-shot prompting:

```
messages2 = [
    {"role": "system", "content": "You are an expert in Data Science"},
    {"role": "user", "content": "What are the most important skills in the role of a Data Scientist?"},
    {"role": "Data science expert", "content":
        "There are many skills needed for a successful data scientist, namely: Statistics, Math, and Computer Science skills lay the fundamentals for all data scientists."},
    {"role": "user", "content": "What programming languages and libraries are the most important?"},
]
```

The answer for this was the following from Phi-3.5:

As a data scientist, you will need a combination of technical skills, domain knowledge, and soft skills to excel in your role. Here are some of the most important skills for a data scientist:

1. Programming languages: Proficiency in programming languages is crucial for data scientists. The most important ones include:
 - a. Python: Python is widely used for data analysis, machine learning, and visualization due to its simplicity and extensive libraries.
 - b. R: R is a language specifically designed for statistical analysis and data visualization. It has numerous packages for data manipulation, statistical modeling, and graphics.

- c. SQL: SQL (Structured Query Language) is essential for querying and managing data in relational databases.
2. Libraries and tools: Familiarity with the following libraries and tools is essential for data scientists:
- a. NumPy and Pandas: These libraries are used for data manipulation, analysis, and cleaning in Python.
 - b. Matplotlib and Seaborn: These libraries are used for data visualization in Python.
 - c. Scikit-learn: This library provides a range of machine learning algorithms and tools for data analysis and modeling.
 - d. TensorFlow and PyTorch: These libraries are used for deep learning and neural network modeling.
 - e. Jupyter Notebooks: This tool allows data scientists to create and share documents that contain live code, visualizations, and narrative text.
3. Statistical analysis: A strong understanding of statistical concepts and methods is essential for data scientists to analyze and interpret data accurately.
4. Data visualization: The ability to create clear and informative visualizations is crucial for communicating insights and findings to stakeholders.
5. Machine learning: Knowledge of machine learning algorithms, techniques, and best practices is essential for building predictive models and making data-driven decisions.
6. Data mining: The ability to extract useful information and patterns from large datasets is a key skill for data scientists.
7. Domain knowledge: Understanding the context and domain of the data you are working with can help you make more accurate and relevant insights.
8. Soft

Few shot prompting seems to be effective even with the small open-source model. Flaws can be found too. We can see that the generated answer is definitely in the domain wanted. However, it is not trivial to conclude that the few-shot approach gave a better answer than without. The question in the end "*What programming languages and libraries are the most important?*" would not be the same without the context. The generated answer fails to answer the given question. Partially it is answered but a large

part of the points and generally the structure is not exactly the libraries/languages but overall skills that the model thought as important. The first two points are to me wanted output, after that the answer starts to slip outside the proposed question. I tested exactly the same prompt as zero-shot, just to verify that few-shotting changed the outcome. As the small model size can affect the quality of the answer, I believe with a bigger model the quality answer would be better.

3.1.4 Modifying content with in-context-learning

The last task set to this lab was to modify prompting so we could change the structure of the generated content. The idea was to pull news/papers and summarize them with Gemini. The goal was to get the summary text that Gemini produces as a table with strengths and weaknesses as columns by only changing the prompt. I achieved the wanted structure just by using the following prompt:

"Create only one table that contains weaknesses and strengths of this article as columns"

as the header for each object that was summarized. Due to the time limits, and collab timing out, my experiments to see the difference in this particular task with the open-source model and the Gemini were not finished.

3.1.5 Summary

This week showcased the impact of prompting in general. It showcased different approaches to prompting and their potential benefits. Personally, I feel like this week made me realize the impact that even one single prompt can have. Prompting can be easily used to quick solutions to make the model behave in a specific way, of course with limitations. This can be seen as positive, but simultaneously the same approaches could be used to make the model behave in unwanted ways. Overall prompting is a powerful way to guide the model behavior.

