UiT The Arctic University of Norway

Faculty of Engineering Science and Technology
Department of Computer Science and Computational Engineering

Fine-tuning Large Language Models on historical causes of death data

Kristoffer Berg Wilhelmsen

Master’s thesis in Applied Computer Science, May 2024

"Stay hard."
–David Goggins

"Yeah, Mr. White! Yeah, science!"
–Jesse Pinkman

Abstract

This thesis assesses the impact of fine-tuning and RAG on LLMs in accurately assigning ICD-10 codes to historical causes of death. Using funeral records from Trondheim, Norway (1830-1920), we fine-tuned Llama 3 and Mistral on 2000 records. Twelve experiments were conducted on 2000 additional records to evaluate the accuracy of each knowledge-injection technique, as well as a combination of the two.

The results indicate that fine-tuning as a standalone knowledge-injection technique achieved the highest accuracy, generating 88% full matches and 2% partial matches for ICD-10 codes, up from 58% full matches and 25% partial matches in previous research. However, concerns regarding memorization of training data due to the lack of diversity in the available dataset remain. Moreover, combining RAG with fine-tuning led to a decrease in accuracy, while a sole RAG approach decreased the results even further. These findings serve as proof-of-concept for the automatic assignment of ICD-10 codes to historical causes of death, paving the way for future research.

Acknowledgements

I would like to thank Rune Dalmo for admitting me into this course, giving me the opportunity to write this thesis.

Special thanks to Bjørn-Richard Pedersen, who allowed me to work on this task. It has been exciting to work on, and a lot of knowledge was gained along the way.

Thank you to my former classmates Joachim Kristensen, Stian Endrè Jakobsen, Asal Asgari, Håkon Berg Borhaug, Johanne Holst Klæboe and Jan Einar Kristensen. I have spent a lot of time throughout the years working by myself, but you have all helped me immensely at some point in time, one way or the other.

Thank you to my father Arild Wilhelmsen for proofreading.

Lastly, I would like to thank Shayan Dadman and Andreas Dyrøy Jansson for volunteering as supervisors.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

BERT Bidirectional Encoder Representations from Transformers

DL Deep Learning

EOS End of Sequence

GRU Gated Recurrent Unit

ICD International Classification of Diseases

LLM Large Language Model

LSTM Long Short Term Memory

ML Machine Learning

NHDC Norwegian Historical Data Centre

NLP Natural Language Processing

PEFT Parameter-Efficient Fine-Tuning

PLM-ICD Pretrained Language Models-International Classification of Diseases

QLoRA Quantized Low-Rank Adaptation

RAG Retrieval-Augmented Generation

RNN Recurrent Neural Network

UiT University of Tromsø

WHO	World Health Organization

1 Introduction

The introductory chapter gives an overview of the thesis by first providing background information and the motivation behind the research. The objectives are then expanded upon to outline the research goals. To ensure the reader's understanding of the material, essential theory is described. Next, the state of the art is reviewed to identify the current experimental results and gaps in existing research. Lastly, the research strategy is defined to map out the methodology that will be used to achieve the set objectives.

1.1 Background

In the domain of historical research, assigning ICD codes to historical causes of death is an important yet difficult challenge. There is a need for statistics in the form of processed, standardized data, to leverage computers and international cooperation for research purposes [1]. Using historical statistics, researchers are able to analyze how trends have changed over time. This allows us to learn from history and make adjustments, such as allocating more resources towards researching a particular cause of death. Moreover, it opens the door for international cooperation such as comparing causes of death for a certain period of time in Norway with other countries.

According to Bjørn-Richard Pedersen from the Department of History and Religious Studies at University of Tromsø (UiT), who is one of the supervisors for this thesis, the amount of data in need of processing is essentially unlimited. New historical data is being digitalized continuously, hence we have only scratched the surface and there will always be more data available.

Typically, assigning ICD codes has been a manual process performed by experts [2], [3]. This has been both costly and time-consuming, due to the need for specialized knowledge and the complex, hierarchical nature of the ICD medical classification system [4]. The Norwegian Historical Data Centre (NHDC) has identified this bottleneck, and recently explored the potential of utilizing LLMs to automate the process [5]. Their initial research concluded that while LLMs showed potential, the overall performance was inadequate, prompting further research.

The NHDC proposed investigating knowledge-injection techniques such as fine-tuning, RAG or a combination of both, to improve the results of their initial findings. A potential solution could have significant impact by serving as proof-of-concept, indicating what areas future research should focus on. The end-goal of automating the process would not only reduce time spent assigning codes, allowing for more data to be processed in less time, but also cut costs associated with hiring experts.

1.2 Objective

The goal of this thesis is to act as proof-of-concept for future research, as stated in Section 1.1. We want to assess and compare how two different knowledge-injection techniques, namely fine-tuning and RAG, as well as a combination of the two, affect the accuracy of LLMs. The specific strategy for implementation and criteria for success will be addressed in Section 1.5.

The research objectives are clearly stated in the task description, and are as follows:

1. Will fine-tuning a LLM lead to results which could be put into production? If so, would this be the case for other types of LLMs?
2. Would adding external knowledge bases boost the results even further?
3. Is it possible to achieve adequate results by only using external knowledge bases?
4. Would this process be financially feasible?

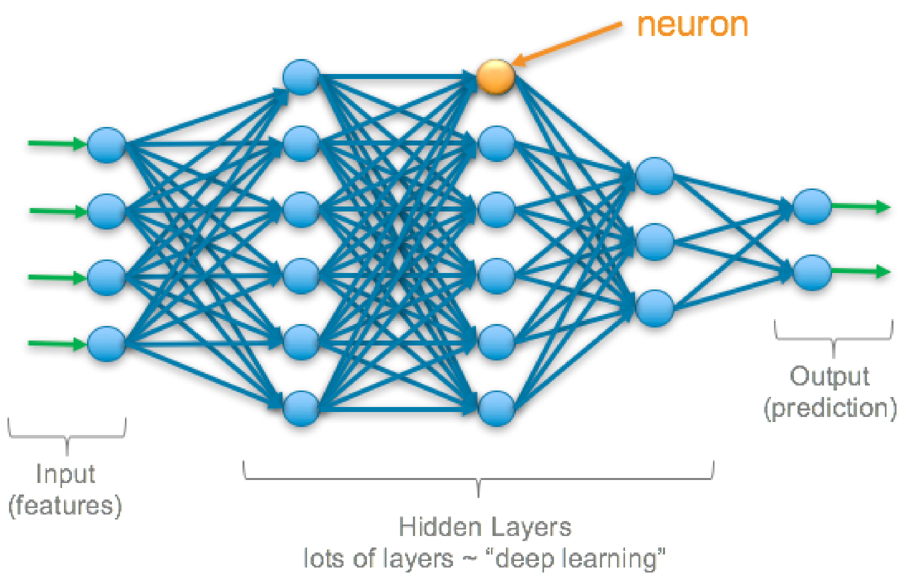

Figure 1.1: The architecture of a deep learning network [10].

1.3 Theory

As this thesis is focused on the practical application of LLMs, basic familiarity with the related topics and concepts would greatly benefit the reader. In this section, a brief introduction is given to the most essential theory.

1.3.1 Natural Language Processing

Machine Learning (ML) is a field within Artificial Intelligence (AI) that enables computers to learn from data and perform tasks without explicit programming [6]. Deep Learning (DL) is a subset of ML that uses artificial neural networks with many layers to learn from data [7]. Figure 1.1 illustrates the structure of a deep learning network, consisting of neurons that mimic the human brain [8]. This deep, comprehensive structure allows for highly complex patterns to be learned. According to [9, p. 4], "deep learning is, at its core, a culmination of achievements in fields such as calculus, linear algebra, and probability". Thus, DL essentially boils down to mathematics.

Within DL, Natural Language Processing (NLP) focuses on the interaction between humans and computers through language or speech. Using NLP, computers are able to understand and generate human language for various tasks in a meaningful manner [11]. Initial research dating back as far as the 1950s focused on translation between different languages by using machines

[12], however the field has come far since then.

In modern times, NLP has been achieved by leveraging ML and artificial neural networks. Recurrent Neural Networks (RNNs) such as Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been widely used in the past, however these have largely been replaced by the transformer architecture [13] which will be discussed further in subsubsection 1.3.1. NLP spans across many tasks. HuggingFace, a community platform where users can share and acquire models and datasets for AI purposes, lists possible tasks as text classification, question answering, translation, summarization, text generation, sentence similarity and more [14].

Transformer

The transformer is a deep learning architecture originally introduced in a paper from 2017 [13], offering more efficiency and better quality than previous architectures based on RNNs. The paper details the technical specifications, where a given input sequence is first split into a set of tokens, then converted to numerical representations in the form of embedding vectors. Semantic meanings and token positions are included in these vectors. The vectors are passed through several layers of encoder stacks, with each stack having two sub-layers of feed-forward neural networks that perform calculations that contribute to the model output, and self-attention mechanisms.

The self-attention mechanism is the core-component of the transformer. It computes attention scores between all pairs of tokens and weighs the importance of each token with respect to the others. This process is run many times in parallel using multi-head attention to process different parts of the input simultaneously, enabling models to capture long-range dependencies. The results of the parallel calculations are combined to form the final attention score.

Output from the encoder stacks are passed to the decoder stacks. The decoder consists of the same two sub-layers as the encoder, with an additional sub-layer that processes the output from the encoder. The additional sub-layer uses a technique called masking, which ensures that the model only makes predictions based on information known at the current position.

The final layer of the decoder stack produces output representations that are passed through a linear transformation and a softmax activation function. These produce the probability for a given word in the output sequence, allowing the model to generate the next word accordingly. Figure 1.2 shows its architecture from the original paper, consisting of an encoder (left) and a decoder (right).

