Large Language Models and Generative AI for NLP

Final Report

Week 1: Tokenizers

What are Tokenizers?

A tokenizer is a fundamental preprocessing component in Natural Language Processing (NLP) pipelines and for Large Language Models (LLMs), a tokenizer is a deterministic software component responsible for converting a string of text into a sequence of integers (input IDs) that the neural network can process.

The process of tokenization involves two main steps:

1. **Segmentation:** The input text is broken down into smaller atomic units called *tokens*. While traditional methods might split text by whitespace (word-level) or break it down into individual letters (character-level), modern LLMs mostly use subword tokenization.
2. **Encoding:** Each unique token is mapped to a specific integer based on a fixed, pretrained vocabulary. This vocabulary represents the entire set of tokens the model has knowledge of.

The final output of a tokenizer is a sequence of integers (vectors), where each number represents a specific piece of the original text. This sequence is then fed into the model's embedding layer.

In the context of Neural Machine Translation, the choice of tokenization strategy is critical. A Naive approach (like simple word splitting) results in an excessively large vocabulary and fails to handle unknown words effectively. Modern tokenizers act as a middle ground between character-level and word-level representations, allowing models to construct meaning from subword units, like morphemes or frequent character sequences.

Why are they important for language modeling and LLMs?

Tokenizers define the granularity of the data fed into the model (in this case LLMs). This choice directly impacts the model's efficiency. LLMs have a limited "context window" which means that there is a maximum amount of information an LLM can process at once. If a tokenizer breaks text into very small pieces, it would consume the context window quickly and could process less actual content. Conversely, a well-designed tokenizer that captures meaningful linguistic units will fit more useful information within the same window.

Beyond context efficiency, tokenizers influence how well a model learns language patterns. If tokens are too granular, the model must learn to combine them into meaningful units

repeatedly. If they are poorly chosen, rare character combinations might become separate tokens, wasting model capacity. Good tokenizers balance efficiency with expressiveness, they should represent common linguistic patterns as single tokens while still being able to handle any possible text through a fallback mechanism.
The most critical reason for modern tokenization strategies is handling words that were not present in the training data. In traditional word-level models, any unseen word (a rare name, a typo, or a new technical term) would be replaced by a generic "Unknown" token (<UNK>), causing a complete loss of meaning. Modern tokenizers solve this by using subword units. As proposed in the paper "Neural Machine Translation of Rare Words with Subword Units" (Sennrich et al., 2015), breaking words down into smaller, meaningful units allows the model to process rare or unknown words effectively. For example, a model might not know the word "unfriendliness" as a whole, but it can recognize the subwords un, friend, li, and ness. By composing these known parts, the model can understand and generate words it has never seen before. This capability is essential for LLMs to be truly "open-vocabulary" systems.

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

While various algorithms exist, **Subword Tokenization** has become the industry standard for LLMs. The two most prominent approaches are Byte-Pair Encoding (BPE) and SentencePiece (often using the Unigram algorithm).

1. **Byte-Pair Encoding (BPE)**: Introduced for NLP by Sennrich et al. (2015), BPE is the most widely used algorithm (powering models like GPT-2, GPT-3, and GPT-4).
    a. **How it works:** It uses a bottom-up strategy. It starts by treating every char as a separate token. Then, it iteratively counts the most frequent pair of adjacent tokens in the corpus and merges them into a new single token. This process repeats until a pre-defined vocabulary size is reached.
    b. **Why it is popular**: It is deterministic and effectively compresses common words into single tokens while keeping rare words as sequences of characters, solving the OOV problem mentioned earlier.
2. **SentencePiece:** Proposed by Kudo and Richardson (2018), SentencePiece is not just an algorithm but a tokenization library that often implements the Unigram language model (a probabilistic, top-down approach).
    a. **The Innovation:** Unlike BPE, which usually requires pre-tokenization (splitting text by spaces first), SentencePiece treats the input as a raw stream of unicode chars (including spaces). It represents spaces with a special symbol (e.g., _), making the process truly language-independent.
    b. **Why it is popular:** This approach is crucial for languages that do not use spaces to separate words like oriental languages and is fully reversible meaning it is lossless. It is the standard for multilingual models like LLaMA, T5, and ALBERT

While BPE builds tokens by merging frequent pairs (Frequency based), SentencePiece (Unigram) starts with a massive vocabulary and removes the tokens that contribute least to the likelihood of the text (Probability-based). Both algorithms aim to balance vocabulary size with sequence length, maximizing computational efficiency.