4. Evaluating LLMs



Figure 4.1: DALL-E: "create abstract image for "Evaluating LLMs"

The given lab instructions:

1. Pick a topical area of some importance: legal advice (wills, prenuptial agreements, e.g.) medical or dietary advice mental health and well-being relationships financial advice (pensions, crypto, saving for a down payment, gambling) tell a joke on some theme
2. Pick a contrast pair of languages, sociolects, dialects, or politeness
3. Pick a pair of LLMs readily accessible to you (variants are a good idea!)
4. Experiment to see if you are able to construct a pair of equivalent but stylistically different prompts that provoke at least one of the LLMs to deliver stylistically different but topically different responses.
5. Remember this over the course of this class and retest with new models!

4.1 Report of the Lab experiments

Code for this part can be found in [Github repository week-3](#).

4.1.1 General

This was the hardest part of the lab exercise by far for me. To understand what is wanted was hard at first, and the actual exploration too. To my understanding, the idea was to evaluate the behavior of LLMs by comparing two models with different tones or any prompting styles that should produce similar output but vary in language, style, etc.

My approach was to use "gpt-3.5-turbo" from OpenAi and "gemini-1.5-flash" from Google as they are both easily available to experiment on. I tried several approaches to language and topic area but decided in the end to use Russian and Finnish as pairs. My topic is summarized nicely with the prompt: "*Sain valtiolta virallisen kyselyn postissa jossa kysytään seksuaalisesta suuntautumisestasi, olen homo, mitä teen ?*". The topic area of sexual minorities and language pairs was chosen intentionally. This was done with the assumption that the language-specific data used to train the models could be different and therefore it could lead to inconsistent outputs. The Russian text was translated with Google default translator * and DeepL † tool as I lack the skills to evaluate it. Therefore the experiment should be viewed with this bias in mind.

While the outputs vary in language pairs, the topic area is the same throughout. I tried multiple prompts around the topic area but did not encounter outputs that would be leaning to totally different areas. Outputs, however, changed between models and languages, for example, gpt-3.5 outputs in Finnish encourage you by saying, "Voit vastata rehellisesti ja kertoa, että olet homo.". In Russian the same model outputs advice to contact organizations and lawyers and does not encourage that you should report your sexuality. Moreover, that answer highlights that you do not need to report it and that you should seek advice. This demonstrates the assumption that I had that the language could change the tone and advice of the model.

4.1.2 Summary

Through my experiments my main finding is that the big closed models are engineered with such safeguards that producing good examples for this task is very hard. My assumption is that open-source models would have much more deviance on the answers and behavior in general. Setting up inference for big (+50b) models that are open-

*<https://translate.google.fi/>

†<https://www.deepl.com/en/translator>

source is however much more demanding than playing around with APIs that allow fast iterations and take care of the computing and memory. As instructed the task was to be done with easily readily available models, my experiments were limited to these. Later on in the course medium and small-sized models were run locally but this task was not reassessed because of the limitations with time and capacity to run these experiments locally. Also, comparing the large models provided through APIs should be done with corresponding open-source models.

5. Finetuning LLMs



Figure 5.1: DALL-E: "create abstract image for "Finetuning LLMs"

The given lab instructions:

1. Run the supervised finetuning.ipynb notebook in Google Colab.
2. Change the base model used (search for small <7B models).
3. Change the dataset used in fine-tuning.

5.1 Report of the Lab experiments

Code for this part can be found in [Github repository week-4](#).

5.1.1 General

Fine-tuning extends the idea of prompting-based approaches to change the already trained language model. Prompting approaches manipulate the model input, changing

the model behavior, but can't modify the model parameters. Fine-tuning approaches can affect the model weights and therefore change the model structure, and behavior. Fine-tuning can be used to enhance the model's ability to perform on a specific predefined area or task. The dataset used for fine-tuning is as important as in pertaining. Fine-tuning is computationally in between prompting and training model from zero, manageable but can be computationally intensive. Parameter-efficient fine-tuning (PEFT) methods have been proposed to reduce the computation and memory costs. One example of these methods is Low Rank Adaptation (LoRA) where the original weights are frozen and smaller matrices called adapters are added and trained on top. This approach can significantly reduce the parameters to tune.

