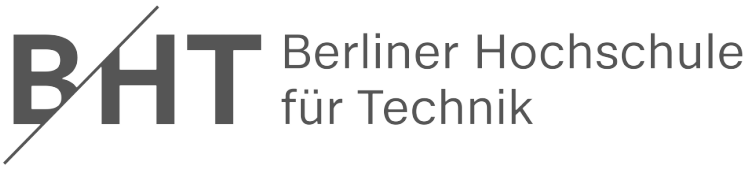

Evaluation reasoning capabilities of language model for clinical outcome prediction

Master Thesis of

Sebastian von Rohrscheidt

Matriculation Number: 102090

Berliner Hochschule für Technik (BHT)
Department VI - Data Science (Master of Science)
Data Science and Text-based Information Systems (DATEXIS)

First Reviewer:	Prof. Dr.-Ing. habil. Alexander Löser
Second Reviewer:	Prof. Dr. Selcan Ipek-Ugay
Advisors:	Dennis Fast, Paul Grundmann

November 29, 2025

Abstract

The development of large language models has progressed rapidly in recent years, and the capabilities of these models are continuously increasing. The in-context learning paradigm assumes that, after training, the models are capable of solving many different tasks and that these capabilities only need to be induced through instructions (prompt). To achieve this, the model does not need to be specifically trained for the task; instead, the instruction must be adapted to the task (e.g. via prompt engineering). The thesis clearly shows that prompts adapted to the task outperform more general prompts. However, this adaptation is not trivial and many attempts are necessary to find high-performing prompts. The thesis shows that prompt adaptation and optimization can be automated. This automation can be achieved because language models can also design and generate prompts. A sophisticated multi-step framework or an agent is necessary to ensure that this adaptation is task-specific and optimized. The thesis shows that in 2025, such agents can be created with manageable effort. The prompts produced in this way significantly outperform manually written prompts (human prompt engineering) and general prompts from the literature.

Acknowledgement

Special thanks go to Dennis Fast and Paul Grundmann, who supervised me during the thesis and always found time for me. They also introduced me to NLP over the last two years and laid the foundations for my knowledge in this area. I would like to thank Prof. Alexander Löser, who made my work at DATEXIS and this thesis possible in the first place. He also provided the impulse for the topic and discouraged others, which made the success of the topic possible. Finally, I would like to thank my friends and family, especially my partner, who supported and endured me throughout the work.

List of Abbreviations

Abbreviation	Definition
NLP	Natural Language Processing
GPT	Generative Pre-trained Transformer
QA	Question-Answering
STEM	Science, Technology, Engineering, and Mathematics
ICL	In-Context Learnin
CoT	Chain of Thought
RAG	Retrieval Augmented Generation
SFT	Supervised Fine-Tuning
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
DNN	Deep Neural Network
PPO	Proximal Policy Optimization
RM	Reward Modeling
ProTeGi	Prompt Optimization with Textual Gradients
ICD	International Classification of Disease
ICU	Intensive Care Unit
HOSP	Hospital
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
DSPy	Declarative Self-improving Pipelines
RQ	Research Questions
H	Hypothesis

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past five years, the development of large language models has accelerated, demonstrating the impressive potential of these models. The capabilities of these models are growing rapidly, as well as the possible fields of application for language models. One type of language model, known as decoder-only architectures, is trained on large text corpora and is then able to generate long texts in natural language. It has become apparent that these models are capable of solving tasks for which they have not been specifically trained. This approach is known as *in-context learning* and was introduced by OpenAI in 2020 with the launch of GPT-3. While fine-tuning a task involves high computational costs, in-context learning can be applied directly and is therefore more cost-effective and straightforward to implement.

This thesis examines in-context learning approaches and methods exclusively, particularly those related to reasoning. Reasoning refers to the model explaining itself and providing information regarding the solution process, i.e. generating how it arrived at a result. These explanations are known as chain of thought. The idea of chain of thought was introduced in 2022, whereby a model explains in chains of thought how the task is to be solved or presents an argument before generating the final solution. The object of this thesis is to discuss the capability of these reasoning argumentation chains, i.e. chains of thought, on a medical task. An important part of the medical field is making diagnoses. These diagnoses can also be produced by language models. This thesis will investigate whether language models can be used to make these diagnoses, i.e. identify diseases. The underlying assumption is that reasoning improves the identification of diseases and also enhances transparency.

As already mentioned, the model is not trained further for the task, so the main influence on the task is the input in form of text that is given to the model, i.e. the prompt. The thesis will quantitatively evaluate different prompts and approaches to writing these prompts in order to identify the influence of these prompts on clinical, i.e. diagnostic, predictions. Running these experiments requires a certain level of computer science expertise. In addition to discussing scientific literature, the work involves preparing, running, and evaluating these experiments. It will be shown that handwritten instructions, or prompts, do not always have the desired effect. The models are very sensitive to these instructions, and it is difficult to predict whether one prompt will produce better results than another. In the second part of the thesis, it is attempted to automate the process of finding and optimizing these instructions. There are many different approaches to automating this process in the literature, usually referred to as multi-step frameworks or agents. Such an agent is developed in the second part, and it should be examined whether these frameworks can produce the expected results, i.e. high-performing prompts. The evaluation of the experiments will show that the automatically generated prompts even outperform those written by humans with respect to their score.

The approaches discussed here are applied exclusively to a medical dataset and evaluated on this dataset. However, it is possible that the methods can also be applied to other areas of NLP. The thesis will show that it is at present possible for a master's student to create such an automated framework or agent.

1.1 Motivation

Improving clinical diagnoses with the support of language models seems reasonable, and it is evident that such systems can offer advantages to physicians and their patients. Diagnoses can be made more quickly and with higher accuracy, physicians can communicate with language models as they would with colleagues, and rare diseases and complications can be highlighted or suggested. However, it has been shown that not every language model is appropriate, and even a suitable language model must be controlled using an adequate method. Previous research by the DATEXIS team at bht has shown that different models and methods to use them lead to significant differences in the performance of clinical diagnosis predictions. A benchmark named CliniBench was developed to compare many different models and the influence of input (prompt) created with different methods

such as zero-shot, few-shot, CoT, and RAG was demonstrated [Grundmann et al. 2025].

This thesis builds on this research and examines the capabilities of a specific prompt technique known as zero-shot CoT prompting. It utilizes the ability of new models (since the end of 2024) to produce reasoning. Specifically designed instructions (prompts) are expected to deliver better results on this task. Therefore, approaches for creating and optimizing such prompts will be investigated. If it can be shown that adapting the prompt with task specific reasoning can improve the quality of predicted diagnoses, this would represent another step toward a helpful medical AI assistant.

1.2 Objective

Many approaches in the literature are intended to improve the reasoning capabilities of models by optimizing the input (prompt). The objective of the thesis will be to examine these approaches and evaluate them experimentally. Handwritten prompts will be compared with inputs generated by a language model, and finally the prompt optimization process will be automated with the help of such language models. These objectives can be summarized in two research questions:

> **RQ1**: Can prompts that induce task-specific reasoning outperform general CoT prompts, i.e. does this approach lead to sophisticated reasoning capabilities and thus better clinical outcome prediction?
>
> **RQ2**: Can the process of writing and optimizing a task-specific prompt be automatized, i.e. can a multi-step framework perform task-specific prompt engineering at a human level or better?

1.3 Outline

To ensure a clear structure, the work has been organized into the following chapters. These chapters are now briefly outlined:

Chapter 2 Background: Every year, new language models are released that offer more capabilities and possibilities than their predecessors. This chapter discusses the training of the models, in particular why the reasoning capabilities increased significantly from 2024 to 2025 and how such CoT abilities are trained.

Chapter 3 Related Work: In order to establish consistent and clear terminology in the work, brief definitions of important terms are provided here. In addition, approaches from the literature on prompt optimization are introduced. These have been divided into methods that call the language model exactly once (single-step) and methods that request the language model multiple times (multi-step). This distinction (single-step and multi-step) is also applied in methodology, implementation, and evaluation to ensure a clear and uniform structure.

Chapter 4 Preliminary Analysis: This chapter deals with three important aspects of the thesis that need to be discussed before planning the experiments: The dataset used, the models used, and the notation used for formalization.

Chapter 5 Methodology: This chapter deals with the approaches of the thesis and the experiments. Six hypotheses are formulated, which will be experimentally tested and answered in the thesis. In addition, the methods used are justified and it is explained why certain methods will not be investigated further. A metric is introduced to ensure uniform evaluation of the experiments and answering of the hypotheses.

Chapter 6 Implementation: The structure of the source code is described without explaining the code line by line. Three pipelines are presented for implementing the experiments, which are combined in a multi-step framework (similar to an agent). The prompts, pipelines, local models, and metrics are described in detail so that they can be reproduced methodically.

Chapter 7 Evaluation and Discussion: The experimental results are presented and analyzed. The six hypotheses are either accepted or rejected. In addition, interpretations are suggested that could explain the observed phenomena.

Chapter 8 Conclusion and Future Work: The thesis is reflected and summarized in terms of its methods and results. The findings are highlighted and interpreted. This includes a discussion of what has not been investigated and what could be the subject of future work. Finally, a possible contribution to research is outlined along with where the prototype agent could lead.

Chapter 2

Background

This chapter will discuss the training of decoder models and examine the two phases of training, pre-training and post-training, in more detail. It is expected that a better understanding of how the models are created will contribute to understanding their properties and potential. The use of the models should become more effective and a theoretical basis for researching their capabilities should be established.

2.1 Training of Decoder Models

The following section deals with the training of decoder models. On the one hand, pre-training is discussed and, on the other hand, the two most important approaches for post-training, namely fine-tuning and RLHF. For pre-training, Qwen models are mainly considered, which are also used for the experiments. For RLHF, instead, models from OpenAI (GPT) are examined, as these were the pioneers of RLHF. A detailed discussion of the Qwen models used in the thesis and their post-training can be found in section 4.2. An attempt is made to indicate a temporal development, as both pre-training and post-training have changed significantly over time. Please note that the training phases refer only to the phases of the model developers and not to possible fine-tuning by third parties.

2.1.1 Pre-training

The first step in training a decoder from scratch is pre-training, whereby the model is trained on a huge text corpus [Sun and Dredze 2024]. The texts are fed into the model in chunks, and the model's task is to continue writing the texts, i.e. first the first token

of the text is entered and the second token is to be generated. Then the first and second tokens are given and the third is to be generated, and so on.

According to many researchers, extending or continuing pre-training significantly improves the capabilities of the models, even if these capabilities only become apparent after fine-tuning [Sun and Dredze 2024]. While the fine-tuning aspect will be discussed in the next section, it should be highlighted here that increasing and improving the training data (especially in pre-training) contributes significantly to the major improvement in the capabilities of the models:

> "Llama 3 uses a standard, dense Transformer architecture [...]. It does not deviate significantly from Llama and Llama 2 [...] in terms of model architecture; our performance gains are primarily driven by improvements in data quality and diversity as well as by increased training scale" [Dubey et al. 2024, p. 6].

The assumption that the amount of data is essential for improving the models is confirmed by the fact that the text corpus grow significantly from generation to generation. New models, i.e. next-generation models compared to previous-generation models, use datasets that are more than twice as large their predecessors for pre-training. For example, Meta uses a pretrain dataset with 1.8 trillion tokens for Llama 2 and approximately 15 trillion tokens for its Llama 3 model family in 2024 [Dubey et al. 2024]. And in 2025, they report a pretrain datasets with 22-40T tokens on the Huggingface model card of Llama 4 (no official paper on the technical report available) [meta-llama n.d.].

The generations of Qwen models from Alibaba make this trend even clearer and are well documented. The first Qwen model was trained on a dataset of up to 3 trillion tokens [Bai et al. 2023]. For Qwen2 models, 7 trillion tokens were used in pre-training [A. Yang, B. Yang, Hui, et al. 2024]. Qwen2.5 has already been trained on 18T token datasets, which was achieved, for example, through multilingual datasets [A. Yang, B. Yang, Zhang, et al. 2024]. And for the latest Qwen3 models, 36T tokens were used for pre-training, including 119 languages and dialects compared to 29 languages for Qwen 2.5 [A. Yang, Li, et al. 2025].