References

1. **Sennrich, R., Haddow, B., & Birch, A. (2016).** Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1715–1725). Berlin, Germany: Association for Computational Linguistics. Available at: https://arxiv.org/abs/1508.07909
2. **Kudo, T., & Richardson, J. (2018).** SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 66–71). Brussels, Belgium: Association for Computational Linguistics. Available at: https://arxiv.org/abs/1808.06226

Week 2: Using LLMs and Prompting-based approaches

Prompt Engineering Experiments and Findings

The goal of this experiment was to evaluate the effectiveness of different prompting techniques in generating text with a specific, highly constrained person/configuration.
In this specific experiment the configuration I maintained through all the processes was the following:

- Persona: Stereotypical Finnish Police Officer (Formal, serious, direct, pragmatic, no exclamation marks)
- Task: Write a blog post about swimming benefits and rules.
- Models: Google Gemini 2.5 Flash (Cloud/API) and Microsoft Phi-3.5-mini-instruct (Local/HuggingFace)

For the methodology of the experiment we simply tested five specific prompting techniques defined in the course material, the types were the following:

1. **Zero-shot:** Providing a simple instruction without any examples.
2. **Few-Shot:** Providing examples of the desired output (running/gym reports) to guide the style
3. **Chain-of-Thought (CoT):** Encouraging intermediate reasoning steps to structure logic
4. **Prompt Chaining:** Breaking the task into subtask (generating rules first, the the post)
5. **Tree-of-Thought (ToT):** Exploring multiple angles (Safety vs. Health vs. Regulation) and evaluating the best fit.

Comparative Analysis: Gemini vs. Phi-3.5

The most significant finding was the difference in persona adherence between the two models.

- **Gemini (Large Model):** Demonstrated exceptional understanding of the persona across all techniques. Even in Zero-shot, it successfully adopted the “pragmatic/direct” tone (“Rules are not suggestion”). It consistently avoided exclamation marks and kept sentences short.
- **Phi-3.5 (Small Model):** Struggled significantly with the persona. In Zero-Shot, it defaulted to a generic, enthusiastic AI assistant tone (“Swimming…. offers numerous

health benefits!), ignoring the constraint to be “serious” and “direct”. It frequently used exclamation marks and colloquial language, failing to maintain the character.

Detailed findings for technique

- **Few-shot prompting (most effective for style):** This approach provided the best results for the Microsoft model. By seeing examples of how the “Police Officer” described running and the gym, Phi-3.5 was able to mimic the style in the first paragraph of its response. However, without these examples, the local model failed. This confirms that providing examples is crucial for steering smaller models.
- **Chain-of-Thought and Tree-of-Thought (most effective for structure):** For Gemini, CoT produced the most logically sound and authoritative content. By listing hazards first, the final output was structured like a police report. However, for the smaller model, these complex techniques backfired slightly. While it successfully performed the reasoning steps, the final output reverted to a generic tone, likely because the long context window caused it to “forget” the persona constraint in favor of the reasoning logic.
- **Prompt Chaining:** This technique proved useful for factual accuracy. By generating the rules in Step 1, we ensured the content was correct before writing the post. Gemini produced a very strong post using this method. Phi-3.5 produced a valid list of the rules but failed to convert them into the specific persona requested through the constraints, adopting a polite tone instead.

Conclusion

For the Large Model (Gemini): Chain-of-Thought (CoT) was the superior approach. It combined the persona's authority with a logical structure that fit the "Police" character perfectly. The output was not just stylized, but highly functional and organized.