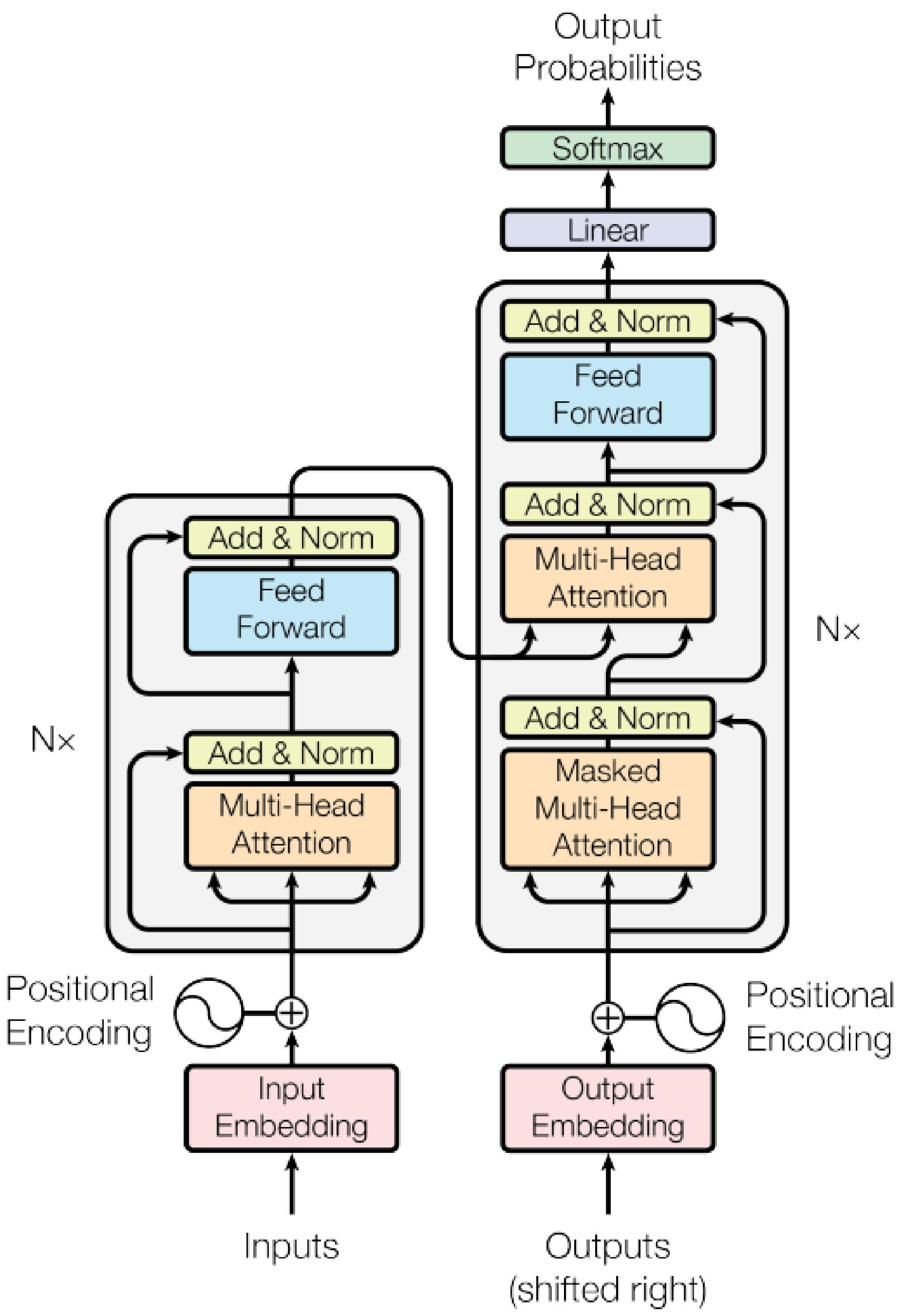

Figure 1.2: The original transformer model architecture [13].

1.3.2 Large Language Models

LLMs are models that incorporate neural network architectures such as the transformer to understand, process and generate human-like text. The term "large" refers to the amount of data used during training, and the high number of parameters these models contain. The parameters are responsible for storing the knowledge learned from training. Generally, there is a correlation between an increase in parameters and broader knowledge in models [15, pp. 81–84].

For instance, OpenAI's GPT-3, one of the most well-known models, was trained on more than 570 GB of filtered data, with the largest version consisting of 175 billion parameters [16]. Given the aforementioned correlation between performance, parameters and training data, a natural assumption is that top-performing models such as GPT-4 have been trained on significantly more data and consist of even more parameters. However, these details have not been publicly disclosed in OpenAI's technical report [17].

Fine-Tuning

Fine-tuning is a technique where a LLM that has already been pre-trained on large amounts of data, is further trained on a smaller dataset for a specialized task. This allows the bulk of learning to be done during pre-training, significantly lowering the amount of data and computational power needed [15, pp. 114–115].

Despite the concept of fine-tuning lowering computational demands, resource requirements can still be high. Quantized Low-Rank Adaptation (QLORA) further reduces these requirements by freezing the original model weights, quantizing them from higher-precision floats to 4-bit precision, and integrating trainable low-rank adapters that are updated during training. In their paper, the authors introduce three main innovations: NF4 (NormalFloat 4-bit), a new data type designed for quantizing weights into 4-bit precision; double quantization, a nested type of quantization; and lastly, paged optimizers that prevent memory spikes during training, by transferring data between the GPU and CPU. These innovations significantly reduce memory consumption, without sacrificing performance [18].

Retrieval-Augmented Generation

RAG is another technique where a retrieval component is integrated into the LLM pipeline. Before a prompt is passed to the model, the retriever fetches

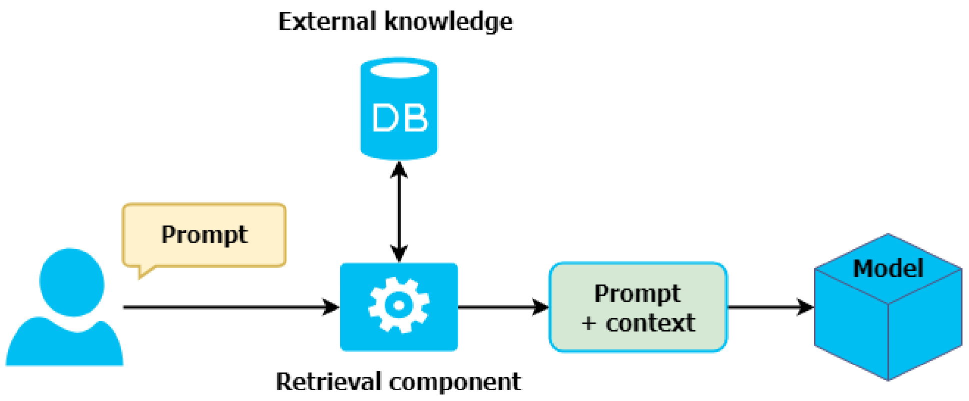

Figure 1.3: Basic RAG concept where context is fetched from an external knowledge source by a retrieval component, and injected into the original prompt.

additional information relevant to the prompt from an external knowledge source, such as a collection of user-defined documents. The retrieved information is then added to the prompt, and will be used by the model to make its prediction. The basic concept is shown in Figure 1.3. RAG can be particularly useful as it allows for knowledge to be updated retroactively, without having to re-train the model. Additionally, it can contribute towards reducing hallucinations, by providing a specific context and information for the model to base the prediction on [19].

1.3.3 International Classification of Diseases

ICD is a standardized system for classifying diseases, with its initial version dating back to 1893. The World Health Organization (WHO) has been responsible for revising and expanding the system since 1946 [20]. Although the most recent version ICD-11 came into effect in 2022, its predecessor ICD-10 from 1993 [21] has largely been used in recent research. This can be attributed to most of the available data being encoded with ICD-10 due to its maturity, as is the case for the data used in this thesis.

The ICD-10 instruction manual [22, pp. 14–17] explains its structure, which at the highest level is divided into 22 different chapters. Each chapter is a broad category of diseases or health-related issues based on organ systems or specific conditions, and is labeled with letters representing the first character of a code. Chapters are further divided into blocks, providing a more focused categorization. Blocks are assigned a range of letters and numbers, and contain categories that represent specific groups of related diseases or conditions. Categories are assigned specific letters and numbers, and have subcategories

Table 1.1: Top-down example of ICD-10 structure from chapter to subcategory [23].

[Chapter I]

A00 - B99 Certain infectious and parasitic diseases

↪ [Block]

  A00 - A09 Intestinal infectious diseases

  ↪ [Category]

    A00 Cholera

    ↪ [Subcategory]

      A00.0 Cholera due to Vibrio cholerae 01, biovar cholerae

      A00.1 Cholera due to Vibrio cholerae 01, biovar eltor

      A00.9 Cholera, unspecified

that are even more specific.

A code is typically three or five characters long, and represents the hierarchical structure of the system. In five-character codes, the first three characters are followed by a decimal point to separate them from the fifth character. Table 1.1 provides a visual example of the hierarchy, including how letters and numbers relate to chapters, blocks, categories and subcategories.

1.3.4 Benchmarks

During the literature review in Section 1.4, we encountered several types of benchmarks. For this thesis, we chose to focus on a form of string matching. There were a few different approaches to how this was used, but we chose to focus on the specific approach from [5].

String matching is a more primitive type of metric, that simply compares one string to another. In the aforementioned paper, they used partial and full matches for ICD-10 codes. A partial match is when the LLM predicts the first three characters of the code correctly, while a full match is a correct prediction for all five characters. This is simple, yet effective due to the structure of ICD-10 codes described in Subsection 1.3.3.

1.4 The State of the Art

The research on LLMs in historical data analysis is limited. However, a paper that prompted this thesis recently explored assigning ICD-10 codes from historical causes of death using GPT-4, GPT-3.5 and Llama 2 [5]. They found that while the models showed potential, with GPT-4 producing the best results consisting of 58% full matches, 25% partial matches and 17% errors, they were not sufficiently accurate for professional use. Inaccuracies were often caused by difficulties understanding historical terms, the use of abbreviations in data, assigning rare codes, and longer cause of death descriptions. Additionally, the models were instructed to output an error code in the case that they were unable to assign a code, but would instead generate false information, known as hallucinations. To improve these shortcomings, the authors suggested incorporating fine-tuning or frameworks such as RAG.

Despite the limited research in historical data analysis, several papers have used LLMs in medical settings. [24] combined LLMs with human annotators, reducing overall human annotation time by 58% while maintaining high-quality outputs. [25] generated ICD codes from different sets using their descriptions with GPT-3.5 and GPT-4, but found poor overall performance. [26] used GPT-3.5 to generate data for rare ICD codes, and then further train the model with the augmented dataset. This caused a slight drop in overall performance, but improved rare code identification. [27] analyzed GPT-3.5's ability to generate ICD codes from mock retina clinic encounters, generating a partially correct code in 70% of encounters and a fully correct code in 59% [27]. Similarly, [28] used Llama 2 (70B-chat), GPT-3.5 and GPT-4 to generate ICD codes for clinical data, however they also introduced a tree-search approach. This approach was better at predicting rare codes for all models, although it caused GPT-4's overall performance to drop. The authors suggested this may have been due to predicting leaf codes that were similar.