Finally, it should be mentioned that pre-training, which always takes place at the beginning, is also becoming increasingly differentiated and divided into finer steps. 1. General Stage with 30T tokens "language proficiency and general world knowledgeusing" in sequences of 4k tokens [A. Yang, Li, et al. 2025, p. 4]. 2. Reasoning Stage with 5T

Date	Model	Data size	Additional Data
Sep 2023	Qwen	3T tokens	web documents, encyclopedia, books, code, English, Chinese
Jul 2024	Qwen 2	7T tokens	high-quality code, mathematics, multilingual data in 30 language
Dec 2024	Qwen 2.5	18T tokens	synthetic data, down- or up-sampling of low or high-value domains, larger multilingual corpus
May 2025	Qwen 3	36T tokens	STEM data, reasoning tasks, synthetic data (textbooks, QA, Instructions, code snippets), 119 languages

Table 2.1: Generations of Qwen models and the size of the text corpus used to pre-train them.

tokens higher-quality "STEM, coding, reasoning, and synthetic data" in sequences of 4k tokens [A. Yang, Li, et al. 2025, p. 4]. 3. Long Context Stage with less than 1T tokens of high-quality long context data in sequences up to 32k tokens [A. Yang, Li, et al. 2025].

2.1.2 Supervised Fine-Tuning (SFT)

For transformer models (encoders and decoders), fine-tuning initially refers to the process of further training an already pre-trained model on a labeled dataset [Devlin et al. 2018]. This process differs significantly from pre-training, in which unlabeled data is masked and individual tokens that were masked in the input are predicted. Instead, an unmasked task (as input) is given, and the generated response is then compared with the ground truth (label). In older, smaller encoder models such as BERT, one fine-tuned dataset is used for one specific task, and the fine-tuned model is customized for that task. In this way, many fine-tuned models for many different tasks can be trained from one base model.

There are many differences between encoders and decoders, not only in terms of architecture, but also in their pre-training and fine-tuning approaches [Devlin et al. 2018]. However, this thesis discusses decoder models exclusively and will therefore focus on their characteristics in the following.

After pre-training, decoder models have a good understanding of language, but they cannot yet follow instructions [Dubey et al. 2024]. Simplified, after pre-training, these models are trained to continue writing the given text (input) and not, for example, to

answer questions. At this stage, the model would rather continue writing a question than answer it.

Finetuning therefore adjusts the model's behavior to match user expectations. This kind of adjustment is often referred to as *model alignment* and takes place during post-training [Sun and Dredze 2024]. The next two sections will deal with further methods of post-training. In general, supervised fine-tuning takes place after pre-training and requires a labeled dataset.

2.1.3 RLHF

Reinforcement learning from human feedback (RLHF) was first used in the 2022s in post-training of decoder-only models and improved the alignment of these models significantly [Ouyang et al. 2022]. The OpenAI researchers aimed not only to improve task-following capabilities but also a total of three properties that the models should learn through alignment:

- *helpful*: help the user with tasks, very similar to task-following.
- *honest*: don't make up facts (hallucinating) or be misleading.
- *harmless*: don't harm people, e.g. through being biased or toxic.

To come closer to these goals, an attempt should be made to have the decoder models learn human preferences through a reinforcement algorithm. In general, there are two main challenges when fine-tuning decoder models: 1. How to quantify the correctness of an answer. 2. How to scale up the amount of data. Reinforced learning is intended to address both challenges. On the one hand, a reward function should be created that decides how correct or adequate an answer is. On the other hand, this reward algorithm can evaluate the output of a decoder using large amounts of unlabeled data. But how to create such a reward function?

The key concept is to learn the reward function from human feedback [Christiano et al. 2017]. Another research team (OpenAI and Google) achieved good results with such an approach in 2017, but with Atari games and simulated robotic tasks. Rather than manually defining a reward function, they trained a secondary DNN using human feedback, which performs the task of the reward function, i.e. quantify the correctness of an output. Humans were given two outputs and had to decide which would be the better

option. A model (DNN) was trained to predict these human decisions. Based on this prediction, a reward could be passed to the actual model (a robot-controlling DNN) in order to optimize it via RL.

Together with the RL algorithm PPO also developed at OpenAI the InstructGPT team has succeeded in using RLHF for decoder models on a large scale [Ouyang et al. 2022]. In addition to the technical implementation, which will not be discussed in detail here as it would go beyond the limits of this thesis, there is another challenge to the successfully realization of RLHF: The creation of a dataset that contains human feedback.

A clear weakness of RLHF is the people who provide this feedback. In the best case scenario, the model learns to adopt their decisions and values. That is why the InstructGPT team has put a lot of effort into keeping the human feedback process as general as possible: 1. They hired 40 labelers who had previously undergone a test to determine their suitability [Ouyang et al. 2022]. Then they hired additional held-out labelers who would later serve as a validation set. 2. They quantitatively measured the labelers decisions. For example, the overlap or agreement between labelers was about 73%. 3. The majority of the prompts used in the RLHF were not written by OpenAI researchers or the hired labelers, but came from the OpenAI playground API. This is a beta version interface that allows customers to interact directly with OpenAI language models. 4. They grouped the prompts (with and without labels) into categories to ensure the diversity of the datasets. And these were just a few of the approaches they used to address the problems of collecting a diverse dataset containing human feedback.

Finally, they created three distinct datasets for three different phases of post-training: Supervised fine-tuning (SFT) of the InstructGPT model, finetuning of the Reward model (RM) on a downstream task, and finally Reinforcement learning (RL) with PPO of the InstructGPT model after SFT [Ouyang et al. 2022]. These datasets contain different mixes of data created by the hired labelers and customers of the playground API.

Post-train Phase	**Data Size**	**Data Creators**
SFT	13k prompts & answers	more labelers, less customers
RM	33k prompts & feedback	more customers, less labelers
RLHF	31k prompts	customers only

Table 2.2: Datasets of Post-training phases of InstructGPT.

This considerable effort was undertaken in order to better align the models with user expectations and directions. In particular, a decoder model should also correctly follow instructions that did not explicitly occur in post-training. It should generalize this learned behavior:

> "We've seen some evidence that InstructGPT generalizes 'following instructions' to settings that we don't supervise it in, for example on non-English language tasks and code-related tasks. This is an important property" [Ouyang et al. 2022, p. 17].

2.1.4 Changes in post-training

In the previous subsections, an attempt was made to describe the phases of training a decoder model in broad terms. However, the phases vary between different model publishers and also between different generations of models. Unfortunately, the level of detail in the technical reports also varies, and a shift can be seen, for example, at Open AI. Starting with the report for GPT-4, Open AI will no longer publish details about the datasets used and the individual training methods:

> "Given both the competitive landscape and the safety implications of large-scale models like GPT-4, this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar" [OpenAI 2023, p. 2].

Nevertheless, this section examines the development of GPT models, as Open AI was the first to use RLHF for training decoder models and was also a leader in other training methods. It can be assumed that the majority of decoder model publishers follow Open AI's approaches. Following the changes over time in the post-training of GPT models, two aspects become apparent. On the one hand, classic SFT is increasingly being replaced by sophisticated RLHF. On the other hand, the goal of post-training is shifting from simple instruction following to policy following, i.e. determining what behavior is desirable and what is not.

First of all, decoder models are always trained in the pre-training and post-training phases, and this order always remains the same. The model first learns general language knowledge and is then fine-tuned. The learning of instructions mainly takes place in post-training [OpenAI 2023]. More complex instructions, such as generating chains of thought,

will be learned additionally in post-training. Newer models like *OpenAI o1* (end of 2024) will even be specifically trained to generate CoT, which will improve performance across many tasks [Jaech et al. 2024]. Unfortunately, Open AI does not publish any details about how the ability to CoT is trained, only that it is done through RLHF and thus in post-training. It can be assumed that the growing demands on post-training are responsible for classic SFT being replaced by RLHF, because it reflects these small variations more accurately and provides a better overall generalization.

Model (Year)	**SFT**	**RLHF**	**CoT**
GPT-3 (2020)	No (not in paper)	No	No
InstructGPT (2022)	Yes	Yes	No
GPT-4 (2023)	Yes (minor)	Yes (major)	No
OpenAI o1 (2024)	No (not disclosed)	Yes	Yes
GPT-5-thinking (2025)	No (not disclosed)	Yes	Yes

Table 2.3: Timeline of GPT models and their post-training.

Even though Open AI, as mentioned, hardly publishes details about training, especially post-training, the development is clearly recognizable. With InstructGPT, SFT is still an important phase in post-training [Ouyang et al. 2022]. In the GPT-4 paper, RLHF is reported as the main method and SFT is only briefly mentioned [OpenAI 2023]. From the *OpenAI o1* paper onward, SFT is no longer mentioned at all [Jaech et al. 2024]. In some papers, RLHF is referred to as a fine-tuning method, but this should not be mistaken for SFT.

The second aspect is that model alignment is no longer an approach to satisfy user preferences only, but rather a task to mitigate the risks of decoder models. The process of counteracting various types of misuse is often referred to as *mitigation* [OpenAI 2023]. While in the subsection 2.1.3 for InstructGPT three main expectations were defined for alignment - i.e. helpful, honest, and harmless - the list of risks for GPT-4 is longer and more precise: Hallucinations, Harmful content, Disinformation and influence operations, Proliferation of weapons (dual-use), Privacy, Cybersecurity, Potential for risky emergent behaviors, and many more. Risk mitigation seems to be a major focus of the GPT papers from GPT-4 onward. This could also be because other technical details are no longer being published. However, it is possible that other model publishers do not prioritize risk

mitigation to the same level. It is to be hoped that other publishers are aware of their responsibility, as the examples provided by GPT-4 for a model that is not sufficiently aligned are distressing (Please note the "Content Warning") [OpenAI 2023, p. 41]. Since no GPT models are used in the thesis, only one aspect of Open AI's safety policy will be discussed here as an example, namely gray areas such as *dual-use* cases.

Dual-use usually refers to the possibility that a product can be used for both commercial and military purposes. The GPT-5 paper mainly highlights biological and cybersecurity cases [OpenAI 2025]. But there are other problematic gray areas that do not fall under the militarily occupied term dual-use. An example is the distinction between sexual and medical content. This problem is not explicitly mentioned by Open AI, but is part of their safety policy: It is permitted to respond with "contextualized sexual content such as medical, non-pornographic discussion about sexualized content, and erotic jokes" [OpenAI 2023, p. 83]. Furthermore, this problem may also arise in the experiments in this work, since, for example, sexually transmitted diseases may include descriptions of sexual content and could therefore be treated differently from other diseases (due to RLHF training). However, since safety policy implementations are very publisher-specific (in this case, Open AI), a few more general limitations of RLHF will be discussed below.

Another aspect of RLHF's usefulness is testing on various benchmarks, which quantitatively measure the capabilities of the models in different domains. There is evidence that sometimes RLHF has only a minor influence on the performance of specific tasks. For example, Open AI tested the GPT-4 model before and after RLHF on an exam benchmark with multiple choice questions and measured an average performance increase from 73.7% to 74% [OpenAI 2023]. One explanation for this could lead to the continuous improvement of the datasets for pre-training. Datasets are being more filtered and curated over time. It could be assumed that this also increases capabilities without RLHF and that some post-training phases do not further improve performance. However, alignment remains an important post-training task. In addition, RLHF approaches can also lead to the acquisition of undesirable behavior. In its early stages of development, GPT-5 tended to cheat or deceive to obtain higher rewards in RLHF [OpenAI 2025]. This can be particularly problematic when the model makes up missing information in order to generate a complete, reasonable argument. To mitigate this problem, the GPT-5 team attempted to teach the model "to fail gracefully when posed with tasks that it cannot solve", mentioning several times that this is an "open research challenge" [OpenAI 2025, p. 13].