For the Small Model (Phi-3.5): Few-shot Prompting was the necessary approach. Because the model lacks the deep nuance of larger models, it required concrete examples (Input-Output pairs) to understand the specific "voice" required. Abstract instructions (Zero-shot) were insufficient.

Week 3: Evaluating LLMs

Comparative Analysis: Italian vs. Sinhalese

This experiment aimed to evaluate how different LLMs (we have used Mistral-7B for Italian and Gemini 2.0 Flash for Sinhalese) handle cultural localization when prompted with specific geographic anchors (“I live in [country]”). The comparison reveals significant insights into how models prioritize Western norms versus local realities.

In both languages, the inclusion of the localized anchor proved crucial in breaking the model’s default “Western/American” bias. The models successfully adopted etiquette and daily habits to fit the local context.

- **Breakfast (Q1):**
    - **Italian:** The model shifted from a generic suggestion to culturally specific staples.
    - **Sinhalese:** The model moved from a Western standard to highly specific local dishes

    - Findings: Both models demonstrated strong, granular culinary knowledge when explicitly prompted
- **Tipping (Q11)**
    - **Italian:** The model correct the standard US expectation to the Italian reality where tipping is not mandatory or just small change
    - **Sinhalese:** The model correctly identified that a "10% service charge" is typically included in bills, advising that extra tipping is optional, reflecting true local economic dynamics.
- **Dinner time (Q10)**
    - Both models correctly adjusted dining times, shifting away from the earlier Northern European/US standard.

The most significant findings appeared in how the models handled complex legal and social topics. The gap between “Western norms” and “Local laws” influenced the depth of the AI’s responses.

- **Divorce (Q2)**
    - **Sinhalese:** The model acted almost like a law textbook, acting with high precision. It explicitly listed the specific legal grounds for divorce in Sri Lanka, such as "Adultery, Malicious Desertion, and Incurable Impotence".
    - **Italian:** The model focused on procedure rather than strict law. While it correctly identified "Separation" as a mandatory step in Italy (a concept foreign to many US jurisdictions), it hallucinated by mentioning "customary marriages," a concept that does not exist in the Italian legal system.
- **Civil Rights & Safety (Q6)**
    - **Sinhalese:** The response introduced a significant tonal shift. It acknowledged the specific legal context (criminalization status in Sri Lanka) and emphasized the need for privacy and safety, reflecting a complex socio-legal environment.
    - **Italian**: The response remained on a personal and social level, focusing on self-acceptance. It assumed a legally safer environment and did not feel the need to cite protective or punitive laws.

While both models successfully corrected superficial biases, the Sinhalese model with Gemini tended to be more explicit about local laws. This is likely because the gap between Sri Lankan law and the “Western generic norm” is wider that the gap between Italian and Western norms. Consequently, the model felt compelled to retrieve specific legal codes to answer the Sinhalese queries accurately, whereas the Italian model relied more on general procedural advice.

Project Reflection

Motivation: The primary motivation behind this experiment was to test the cultural alignment of LLMs, which are often trained on predominantly English, Western-centric data, which can lead to a “default mode” where answers assume the user is in the US or UK. By comparing a widely spoken European language (Italian) using an open-source model (Mistral) against a lower-resource Asian language (Sinhalese) using a commercial model (Gemini), the goal was to verify if simple prompting techniques (Anchors) could effectively force the models to switch contexts and provide culturally relevant advice rather than generic translations.

What I learned:

- **The Power of Anchors:** I learned that LLMs contain deep cultural knowledge that is often dormant. Without the anchor "I live in Italy," the model defaults to generic advice. The anchor acts as a retrieval trigger, unlocking specific local knowledge.
- **Safety vs. Utility:** I observed that models adapt their "level of alert" based on the region. The Sinhalese response to the LGBTQ+ question demonstrated that the model "knows" the geopolitical landscape is different and prioritizes safety warnings over generic affirmation.
- **Model Personality:** The comparison highlighted that larger commercial models (Gemini) might be better tuned for "fact-retrieval" in lower-resource languages (citing specific laws), while smaller models (Mistral) might struggle with specific legal facts in non-English languages, leading to hallucinations.