The Pretrained Language Models-International Classification of Diseases (PLM-ICD) framework proposed by [29] achieved state-of-the-art performance by several metrics. Using fine-tuned models based on Bidirectional Encoder Representations from Transformers (BERT) [30], the framework addressed three primary issues affecting performance; large label space, long input sequences and domain-specific pre-training. Firstly, there are thousands of possible labels when assigning ICD codes, causing a high level of complexity. By introducing a label attention mechanism, the model was able to focus on certain parts of the input, relevant to the specific labels. Secondly, the number of tokens a model can process as input (context length), can differ between models. Clinical data often consists of long documents, resulting in truncation. By splitting documents into shorter segments, encoding each segment and using an aggregated representation, the models were able to fully consider the content without truncation. Lastly, the effectiveness of domain-specific pre-training was highlighted by comparing two models. A base-model trained on a variety of data, and a second version of the same model pre-trained on domain-specific data. The latter model performed slightly better, indicating that domain-specific pre-training is important.

Fine-tuning and RAG are both knowledge injection techniques used in LLMs. [31] compared the two approaches for a range of tasks requiring factual knowledge, and found that although fine-tuning did show an increase in performance, it was consistently outperformed by RAG. Fine-tuning struggled to adapt to new information, whereas RAG demonstrated its ability to understand both existing and new knowledge. The results on combining fine-tuning and RAG were inconsistent, but showed that it could outperform a pure RAG approach in some scenarios. They recommended future work to focus on exploring different combinations of techniques to optimize LLMs.

1.5 Strategy

The first, second and third objective of this thesis all concern the effectiveness of knowledge-injection techniques, more specifically fine-tuning and RAG. To measure the impact each technique, as well as a combination of the two, has on the performance of different LLMs, we will run six experiments on two models. This will result in a total of twelve experiments, with the following configurations:

1. **Base:** Model without any modifications, providing a baseline (two experiments).

2. **Fine-tune:** Model fine-tuned on historical causes of death, isolating the fine-tuning component (two experiments).

3. **RAG 1 & 2:** Model integrating an external knowledge base, isolating the RAG component. Two different knowledge bases will be tested on both models (four experiments).

4. **RAG 1 & 2 + fine-tune:** Model fine-tuned on historical causes of death, while simultaneously integrating an external knowledge base, combining both knowledge-injection techniques. The same knowledge bases as mentioned above will be tested on both models (four experiments).

To carry out the experiments, we will write two different scripts:

1. **Fine-tuning:** Fine-tuning each model on historical causes of death, with assigned ICD-10 codes. This is only for updating the models' weights, and will not output any predictions.

2. **Inference:** With a modular design, a single script will run all twelve experiments. A set of parameters will be defined by the user before runtime, determining which configuration to load.

The fourth and final objective addresses the financial viability of the process as a whole. This is particularly important, as this thesis has not received any financial support. With a budget of zero, all experiments will have to be performed on personal hardware. Thus, the fourth objective will investigate whether viable results can be acquired with limited resources, by choosing minimalistic model configurations wherever applicable.

For the experiments to be considered a success, the ideal scenario is for the results to clearly identify a single knowledge-injection technique configuration that outperform the others. Considering that the focus of this thesis is proof-of-concept and that a commercial product for practical use is out of scope, a specific accuracy for the model predictions is less important. The main criterion is identifying a path forward for future work.

2 Tools and Methods

The second chapter provides a detailed description of the tools utilized in this thesis. It discusses the methods that were applied and the rationale for their selection. By meticulously documenting the tools and methods used, this chapter serves as a comprehensive guide for reproducing and validating the results obtained in this thesis.

2.1 Tools

This section contains the tools and resources used to conduct the experiments defined in Section 1.5. It encompasses the programming language, dataset, hardware and specific models that were utilized.

2.1.1 Programming Language

When selecting the programming language for this thesis, we considered two primary factors: previous experience and the industry standard.

Initially, we reviewed which programming languages were previously used throughout the master's curriculum. There have been two courses involving AI. Both of them, DTE-3608 Artificial Intelligence and Intelligent Agents - Concepts and Algorithms [32] and DTE-3606 Artificial Intelligence and Intelligent Agents

Table 2.1: Columns and related descriptions of dataset containing causes of death from 1830 to 1920 in Trondheim, Norway [36].

Variable Name	Description
id_ori	Person identifier
cod_ori	Original cause of death
cod_literal	Standardized cause of death
ICD10h	Historical code based on ICD-10
icd10	ICD-10 code [37]
icd10_desc	English description of main category
maincat	Historical main category
englishname	English translation of cod_literal
subcat	Historical subcategory of maincat

- Project [33], used Python [34] as their programming language. Our existing experience provided a strong foundation for selecting Python.

When reviewing the literature referenced in Section 1.4, we found that all the papers involving programming tasks had used Python. Additionally, every model we reviewed on HuggingFace [14] provided sample code for running inference written in Python. This reinforced the foundation further by asserting Python as the industry standard within AI community.

Python offers extensive library support through its package installer pip [35], in addition to a large amount of available online documentation due to its widespread use. For these reasons, Python was a natural choice and therefore selected as the programming language for this thesis. More specifically, we used Python 3.11.4.

2.1.2 Dataset

The dataset used for the experiments in this thesis was provided by the NHDC and contains historical records of causes of deaths from the period 1830 to 1920 in Trondheim, Norway. It consists of 32,754 entries, where each entry details a person's cause of death along with other associated variables as shown in Table 2.1.

The experiments conducted in this thesis focused on predicting the correct ICD-10 code (*icd10*) for a given cause of death (*cod_ori*). An important distinction to make is that the *icd10h* column contains a type of code used for historical causes of death, for which documentation is unavailable online as discussed by [5], making it unsuitable for predictions. The remaining columns in the dataset

Table 2.2: Computer specifications used to run experiments.

Name	Description
Motherboard	ASUS ROG STRIX Z390-E GAMING
RAM	HyperX Fury DDR4 3200MHz 32GB
Disk	Samsung 970 EVO 1TB M.2 SSD
CPU	Intel Core i9-9900K
GPU	MSI GeForce RTX 2080 Ti (11GB VRAM)
PSU	Corsair RM850x
OS	Ubuntu 22.04.4 LTS x86_64

were not required for our experiments and were thus not utilized.

2.1.3 Hardware

All experiments were conducted on a personal machine as stated in Section 1.5. The complete specifications of the machine are listed in Table 2.2. Considering that the hardware was purchased towards the end of 2018, it was particularly important to make minimalistic decisions regarding model selection and hyper-parameters in the development process, while also prioritizing performance. The most critical aspect was reducing GPU memory consumption as this is a finite resource, in contrast to computation time which simply increases.

2.1.4 Model Selection

The LMSYS Chatbot Arena Leaderboard [38] is a platform that offers a subjective human ranking of various models by comparing two different responses and having users select their preferred one. As of May 9, 2024, the recently released (April 18, 2024) Llama-3-70b-Instruct by Meta, by Meta, is the best performing model available under a community license, ranked #6. The smaller version, Llama-3-8b-Instruct, is not far behind, ranked #16. Given the limited GPU memory available on our hardware, selecting models around the 8 billion parameter range was essential.

Despite findings in Section 1.4 suggesting that domain-specific pre-training increases performance, we opted for Llama-3-8b-Instruct [39] as one of the models for our experiments. Having been released shortly before our experiments began, the availability of versions fine-tuned on medical data was limited. According to the model card [40], Llama 3 showed improvements in all metrics when compared to its predecessor Llama 2, making it an interesting choice despite not being pre-trained on domain-specific data. Moreover, we planned specific additional training using our own dataset, ensuring that the model would receive domain-specific training. The main goal of this thesis was to investigate RAG and fine-tuning, which further justified our model choice.

For the second model, we selected Mistral-7B-Instruct-v0.2 [41]. According to the findings in their paper [42], this model performed better than both the 7B and 13B versions of Llama 2, although by lesser margins than Llama 3. While models with domain-specific training such as BioMistral-7B [43] were available, we chose not to pursue a fine-tuned model, for a more accurate comparison with Llama 3.

2.2 Methods

This section details the practical implementation of the experiments that were conducted for this thesis. It outlines the preprocessing applied to the dataset, describes the prompt used as input for the models during the experiments, explains how the models were fine-tuned, and the process by which predictions were generated during inference.

2.2.1 Preprocessing

In practical applications, the input for a given model would likely be the original cause of death, denoted as *cod_ori* in the dataset from Table 2.1. However, the original data was provided in Danish. Since both models we selected are intended for use in English [40], [44], the original string could not serve as input. Although a possible option could have been using the English translation (*englishname*), this column was processed by humans and highly standardized. Therefore, we instead opted to write a script to translate the original cause of death in Python, with the intention of preserving nuances in each entry.

We performed some initial testing using small samples with different libraries such as dl_translate [45] and googletrans [46], but found that deep_translator [47], which supported using Google Translate's API, provided the best translations. Prior to translating, we filled rows where *cod_ori* was empty with "blank" to avoid any exceptions from being raised. Then, we looped through the entire dataset, translating *cod_ori* from Danish to English. Of the original 32,754 rows, there were only 4,729 unique entries for *cod_ori*. This was reduced to 3,887 after translating.

The total number of rows in the dataset exceeded the number of samples needed by far, particularly considering we intended on running several experiments using limited hardware within a short period of time. When analyzing the dataset for imbalances, we found that it only contained 277 unique ICD-10 codes. Of these, 83 of them had 2 or fewer examples, while 129 had 10 or more examples. Factoring in the relatively small amount of diversity of codes, we used pandas.sample [48] with `frac=1` and `random_state=1337` for reproducibility to shuffle the dataset. We then took the first 2000 rows for fine-tuning, the next 500 for validation and another 2000 for testing. This resulted in training data consisting of 141 unique ICD-10 codes, validation data of 92 unique codes, and test data of 131 unique codes.

2.2.2 Prompt

For fine-tuning and inference, we used the prompt from [5] as a starting point. Based on responses we obtained through trial and error using smaller samples, we made several changes to the language with the intention of being more short and concise, in addition to the following specifics:

1. Added an example of a 3-character code in addition to the existing 5-character code to accommodate both possible responses.
2. Instructed the model to assign unknown or blank causes of death the code ’R99’ to avoid confusion with the anti-hallucination code ’Æ99.9’. The code ’R99’ is commonly used for unspecified causes of death [49].
3. Specifically instructed the model to not explain its answer.
4. Added a ’context’ variable for the RAG implementations.