2.2 Summary

The two training phases, pre- and post-training, of decoder models were discussed on a methodological level. The two steps, SFT and RLHF, of post-training were examined in order to understand how the model could evolve from a pure next token predictor to a helpful assistant. Problems and difficulties were also addressed, such as the misuse or risks of these models and how they can be mitigated.

An important aspect for the further course of the thesis is the ability to generate reasoning, i.e. chains of thought. An attempt was made to identify how this ability is developed and manifested in pre- and post-training. A significant change and progression was observed in both phases. Reasoning skills are not only teached in post-training, but are also learned in pre-training with long contexts and argumentation-focused texts, e.g. scientific text, especially math and code. The skills acquired in this way are then to be manifested in post-training, whereby different model publishers weight SFT and RLHF differently. The purpose remains the same: CoT behaviors are encouraged and reinforced. In addition to alignment and risk mitigation, the models should argue and reason more by default.

Chapter 3

Related Work

This chapter deals with recent research and research from the last five years. It highlights some exemplary prompt techniques and traces the evolution of prompt techniques and the use of large language models in the sense of prompt engineering. Furthermore, framework and agent approaches will be discussed, whether and in which way they can be used and implemented, and what could help to compensate for variances across models and datasets.

3.1 Terminology

In order to ensure consistent and clear terminology in this paper, some terms are briefly defined. The definitions presented in the paper "The Prompt Report: A Systematic Survey of Prompt Engineering Techniques" have been followed in order to use terms in accordance with current research [Schulhoff et al. 2024, p. 1].

3.1.1 Models

Decoder model or decoder-only model is a generative transformer that uses only the decoder stack, trained with causal masking to predict the next token in a sequence, making it effective for autoregressive text generation. This term is used instead of the frequently used term LLM to reduce the number of models addressed. Since exclusively decoder-only models are discussed—no encoders and no encoder-decoder models—the word *only* is omitted in the following and this type of model is referred to as decoder models.

3.1.2 Prompts

Prompt refers to the entire input that is fed into a decoder model, excluding model hyperparameters such as temperature or maximum output tokens [Schulhoff et al. 2024]. This thesis works exclusively with text input in natural language. In other cases, input may also include images or audio.

Prompt Template is a blueprint for a prompt that contains gaps or variables that will be filled in later. A simple example is *Please answer this question:* {*Question*}, where the variable {Question} will later be replaced by a question. This allows large datasets with many questions to be handled uniformly.

3.1.3 Prompt components

Directive is a part of the prompt that contains a instruction or a work order. It is probably the most important part of the prompt, as it significantly influences the output. A simple example using the prompt template given above: *Please answer this question:* {*Question*} or *Please rephrase this question:* {*Question*} However, the following terms, such as output format, style, or role, can also contribute to the directive or be part of it.

Shots are examples for the given task. These can be, for example, labeled data points from a training set. A few-shot prompt is a prompt that contains several examples. In contrast, a one-shot prompt contains exactly one example, and a zero-shot prompt contains no examples.

Output Formatting attempts to instruct the decoder model to generate a specific format, such as CSV or JSON. However, not all models are capable of doing this; older and smaller models in particular often fail.

Style Instructions can be considered a type of output formatting. A simple example would be: *Answer briefly and clearly.* However, a major problem when working with decoder models is parsing of outputs. This is crucial when only parts of an output are to be processed further. Style Instructions can help here. An example of this would be: *Start your answer with **Answer**.*

Role used to be a method of placing the model in a person or situation: behave like a doctor. Today, this term mainly refers to the roles that the model learns during post-training. This enables the model to better understand the chat history in a chat situation. System roles fulfill the task of assigning a person or situation to the model in which the

model should respond. An example of this would be *System: You are a writer. User: Please write a poem. Assistant: Roses are red. User: Use more detail. Assistant: Roses are very red.* Here, only the text after Assistant: is generated by the model, and the text after System: and User: belongs to the prompt.

Additional Information is the part of the prompt that provides the model with insights needed to solve the given task. The term *context* is very overloaded and should be avoided in the following, if possible.

3.1.4 Prompting Terms

Prompting is the process of giving an input to a decoder model and letting the model generate an output. The following terms are based on this concept.

Prompt Chain is a method where two or more prompts are given to the model sequentially. A first prompt (usually a prompt template) is used to generate an output. This output (usually only part of the output) is then entered into a second prompt template to generate the final response (second output).

Prompting Technique is the general term for methods that define and suggest a specific structure for a prompt to improve the success of the given task. The prompt should be designed in such a way that it leads to a specific behavior. In some cases, a phrase is given that can be simply integrated into the prompt; in other cases, a basic structure or idea is presented that should lead to the creation of such prompts. Well-known examples are Few-Shots, Zero-Shot CoT, and Plan-and-Solve Prompting.

Prompt Engineering is the process of creating and improving prompts. In most cases, the prompt is changed over several iterations, e.g. rephrased or choosing a different prompt technique, to improve the prompt. Prompt optimization mainly consists of prompt engineering, but can also involve searching for additional information or shots (examples).

Prompt Engineering Technique Prompt engineering technique is a structure or strategy for performing prompt engineering. In literature and research, the process of prompt engineering is usually automated, which requires a clear procedure. Modern methods of prompt optimization usually include one or more prompt techniques, prompt chains, and an iterative process for improving the current prompt. An example would be: CoT + Few Shots. In addition, the CoT prompt template can be rephrased and other (more suitable) shots can be selected.

3.1.5 Frameworks and Agents

Despite the terms already listed, it is often difficult to distinguish between methods for prompt optimization. There are still gray areas, overlapping terms, or simply different definitions. In order to make this thesis clear, the following three terms will be introduced and briefly defined. These terms focus primarily on the implementation of methods and provide a better overview of what is required for each method.

Single-Step methods or frameworks are prompting processes that use exactly one call of the language model.

Multi-Step methods or frameworks are prompting processes that use two or more prompts or prompt templates. It does not matter whether the prompts are chained and thus entered sequentially or whether they run in parallel and are aggregated, for example, as an ensemble method.

Agent are mostly multi-step frameworks in which a LLM triggers an action outside the model through parseable output. This also includes "LLMs which write and record plans" which is why the framework presented here is referred to as an agent [Schulhoff et al. 2024, p. 24].

3.2 Single-Step Prompting

Single-step prompting was defined in subsection 3.1.5 and refers to all approaches that use only one prompt for the final output. The techniques presented here are to be implemented, evaluated, and discussed in this thesis. The selection was made in order to verify reasoning approaches through prompt techniques, and it was ensured that each data point required only one prompt (prompt template). However, there are many other prompt techniques that were less suitable for the thesis and are therefore not mentioned. Survey papers such as "The Prompt Report" provide a good overview of frequently used techniques [Schulhoff et al. 2024, p. 1].

3.2.1 In-Context Learning (ICL)

The field of research investigating prompting methods for large language models begins with the concept of "in-context learning" [Brown et al. 2020, pp. 3–4]. This describes the approach of using a pre-trained (and later also post-trained) decoder model directly

for a task without further fine-tuning it for that task [Brown et al. 2020]. The new task is learned through the context contained in the prompt. *Learning* could be misleading here, as *in-context learning* is meant to imply that the task was not previously learned through fine-tuning, but has now been newly teached through the context [Schulhoff et al. 2024]. But this does not mean that the skill was learned during inference; it may have already occurred during training and thus been learned, and then only emerged through corresponding instructions (context) [Reynolds and McDonell 2021]. However, the term refers to the paradigm that a large language model does not need to be further trained for specific tasks and that it is sufficient to use a well-designed prompt to perform specific tasks.

ICL is also a good example of the problem that terminology can hardly keep up with the speed at which models and methods are developed. As a result, terms are constantly being applied to new methods that did not exist when the terms were introduced. This describes a trade-off between consistency of terms and the inclusion of new methods. New terms are slow to become established, while old terms often do not fit well with all new methods. To illustrate the problem, the timeline for ICL is outlined here.

The term *in-context learning* was introduced by OpenAI in 2020 when they presented their new model GPT-3 in the paper "Language Models are Few-Shot Learners" [Brown et al. 2020, p. 1]. As the name of the paper suggests, ICL mainly referred to few shot approaches at that time and until early 2023. Some researchers still follow this position and equate the term with few shots: "ICL, also called few-shot learning" [Sun and Dredze 2024, p. 3]. When OpenAI then introduced Instruct-GPT at the end of 2022 and demonstrated the capabilities of their new models to the public with ChatGPT (GPT-3.5), more work was done on prompt methods [Ouyang et al. 2022][OpenAI 2023]. This led to the development of methods such as zero-shot CoT, which do not include examples. Some researchers also see ICL as an overarching term for zero-shot and few-shot approaches, while others use the term ICL to refer to few-shot learning only. In 2024, OpenAI is still discussing the usage of the term, whereby they have chosen the definition that will also be used in this thesis: ICL refers to not fine-tuning and task adaptation through prompting (with few-shots or zero-shot instructions) [Schulhoff et al. 2024]. It is foreseeable that the term *context* will also be affected by this uncertainty, which is why it is rarely used in this thesis.

3.2.2 Few-Shot Prompting

Few-shot prompting is an in-context learning method that instructs the model to perform a new task using examples in the prompt [Brown et al. 2020]. It is frequently debated whether this involves learning or whether the model already possesses the knowledge (through pre-training) and the few-shot prompt simply triggers this knowledge [Schulhoff et al. 2024]. In this thesis, few-shot prompting is considered a kind of context approach, whereby the examples contained in the prompt trigger the pre-trained knowledge.

3.2.3 Zero-Shot Prompting

In contrast to few-shot prompting, zero-shot prompting does not include any examples in the prompt [Schulhoff et al. 2024]. This method was introduced a year after few-shot prompting, because the models were only gradually able to follow instructions precisely. However, in 2021, researchers showed that fine-tuned models (GPT-3) can perform zero-shot [Wei, Bosma, et al. 2021]. A later example is zero-shot CoT, which is already based on InstructGPT [Kojima et al. 2022].

3.2.4 Few-shot Chain of Thought

There is a distinction between two main types of CoT prompting: *Few-shot CoT* and *Zero-shot CoT* [Kojima et al. 2022]. Both were invented in close succession, mainly by authors from the Google research team. One of the first papers that systematically use CoT prompting without fine-tuning the model is the paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" [Wei, X. Wang, et al. 2022, p. 1]. They invented the *few-shot CoT* prompting technique and started a entire field of research for CoT prompting, and later also learned CoT (as described in subsection 2.1.4).

It is interesting to note that they already use the term "chain of thought reasoning," i.e. on the one hand, reasoning through chains of thought and, on the other hand, chains of thought that contain reasoning [Wei, X. Wang, et al. 2022, pp. 3, 5, 9, 16]. To this day, this is the most common interpretation of CoT: The model uses chains of thought to argue, i.e. perform reasoning. However, the authors use few-shot examples to induce the model to generate CoT. They incorporate examples into the prompt (few shots), each of which contains a chain of thought. The model is thus encouraged to develop its own chain of thought, as shown in the examples given within the prompt. They refer to this as

"in-context few-shot learning via prompting" [Wei, X. Wang, et al. 2022, p. 2]. It should also be mentioned that the few-shot technique for generating CoT is very complex, as the examples (few shots) are not just question-answer pairs, as in classic supervised learning, but also contain a third part: Question, chain of thought, answer [Wei, X. Wang, et al. 2022].

This chain-of-thought part usually has to be written by humans, which greatly reduces the scalability and general applicability of CoT few-shot examples [Kojima et al. 2022]. It is difficult to measure the influence on performance that the quality of the CoT in the examples has. Although it can be assumed that CoT reasoning written by experts lead to better results, as they contain high-quality arguments. But these experts are not always available, especially for difficult or special tasks. Furthermore, the authors of few-shot CoT themselves address another huge problem, namely that the order of the few shots is not irrelevant [Wei, X. Wang, et al. 2022]. They refer to a paper from January 2021 and also base their argument for the performance variance of permuted few shots. One year before CoT was invented, the paper reports the highest measured performance difference of 54% to 91% when the same few shots are permuted. The authors suggest that two factors could explain these results: Majority Label Bias and Recency Bias [Zhao et al. 2021]. The problems could apply to all few-shot approaches, even though the paper only describes a downstream classification task. Nevertheless, it will be briefly described below.