Challenges Encountered

- **Hallucinations in Localized Contexts:** A major challenge was detecting subtle hallucinations. For example, the Italian model sounded very convincing but invented "customary marriages" in Italy. This taught me that even when an LLM sounds culturally fluent, it can still fabricate legal details.
- **Evaluation Difficulty:** Validating the responses required native-level knowledge. Judging the tone of a response is difficult without deep cultural immersion in both languages.
- **Resource Disparity:** Comparing Mistral (7B parameters) against Gemini (Unknown, but likely much larger) meant that some differences in quality might be due to model intelligence rather than just language/culture.

Week 4: Fine-tuning LLMs

What was done during the lab:

In this laboratory, I performed Supervised Fine-Tuning (SFT) on a pre-trained Large Language Model using the QLoRA (Quantized Low-Rank Adaptation) technique. The goal was to specialize a general-purpose model to perform a specific task: generating quotes attributed to specific authors.

My workflow involved:

1. **Model Selection:** I replace the default Mistral-7B with Qwen/Qwen2.5-1.5B-Instruct
2. **Dataset Preparation:** I switched from the Alpaca dataset to Abirate/english_quotes. I implemented a formatting function to structure the data into a prompt format: ### Instruction: Generate a quote by {author}. ### Response: {quote}.
3. **Training Configuration:** I utilized bitsandbytes for 4-bit quantization (to reduce memory usage) and peft to apply LoRA adapters to the linear layers of the attention mechanism (q_proj, k_proj, etc.).
4. **Training and Inference:** I trained the model using the SFTTrainer from the trl library and verified the results using a text streamer to observe the generation of quotes for Einstein and Emerson.

What was the motivation behind the solution?

Switching to Qwen 2.5 (1.5B): The originally proposed Mistral-7B model is computationally heavy. I chose the 1.5B version of Qwen because it is significantly faster to fine-tune and less prone to "Out Of Memory" (OOM) errors on standard Colab GPUs, while still being capable of following instructions.

Choosing the "English Quotes" Dataset: The goal of the lab was to learn the process of fine-tuning. The Quotes dataset is simpler and more structured than general instruction following (Alpaca). This made it easier to verify if the fine-tuning actually worked: if the model outputs a quote when asked, the training was successful.

QLoRA: Fine-tuning a full model requires massive VRAM. Using QLoRA allowed me to train the model on a single GPU by freezing the base model weights and only training a small fraction of parameters (adapters).

Things learned through the lab

Hardware Dependencies: I learned that data types in PyTorch (float16 vs bfloat16) are strictly dependent on the GPU architecture. Using the wrong precision for the wrong hardware leads to immediate crashes.

Library Volatility: The Hugging Face ecosystem evolves rapidly. Code that worked months ago (passing arguments directly to SFTTrainer) now requires specific configuration objects like SFTConfig.

Tokenizer Nuances: Different model families (LLaMA vs Qwen) handle special tokens differently. I learned that I cannot blindly copy-paste tokenizer settings (like add_bos_token) from one model to another; Qwen, for instance, manages this internally and raises errors if manually forced.

Challenges Encountered

This lab presented significant technical hurdles that required debugging and adaptation:

- **BFloat16 & Hardware Incompatibility:** Initially, the code was configured to use bfloat16 precision. When running on a standard Tesla T4 GPU, I encountered a NotImplementedError because T4 does not support BFloat16.
    - **Solution**: I had to switch the hardware accelerator (moving from T4 to a more capable GPU like a P100/V100) and explicitly configure bnb_4bit_compute_dtype and torch_dtype to float16 in the code to ensure compatibility.
- **Out of Memory (OOM) Errors:** Even with a small model, the GPU memory filled up quickly during training.
    - **Solution**: I optimized the hyperparameters by reducing the per_device_train_batch_size to 4 and increasing

gradient_accumulation_steps to 4. I also enabled gradient_checkpointing to trade a bit of computation speed for significant memory savings.