We used the same prompt for both fine-tuning and inference of all configurations, except for the ’context’ variable, which was only present when RAG was utilized during inference. Note that both `{cause_of_death}` and `{context}` are variables that were substituted during runtime, and will be elaborated on in Subsection 2.2.3 and Subsection 2.2.4.

```
You are to assign an ICD-10 code to a cause of death using
the following instructions:
- Use standard ICD-10 codes, not ICD-10-CM billing codes.
- Each ICD-10 code should be 3 or 5 characters long, for
  example: ’X01.0’ or ’C15’.
- If the cause of death is ’unknown’ or ’blank’, use code
  ’R99’.
- If you lack sufficient information to assign a code, do
  not try to guess. Instead, use code ’Æ99.9’.
```

```
- Your response should only contain a single ICD-10 code
  using this format: '<ICD-10 CODE>'.
- Do not explain your answer.

Cause of death: {cause_of_death}

Context: {context}
```

2.2.3 Fine-Tuning

The fine-tuning of both models was carried out by a single Python script, primarily using components from the transformers library [50]. We chose transformers based on its large number of components and comprehensive documentation [51]. The library is provided by HuggingFace and supports both models we selected.

The models supported by the script are programmatically defined as constants. Before runtime, users must specify which model should be loaded. The tokenizer is loaded from AutoTokenizer, and the pad token used during training is automatically set depending on which model is loaded. For Mistral, we set the pad token to the tokenizer's "unknown" token. For Llama, we set it to the End of Sequence (EOS) token. The difference in pad token stems from Mistral utilizing the EOS token during training, while Llama did not. Setting Mistral's pad token to EOS will as of now (14th May 2024) cause indefinite text-generation [52].

Using AutoModelForCausalLM and BitsAndBytesConfig, we load a quantized version of the model. For BitsAndBytesConfig, we enabled both 4-bit loading and double quantization, while setting the quantization data type to nf4 and computation data type to bfloat16. The configuration parameters were chosen to minimize GPU memory usage.

For efficient QLORA fine-tuning, we used the Parameter-Efficient Fine-Tuning (PEFT) [53] library. We defined a LoraConfig with a rank of 8 and an alpha of 16. Target modules were set to all linear layers, with a dropout of 0.05. We did not use any bias, and specified CAUSAL_LM as the task type. The parameters were chosen based on recommendations for small models in the 7 billion parameter range from the QLORA paper [18].

The 2000 rows of training data and 500 rows of validation data we preprocessed in Subsection 2.2.1 were loaded from text files using pandas [54], and then inserted into a Dataset object from the dataset [55] library. The data was formatted to suit each model by using the apply_chat_template from the

Tokenizer class, and passed to the data collator for processing.

The final step was training the model, for which we first defined a set of hyperparameters using TrainingArguments from transformers. Again, we looked to the QLORA paper for specific recommendations, where we found that a learning rate of 2e-4, a batch size of 16 and training for 3 epochs was recommended. Due to memory constraints, we had to use gradient checkpointing to reach the batch size. We set the gradient-accumulation steps to 8 along with a batch size of 2, for an effective batch size of 16. We used the paged version of the 8-bit adamw optimizer to avoid spikes in memory. As overfitting was not a concern, validation was only performed once at the end of each epoch. We used a warmup ratio of 0.1, and a weight decay of 0.01. Lastly, we used the Trainer class from transformers to train and update the weights for each model.

2.2.4 Inference

We initially intended on using LangChain [56] for inference, based on a recommendation during a meeting in the early stages of the thesis. It offers components that serve as building blocks, which are assembled into chains that can be executed. The purpose of LangChain is to simplify the development process, while also offering customization. However, from practical testing, we found inference using LangChain to be slow, while also adding a layer of complication. Considering the limited scope of this thesis, we opted for more of a manual approach using transformers.

Similarly to fine-tuning, we used a single script to run inference for both models. Before runtime, parameters specifying the model name, whether fine-tuning or RAG should be used, and files for both input and output must be defined. A model with the corresponding configuration is then loaded, using the same libraries in the same manner as with fine-tuning.

In this case, we specified the 2000 rows of test data from Subsection 2.2.1 as the input. This was loaded into a DataFrame using pandas and then iterated through. For each iteration, we substituted the variables from Subsection 2.2.2 with the translated cause of death, and relevant context if RAG was enabled.

For the RAG knowledge sources, we used two different PDFs. The first source contained all the translated causes of deaths from the test dataset, with correctly assigned ICD-10 codes. The second source was the complete ICD-10 tabular list [22], containing every possible ICD-10 code with the related causes. We wanted to see how information quality would impact the results, specifically how accuracy would be affected if the RAG component undoubtedly had the correct information available, and whether simply using the complete tabular list would be beneficial, despite possible differences in terminology.

The RAG implementation was put together using components from LangChain. The specified source of external knowledge was first loaded using PyPDFLoader and split by RecursiveCharacterTextSplitter, using a chunk size of 400 without overlap. The chunks were stored as embeddings with Chroma, using NeuML/pubmedbert-base-embeddings [57] for embeddings, which has been trained on medical data. Context is retrieved using a similarity search from Chroma, with the cause of death as the parameter. Text from the four best matching chunks is fetched, concatenated and injected into the prompt.

We use different methods to run the actual predictions based on which model was loaded, as Mistral and Llama3 differ slightly in how responses are generated. Both models used a temperature of 0.7 and a top_p of 0.9. The resulting prediction generated by the model was added in a new column in the same row at the cause of death, which we used to create the results presented in the next chapter.

Figure 2.1 illustrates a simplified version of the inference implementation. First, the model is loaded from the user-defined configuration. If RAG is enabled, the database is created. As the dataset is iterated over during runtime, the cause of death is set for each prediction, while context is injected if RAG is enabled. At the end of the loop, the updated data frame containing including the new predictions is saved to the defined output file.

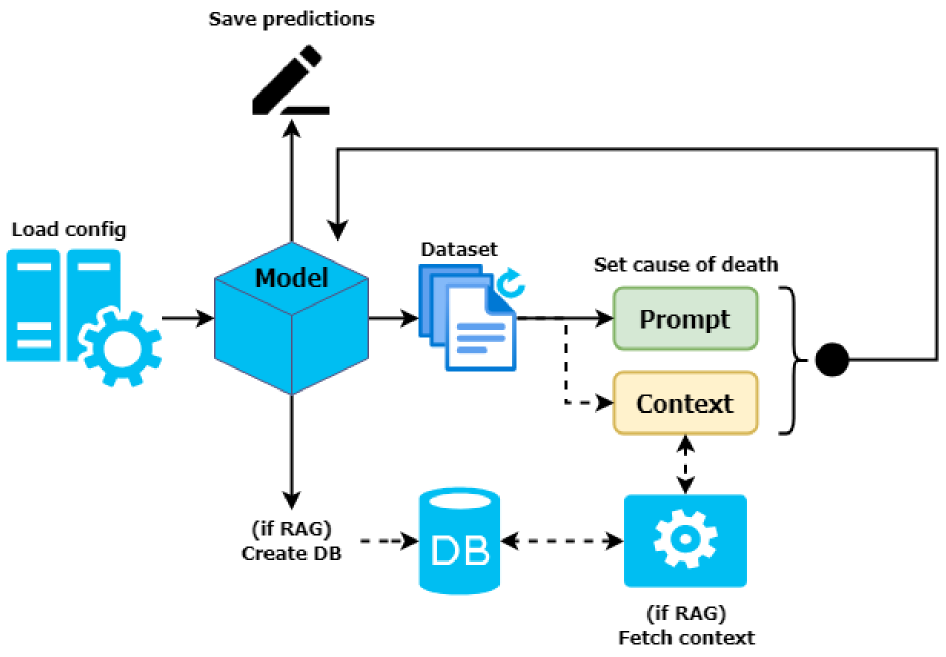

Figure 2.1: Simplified illustration of the inference implementation's modular design.

3 Results and Discussion

The third chapter presents the quantitative results obtained from a series of experiments conducted on LLMs, using various knowledge-injection configurations. The results consist of metrics from the fine-tuning process, as well as benchmarks from the inference stage. Subsequently, the implications of the findings are discussed.

3.1 Results

This section contains the results from the experiments we conducted. It is divided into two parts, of which the first presents metrics obtained during the fine-tuning stage, including trainable parameters, runtime, memory usage and training loss. The second part focuses on the outcomes of the twelve experiments defined in Section 1.5, in addition to showing various metrics similarly to the fine-tuning stage.

3.1.1 Fine-Tuning

To train the models using data specialized for our task, assigning ICD-10 codes from historical causes of death, we used QLORA to only update a small fraction of the model parameters. When running the fine-tuning script, the number of total and trainable parameters for each model was printed as output.

Table 3.1: Number of total and trainable parameters from fine-tuning.

Model	Total parameters	Trainable parameters
Llama3	8,051,232,768	20,971,520 (0.26%)
Mistral	7,262,703,616	20,971,520 (0.29%)

Table 3.2: Comparison of runtime and memory usage for both models during fine-tuning.

Model	Runtime (min)	Memory Usage (GB)
Llama3	62.37	10.07
Mistral	64.88	6.47

The parameters are listed in Table 3.1, showing how much we were able to reduce the trainable parameters by as a result of the LoraConfig described in Subsection 2.2.3.

When the fine-tuning script had concluded for each model, we recorded the exact runtime and final memory usage for the specific Python process. These metrics provide insight into the resource demands and efficiency of QLORA, which was particularly important considering our hardware constraints. Table 3.2 lists the runtime in minutes and memory usage in gigabytes for each model. In addition to the memory used by the Python process, the operating system required approximately 500 MB. The total available GPU memory was 11 GB as listed in Subsection 2.1.3.

Lastly, we plotted the training and validation loss for each model to assess their learning. The training loss indicates how quickly the model was able to learn from the data, while the validation loss determines if the model is generalizing well to new data or simply memorizing the training set. In Subsection 2.2.1, we allocated 2000 rows for training and 500 rows for validation. Each model was trained over 3 epochs, and validation occurred once at the end of each epoch. Training loss was calculated at each step with a batch size of 16. Figure 3.1 and Figure 3.2 show the training and validation loss graphs for Llama3 and Mistral, respectively.