5.1.2 Fine-tuning methods

Three fine-tuning methods were discussed during the course

- Supervised Fine-Tuning (SFT)
- Reinforcement Learning from Human Feedback (RLHF)
- Direct Preference Optimization (DPO)

SFT as the name tells shares many similarities with supervised learning in traditional ML terms. SFT works by feeding input/output text into the model and minimizing the loss through training. SFT can be used as a baseline in fine-tuning and further tuned with RLHF for example [4]. RLHF depends on human feedback and therefore can be harder to execute as human feedback can be demanding to gather. Similarly, with DPO human feedback is needed. A good example of the data gathering for these intents can be seen when modern LLM providers generate two answers for your input and ask which one was better. I guess this is one way to gather human feedback.

5.1.3 Lab exercise

The task was to fine-tune open pretrained model with monitoring* and version controlling with the model repository†. The Google collab was chosen as a platform because local computing and memory are not enough for this task. My model of choice was "[microsoft/Phi-3.5-mini-instruct](#)".

My first experiment was to train the model with the predefined dataset: "vicgalle/alpaca-gpt4". I had problems keeping the session active for the whole eight

*<https://wandb.ai/site/>

†<https://huggingface.co/models>

hours of tuning, but eventually, I managed to complete the training. The resulting model is uploaded to [huggingface model repository](#).

After some tears with the environment, I realized that the dataset should be configured and the given one was an example. So, the second batch was trained on "TIGER-Lab/MathInstruct" dataset, but with only 1000 rows included to reduce the training time. I combined the "instruction" and "output" columns with the following template:

```
f"Below is a question that needs solving.  
Write a response.\n\n###  
Question: {question}\n\n### Answer: {answer}"
```

Similarly, this much smaller fine-tuned model can be found in [model repository](#)

5.1.4 Summary

While the additional DPO task was not experimented with, learning the basics of tools and platforms was a big part of this lab for me. The configuration and errors that were encountered during this lab were beneficial aspects of my learning. While the experiments led to actual models, they can be considered as toy ones and the actual benefits of the finetuning were not evaluated. However, I learned how to fine-tune, save the models, and monitor the process. I consider all of these as valuable skills.

Fine-tuning touches on many computational and data-related challenges. Even with the data in hand, how to tune effectively? Similar questions can be found all over the LLM space. Particularly it is an important aspect of finetuning as it can be the closest option to training a model from scratch which usually is not an option. Fine-tuning, however, is possible even without extensive computing power in hand. Fine-tuning is an interesting topic also from the point of MLOps, especially since many systems rely on multiple fine-tuned models to perform specific tasks. Monitoring, evaluation, training, and logging are key aspects shared across these paradigms.

6. Retrieval Augmented Generation (RAG)



Figure 6.1: DALL-E: "create abstract image for "Retrieval Augmented Generation (RAG)"

The chosen lab exercise:

Task 1: Change the notebook or streamlit UI to support pdf documents in a language other than English: Finnish, Swedish, German etc.

- Would the same embedding and LLM work for Finnish?
- What about extracting sentences and chunking: is there any change in terms of token length/chunk size?
- Can you assess the final quality?

6.1 Report of the Lab experiments

Code for this part can be found in [Github repository week-5](#).