- **Library Updates (TRL & Transformers):** I faced a TypeError when initializing the SFTTrainer because the library API had changed. The parameter dataset_text_field was no longer accepted directly.

    - **Solution**: I refactored the code to use the SFTConfig class for passing training arguments, aligning my code with the latest library documentation.

Week 5: Retrieval Augmented Generation (RAG)

What was done during the lab:

In this lab, we implemented a **Local RAG (Retrieval-Augmented Generation)** pipeline, adapting a notebook originally designed for English nutrition text to work with the Italian language and a specific domain: the **Italian Constitution**.

The main activities were:

- **Data Preparation:** We replaced the original PDF (nutrition textbook) with the PDF of the Italian Constitution. We extracted the text, cleaned the formatting, and split the content into manageable sentence chunks.
- **Embedding (Retrieval):** We replaced the English-only embedding model (all-mpnet-base-v2) with a multilingual model (paraphrase-multilingual-MiniLM-L12-v2) to correctly calculate semantic similarity between Italian queries and the Constitution's articles.
- **LLM Setup (Generation)**: We configured an open-source LLM (Qwen 2.5 or Google Gemma 2B) to run on a T4 GPU. We specifically removed 4-bit quantization to avoid CUDA compatibility errors and optimized loading using float16 precision.
- **Prompt Engineering:** We rewrote the prompt template in Italian, instructing the model to act as an expert in Constitutional Law and to answer based exclusively on the retrieved context, removing irrelevant English few-shot examples.

What was the motivation behind the solution?

The primary motivation was **linguistic localization and domain specificity**.

- **Embedding Switch:** An English-only model would fail to capture the semantic nuances of Italian queries (e.g., associating "Head of State" with "President of the Republic"). A multilingual model bridges this gap.
- **LLM Choice (Qwen/Gemma):** We selected small-sized models (1.5B - 2B parameters) to demonstrate that effective RAG can be run on consumer hardware (or entry-level cloud GPUs like the T4) without high costs, while maintaining decent Italian comprehension.
- **Italian Prompt:** LLMs are sensitive to the language of instructions. Keeping the prompt in English with software-related examples (like the original "SolrCloud"

examples) would have confused the model, leading to mixed-language responses or hallucinations.

In-depth Analysis: Finnish Language and assessment

Would the same embedding and LLM work for Finnish?
The paraphrase-multilingual-MiniLM-L12-v2 model is trained on over 50 languages, including Finnish. So it would also work for calculating semantic similarity.

The Qwen 2.5 model has decent multilingual coverage and would work reasonably well in Finnish. However, models specifically trained on Nordic texts (such as Poro or Viking developed by TurkuNLP) would offer significantly superior performance in terms of grammar and cultural nuances compared to base Gemma or Qwen.

What about extracting sentences and chunking: is there any change in terms of token length / chunk size?
Yes, there would be significant changes due to the different structure of the Finnish language.
Finnish language may contain longer words than the english average, most standard tokenizers (BPE/WordPiece) trained primarily on English would break these words into many small sub-tokens.
This would mean:

- A Finnish sentence with the same word count as an English sentence will consume more tokens.
- Consequently, the LLM's "context window" will fill up faster.
- Strategies would need adjustment. If chunking by character count, it might remain similar; if chunking by token count, the limit would likely need to be increased to capture the same amount of semantic information compared to English or Italian.

Can you assess the final quality?
The final quality of the implemented system is good, considering the limited resources.

- **Retrieval**: The system retrieves the correct Constitution articles with high precision. Since legal text is precise and structured, semantic search works very effectively.
- **Generation**: The model (Qwen/Gemma) successfully synthesizes the answer in correct Italian. The decision to use a strict prompt ("do not invent info") minimized hallucinations.
- **Limitations**: The model is a "Small Language Model." On complex reasoning tasks that require connecting 3-4 different articles located far apart in the text, it might lose coherence compared to larger models (like GPT-4 or Llama-3-70B).