3.1.2 Inference

In the same manner as fine-tuning, we recorded the exact runtime and final memory usage by the specific Python process for each model configuration. Figure 3.3 shows a comparison of runtime in minutes for each model configuration,

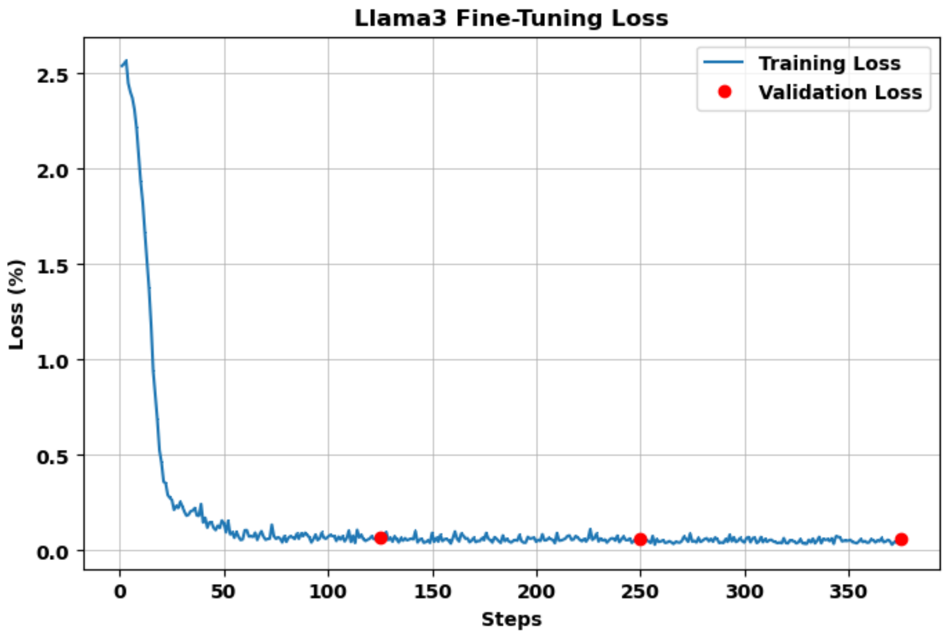

Figure 3.1: Llama3 training and validation loss from fine-tuning.

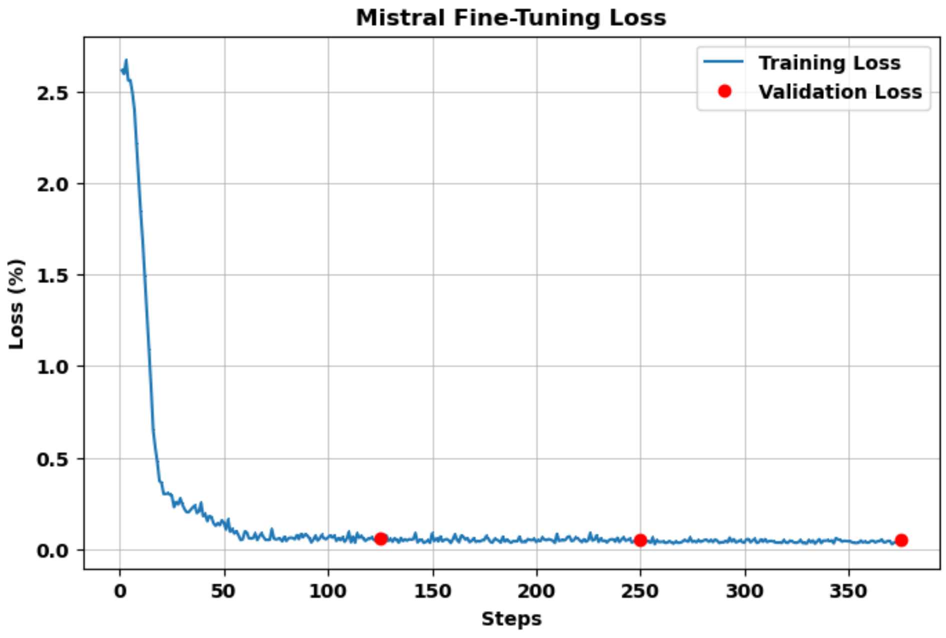

Figure 3.2: Mistral training and validation loss from fine-tuning.

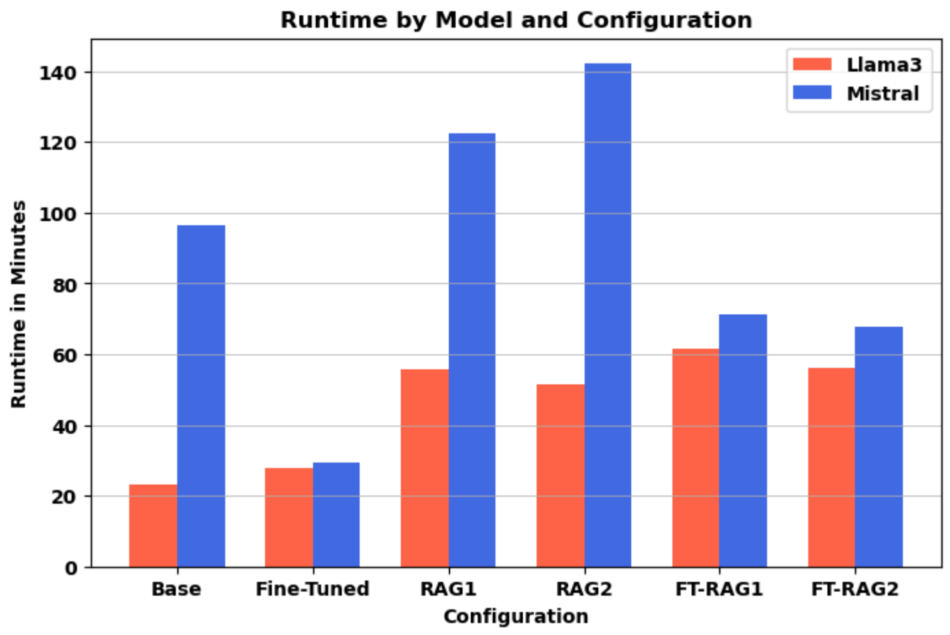

Figure 3.3: Inference runtime for both models and all configurations.

while Figure 3.4 shows the final memory usage for each Python process.

The predictions from inference were categorized into five match types. A "full match" required the generated prediction to be an exact match of the true ICD-10 code. A "partial match" was found when the first three characters of the codes matched. The "error code" Æ99.9 is what we instructed the model to output if it was uncertain, to reduce hallucinations. Subsequently, a "hallucination" is noted as an incorrect prediction, despite the model being instructed to refrain from guessing, in the case of insufficient information. Lastly, "no code" indicates that the generated response did not contain a valid ICD-10 code or the defined error code. A visual chart can be seen in Figure 3.5, while exact numeric values are listed in Table 3.3. Note that we have used the following, shortened configuration names in certain figures and tables as necessary:

- Llama3 $\longrightarrow$ L3
- Mistral $\longrightarrow$ M
- Base $\longrightarrow$ B
- Fine-Tuned $\longrightarrow$ FT
- RAG1 $\longrightarrow$ R1

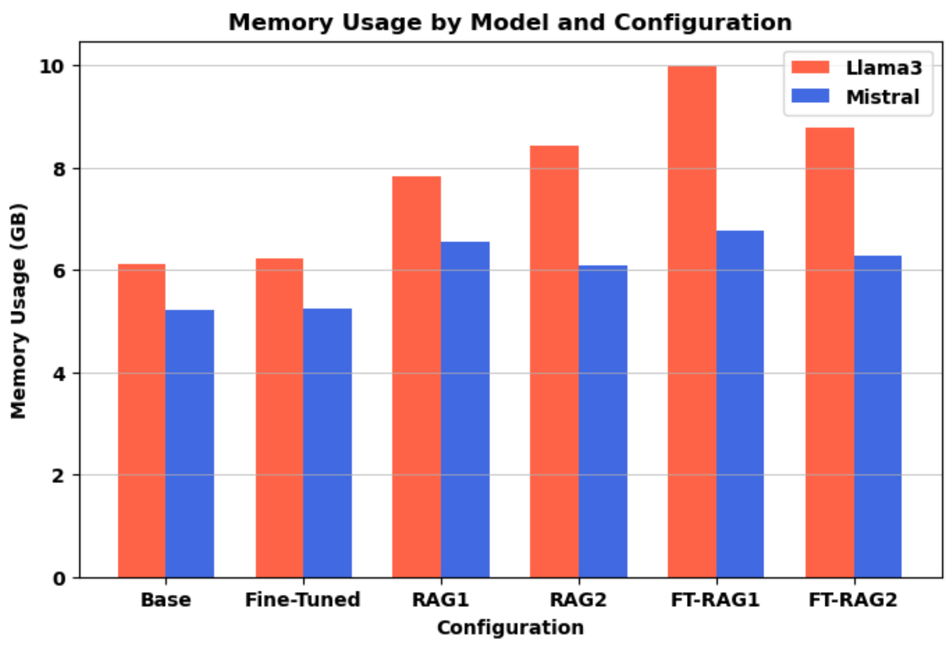

Figure 3.4: Inference memory usage for both models and all configurations.

- RAG2 ⟶ R2

The final metric we recorded was the average character length for predictions generated by each model configuration, listed in Table 3.4. As we instructed the models to only output an ICD-10 code without explaining their answers, this metric indicates how well they were able to follow instructions.

3.2 Discussion

This section discusses and interprets the results presented in the previous section. It follows the same structure as the results, with the first part focusing on findings from fine-tuning and the second part addressing inference.