They refer to *Majority Label Bias* as the observation that the model tends to predict the label that occurs most frequently in the few shot examples. Even stronger seems to be the influence of *Recency Bias*, in which the label of the last few shots is predicted, i.e. the example that appears latest in the prompt is preferred. If this last example has the label *negative*, the average negative prediction rate is much higher (considering a binary classification). However, the Google research team reports permutation-related performance variations of only 1-2% in most cases [Wei, X. Wang, et al. 2022]. This could be explained by the fact that the authors used GPT-3 2.7B in 2021 [Zhao et al. 2021], while the Google research team used models such as GPT-3 175B (as well as PaLM 540B and LaMDA 137B) in 2022 [Wei, X. Wang, et al. 2022]. The author of this thesis assumes that models whose size differs by a factor of 100 are difficult to compare. In addition, the experiments in this thesis will show that the smaller model (Qwen3 8B) shows higher variances in response to small changes than the large model (Qwen3 32B). Although the Google research team reports a dataset (coin flip) with a permutation-related variance of

8-11%, but this dataset again contains a binary classification problem. Since this thesis deals with multi-label classification involving many thousands of labels (ICD-10 codes), the permutation problem discussed here will only be addressed peripherally.

3.2.5 Zero-shot Chain of Thought

In January 2022, the Google research team used the phrase *step-by-step* in their paper, but do not include this phrase in their prompts [Wei, X. Wang, et al. 2022]. In May 2022, the paper "Large Language Models are Zero-Shot Reasoners" was published, which uses such a phrase in the prompt: "Let's think step by step" [Kojima et al. 2022, pp. 1–2]. Thus, they invented zero-shot CoT prompting, which is probably the most frequently used prompting technique to date. Their method consists of adding the phrase *Let's think step by step* to any task encouraging the model to think in chains of thought. In other words, they use a prompt template that ends with this phrase and insert an arbitrary task into this template.

Zero-shot CoT prompting offers two key advantages: First, examples are no longer needed, which can be expensive and problematic to create, as mentioned above [Kojima et al. 2022]. Second, the prompt template no longer needs to be changed for different tasks, which simplifies implementation and improves comparability. They show that zero-shot CoT performs better than zero-shot. If the phrase mentioned above is added to a prompt, the result usually improves. The situation is slightly different when compared to the few-shot CoT method. If the few-shot examples match the task well, few-shot CoT is better. But if the examples are very different from the task, zero-shot CoT performs better. They found that the performance of few-shot CoT fluctuates significantly when the examples in the few-shot prompt do not match the actual tasks.

Finally, an important method used in zero-shot CoT and frequently applied in the following methods should be mentioned, namely the extraction of the answer. Especially with answers in natural language, extraction, usually parsing with regex patterns, can be problematic. To solve this, a second prompt is created and the phrase "Therefore, the answer is:" is appended [Kojima et al. 2022, p. 18]. The following prompt techniques also use similar methods, which is why a uniform formalization is chosen here, regardless of whether the paper uses a different formalization. However, the formalization serves only to clearly illustrate how the prompts are composed and is not intended as a mathematical proof. Nevertheless, the definition should be formally correct, see also section 4.3 for more

details. Following the paper "Formalizing BPE Tokenization", prompts can be defined very simply as a sequence of tokens, although some details have been omitted here for simplicity [Berglund and Merwe 2023, p. 1].

Let $\mathcal{V}$ denote the vocabulary, i.e. the finite set of tokens and $\mathcal{V}^*$ as its Kleene closure, i.e. the set of all finite token sequences over $\mathcal{V}$. Each prompt $\mathbf{P}$ and answer $\mathbf{A}$ is an element of $\mathcal{V}^*$, and the language model is represented as a mapping function $\mathcal{M}$:

$$\mathcal{M} : \mathcal{V}^* \to \mathcal{V}^*, \qquad \mathcal{M}(\mathbf{P}) = \mathbf{A}. \tag{3.1}$$

The prompt $\mathbf{P}$ itself consists of sub-sequences derived from $\mathcal{V}^*$, namely a question $\mathbf{Q}$ and an optional trigger sentence $\mathbf{T}$:

$$\mathbf{P} = \langle \mathbf{Q}, \mathbf{T} \rangle, \qquad \mathbf{P}, \mathbf{Q}, \mathbf{T} \in \mathcal{V}^*, \tag{3.2}$$

which denotes the concatenation of its elements in the given order.

Now it is straightforward to formalize how the zero-shot CoT authors composed their two prompts. They used precisely a sequential two-step prompt technique, in which they created a prompt to generate an answer and another to extract the part of the answer that contains the solution. This can be, for example, a binary decision or a multiple choice, whereby this type of task is often referred to as a classification problem. First, they create a prompt $\mathbf{P_1}$ and enter it into the model to generate CoT as described above, which leads to a solution and almost always contains the solution. Second, they use the first answer $\mathbf{A_1}$ to create a second prompt $\mathbf{P_2}$, whose output $\mathbf{A_2}$ then only contains the solution, which is much easier to parse:

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{Q}, \mathbf{T_1} \rangle, \tag{3.3}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{P_1}, \mathbf{A_1}, \mathbf{T_2} \rangle, \tag{3.4}$$

where M represents the model, $\mathbf{P_1}$ is the first prompt, and $\mathbf{P_2}$ is the second. $\mathbf{A_1}$ is the first output of the model and $\mathbf{A_2}$ is the second. $\mathbf{Q}$ is the question or, in most cases, a data point from a dataset, i.e. the actual task. $\mathbf{T_1}$ and $\mathbf{T_2}$ are so-called trigger sentences, i.e. the actual instruction. In the terminology section, this part of the prompt was referred to as a *directive* in order to avoid overloading the frequently used term instruction. While

Q represents the question of the data point and therefore changes across the dataset, the trigger sentences remain the same across a dataset. $\mathbf{T_1}$, the already mentioned sentence "Let's think step by step", is the essence of zero-shot CoT, and $\mathbf{T_2}$ can vary depending on the dataset, i.e. different task types [Kojima et al. 2022, p. 2]. For multiple-choice questions, the authors use, for example, "Therefore, among A through E, the answer is" and for binary decisions or numerical solutions, they use the $\mathbf{T_2}$ trigger sentence "Therefore, the answer (arabic numerals) is" [Kojima et al. 2022, p. 4].

Although this is technically a multi-step prompting, it is considered single-step prompting in this thesis for two reasons. Firstly, the second step ($\mathbf{P_2}$) does not contribute to the solution, but only extracts it. And secondly, the second step becomes obsolete for the (more recent) models used in this thesis, as it can already be completed in the first step (see subsection 6.1.1). This also applies to single-step prompting techniques mentioned below. Straightforwardly, a few more prompt techniques from the literature can be referenced, all of which build on zero-shot CoT and develop it further in a certain way.

3.2.6 Plan-and-Solve Prompting

In May 2023, researchers from Singapore and China introduced the prompt technique "Plan-and-Solve Prompting" [L. Wang et al. 2023, p. 2]. It is implemented similar to zero-shot CoT, as they use the same setup but with a different trigger sentence. However, this method brings with it a new way of looking at chains of thought, namely task decomposition. While classic chains of thought involve a kind of argumentation sequence, task decomposition specifically addresses subproblems that are first identified and then solved sequentially. Task decomposition could therefore be seen as a subcategory of CoT. approaches to task decomposition already existed before the plan and solve paper, but with multi-step prompting methods, which will be discussed in the next chapter.

The plan and solve authors found a few weaknesses in the use of zero-shot CoT, such as missing reasoning steps and semantic misunderstandings. Although they suspect that the cause of these problems lies in the capability of the model, they attempt to prevent them. In contrast to the simple zero-shot CoT directive *Let's think step by step* they expand the trigger sentence with a type of task decomposition: "Let's first understand the problem and devise a plan to solve the problem. Then, let's carry out the plan and solve the problem step by step" [L. Wang et al. 2023, p. 2].

Using the notation introduced in zero-shot CoT, their approach can be formalized as

follows:

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{Q}, \mathbf{T_1} \rangle, \tag{3.5}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{P_1}, \mathbf{A_1}, \mathbf{T_2} \rangle, \tag{3.6}$$

where a mapping function $\mathcal{M}$ represents the model, $\mathbf{P_1}$ is the first prompt, and $\mathbf{P_2}$ is the second prompt. Likewise, $\mathbf{A_1}$ is the first output of the model and $\mathbf{A_2}$ is the second output, which only contains the extracted solution. The major difference in the method consists of the trigger sentence $\mathbf{T_1}$, which is "Let's first understand the problem and devise a plan to solve the problem. Then, let's carry out the plan and solve the problem step by step," while $\mathbf{T_2}$ "Therefore, the answer (arabic numerals) is" is the same as in zero-shot CoT [L. Wang et al. 2023, pp. 3–4].

Moreover, for some datasets that contain calculations, they attempted to further improve their trigger sentence by adding subphrases such as "extract relevant variables and their corresponding numerals" and "calculate intermediate results" [L. Wang et al. 2023, p. 3]. This resulted in a much longer trigger sentence $\mathbf{T_1}$ that was better suited to the task: "Let's first understand the problem, extract relevant variables and their corresponding numerals, and devise a plan. Then, let's carry out the plan, calculate intermediate variables (pay attention to correct numeral calculation and commonsense), solve the problem step by step, and show the answer." [L. Wang et al. 2023, p. 12].

Interestingly, they sometimes replaced the phrase "devise a plan" with "make a complete plan" which increased performance in some of the datasets and decreased performance in others [L. Wang et al. 2023, p. 13]. The observation that small changes in the prompt can lead to significant impacts was also made in the experiments conducted in this thesis and will be discussed later.

3.2.7 Thread-of-Thought

In November 2023, researchers at Microsoft introduced "Thread of Thought" (ThoT), a kind of advanced zero-shot CoT [Yucheng Zhou et al. 2023, p. 2]. Their structure is very similar to the zero-shot CoT approach, except for a context part $\mathbf{C}$ in the prompt and a different trigger sentence $\mathbf{T_1}$: "Walk me through this context in manageable parts step by step, summarizing and analyzing as we go" [Yucheng Zhou et al. 2023, p. 3].

As in zero-shot CoT, they use a two-step prompting method, whereby the answer $\mathbf{A_1}$

(including the solution) is generated in the first step and the solution $\mathbf{A_2}$ is extracted from the answer $\mathbf{A_1}$ in the second step. In addition, they add a prompt part with context $\mathbf{C}$ in step one ($\mathbf{P_1}$). This context can be data point specific or dataset specific, i.e. the same across a dataset. In the notation established above, they combine their two prompts as follows:

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{C}, \mathbf{Q}, \mathbf{T_1} \rangle, \tag{3.7}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{P_1}, \mathbf{A_1}, \mathbf{T_2} \rangle, \tag{3.8}$$

where $\mathcal{M}$ represents the model, $\mathbf{P_1}$ is the first prompt, and $\mathbf{P_2}$ is the second. $\mathbf{A_1}$ is the first output of the model and $\mathbf{A_2}$ is the second. $\mathbf{Q}$ is the question or, in their setting, a query that asks for information from a chat history, for example. $\mathbf{C}$ is the context, in this example, the chat history. The essence of prompt technology is again the trigger sentence $\mathbf{T_1}$: "Walk me through this context in manageable parts step by step, summarizing and analyzing as we go" and $\mathbf{T_2}$ is again used for answer extraction and is specified as "Therefore, the answer:" [Yucheng Zhou et al. 2023, p. 4].

Overall, this prompt method was invented for a specific situation. While Plan-and-Solve mainly addresses mathematical and logical questions, the ThoT approach focuses on tasks with chaotic or long contexts. They report that, on the one hand, established CoT techniques have problems with chaotic contexts and, on the other hand, long contexts are difficult to tackle with retrieval approaches, for example. In both cases, they are attempting to develop a single-step prompting method (such as zero-shot CoT) again.