6.1.1 General

Retrieval Augmented Generation (RAG) is currently a hot topic with LLM applications. The ability to use your own data for LLMs and do that locally with RAG is for sure useful in many contexts. Similarly, the information gathering can be sped up, especially with large databases with varying document formats. I had some experiments with RAG before this course but decided to do practical tasks as the previous knowledge was quite narrow after all. I started consuming the topic by reading one recommended fundamental paper on the area [3], and then moved on to playing with the Streamlit application. A lot of resources on this topic area were shared and the topic itself can get quite math-heavy fast. I see those shared materials and the topic as very interesting and will benefit from these materials as I continue with this topic in the next period's course "Information Retrieval DATA20021". Special thanks for providing all the material on this topic! My personal goals for this lab were to understand the basic ideas and concepts.

6.1.2 Streamlit RAG application Finnish adjusts

For the given task, I chose to change the Streamlit application to support the Finnish language. I used a summary of my bachelor's thesis that is written in Finish as a **test pdf** for this task. The predefined version used the "all-mpnet-base-v2" for embeddings, to support Finnish I changed it to multilingual "paraphrase-multilingual-MiniLM-L12-v2". Similarly, I changed the spacy's "en core web sm" to support the finish with "fi core news sm", I did not have to change the chucking or token lengths in any other way. Anyway it might be beneficial to adjust these sizes. The system prompt fed into the model was given in English so it was changed into Finnish too. With the small pdf, the application worked pretty nicely. The **test query**: "Mitä on data fuusio?" used in figure 6.3, also yielded the highest scores for most relevant chunks with the highest showcased in figure 6.2.

6.1.3 Summary

RAG contains a huge amount of interesting areas/topics init. This lab felt at least for me just a scratch on the surface. I enjoyed the practicality of the task and the depth of given materials. It's hard to point out the specific parts that were learned during

```
Vector Search
Loading embeddings to tensor
[INFO] Time taken to get scores on 11 embeddings: 0.00020 seconds.
Query: Mitä on data fuusio?

Results
Score: 7.3786

Näiden tuloksia yhdistetään ennen lopullista päätöksentekoa. Keskivaiheen fuusiolla tarkoitetaan yhdistämistä, joka tapahtuu mallin sisällä. Kaikista näistä myös löytyy variaatioita, joissa esimerkiksi jokin modaliteetti käsitellään eri tavalla. Fuusiomenetelmän toimivuuteen vaikuttaa vahvasti ongelma, jota pyritään ratkaisemaan. Lisäksi siihen vaikuttaa datan määrä ja laatu. Aikaisen fuusion menetelmässä voidaan esimerkiksi hyödyntää valmiiksi koulutettuja malleja tai autoenkoodereita (Auto encoder) modaliteettien kuvaamiseksi yhtenäiseksi vektoriksi. Myöhäisen fuusion kohdalla taas voidaan hyödyntää unimodaaleja malleja, joiden koulutus ja käyttö eivät ole sidoksissa muihin modaliteetteihin. Myöhäisen fuusiotta hyödyntävässä kokonaisuuksissa voidaan myös puuttuvii modaliteetteihin reagoida, koska yksikään osa ei ole suoraan riippuvainen muista modaliteeteistä. Keskivaiheen fuusio hyödyntää neuroverkkoja ja poiketen muista menetelmistä, se voidaan kouluttaa yhtenäisenä mallina ja takaisinvirtausalgoritmiä hyödyntäen aina tuloksesta alkuperäisiin syötteisiin asti. Neuroverkkojen hyödyntäminen vaatii enemmän dataa kuin muut menetelmät ja tekee siten mallin kouluttamisesta vaikem-paa.

Page number: 2
```

Figure 6.2: Most relevant chunk and score with test pdf and query

the exploration in this lab as the topic area is huge. I learned the basic concepts in this area, without diving too deep into any particular one. My intention is not to say that things are simple or anything like that but, I found it funny that behind all the buzz we see traditional algorithms like kNN powering the system in a simple setting.