3.2.1 Fine-Tuning

From using QLORA for fine-tuning, we were able to significantly reduce the trainable parameters to 0.26% of the original amount for Llama3 and 0.29% for Mistral. Despite the hardware constraints, we were able to train both models on 2000 samples in around an hour of computation time each. For

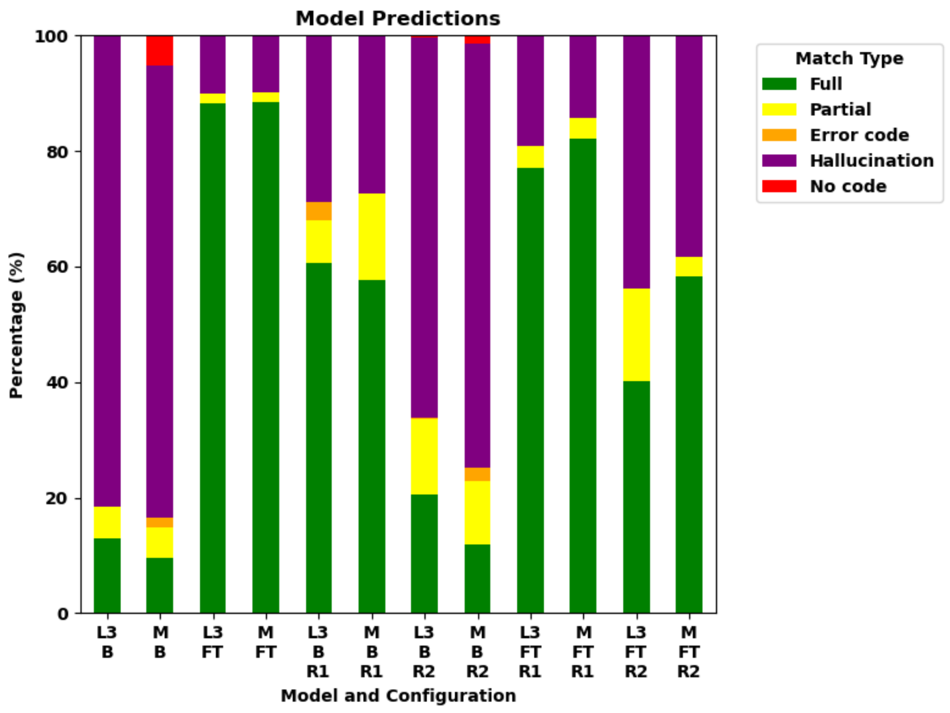

Figure 3.5: Predicted ICD-10 codes categorized by match type.

Table 3.3: Numeric values for predicted ICD-10 codes, categorized by match type.

Config	Full	Partial	Error C.	Halluci.	No Match
L3 B	259 (13%)	110 (6%)	1 (0%)	1630 (82%)	0 (0%)
M B	193 (10%)	105 (5%)	31 (2%)	1569 (78%)	102 (5%)
L3 FT	1767 (88%)	31 (2%)	0 (0%)	202 (10%)	0 (0%)
M FT	1769 (88%)	36 (2%)	0 (0%)	195 (10%)	0 (0%)
L3 B R1	1212 (61%)	147 (7%)	64 (3%)	577 (29%)	0 (0%)
M B R1	1153 (58%)	301 (15%)	0 (0%)	542 (27%)	4 (0%)
L3 B R2	409 (20%)	264 (13%)	4 (0%)	1318 (66%)	5 (0%)
M B R2	238 (12%)	218 (11%)	48 (2%)	1469 (73%)	27 (1%)
L3 FT R1	1540 (77%)	78 (4%)	0 (0%)	381 (19%)	1 (0%)
M FT R1	1643 (82%)	72 (4%)	0 (0%)	285 (14%)	0 (0%)
L3 FT R2	801 (40%)	323 (16%)	0 (0%)	875 (44%)	1 (0%)
M FT R2	1168 (58%)	66 (3%)	0 (0%)	766 (38%)	0 (0%)

Table 3.4: Average character length for predictions generated by models.

Model	Length
L3 B	4.1
M B	156.1
L3 FT	4.3
M FT	4.3
L3 B R1	4.2
M B R1	134.0
L3 B R2	4.6
M B R2	195.4
L3 FT R1	4.3
M FT R1	4.3
L3 FT R2	4.3
M FT R2	4.4

Mistral, the memory usage was well below what we had available, however Llama3 used significantly more memory. When factoring in memory used by the operating system, fine-tuning Llama3 required 10.6/11 GB, even though we made minimalistic choices at every turn. Thus, although we were able to fine-tune both models within our constraints, the margins were razor-thin.

The training and validation losses may appear satisfactory at first glance, however they come with a caveat. When we analyzed the dataset in Subsection 2.2.1, we found that there were only 4,729 unique entries despite there being 32,754 rows. After translation, the number of unique entries was reduced to 3,887. The scarce diversity could suggest that despite using different splits for training and validation, both splits were largely equivalent. The primary difference was likely the inevitable outliers of codes with few occurrences, as we used a random split without considering data imbalance.

The bulk of the learning was done in the first 20 steps, indicated by the steep curves in Figure 3.1 and Figure 3.2. After the first 50 steps, loss did not significantly drop. Given that the training data contained 141 unique codes with an effective batch size of 16, we can make the following assumptions for an ideal scenario:

$$\begin{gathered}16 \text{ batch size} \times 20 \text{ steps} = 320 \text{ samples seen} \\ \frac{320 \text{ samples}}{141 \text{ unique codes}} \approx 2.27 \text{ samples per code}\end{gathered} \tag{3.1}$$

Equation 3.1 shows how 320 samples would have been seen by the model after 20 steps. With 141 unique codes, this equals around 2 samples per unique code, given an even distribution.

$$\begin{gathered}16 \text{ batch size} \times 50 \text{ steps} = 800 \text{ samples seen} \\ \frac{800 \text{ samples}}{141 \text{ unique codes}} \approx 5.67 \text{ samples per code}\end{gathered} \tag{3.2}$$

Equation 3.2 shows the equivalent calculations for 50 steps, where between 5 and 6 samples per unique code would be required. Given that each epoch consisted of 125 steps, it is possible we could have gotten similar results with less than half the data. Moreover, the number of training epochs could potentially have been reduced, lowering training time. However, we are unable to confirm this as we did not do any practical tests.

The prompt we used from Subsection 2.2.2 explicitly stated that the model should not explain its answer. Despite this, as reflected in the average response length in Table 3.4, we can see that Mistral failed to comply with this instruction in all three configurations that were not fine-tuned. The average response lengths ranged from 134 to 195.4, while an ICD-10 code can only be between 3 and 5 characters. With fine-tuning however, the model responded with a satisfactory average character length. Llama3 on the other hand followed this instruction in all its configurations.

3.2.2 Inference

The inference runtime in Figure 3.3 showed minor differences in the fine-tuned configurations, with Llama3 having shorter runtime. On the contrary, the Mistral base configurations were up to several times slower than the Llama3 base configurations. When considering the average prediction length in Table 3.4, it becomes apparent that the significant increase in runtime stems from the number of output tokens, as a consequence of not following the specified instructions. When fine-tuned, Mistral was able to output a response in the format we instructed, resulting in a significantly shorter runtime.

Memory usage was fairly proportional across all configurations for both models. The addition of RAG caused memory consumption to increase, due to the prompt containing more tokens when context was injected. Combining fine-tuning with RAG caused an even further increase, despite the non-RAG base and fine-tuned configurations consuming the same amount. The specific cause for this is uncertain, though seeing memory usage over time as opposed to the final number could have been beneficial.

The predictions shown in Figure 3.5 and numeric values from Table 3.3 give us clear indications. The results for both models are quite similar with the same configurations, showing the techniques are generalizable for different models. Both base configurations performed terribly, matching the findings from [5]. The fine-tuned configurations performed the best by a fair margin, surpassing both RAG and the combination of techniques. This is in contrast to findings by [31], who found that RAG outperformed fine-tuning, although their experiments concerned factual knowledge. Additionally, they cited that fine-tuning struggled with adapting to new knowledge, which we are unable to verify due to the nature of the dataset we had available. Their findings on combining RAG and fine-tuning were inconsistent, however they found that a combination could outperform a pure RAG approach in some scenarios. In our experiments, RAG was consistently outperformed by the combination approaches. However, the use of RAG caused the results to deteriorate.

From the two different data sources we used in RAG, we can see that information quality has a clear impact on the results. For RAG1, we supplied the model with correctly assigned codes. For RAG2, we used the tabular list of every possible code. Although RAG2 did result in increased accuracy, RAG1 achieved significantly better results.

Despite all codes being available in the data sources, we did not verify whether they were present in the injected context. This implies that the incorrect predictions could have been caused by the similarity search not retrieving chunks containing the correct code, as opposed to the model not interpreting the context correctly.

Whether the context included the correct code becomes less relevant when factoring in the error code we specified. The intention behind specifying an error code was to reduce hallucinations, by persuading the model to refrain from guessing. From Table 3.3, we can see that the error code was not output a single time by most configurations. For the few configurations that did output the error code, there is no clear pattern, as they were different models with different configurations. A factor to consider is that our training data did not contain a single occurrence of the error code. Adding synthetic rows containing the error code could lead to the model generating it more frequently.

4 Future Work and Conclusion

The fourth and final chapter presents suggestions for future work within the field, based on the findings from this thesis. Finally, the objectives set for the work are addressed, followed by the associated conclusions.

4.1 Future Work

The approach taken in this thesis has been quite broad, experimenting with several knowledge-injection techniques. Consequently, we were unable to delve too deep into the details of each individual experiment. This introduces many potential avenues for future work.

Using two different models, we showed that the results from introducing different knowledge-injection techniques were generalizable across models. As a general way forward, we recommend experimenting with a single, high-performing model such as Llama3, as opposed to running the same experiments on multiple models. The repetitive nature of using multiple models adds a layer of complexity, while also being significantly more time-consuming.

Fine-tuning as a sole technique produced the best results by far. Therefore, the most obvious path for future work is optimization. We used a single set of hyperparameters for the LoraConfig, TrainingArguments and inference. Experimenting with different hyperparameters to see the effect they have on the accuracy would be highly relevant. Additionally, the frequency of which the correctly predicted ICD-10 codes occurred in training data for incorrect predictions could be assessed. Emphasizing a balanced dataset for both training and testing is also a viable option.

Considering the dataset challenges discussed in Subsection 3.2.1, performing training and testing using different datasets could pinpoint to which degree the model is able to learn, as opposed to memorizing the training data. This approach assumes that both datasets contain the same codes, with distinct wording for the causes of death. A possible alternative to acquiring multiple datasets could be experimenting with synthetic data similarly to [26], for instance by leveraging LLMs ability to paraphrase.

RAG produced worse results than fine-tuning, even deteriorating the results from fine-tuning alone when combined. Nevertheless, more in-depth research on RAG is warranted. We retrieved and concatenated the same number of results from the similarity search for all RAG configurations, without regard for the knowledge source. Moreover, we used the same chunk size. Additional research could experiment with using a different number of results from the similarity search in the injected context, as well as adjusting the chunk size depending on the knowledge source. Another crucial aspect is investigating the contents of the retrieved context, such as the correct code being present, to verify whether incorrect predictions had the necessary information available.