3.3 Multi-Step Prompting

3.3.1 Least-to-Most Prompting

A Google Research Team invented *Least-to-Most Prompting* in early 2022 [D. Zhou et al. 2022]. This prompting technique is intended to serve here as a leading example of multi-step frameworks. Furthermore, it was developed a year before Plan-and-Solve prompting and is one of the first papers to address task decomposition as a further development of CoT. However, it is not implemented, therefore it will only be discussed briefly. First, it can be noted that least-to-most prompting is a few-shot method. The authors do not use

trigger sentences or additional directives, as in zero-shot CoT or plan-and-solve. Instead, they use the same approach as few-shot CoT. The model is given examples of how to solve the current task and is therefore encouraged to follow this solution scheme. This applies to all steps in their multi-step approach. First, the actual task is divided into at least two sub-tasks, usually questions. The first sub-question is an intermediate step and the second sub-question refers to the final answer, i.e. the solution.

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{F_1}, \mathbf{Q} \rangle, \tag{3.9}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{F_2}, \mathbf{Q}, \mathbf{A_{11}} \rangle, \tag{3.10}$$

$$\mathcal{M}(\mathbf{P_3}) = \mathbf{A_3}, \qquad \mathbf{P_3} = \langle \mathbf{F_3}, \mathbf{Q}, \mathbf{A_{11}}, \mathbf{A_2}, \mathbf{A_{12}} \rangle, \tag{3.11}$$

where $\mathcal{M}$ represents the model, $\mathbf{P_1}$ is the first prompt, $\mathbf{P_2}$ is the second, and $\mathbf{P_3}$ is the third. $\mathbf{A_1}$ is the first output of the model, $\mathbf{A_2}$ is the second , and $\mathbf{A_3}$ is the third. $\mathbf{Q}$ is the question, e.g. math word problem. $\mathbf{F_i}$ denotes the few shots for every step. $\mathbf{F_1}$ contains examples that only show task decomposition and not how to solve theses subtasks. $\mathbf{F_2}$ provides examples showing how to solve the first subtask, and $\mathbf{F_3}$ shows how to solve the second subtask. $\mathbf{A_1}$ contains the subtasks into which the original question $\mathbf{Q}$ was divided. In the case of two subproblems, $\mathbf{A_1}$ contains subtask $\mathbf{A_{11}}$ and subtask $\mathbf{A_{12}}$. $\mathbf{A_2}$ also contains the solution to $\mathbf{A_{11}}$. All three, $\mathbf{A_{11}}$, $\mathbf{A_{12}}$, and $\mathbf{A_2}$, are included in the third prompt $\mathbf{P_3}$ to generate $\mathbf{A_3}$. In this way, all previous subtasks and their solutions are added to the next prompt. This procedure can easily be generalized to decompositions with more than two subtasks.

Unfortunately, the authors do not specify how such a multi-framework can be implemented and, in particular, automated. The paper does not explain how the subtasks $\mathbf{A_{1i}}$ are extracted from the response $\mathbf{A_1}$. It cannot be completely excluded that this is done manually by humans. Nevertheless, least-to-most prompting is an interesting paper, as it introduces both decomposition and multi-step prompting for solving a higher-level problem.

3.3.2 Meta Prompting

Meta prompting was already mentioned at the beginning of 2021: "[W]e introduce the concept of *metaprompt programming*, an approach which offloads the job of writing a

task-specific prompt to the language model itself" [emphasis in original] [Reynolds and McDonell 2021, p. 2]. This work is also interesting because it describes a paradigm that this thesis also follows.

First, they argue that zero-shot prompting can be more effective than fine-tuning or few-shot learning to extract knowledge or task-specific behavior learned in pre-training. Fine-tuning is not required because the tasks have already been learned in pre-training. They do not consider few-shot prompting to be actual learning through context, but rather as guidance for locating the task in the already learned space of behavior. This is why zero-shot methods also show similar or sometimes better performance, even though they do not contain any examples that could be learned from. They argue that the goal is to find an instruction (directive) that leads as well as possible to task-specific behavior.

Secondly, the authors consider having a good understanding of how the models are trained to be an important aspect of working with prompting techniques. They argue that language models, like other models, are trained to approximate a function. "It is the function of human language" as it appears in the training data, e.g. books, articles, websites [Reynolds and McDonell 2021, p. 4]. In the process, the models learn to understand not only language, but also human behavior and details about the physical world that have been expressed in language (training data). In pre-training, the models learn to predict the next token, i.e. the next word in the original source (training data). However, the source can also be a "conversation between theoretical physicists" [Reynolds and McDonell 2021, p. 4]. This would require not only language skills, but also domain-specific knowledge. This knowledge is therefore incorporated by the pre-trained (and later also post-trained) model and should be made apparent or localized through appropriate instructions, via zero-shot directives, few-shots, CoT, execution plans, etc.

The authors suggest that task-specific prompts perform better than general prompts, which, for example, only stimulate general behavior (like zero-shot CoT). They propose using the model itself to perform this task specification. However, they use this idea in a single-step setting. They let the model first generate a task description and then the solution to the task, all in one step. This method is more similar to zero-shot CoT methods such as plan-and-solve or thread-of-thought, which is impressive since these methods were developed 1-2 years later.

On the other hand, another paper, "Large Language Models Are Human-Level Prompt Engineers" from November 2022, introduces a meta prompting method that is frequently

used in the following years [Yongchao Zhou et al. 2022, p. 5]. They let a model create a zero-shot prompt with instructions, which can then be tested on a benchmarking sub-dataset [Yongchao Zhou et al. 2022]. In the simplest case, the procedure can be described as follows:

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{D}, \mathbf{C} \rangle, \tag{3.12}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{A_1}, \mathbf{Q} \rangle, \tag{3.13}$$

where $\mathcal{M}$ represents the model, $\mathbf{P_1}$ is the first prompt, and $\mathbf{P_2}$ is the second. $\mathbf{A_1}$ is the first output of the model and $\mathbf{A_2}$ is the second. $\mathbf{D}$ is a directive that asks the model to write a prompt: "Generate a variation of the following instruction while keeping the semantic meaning" [Yongchao Zhou et al. 2022, p. 5]. $\mathbf{C}$ is task-specific content, like a prompt that was written by hand before. But $\mathbf{C}$ can also consists of few-shots, which are examples from the task, or a task description. $\mathbf{A_1}$ is then the prompt instruction produced by the model, which is entered into the model as the second step ($\mathbf{P_2}$) together with the actual task $\mathbf{Q}$. This second step usually covers an entire dataset and is very similar to the single-step zero-shot methods already discussed.

3.3.3 Automatic Prompt Engineer (APE)

The authors of "Large Language Models Are Human-Level Prompt Engineers" present not only a meta prompting method in which a single prompt is generated and tested, but also a multi-step framework that generates various prompt candidates, tests them, and then selects the best prompt [Yongchao Zhou et al. 2022, p. 5]. They call this multi-step framework "Automatic Prompt Engineer (APE)" [Yongchao Zhou et al. 2022, p. 2]. They show that it can be used to refine (task adaptation) and improve (performance) zero-shot CoT prompts, which is also the goal of this thesis [Yongchao Zhou et al. 2022]. Furthermore, they also demonstrate that few-shot methods can be improved by APE or other desired behaviors, such as truthfulness or informativeness. Since this thesis is limited to zero-shot CoT methods, only these are discussed here.

The paper discusses two different approaches to finding or optimizing a prompt. One method involves providing few shot examples and using them to create a prompt. The other method is based on an initial prompt, which is then to be improved. In the first

method, they refer to a naive Monte Carlo search, and in the second, to an iterative Monte Carlo search method. With Monte Carlo search, they express that the model creates a prompt more arbitrarily than systematically and refer to this as a black-box optimization problem. By naive, they mean a one-step process, and by iterative, they mean a repetitive Monte Carlo search process. Nevertheless, both processes are structured similarly and can be formalized in the same way, as described in subsection 3.3.2:

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{D}, \mathbf{C} \rangle, \tag{3.14}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{A_1}, \mathbf{Q} \rangle. \tag{3.15}$$

The difference between the two methods consists in the given context $\mathbf{C}$. In the first approach (meta prompting), several few shots (always with ground truth) are shown as context, and the model is instructed to design a prompt that predicts the labels for these examples. In the second approach (APE), a previously created prompt appears in the context and is used as an initial prompt, with the directive that this prompt should be further improved.

At this point, an important disadvantage of meta-prompting should be mentioned, particularly with regard to APE. This difficulty is rarely mentioned in research, but is a direct consequence of it. Consider a two-step process, where the first step is to create a prompt for the task and the second step is to test this prompt, i.e. assign it a score. However, the first step also requires a prompt template, e.g. a directive to craft a prompt for step 2. This first directive can also be improved, for instance in an automatic framework. This results in three steps. If each step is now executed with, for example, 10 variations, the total number of iterations is 10^3. The number of steps is in the exponent, which is why this problem grows exponentially with each step and therefore scales very poorly.

3.3.4 Prompt Optimization with Textual Gradients (ProTeGi)

In May 2023, researchers at Microsoft introduced a multi-step framework designed to make automatic prompt optimization less random [Pryzant et al. 2023]. They still refer to a Monte Carlo search process that paraphrases the prompt. But they try to perform this along a textual gradient that is supposed to describe the error of the initial prompt. In this sense, they refer to it as a *gradient*. Whether such text gradients exist and whether they

can be calculated has not been clarified conclusively in research, nor is such a discussion part of this thesis. However, a trivial counterargument can be made: A token is a vector in the latent space of the model. A sentence consisting of several words usually also consists of several tokens. But this token sequence is a sequence of vectors and not a single vector. It has not been proven that sequences of vectors can be used to perform calculations and that, for example, gradients between these sequences can be found, or that such sequences of vectors can serve as gradients.

The authors approach is very detailed and includes several algorithms that will not be described in detail here. in a nutshell, their procedure is as follows: 1. Test an initial prompt on a labeled dataset. 2. Give the model examples of incorrect predictions along with the ground truth and output error strings, which should serve as gradients. 3. Provide the model with these error strings and the initial prompt, requesting that the prompt be rewritten to avoid the listed errors. The iteration could end here, but they add another step: 4. Feed the possible new prompt into a model to paraphrase it and thus explore the "local monte carlo search space around the new prompt candidates" [Pryzant et al. 2023, p. 3]. The function of the fourth step seems to be to expand the set of possible new prompt candidates and enable parallel testing of many prompt candidates.

The following is intended to be a simplified formalization that gives an idea of ProTeGi, but is not a complete description of their framework (step 4 has been omitted for simplicity):

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{D_1}, \mathbf{Q} \rangle, \tag{3.16}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{D_2}, \mathbf{Q}, \mathbf{A_1} \rangle, \tag{3.17}$$

$$\mathcal{M}(\mathbf{P_3}) = \mathbf{A_3}, \qquad \mathbf{P_3} = \langle \mathbf{D_3}, \mathbf{A_2}, \mathbf{D_1} \rangle, \tag{3.18}$$

where $\mathcal{M}$ represents the model, $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ represent the input, and $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$ represent the corresponding output. $\mathbf{D_1}$, $\mathbf{D_2}$, $\mathbf{D_3}$ is the directive for the corresponding step, e.g. find the errors or rewrite the prompt. $\mathbf{Q}$ is the actual task from the dataset, i.e. a data point. $\mathbf{A_1}$ contains the predictions for data points, $\mathbf{A_2}$ contains the errors detected by the model in $\mathbf{A_1}$, and $\mathbf{A_3}$ contains the newly constructed prompt, or more precisely, the directive $\mathbf{D_1}$ of the first prompt $\mathbf{P_1}$. Thus $\mathbf{D_1}$ is improved iteratively, and it is assumed that $\mathbf{D_1}$ will eventually end up at a local maximum and the score of $\mathbf{P_1}$ will not improve

any further. However, this could not be proven experimentally in this thesis.