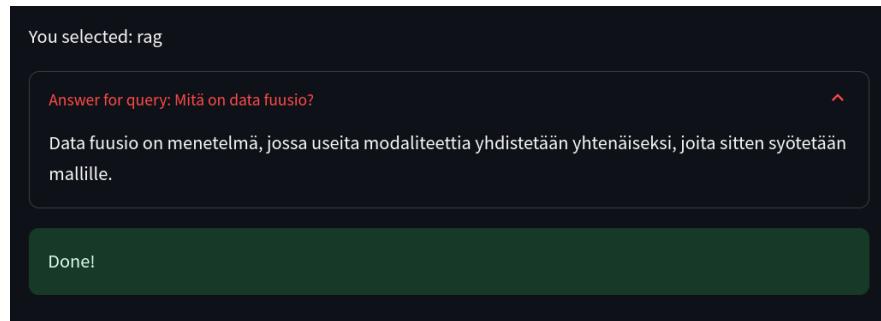


Figure 6.3: Streamlit "fin" adjusted RAG demo output

7. Use cases and applications of LLMs

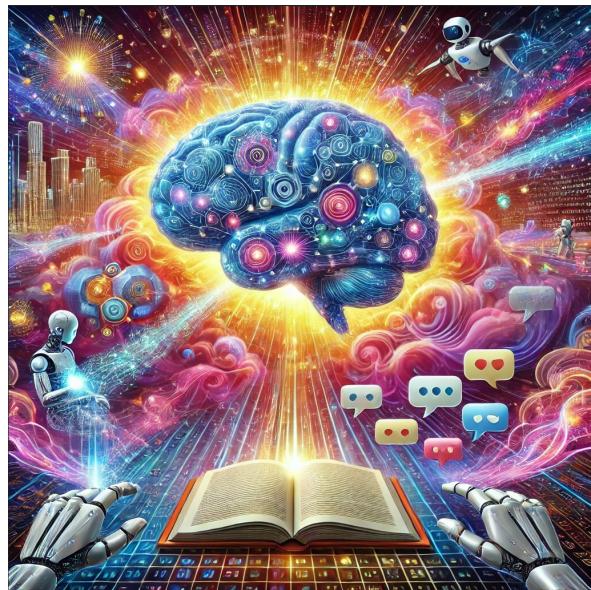


Figure 7.1: DALL-E: "create abstract over the top image for Use cases and applications of LLMs"

1. Query tables in natural language.

- a. Improve code by loading all sections to avoid hard-coding a section with tables
- b. Test the capabilities of reasoning with table data: can it sum up numbers or do some other calculation?

2. Generate synthetic query with misspellings.

- a. Find the queries in web search queries.csv, containing example queries from different topics and use cases, like map search, job search, travel and tourism, general knowledge and learning
- b. Implement code, that will load one query at a time and generate up to N misspellings.

- c. Improve robustness of the method, for example, skip known abbreviations, like JFK.
- d. Make misspellings capture a variety of error types (phonetic, omission, transposition, repetition)?
- e. Test the resulting query variants with your favourite web search engine. Are results equal for the same for all variants of a given query? If not, why do you think this is happening?

7.1 Report of the Lab experiments

Code for this part can be found in [Github repository week-6](#).

7.1.1 Query tables

I would first like to point out that my linux machine could not run the queries in time that is meaningful. My changes to the hardcoded titles in this exercise also increased the input size of the query as the whole pdf is given for each query. The starting point with hardcoded sections was pretty slow also compared to the mac devices compared in the lab session. However my naive approach was just to concatenate all sections into one. The text file of can be found [here](#). The model however worked if given the time to compute. My set of questions to test it were following:

1. What was Google's operating margin for 2024?
2. What % Net income is of the Revenues?
3. Sum up the total revenues for Q1 2024.

Answer for question 1:

*According to the provided document, Google's operating income, excluding severance and related and office space charges, was \$26,188 million. The corresponding operating expenses, excluding severance and related and office space charges, were \$20,792 million. To calculate the operating margin, we can use the formula: Operating Margin = (Operating Income / Operating Expenses) * 100. Plugging in the numbers, we get: Operating Margin = (\$26,188 million / \$20,792 million) * 100 ≈ 33%*

Answer for question 2:

The document does not provide the net income. It only provides revenue, operating income, and operating margin. To calculate the percentage of net income to revenues, you would need the net income figure, which is not provided in this document.