Given the wide array of research opportunities specifically aimed at fine-tuning or RAG, we would recommend prioritizing research focusing on a single knowledge-injection technique as opposed to combining multiple techniques. As the impact of the factors we discussed for each technique within the specific domain of assigning ICD-10 codes from causes of death is more well-established, research combining multiple techniques could be resumed at a later stage.

4.2 Conclusion

The purpose of the research conducted in this thesis was to assess the impact of different knowledge-injection techniques such as fine-tuning and RAG, when predicting ICD-10 codes using historical causes of death. Furthermore, we sought to address the financial feasibility of the process.

Fine-tuning both LLMs that were selected resulted in an accuracy of 88% full matches and 2% partial matches. This is a significant increase from previous research, where the highest accuracy was produced by a base-version of GPT-4, with 58% full matches and 25% partial matches [5]. When considering the accuracy increase by itself, the results could potentially be put into production. However, due to concerns regarding knowledge gained and memorization of training data, further research is warranted for an implementation to be deemed production-ready. On the contrary, we were able to show that fine-tuning produced results were similar for multiple LLMs, ascertaining generalizability.

Adding external knowledge bases on their own did indeed boost the results from a base model, although less than those of fine-tuning. More importantly, combining RAG with fine-tuning produced worse results than solely utilizing fine-tuning. Both pure-RAG configurations performed similarly, with Llama3 producing 63% full, 7% partial match and Mistral producing 58% full, 15% partial match. Considering that the accuracies for both models are around 20% lower than the fine-tuning results, they cannot be deemed as adequate for achieving results using only an external knowledge base.

The financial feasibility of the process was assessed by conducting all development and experiments on personal hardware, without any financial support. The sole expense associated with this work has been electricity, which is negligible given the runtime and power supply limitations on consumer hardware. As we were able to improve the accuracy seen in previous research with these constraints, the process can be labeled as feasible with a relatively low barrier of entry.

Finally, we evaluate the outcome of the thesis as a whole. The ideal scenario for the experiments to be considered a success, was identifying a single knowledge-injection technique that outperformed the others. The results clearly indicate fine-tuning as the best performing knowledge-injection technique. Despite being unable to recommend that fine-tuning be put into production without more specialized research, the focus of this thesis was to act as proof-of-concept. As a commercial product was out of scope, the experiments can be considered successful, with fine-tuning being identified as the primary path forward for future work.

Bibliography

[1] M. A. Alharbi, G. Isouard, and B. Tolchard, “Historical development of the statistical classification of causes of death and diseases,” *Cogent Medicine*, vol. 8, no. 1, p. 1 893 422, 2021.

[2] L. Zhou, C. Cheng, D. Ou, and H. Huang, “Construction of a semi-automatic icd-10 coding system,” *BMC medical informatics and decision making*, vol. 20, pp. 1–12, 2020.

[3] D. Arifoğlu, O. Deniz, K. Aleçakır, and M. Yöndem, “Codemagic: Semi-automatic assignment of icd-10-am codes to patient records,” in *Information Sciences and Systems 2014: Proceedings of the 29th International Symposium on Computer and Information Sciences*, Springer, 2014, pp. 259–268.

[4] W. H. Organization, *International Statistical Classification of Diseases and Related Health Problems 10th Revision*, `https://icd.who.int/browse10/2019/en`, Accessed: 19.02.2024, 2019.

[5] B. Pedersen, M. Islam, D. T. Kristoffersen, *et al.*, *Coding historical causes of death data with large language models*, 2024. arXiv: `2405.07560 [cs.LG]`.

[6] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[7] J. D. Kelleher, *Deep learning*. MIT press, 2019, p. 1.

[8] A. Dongare, R. Kharde, A. D. Kachare, *et al.*, “Introduction to artificial neural network,” *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 2, no. 1, pp. 189–194, 2012.

[9] N. Buduma, N. Buduma, and J. Papa, *Fundamentals of deep learning*. " O’Reilly Media, Inc.", 2022, p. 4.

[10] M. Verma, “Prediction of compressive strength of geopolymer concrete using random forest machine and deep learning,” *Asian Journal of Civil Engineering*, vol. 24, Apr. 2023. DOI: `10.1007/s42107-023-00670-w`.

[11] K. Chowdhary and K. Chowdhary, “Natural language processing,” *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[12] W. N. Locke and A. D. Booth, “Machine translation of languages,” eng, *American documentation*, vol. 7, no. 2, pp. 135–136, 1956, ISSN: 0096-946X.

[13] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[14] HuggingFace, *Models*, `https://huggingface.co/models`, Accessed: 24.03.2024, 2024.

[15] T. Amaratunga, *Understanding Large Language Models : Learning Their Underlying Concepts and Technologies*, First edition. Apress Media LLC, 2023, pp. 81–84, 114–115, ISBN: 979-88-6880-017-7.

[16] T. Brown, B. Mann, N. Ryder, *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[17] J. Achiam, S. Adler, S. Agarwal, *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.

[18] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient finetuning of quantized llms,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[19] P. Lewis, E. Perez, A. Piktus, *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[20] W. H. Organization, *History of the development of the ICD*, `https://cdn.who.int/media/docs/default-source/classification/icd/historyoficd.pdf`, Accessed: 01.04.2024, 2021.

[21] W. H. Organization, *International Statistical Classification of Diseases and Related Health Problems (ICD)*, `https://www.who.int/standards/classifications/classification-of-diseases`, Accessed: 01.04.2024, 2024.

[22] W. H. Organization, *International statistical classification of diseases and related health problems*. World Health Organization, 2016, vol. 2.

[23] W. H. Organization, *ICD-10 Version:2019*, `https://icd.who.int/browse10/2019/en#/A00-A09`, Accessed: 02.04.2024, 2024.

[24] A. Goel, A. Gueta, O. Gilon, *et al.*, “Llms accelerate annotation for medical information extraction,” in *Machine Learning for Health (ML4H)*, PMLR, 2023, pp. 82–100.

[25] A. Soroush, B. S. Glicksberg, E. Zimlichman, *et al.*, "Assessing gpt-3.5 and gpt-4 in generating international classification of diseases billing codes," *medRxiv*, pp. 2023–07, 2023.

[26] M. Falis, A. P. Gema, H. Dong, *et al.*, "Can gpt-3.5 generate and code discharge summaries?" *arXiv preprint arXiv:2401.13512*, 2024.

[27] J. Ong, N. Kedia, S. Harihar, *et al.*, "Applying large language model artificial intelligence for retina international classification of diseases (icd) coding," *Journal of Medical Artificial Intelligence*, vol. 6, 2023.

[28] J. S. Boyle, A. Kascenas, P. Lok, M. Liakata, and A. Q. O'Neil, "Automated clinical coding using off-the-shelf large language models," *arXiv preprint arXiv:2310.06552*, 2023.

[29] C.-W. Huang, S.-C. Tsai, and Y.-N. Chen, "Plm-icd: Automatic icd coding with pretrained language models," *arXiv preprint arXiv:2207.05289*, 2022.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`. [Online]. Available: `https://aclanthology.org/N19-1423`.

[31] O. Ovadia, M. Brief, M. Mishaeli, and O. Elisha, "Fine-tuning or retrieval? comparing knowledge injection in llms," *arXiv preprint arXiv:2312.05934*, 2023.

[32] U. of Tromsø, *DTE-3608 ARTIFICIAL INTELLIGENCE AND INTELLIGENT AGENTS - CONCEPTS AND ALGORITHMS*, `https://uit.no/utdanning/emner/emne?p_document_id=785551`, Accessed: 16.04.2024, 2023.

[33] U. of Tromsø, *DTE-3606 ARTIFICIAL INTELLIGENCE AND INTELLIGENT AGENTS- PROJECT*, `https://uit.no/utdanning/emner/emne?p_document_id=765881`, Accessed: 16.04.2024, 2023.

[34] Python Software Foundation, *Welcome to Python.org*, `https://www.python.org/`, Accessed: 09.05.2024, 2024.

[35] Python Software Foundation, *pip 24.0*, `https://pypi.org/project/pip/`, Accessed: 09.05.2024, 2024.

[36] N. H. D. Centre, the National Archive of Norway, *et al.*, *Historical population register of norway*, Original sources at the National Archive of Norway, [Norsk historiske befolkningsregister 1801-1964, begravelsesprotkoller for Trondheim 1830-1920], UiT the Arctic University of Norway.

[37] Helsedirektoratet, *ICD-10 Den internasjonale statistiske klassifikasjonen av sykdommer og beslektede helseproblemer*, https://finnkode.ehelse.no/#icd10/0/0/0/2596212, Accessed: 19.02.2024, 2024.

[38] W.-L. Chiang, L. Zheng, Y. Sheng, *et al.*, *Chatbot arena: An open platform for evaluating llms by human preference*, 2024. arXiv: 2403.04132 [cs.AI]. [Online]. Available: https://chat.lmsys.org/?leaderboard.

[39] meta-llama, *meta-llama/Meta-Llama-3-8B-Instruct*, https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct, Accessed: 12.05.2024, 2024.

[40] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: %5Curl%7Bhttps://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md%7D.

[41] mistralai, *mistralai/Mistral-7B-Instruct-v0.2*, https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2, Accessed: 14.05.2024, 2024.

[42] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, *Mistral 7b*, 2023. arXiv: 2310.06825 [cs.CL].

[43] Y. Labrak, A. Bazoge, E. Morin, P.-A. Gourraud, M. Rouvier, and R. Dufour, *Biomistral: A collection of open-source pretrained large language models for medical domains*, 2024. arXiv: 2402.10373 [cs.CL]. [Online]. Available: %5Curl%7Bhttps://huggingface.co/BioMistral/BioMistral-7B%7D.

[44] Mistral AI, *Mistral technology*, https://mistral.ai/technology/, Accessed: 10.05.2024, 2024.

[45] X. Han Lu, *dl-translate 0.3.1*, https://pypi.org/project/dl-translate/, Accessed: 09.05.2024, 2024.

[46] S. Han, *googletrans 3.0.0*, https://pypi.org/project/googletrans/, Accessed: 09.05.2024, 2020.

[47] N. Baccouri, *deep-translator 1.11.4*, https://pypi.org/project/deep-translator/, Accessed: 09.05.2024, 2023.

[48] pandas, *pandas.DataFrame.sample*, https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html, Accessed: 09.05.2024, 2024.