Even though some statements in the paper could be criticized as interpretations, two aspects are nevertheless interesting and should be highlighted. First, they use examples of incorrectly predicted data points to improve the prompt. This idea is comparable to classical supervised learning and is therefore plausible. Second, like other prompting approaches, they use *tags* to mark part of the answer and thus make it easier to parse them: "Wrap each reason with <START> and <END>" [Pryzant et al. 2023, p. 10]. Both approaches are implemented in this thesis, whereby the second aspect enables corresponding multi-step frameworks or agents in the first place.

3.4 Summary

The related work chapter highlights the methods from the literature that were used in the thesis or contributed to the development of own methods. First, a terminology was introduced and a notation was developed to categorize and uniformly describe the prompting methods. Second, the two categories of prompt techniques, single-step and multi-step prompting, were discussed. In both categories, the chronological order of the methods was used to indicate their development over time.

Please note that this categorization is implementation-focused and not method-oriented, making overlap and chronological jumps inevitable. However, the effort required for implementation is crucial for the thesis in order to decide which methods and how many of them can be implemented. Finally, larger concepts such as APE or ProTeGi frameworks were outlined, which should lead to a self-developed multi-step framework with meta prompting and self-criticism (to be called agent).

In addition, the chapter highlights the emergence and development of CoT and distinguishes between few-shot and zero-shot methods. It shows why few-shot methods have a different focus than zero-shot methods and thus justifies that this thesis focuses on one, namely zero-shot methods.

Chapter 4

Preliminary Analysis

The objectives of this thesis include a multi-label classification problem, which is to be addressed using a dataset, decoder models, and prompts. The three main components are analyzed in detail in the following chapter in order to identify potential difficulties as well as opportunities. After examining the dataset and the models used, a mathematical formalism for prompts is developed, which was already used in chapter 3. This is intended to document all methods in the thesis in a uniform, clear, and comparable way.

4.1 Dataset MIMIC-IV

A diverse and comprehensive clinical dataset called MIMIC-IV was chosen for the experiments in this thesis. The data was collected by the Beth Israel Deaconess Medical Center (BIDMC) and the Massachusetts Institute of Technology (MIT) and contains over 400,000 notes from approximately 180,000 patients (hosp) [Johnson et al. 2023]. But only a small subset of the MIMIC-IV dataset was used for the experiments in this thesis. This dataset is publicly available "via PhysioNet" [Johnson et al. 2023, p. 4]. But the data is restricted by usage guidelines, which is why it is not published in the thesis repository and is not included in the code for this work. Furthermore, the thesis does not show any qualitative analyses of the exact inputs and outputs of the LLMs, as this could be considered a publication of sensitive information.

The data is divided into two modules, *hosp* and *icu*, where hosp contains patients who were hospitalized and icu only includes patients who were transferred to the intensive care unit (ICU). However, only a small part of the hosp data was used in the thesis to keep

computational costs low. It is possible, of course, to repeat the experiments for all other subsets of the MIMIC-IV dataset.

The ground truth labels of the data are the common ICD code, i.e. International Classification of Diseases (ICD) [Johnson et al. 2023]. These codes are also used, for example, for billing health insurance companies. The dataset contains ICD-9 and ICD-10 codes, although again a restriction was made and only the ICD-10 codes are included in the thesis. The data was not divided and preprocessed by the author, but rather the splits and preparations of previous works by the DATEXIS research team was used to ensure comparability [Aken et al. 2021] [Grundmann et al. 2025]. To further ensure direct comparison with the *CliniBench* benchmark, the validation split of their data was used [Grundmann et al. 2025]. This subset, called *val* dataset, comprises approximately 8k data points, while the train split contains approximately 60k and the test split 20k data points. It was assumed that 8k data points would be sufficient for the purposes of the thesis to highlight the important aspects without unnecessarily increasing the computational costs and duration of the experiments.

The data preprocessing was also conducted by the mentioned researchers from DATEXIS. Only information that was already known at the time of the patients admission to the hospital was extracted [Grundmann et al. 2025]. This resulted in *admission notes* that did not contain any information collected during or after the hospital stay. Please note that patients may appear more than once in the dataset because they had more than one hospital stay. For this reason, when referring to a data point, the term *admission notes* is used instead of *patient* in most cases.

Finally, the labels were separated into two variants: Short codes and long codes. Long codes contain the full ICD-10 codes, e.g. A12345, while short codes are limited to only the first 4 digits, e.g. A123, whereby the short codes ensure better fine-tuning of encoder models [Grundmann et al. 2025]. All evaluations in this thesis were performed on short and long codes, which should contribute to comparability with CliniBench. However, the plots in this thesis were only created using the long code metric to ensure consistency and clarity.

4.1.1 Challenges with the MIMIC-IV dataset

The established MIMIC-IV dataset offers many advantages, such as a large number of patients, hospital stays, and a wide variety of diseases. However, a dataset as large and

diverse as MIMIC-IV also presents a number of challenges. One of these is simply the number of ICD-10 codes that occur. In the following, the validation dataset used in the experiments is examined, assuming that it adequately represents the hosp module of MIMIC-IV.

Even though the thesis deals with language models and long responses containing chains of thought, the dataset implies a traditional (downstream) multi-classification problem. Diseases are considered labels (ground truth) in form of ICD-10 codes. The more different labels there are in the dataset, the more difficult it could be to distinguish and precisely assign these labels. The val dataset contains 6341 unique ICD-10 codes and thus labels.

The CliniBench paper discusses the imbalance of labels in more detail [Grundmann et al. 2025]. They refer to the rare ICD codes as the tail of the label space and the frequently occurring ICD codes as the head. It is to be expected that the diseases (codes) assigned to the head are easier to predict than the tail codes and are therefore more frequently predicted correctly. However, for the sake of simplicity, the evaluation in this thesis does not distinguish between individual diseases, but treats all diseases and patients equally.

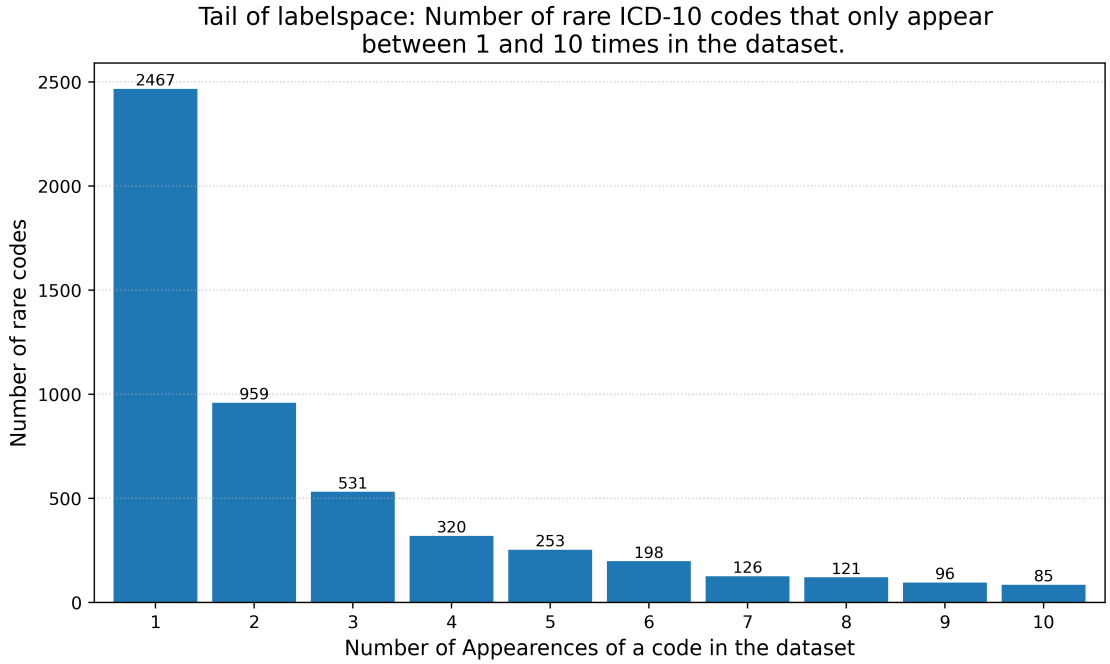

Figure 4.1: The histogram shows the 5k rarest ICD-10 codes from the val dataset (with 6k unique codes). This tail of the label space contains codes that occur in only one to ten admission notes. The histogram shows that the majority of codes occur very rarely.

Therefore, label imbalance is only mentioned briefly. The data from the experiments could also be further examined for this problem, e.g. to identify prompts that are particularly good at predicting rare diseases. However, this was omitted due to the limitations of the thesis.

An analysis of the validation dataset shows the label imbalance mentioned above, with Figure 4.2 indicating the head and Figure 4.1 indicating the tail of the labels. Figure 4.1 shows all rare codes that occur in only 1, 2, or up to 10 admission notes; for example, there are 2,467 codes each of which is mentioned in only one note. All ICD-10 codes that occur in 10 or fewer admission notes are considered, which is the threshold for the tail as defined by the CliniBench authors. The rare codes visualized in Figure 4.1 add up to over 5k codes, which represents over 80% of the unique diseases (6k unique codes in val dataset). This will be a great challenge for classification, and it can be expected that the models will not correctly recognize 100% of the diseases, but rather 10-30% as reported by CliniBench.

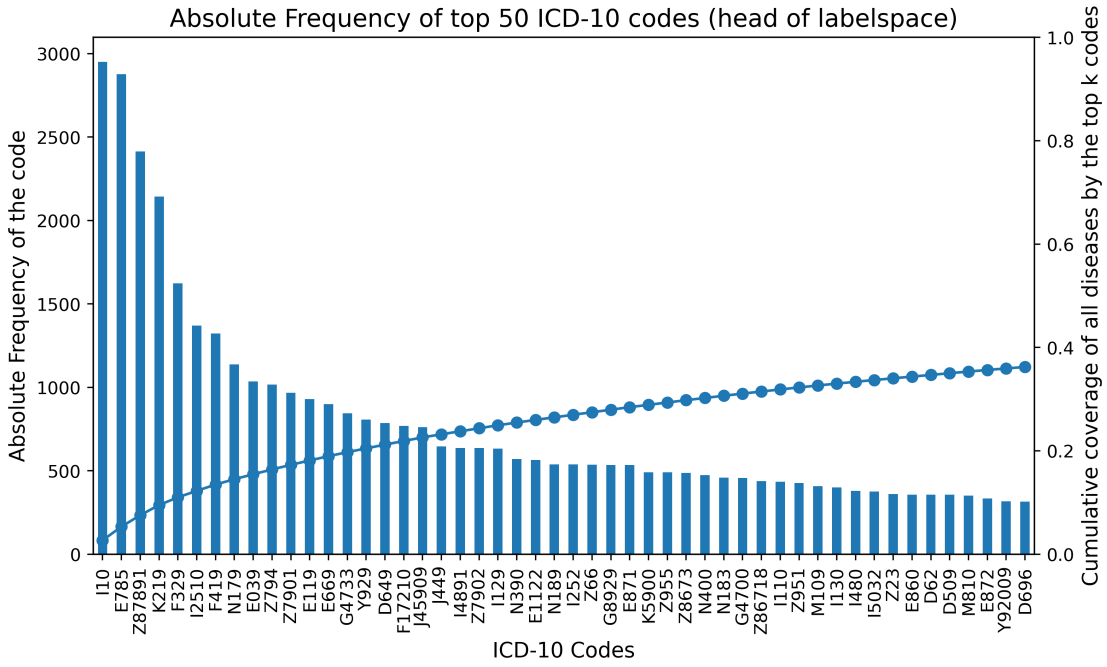

Figure 4.2: The histogram shows the absolute frequency of the 50 most occurring ICD-10 codes (left scale) and their cumulative coverage (right scale) of all diseases in the val dataset. This indicates the head of the label space. The plot shows that a few frequently occurring codes cover the majority of all diseases.

On the other hand, there are also ICD codes that occur very frequently. Figure 4.2

shows some of the head labels mentioned. The first ICD-10 code (I10) occurs in almost 3k admission notes, which is about 35% of all data points (val dataset). In addition, the plot shows that the 50 top ICD-10 codes account for almost 40% of all diagnosed diseases. For example, if a patient has 10 diagnoses in ICD-10 code form, statistically 40% of these are one of the codes shown here. These diseases should be predicted correctly more often because they simply occur more frequently for patients. A language model trained on large text corpora that also contain medical data should therefore be able to recognize and identify these diseases with higher precision.