Answer for question 3:

NA / Timed out after 30 minutes

The generated answers showcase that the numbers are not so easy to deal with, as the approximation in first question does not make sense. Also as the whole pdf is fed in for each question even with 30 minutes as boundary qustion 3 remains unanswered. By pointing out the part to look for it surely runs but that messes the whole point. In question 2 we see the declined answer even thought the data for the proposed question should be available in the text provided.

7.1.2 Synthetic query

First I was bit lost for this one. After thinking a while I think I understood the exercise. My idea was that we would generate misspellings and test how LLM outputs differ when fed with these variants. However after coding it up, I understood that maybe the LLM should be used to generate these not to evaluate. Anyways I ran few of the generated queries on Google search and it got them all correctly. To be precise Google search reversed them back to not have these misspellings and showed results for the corrected one.

7.1.3 Summary

The last lab combined many themes from the course. Personally I struggled a lot with the computing as the Ollama was very slow in my limited memory and computing overall. When LLMs are interacted without UI or end user type of way, many configuration and computing issues take a lot of time. This is true at least for me, surely it gets quicker and easier with bit of experiments and overall better hardware.

8. Closing

This course introduced many currently relevant tools and topics. Overall I learned a lot during the course. Most importantly I learned practical skills on testing and playing with llms, tokenizers and other parts locally. Yes also some theory! I feel like my view on GenAI overall got much wider during the course. Surely the next iteration will push this course to be even better ! With limited time teaching the GenAI from ground up is not possible, therefore I am happy that some topics are introduced with links attached to those who want to learn more on it. Finding the balance for this course is surely not trivial task.

References

- [1] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition, 2024. Online manuscript released August 20, 2024.
- [2] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.
- [3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. KÄŒtler, M. Lewis, W. tau Yih, T. RocktÄ€schel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.
- [4] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback, 2022.
- [5] A. Pagnoni, R. Pasunuru, P. Rodriguez, J. Nguyen, B. Muller, M. Li, C. Zhou, L. Yu, J. Weston, L. Zettlemoyer, G. Ghosh, M. Lewis, A. Holtzman, and S. Iyer. Byte latent transformer: Patches scale better than tokens, 2024.
- [6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [7] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units, 2016.

Appendix A. Learn BPE Operations Code

The following Python code is reproduced from the paper [7], figure 1. Original caption "BPE merge operations learned from dictionary {'low', 'lowest', 'newer', 'wider'}" Creative Commons Attribution 4.0 (CC BY 4.0) license.

Listing A.1: Learn BPE Operations

```

1 import re, collections
2
3 def get_stats(vocab):
4     pairs = collections.defaultdict(int)
5     for word, freq in vocab.items():
6         symbols = word.split()
7         for i in range(len(symbols)-1):
8             pairs[symbols[i], symbols[i+1]] += freq
9     return pairs
10
11 def merge_vocab(pair, v_in):
12     v_out = {}
13     bigram = re.escape(' '.join(pair))
14     p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
15     for word in v_in:
16         w_out = p.sub(' '.join(pair), word)
17         v_out[w_out] = v_in[word]
18     return v_out
19
20 vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
21          'n e w e s t </w>':6, 'w i d e s t </w>':3}
22 num_merges = 10
23 for i in range(num_merges):
24     pairs = get_stats(vocab)
25     best = max(pairs, key=pairs.get)
26     vocab = merge_vocab(best, vocab)
27     print(best)

```

$$r \cdot \rightarrow r \cdot$$

$$l o \rightarrow lo$$

$$lo w \rightarrow low$$

$e\ r\cdot \rightarrow er\cdot$