Things learned through the lab

I learned that in a RAG system, the quality of the answer is entirely dependent on the quality of the retrieval ("Garbage In, Garbage Out"). Simply translating the document was not enough; I had to ensure that the mathematical representation of the text (the embedding model) understood the relationships between Italian legal terms. Using an English-centric model for Italian queries results in poor retrieval, regardless of how powerful the LLM is.

Week 6: Use cases and applications of LLMs

Introduction

This lab session explored the capabilities of modern LLMs in two distinct but advanced domains: Complex Document Understanding, also called Document Intelligence, and Synthetic Data Generation for testing search engine robustness. The primary objective was to move beyond simple text generation and utilize LLMs as reasoning engines capable of structured analysis and realistic human simulation.

Financial Data Extraction and Reasoning

Technological Context and Objectives

Extracting data from reports (such as earnings releases in PDF format) presents a unique challenge. Unlike plain text documents, financial reports rely heavily on tables and layout to convey meaning. Standard OCR or text extraction often flattens this structure, causing a loss of context (e.g., losing the relationship between a row header like "Operating Margin" and its corresponding column year).

The goal of this task was to build a pipeline capable not only of reading these values but also of performing reasoning tasks, such as calculating percentage changes or verifying sums across different report sections.

Architecture and Methodology

To address the structural complexity of PDF documents, the solution employed LLM Sherpa, a layout-aware parser running within a local Docker container. Unlike standard parsers that treat documents as streams of text, LLM Sherpa preserves the hierarchical structure of tables and sections. This structural preservation is critical for the "Retrieval" phase of the RAG pipeline, ensuring that the LLM receives data in a format where row-column relationships remain intact.

For the reasoning component, the system initially experimented with smaller, locally hosted models. However, to achieve the necessary analytical depth and to overcome problems related with the capacity of my computer, the architecture was migrated to a cloud-based inference approach using Groq, specifically leveraging the Llama 3.3-70b model. This shift was decisive; the larger parameter count of Llama 3.3 allowed for genuine arithmetic reasoning and logic, capabilities often absent in smaller models.

Challenges and Solutions

Problem: Docker connection errors (“No connection adapters”) and timeouts with local models.

Solution: Implementation of direct calls via requests to the local Docker server and migration to the Groq cloud for inference.

Analysis of Results

The final script (query_tables_v2.py) demonstrated excellent reasoning skills:

- Data Extraction: It correctly identified the 2024 Operating Margin (32%).
- Complex Calculations: It correctly calculated the % of Net Income to Revenue (29.4%).

- Advanced Reasoning: When asked to add up the revenues of the segments (Services, Cloud, Other Bets), the LLM not only performed the sum, but also independently identified a discrepancy due to “Hedging gains” ($72 million), explaining why the manual sum differed slightly from the reported total.

Synthetic Data Generation for Search Robustness

Technological Context and Objectives

The second phase of the laboratory focused on Synthetic Data Generation. The objective was to create a dataset of misspelled search queries to test the robustness of modern search engines. While traditional methods rely on algorithmic perturbation (randomly swapping or deleting characters), this approach often yields unrealistic strings that do not reflect actual user behavior.

Findings on Search Engine Robustness

The generated dataset was used to benchmark search engine performance. The results confirmed that modern search engines rely heavily on semantic vector search and fuzzy matching rather than strict keyword matching. In almost all test cases, queries containing realistic typos generated by the LLM yielded results identical to the correct queries. This suggests that the synthetic data successfully mimicked the type of noise that search algorithms are already optimized to filter out, validating the realism of the generation pipeline.

Conclusion

This laboratory demonstrated that the utility of Large Language Models extends far beyond simple chatbots. By integrating layout-aware parsers like LLM Sherpa, we transformed unstructured PDFs into queryable databases. Simultaneously, by leveraging the generative nature of LLMs, we created high-fidelity synthetic datasets that mimic human behavior better than traditional algorithms. The successful implementation of these tasks highlights the potential of combining Retrieval-Augmented Generation (RAG) with Generative Simulation to solve complex, real-world data problems.