4.2 Decoder Models Qwen Family

The following section provides an overview of the models used and focuses in particular on the training of thought chains. The experiments were limited to decoder models from the Qwen family in order to make the results as comparable as possible. Nevertheless, two generations (Qwen2.5 and Qwen3) were to be covered, as well as three different sizes from each generation. Although the two generations were released in close succession, i.e. their technical reports, they differ significantly in terms of their capabilities. Since the sizes of the models of both generations are very similar, these differences appear to be mainly due to the training [A. Yang, B. Yang, Zhang, et al. 2024] [A. Yang, Li, et al. 2025]. Table 4.1 shows an overview of the generations and their training, where the details are explained in more detail in the next section.

Date	Generation	Models used	Data size	SFT	RLHF	CoT
Dec 2024	Qwen 2.5	7B, 14B, 32B	18T	Yes	Yes	Yes
May 2025	Qwen 3	8B, 14B, 32B	36T	Yes	Yes	Yes

Table 4.1: All Qwen models used in the thesis with their technical details and a focus on their training.

The table also shows whether the models were trained via supervised fine-tuning (SFT) and RLHF and whether they were specifically trained on chains of thought (CoT). The data size indicates the size of the training data in tokens.

4.2.1 Comparison of Qwen Generations

As mentioned above, there are major differences in the training of the Qwen2.5 and Qwen3 models. This is particularly evident in the size of the training data (see Table 4.1), but also in the explicit mention of CoT training in the paper. Qwen2.5 appears to have been trained only peripherally on CoT [A. Yang, B. Yang, Zhang, et al. 2024]. This is not surprising, as the pioneer OpenAI published explicit CoT training through RLHF in December 2024 [Jaech et al. 2024]. Nevertheless, the authors of Qwen2.5 state that their model is capable of reasoning and that they based their work on the OpenAI o1 paper [A. Yang, B. Yang, Zhang, et al. 2024]. They report that SFT was mainly used to teach the model chains of thought, e.g. through mathematical chain of thought datasets. RLHF, on the other hand, was only used to align the models, e.g. to ensure truthfulness, helpfulness, and harmlessness. They also described the use of long contexts in both pre- and post-training (SFT). The abilities learned through this training, i.e. to understand long contexts and generate longer responses, could also contribute to basic reasoning capabilities.

In the development of Qwen3, on the other hand, CoT was focused in post-training, whereby the model was made capable of flag-based thinking and non-thinking modes [A. Yang, Li, et al. 2025]. Flag here means that the tokens were not specifically trained in post-training, but are known from pretraining and were learned as think triggers in post-training (mainly SFT). Thinking mode refers to the model's response beginning with the flag $< think >$. The model then generates a reasoning string and closes it itself with the flag $< /think >$. In non-thinking mode, the response begins with $< think >< /think >$, leaving the reasoning string empty. However, the non-thinking mode is not further considered in the thesis, as first experiments have shown that it is not suitable for the task, i.e. in non-thinking mode, performance significantly decreases with the same prompt.

In contrast to Qwen2.5, the Qwen3 authors report a SFT with long chains of thought followed by RLHF training focused on CoT mathematics and coding tasks. These two phases represent the first stages of post-training. In the later phases, more general tasks with and without CoT are trained, first with SFT and then followed by RLHF.

In addition to upgrading post-training, the pre-training was also significantly extended with a dataset twice as large (see Table 4.1). This was achieved by using data from different languages (119 languages) as well as synthetic data, i.e. data generated by other models,

such as specialized models like Qwen2.5-Math or Qwen2.5-Coder. The contribution of this pre-training data to reasoning abilities is also evident in the naming of their three training steps: “General Stage (S1)”, ”Reasoning Stage (S2)”, and ”Long Context Stage” [A. Yang, Li, et al. 2025, p. 4]. The models thus seem to learn reasoning capabilities through reasoning data.

4.3 Mathematical Formalism

This section aims to ensure mathematically correct notation. It also demonstrates that neither a perfect prompt can be found, nor can a brute force method be used to find nearly perfect prompts. In order to develop such a mathematical formalism, the thesis follows *Formalizing BPE Tokenization* [Berglund and Merwe 2023]. This mainly involves the use of Zermelo-Fraenkel set theory and a consideration from formal language theory, namely the Kleene star.

Consider vocabulary $\mathcal{V}$ to be a set that contains all tokens on which the model was trained.

$$\mathcal{V} = \{v_1, ..., v_t\}, \tag{4.1}$$

where v is a token and $\mathcal{V}$ is the vocabulary (formally an alphabet). Let $\mathcal{V}^*$ be the set that contains all possible sequences of these tokens:

$$\mathcal{V}^* = \{\langle v_1, ..., v_k\rangle | k \geq 0, v_i \in \mathcal{V}\}, \tag{4.2}$$

where $\mathcal{V}^*$ is the set of all finite-length strings over the alphabet $\mathcal{V}$ (formally $\mathcal{V}^*$ is the Kleene closure of $\mathcal{V}$). Furthermore, $\langle v_1, ..., v_k\rangle$ describes a sequence of tokens or encodes a string of characters. Such a finite sequence can be easily defined in pure set theory as a set of sets:

$$\langle A, B, C\rangle = \{\{A\}, \{A, B\}, \{A, B, C\}\}, \tag{4.3}$$

where A, B, C could be sets and thus sequences as well. The order of the elements is therefore defined with the set of sets, whereby the cardinality of the set determines the position of the new element in the order.

In addition, consider the model $\mathcal{M}$ as a function that maps sequences of tokens (input) to sequences of tokens (output).

$$\mathcal{M} : \mathcal{V}^* \to \mathcal{V}^*. \tag{4.4}$$

Thus, prompt $\mathbf{P} \in \mathcal{V}^*$ can be mapped to answer $\mathbf{A} \in \mathcal{V}^*$ with

$$\mathcal{M}(\mathbf{P}) = \mathbf{A}. \tag{4.5}$$

Furthermore, let

$$\mathcal{E} : \mathcal{V}^* \to \mathbf{S} \tag{4.6}$$

be an evaluation function that maps an output sequence $\mathbf{A} \in \mathcal{V}^*$ to a scalar score $s \in \mathbf{S}$, such as the F_1-score.

By definition of $\mathcal{E}$, there exists

$$\mathbf{P}_{\max} \in \mathcal{V}^* \quad \text{such that} \quad \mathcal{E}(\mathcal{M}(\mathbf{P}_{\max})) = \max_{\mathbf{P} \in \mathcal{V}^*} \mathcal{E}(\mathcal{M}(\mathbf{P})), \tag{4.7}$$

and similarly,

$$\mathbf{P}_{\min} \in \mathcal{V}^* \quad \text{such that} \quad \mathcal{E}(\mathcal{M}(\mathbf{P}_{\min})) = \min_{\mathbf{P} \in \mathcal{V}^*} \mathcal{E}(\mathcal{M}(\mathbf{P})). \tag{4.8}$$

If one selects an element $\mathbf{P} \in \mathcal{V}^*$ uniformly at random, the probability of selecting $\mathbf{P}_{\max}$ is

$$\frac{1}{|\mathcal{V}^*|}, \tag{4.9}$$

with $|\mathcal{V}^*|$ is the cardinality of $\mathcal{V}^*$, i.e. the number of elements that $\mathcal{V}^*$ contains.

The cardinality of $\mathcal{V}^*$ is given by

$$|\mathcal{V}^*| = t^n, \tag{4.10}$$

where t is the number of distinct tokens in the vocabulary (see Equation 4.1) and n is the maximum number of tokens in the input sequence.

This follows since each of the n positions in the input sequence can independently be filled with one of the t tokens, yielding t^n possible sequences.

Qwen3 has a vocabulary size $|\mathcal{V}| = t$ of 151,669 and a native context length $|\mathbf{P}| \leq n$ of 32k or an extended 128k for models greater than 3B [A. Yang, Li, et al. 2025]. This results in the following values in the native case with $n_a = 32k$ and in the extended case with $n_b = 128k$, using Equation 4.10:

$$|\mathcal{V}^*|_a = t^{n_a} = 151,669^{32,000} >> 10^{80} > 4.35 * 10^{17}$$
$$|\mathcal{V}^*|_b = t^{n_b} = 151,669^{128,000} >> 10^{80} > 4.35 * 10^{17}$$

where 10^{80} is the estimated number of atoms in the universe and $4.35 * 10^{17}$ is the estimated age of the universe in seconds. It is therefore practically impossible to test this enormous number of possible prompts.

Even if we consider prompts containing only $n_m = 1,000$ tokens and allow only $t_m = 10,000$ different tokens, which is a more likely scenario, the number of combinations is incredibly large:

$$|\mathcal{V}^*|_m = t_m^{n_m} = 10,000^{1,000} = (10^4)^{1,000} = 10^{4,000}$$

where Equation 4.10 where used. Similarly, the probability of finding this promptly by chance is w.r.t. Equation 4.9 also incredibly small and therefore very unlikely.

4.3.1 Conclusion

It has been shown that it is not possible to test the set of all possible prompts. This also applies to subsets of $\mathcal{V}^*$, such as the set of all prompts that contain natural language, i.e. token combinations that are grammatically correct and make semantic sense to humans. Unfortunately, it cannot be guaranteed that a prompt that does not only contain natural language will have a better score than the best prompts in natural language. However, the intention here was only to show that brute force approaches that test all possible combinations of tokens cannot be used for prompt engineering.

4.4 Summary

The previous analysis examined the data set, model, and formal notation used for prompting, in order to identify possibilities and difficulties at an early stage. The data set provides a considerable challenge for the classification task, as the number of labels is very large. This will significantly reduce the performance of the models and also result in spe-

cial requirements for the metrics (see metric). The analysis of the models has shown that although the Qwen2.5 and Qwen3 generations were released within six months, they probably have major differences in their reasoning capabilities. Since the models are compared in equal sizes, the experiments will show which performance advantage the doubling of the training data (pre-training) and the focus on reasoning responses (post-training) will have. Finally, a mathematical formalism was attempted to ensure the correct notation of prompts and their compositions. This allows the methods in the literature and the approaches developed in the thesis to be noted uniformly. In addition, it was shown that brute force methods are not suitable for prompt optimization, as the number of all possible prompts is incredibly large.

Chapter 5

Methodology

This chapter deals with the problem posed by the thesis and suggests methods for solving it. Approaches are presented that are to be implemented in the experiments. Again, there is a division into single-step and multi-step methods. Hypotheses are formulated that are to be answered for each category. In addition, further methods are suggested that are not implemented in the experiments. Finally, a self-developed multi-step framework (agent) is presented methodically. The metrics that enable the evaluation of the experiments are also discussed.

5.0.1 Problem Definition

This thesis examines the capabilities of language models, focusing exclusively on decoder models, to make clinical predictions. It is a follow-up to previous work on this topic by investigating the capabilities of reasoning in this task. The assumption is that the model's clinical predictions, i.e. diagnoses for patients, will improve if the model performs reasoning.

The MIMIC-IV dataset was chosen to enable comparison with other studies. The dataset contains information on patients admitted to hospital. Each data point covers a patient's hospital stay, and the data has been processed to provide only information available at the time of admission, namely the patient's admission note. The task of the model is to predict the diagnoses on discharge of the patient in the form of ICD-10 codes, which are used, for example, for billing health insurance companies. The actual ICD-10 codes at discharge are also included in the dataset as ground truth and form the labels for each data point. A patient may occur more than once, but each data point represents

an individual hospital stay.

All prompting methods used in this work are in-context learning approaches, i.e. the model is not fine-tuned or otherwise modified in terms of its weights. In addition, the setting was limited to zero-shot (no examples) to reduce the number of variables. Few-shot methods, i.e. examples (few shots) are included in the prompt, are more complex because selecting and combining these few shots leads to a large number of possible combinations. These approaches could be the subject of further work.