[49] World Health Organization, *Chapter XVIII Symptoms, signs and abnormal clinical and laboratory findings, not elsewhere classified (R00-R99)*, https://icd.who.int/browse10/2019/en#/R99, Accessed: 10.05.2024, 2019.

[50] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online:

Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[51] HuggingFace, *Transformers*, https://huggingface.co/docs/transformers/en/index, Accessed: 11.05.2024, 2024.

[52] samchain, *Mistral trouble when fine-tuning : Dont set pad_token_id = eos_token_id*, https://discuss.huggingface.co/t/mistral-trouble-when-fine-tuning-dont-set-pad-token-id-eos-token-id/77928, Accessed: 11.05.2024, 2024.

[53] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*, https://github.com/huggingface/peft, 2022.

[54] pandas, *pandas - Python Data Analysis Library*, https://pandas.pydata.org/, Accessed: 14.05.2024, 2024.

[55] Q. Lhoest, A. Villanova del Moral, Y. Jernite, *et al.*, "Datasets: A community library for natural language processing," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. arXiv: 2109.02846 [cs.CL]. [Online]. Available: https://aclanthology.org/2021.emnlp-demo.21.

[56] LangChain AI, *langchain*, https://github.com/langchain-ai/langchain, Accessed: 12.05.2024, 2024.

[57] NeuML, *PubMedBERT Embeddings*, https://huggingface.co/NeuML/pubmedbert-base-embeddings, Accessed: 12.05.2024, 2024.

A Code Documentation

This appendix serves as documentation for the code written as part of the thesis, which was delivered alongside the report as a compressed archive. The source code is also available on GitHub (`https://github.com/kribw/thesis-submission`), excluding the historical causes of death and fine-tuned adapters. The appendix includes a description of the files submitted, an installation guide, and lastly a user manual describing how to run the fine-tuning and inference scripts.

A.1 File Contents

The following files with associated descriptions are a part of the thesis submission:

`./` (root directory)

benchmark.ipynb Creates the benchmarks for accuracy and character length, referred to in Chapter 3.

create-data.ipynb Splits the translated funeral records into training, validation and test data.

fine-tune.ipynb Fine-tunes the Llama 3 and Mistral models.

inference.ipynb Runs inference for the experiments, producing the predicted ICD-10 codes.

plot-loss.ipynb Plots training and validation loss for both models, in addition to memory usage and runtime from inference.

requirements.txt Libraries required to run all scripts, including the specific versions we used.

translate.ipynb Translates the original causes of death from Danish to English using Google Translate's API.

doc /

icd10h_v3.txt Original dataset provided for the thesis.

predictions.txt Original dataset with columns added for English translation of original cause of death, and the full model response for every configuration.

test.txt Rows 0 to 2000 from translations.txt, used for testing.

train.txt Rows 2500 to 4500 from translations.txt, used for fine-tuning. Columns for the translated causes of death and correctly assigned ICD-10 codes have been extracted, as no other data was necessary for fine-tuning.

translations.txt Original dataset with English translations for original cause of death.

validate.txt Rows 2000 to 2500 from translations.txt with the same shape as train.txt, used for validation during fine-tuning.

llama-ft / Folder containing the fine-tuned Llama 3 adapter.

mistral-ft / Folder containing the fine-tuned Mistral adapter.

rag-sources / Data sources used as external knowledge bases.

correct-test-predictions.pdf Test data with correctly assigned ICD-10 codes.

icd10-tabular-list.pdf Tabular version of every possible ICD-10 code.

A.2 Installation Guide

The libraries required to run every script are specified in the "requirements.txt" file, as stated in the previous section. The specific Python version used was 3.11.4. To install the libraries, simply run the following command in a terminal opened from the directory that text file is located in: `pip install -r requirements.txt`

Additionally, an environment file ".env" must be present in the root directory to load the Llama 3 model. The file must contain a HuggingFace API key specified as `HF_TOKEN=[TOKEN]`, with access granted by Meta. Users can apply for access at: `https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct`.

Lastly, CODA Toolkit is required to run the libraries specific to LLMs. We used version 12.1. Properly configuring this can be challenging depending on the hardware used, and is beyond the scope of this documentation. The toolkit is available at: `https://developer.nvidia.com/cuda-12-1-0-download-archive`.

A.3 User Manual

Once all the libraries and CUDA Toolkit have been installed, the main points of interest are fine-tuning and inference. Both scripts list the necessary modifications at the top of each notebook.

For fine-tuning, the only required adjustment is the model name. This is defined as `model_name =`, where the possible values are `LLAMA3` or `MISTRAL`.

For inference, there are three possible adjustments. The model name is defined in the same manner as for fine-tuning. The `is_fine_tuned=` variable can be set to `True` or `False`, which will load the fine-tuned model adapter accordingly. The RAG source can be specified using `rag_source=`, with possible values being `None` to not use RAG, `RAG1` to load the file containing all correct predictions, or `RAG2` to load the tabular version of all possible ICD-10 codes.

After programmatically defining which parameters should be used, the script can be run in the same manner as any other Jupyter notebook, for instance using Jupyter (`https://jupyter.org/`) or VS Code (`https://code.visualstudio.com/docs/datascience/jupyter-notebooks`).

B Task Description

This appendix contains the original task description for the thesis, which was provided by UiT. The thesis was supervised by Bjørn-Richard Pedersen from UiT-AHR, Andreas Dyrøy Jansson, Shayan Dadman and Rune Dalmo from UiT-IDBI.

UiT THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Engineering Science and Technology
Department of Computer Science and Computational Engineering
UiT - The Arctic University of Norway

Fine-tuning Large Language Models on historical causes of death data

Kristoffer Berg Wilhelmsen

Thesis for Master of Science in Technology / Sivilingeniør

Background

The Norwegian Historical Data Centre (NHDC) has recently performed an exploratory analysis of the performance of Large Language Models (LLMs) when it comes to assigning ICD-10 labels to a dataset consisting of historical causes of death. This study used the default, chat versions of LLMs like GPT-4 and Llama 2 and achieved less than adequate results.

Problem description

Based on the study from the NHDC, one of the findings was that LLMs do have potential for this type of classification task, but additional methods that would boost the models' performance would be required.

The NHDC proposes that experimentation into model fine-tuning be undertaken. A process in which a base version LLM is further trained with highly specific and high-quality data for the intended task. In this case, historical causes of death data from 19th century Trondheim. Additionally, frameworks like Retrieval-Augmented Generation (RAG) which incorporates external knowledge bases into the LLM prompting process could also be used to increase the models' performance.

Scope and limitations
Focus on the academic part and research problems. Implementing a commercial product is out of scope. Proof-of-concept implementation of algorithms are expected to emerge.

Research questions
Will fine-tuning an LLM lead to results which could be put into production?
If so, would this be the case for all/different types of LLM?
Would adding external knowledge bases boost the results even further?
Is it possible to achieve adequate results through the use of external knowledge bases only?
Would this process be financially feasible?

Objectives
Identify the limitations of the problem.
Propose strategies for solving the problem.
Implement a proof of concept / prototype algorithm.

Dates

Date of distributing the task:	<08.01.2024>
Date for submission (deadline):	<21.05.2024>

Contact information

Candidate	Kristoffer Berg Wilhelmsen kwi059@uit.no
Supervisors at UiT-IDBI	Andreas Dyrøy Jansson andreas.d.jansson@uit.no
	Shayan Dadman shayan.dadman@uit.no
	Rune Dalmo rune.dalmo@uit.no
Supervisor at UiT-AHR	Bjørn-Richard Pedersen bjorn-richard.pedersen@uit.no

General information

This master thesis should include:

- ❄ Preliminary work/literature study related to actual topic
  - A state-of-the-art investigation
  - An analysis of requirement specifications, definitions, design requirements, given standards or norms, guidelines and practical experience etc.
  - Description concerning limitations and size of the task/project
  - Estimated time schedule for the project/ thesis
- ❄ Selection & investigation of actual materials
- ❄ Development (creating a model or model concept)
- ❄ Experimental work (planned in the preliminary work/literature study part)
- ❄ Suggestion for future work/development

Preliminary work/literature study

After the task description has been distributed to the candidate a preliminary study should be completed within 3 weeks. It should include bullet points 1 and 2 in “The work shall include”, and a plan of the progress. The preliminary study may be submitted as a separate report or “natural” incorporated in the main thesis report. A plan of progress and a deviation report (gap report) can be added as an appendix to the thesis.

In any case the preliminary study report/part must be accepted by the supervisor before the student can continue with the rest of the master thesis. In the evaluation of this thesis, emphasis will be placed on the thorough documentation of the work performed.

Reporting requirements

The thesis should be submitted as a research report and could include the following parts; Abstract, Introduction, Material & Methods, Results & Discussion, Conclusions, Acknowledgements, Bibliography, References and Appendices. Choices should be well documented with evidence, references, or logical arguments.

The candidate should in this thesis strive to make the report survey-able, testable, accessible, well written, and documented.

Materials which are developed during the project (thesis) such as software / source code or physical equipment are considered to be a part of this paper (thesis). Documentation for correct use of such information should be added, as far as possible, to this paper (thesis).

The text for this task should be added as an appendix to the report (thesis).

General project requirements

If the tasks or the problems are performed in close cooperation with an external company, the candidate should follow the guidelines or other directives given by the management of the company.

The candidate does not have the authority to enter or access external companies' information system, production equipment or likewise. If such should be necessary for solving the task in a satisfactory way a detailed permission should be given by the management in the company before any action are made.

Any travel cost, printing and phone cost must be covered by the candidate themselves, if and only if, this is not covered by an agreement between the candidate and the management in the enterprises.

If the candidate enters some unexpected problems or challenges during the work with the tasks and these will cause changes to the work plan, it should be addressed to the supervisor at the UiT or the person which is responsible, without any delay in time.

Submission requirements

This thesis should result in a final report with an electronic copy of the report including appendices and necessary software, source code, simulations and calculations. The final report with its appendices will be the basis for the evaluation and grading of the thesis. The report with all materials should be delivered according to the current faculty regulation. If there is an external company that needs a copy of the thesis, the candidate must arrange this. A standard front page, which can be found on the UiT internet site, should be used. Otherwise, refer to the "General guidelines for thesis" and the subject description for master thesis.

The supervisor(s) should receive a copy of the the thesis prior to submission of the final report. The final report with its appendices should be submitted no later than the decided final date.