There are two ways to improve the performance of a model in such a zero-shot (CoT) setting: First, the hyperparameters of the model (sample parameters) can be changed. Second, the prompt can be changed or completely rewritten. It will be shown that the influence of the hyperparameters is much smaller than the influence of the prompt. Therefore, the thesis will focus on optimizing this prompt. Three approaches will be considered for this purpose: Prompting techniques, prompt engineering, and prompt engineering techniques (i.e. auto prompt engineering):

- **Prompting techniques** here refers to the implementation and evaluation of various approaches from the literature on how a CoT prompt can be designed.
- **Prompt engineering** builds on prompt techniques and aims to further optimize the prompt to increase performance, by adjusting the prompt to the task.
- **Prompt engineering techniques** describes how the process of prompt optimization (prompt engineering) can be automated to make it scalable and reduce the human factor.

The task of the thesis could therefore be characterized as a prompt engineering project: Empirically identify the best possible prompt that maximizes reasoning ability in the given setting (dataset and model). However, it is unlikely that a single best prompt will consistently provide the best results for a broad range of models and across different clinical tasks. In addition, the scientific nature of such an approach, i.e. trying to change the prompt arbitrarily until performance no longer improves, is criticizable. Therefore, this thesis will focus on showing a systematic way to optimize such a CoT prompt. Thereby, two things should be addressed: Reproducibility and transferability.

Reproducibility aims to ensure that the process of prompt optimization can be traced from beginning to end and, in particular, can be repeated for other models, as only six

different models are used and evaluated in this work. However, there are many more models that may be better or worse suited to the particular task.

Transferability should guarantee that the process of prompt optimization can also be applied to other datasets and other tasks in the field of medicine and clinical prediction. There are other settings in which patients have medical conditions and diagnoses need to be determined, such as during a routine visit to the family doctor.

The approaches presented here are also intended to address a difficulty in the field of LLMs. Sam Altman is credited with the idea that LLMs (mainly decoder models) change rapidly over time [Archiv n.d.] (full quote see section A.1). When using such a model to solve a task, i.e. designing a strategy for how the model can help with or solve the specific task, the following should be taken into account: The advanced capabilities of future models. If an approach is tailored to one generation of models, it could become obsolete as soon as a new generation of models is released. However, if the approach works for different model generations and the performance of the approach increases with the capabilities of the model, then the next generation of models will be a benefit for the approach. This means that an approach to prompt optimization is needed that builds on the fundamental capabilities of the models (i.e. is consistent across different models) and in which the success of the approach grows with the capabilities of these models.

5.1 Single-Step Prompting

To answer the first research question, this chapter presents single-step methods and reflects on the methods from related work. The first research question is: Can prompts that induce task-specific reasoning outperform general CoT prompts, i.e. does this approach lead to sophisticated reasoning capabilities and thus better clinical outcome prediction? Three scientific hypotheses can now be derived from the first research question which will be tested in the experiments. The aim should be to disprove the hypotheses, as this allows a general statement to be made: In general, this is not the case, as there are settings where it is not true. Verification, on the other hand, only allows weaker conclusions to be drawn: Within the strict conditions of this experimental setup, the hypothesis is confirmed.

> **H1**: CoT prompting improves the performance of the models tested on the given dataset.

H2: There are zero-shot CoT prompt templates from the literature that show the high performance across different models, i.e. they are robust to model changes.

H3: Manual prompt engineering or meta prompting can adapt the zero-shot CoT prompt to the task and the dataset. This allows the previous tested prompt templates to be outperformed.

5.1.1 Prompting techniques

The prompting techniques refer to the zero-shot CoT methods from the literature that have already been presented. To test hypotheses 1 and 2, four different prompt templates are created, three for the three methods described in the literature: Zero-shot CoT, plan and solve, and thread of thought. For comparison, a zero-group prompt was created, which is called zero-shot and contains only the directive but no trigger sentence.

5.1.2 Prompt engineering and meta prompting

There are two simple methods for optimizing prompts: Manual prompt engineering and meta prompting. Both methods are used to improve the prompt templates. Four prompts are to be created manually and four prompts are to be developed in chat with a decoder. This chat was conducted with GPT-4o in order to create more complex prompt templates. GPT-4o was one of the most powerful models at the time and freely accessible with a free account. A total of twelve prompts and their performance can be tested against each other: Four prompts from the literature, four prompts created manually, and four prompts created with a decoder. However, this aims to answer hypothesis 3.

Meta prompting can also be used to rewrite or rephrase the prompt. Iterative APE methods are suitable for this purpose. One method was tested to create a prompt consisting of 10 questions (see subsection 5.2.2). These questions were created in a previous step by giving the model a task description and an admission note, along with the directive: Write 10 questions that you would answer before making the diagnosis. Another application of meta prompting was presented in ProTeGi (see subsection 3.3.4) and represents the main part of the multi-step section of this thesis.

5.2 Multi-Step Prompting

The attempt to develop a self-made multi-step framework should also investigate the usefulness of such frameworks. Therefore, the second research question could be addressed, which is: Can the process of writing and optimizing a task-specific prompt be automatized, i.e. can a multi-step framework perform task-specific prompt engineering at a human level or better? Again, this research question can be divided into three hypotheses, which will then be evaluated experimentally.

> **H4**: Automatic prompt engineering (APE) can further improve manually crafted prompts or the earlier meta prompts and thus outperform previous methods.
>
> **H5**: The method used, APE with self-criticism, is not a gradient method, as it does not show continuous improvement (increasing performance for every iteration) and should be described as a random method (e.g. Monte Carlo Search).
>
> **H6**: The agent is able to replaces manual prompt engineering completely, i.e. it is robust against the initial prompt. The agent is therefore capable of producing a high-scoring prompt from a low-scoring prompt.

5.2.1 DSPy

Researchers from Stanford presented "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines" in October 2023 [Khattab et al. 2023, p. 1]. Dspy is a multi-step framework that automatically generates and tests prompts from initial directives and few shots. The idea for developing a self-designed multi-step prompt engineering framework appeared while testing Dspy. The first experiments of this thesis indicate that a manual or automated prompt engineering process of the already created prompts may be necessary. There are ready-to-use frameworks such as Dspy or Textgrad, which are available as Python modules and could be used for this purpose. Therefore, Dspy was implemented in an attempt to craft a better prompt.

It quickly became apparent that Dspy was not designed for this type of experiments and research, but rather for custom users. For three reasons, work with Dspy was discontinued: 1. Dspy does not appear to be designed for local models, but rather to work with API keys from cloud providers that make LLMs available. This makes implementation with a local model very complex and vulnerable to errors. 2. Dspy offers little control over the

processes in the framework. Many desirable parameters, such as the number of prompt candidates for an entire run with a finished result, could not be sufficiently determined. 3. Dspy is not efficient to use; it scales very poorly. Even though the local model was set up with vllm and thus benefited greatly from batches and parallelization (good scalability), Dspy only sent single requests to the model. As mentioned above, Dspy does not seem to be designed for local models, so the batch size settings offered were not sufficient to solve this problem.

However, after two weeks, the experiments with Dspy were abandoned because it was assumed that a self-made implementation would be more efficient and effective. Efficient because the code could be adapted to vllm, and effective because the parameters could be fully controlled and thus better suited to the task.

Nevertheless, working with Dspy led to a crucial insight: Dspy used tags or flags to parse parts of the LLM's output. These tags are not mentioned in the Dspy paper or on their website. However, since Dspy was used with a local model, all inputs and outputs sent to the model could be collected. This made it clear how Dspy operates with regard to the model. The tags used by Dspy, or rather the special characters, were adopted in the implementation of the agent (see Table 6.5).

5.2.2 Context creation by task decomposition

There are many ways to add context to a prompt. One possibility is to search for or create additional information and add it to the prompt. Another is to ask the model to do more reasoning or thinking. A third option, which is often similar to the second, is task decomposition. "Some decomposition techniques are similar to thought-inducing techniques, such as CoT", as the explanation of a thought process usually involves intermediate steps that can be considered sub-tasks [Schulhoff et al. 2024, p. 13].

The third method (task decomposition) will only be tested on a small scale. However, a clear weakness of such approaches can be identified a priori: Let's assume that it should be empirically determined how many subtasks in the sense of questions a prompt should contain as additional information or plan and solve instructions. The aim would be to provide an appropriate amount of context. A very simplified approach could look like this. Answer five questions about the patient before making the final diagnoses. 1. What symptoms does the patient have? 2. What is the main diagnosis? 3. What are the secondary diagnoses? 4. Are there comorbidities? 5. What is noticeable in the laboratory

data?

It is possible that some combinations of questions will produce better results than others. To ensure that no combination has been forgotten, all possible combinations would have to be tested. One could use Equation 4.10 to calculate all possible combinations of these questions, but this would also include the combination in which a question is repeated n times. Instead, consider the number of possible permutation $permut(\mathbf{P})$, where $\mathbf{P}$ consists of n discrete elements p_n [Pak 1997]:

$$permut(\mathbf{P}) = n!, \quad \mathbf{P} = \langle p_1, ..., p_n \rangle, \tag{5.1}$$

where $\mathbf{P}, p \in \mathcal{V}^*$ and $permut(P)$ is the number of permutations of all $p \in \mathbf{P}$. For n = 5, the number of permutations would be 120, which is still feasible, but for n = 10, it would already be more than 3 million, which is problematic. Unfortunately, not only must every permutation of the questions be checked, but also the total number of questions. Consider the sum of the permutations of $\mathbf{P}$ if the number of elements p_i is arbitrary, i.e. combinations may consisting of only one question (p_i) or three questions instead of n questions are also possible [Pak 1997]:

$$\sum_{k}^{n} permut_k(\mathbf{P}) = \sum_{k}^{n} \frac{n!}{(n-k)!}, \tag{5.2}$$

where k is the number of elements selected and n the number of elements from which to select. For $k = n$, Equation 5.1 is equal to Equation 5.2. With five questions, the number of combinations would now be 153. As discussed in the few-shot approaches, the order of the prompt parts has an influence on performance. And with a prompt consisting of the five questions, 153 combinations would have to be tested if all possibilities are to be considered.

Unfortunately, no new prompt or new question was generated or created, but only a single generation of questions. As with meta prompting, however, many such question candidates could be generated with a model, not just 10 but 100 or more. Testing these combinations (permutation Equation 5.2) would not be realistic. It is assumed that instead of testing 5 fixed questions (153 permutations), testing 150 meta prompts would lead to better results. This is especially true when computational resources are limited, as is the case with this thesis.

Before the self-criticism agent was built, such an approach was tested in a small multi-

step framework (not fully automated for testing purposes). An admission note was given to the model (Qwen3-32B) and it was asked which questions could be answered before the final diagnosis is made. The number of questions was tested with 5, 10, and 20. There were no clear results as to whether more or fewer questions led to better performance. Therefore, the 10 questions prompt was decomposed and 10 prompts were created from it, containing an ascending number of questions. The experiment (on a small scale) showed that performance decreased after more than two questions. With regard to the permutation problem, this idea was not investigated further and instead, an agent was attempted that creates a completely new prompt with each iteration, without specifying how many questions or instructions this generated prompt contains.

5.2.3 An APE Agent

It is planned to develop a self-made multi-step framework for automatic prompt engineering (APE) with self-criticism. This framework will be referred to as an agent in the following, as it matches the definition of *The Prompt Report* paper for an agent: An agent is a LLM-based system, whereby the model interacts with systems outside the LLM "and more broadly, LLMs which write and record plans" [Schulhoff et al. 2024, p. 24]. More importantly, and in particular, to clearly distinguish this approach from the other pipelines and frameworks in this work. Methodologically, the agent is similar to modules such as ProTeGi or Textgrad, although it was developed independently of these. However, it should not be claimed that this agent is the first of its kind or that any of the methods used here have been newly invented.

APE refers to the prompt being automatically improved by the model itself, i.e. a kind of meta prompting. Self-criticism should be used to guide the direction of prompt improvement. This should not be mistaken for self-criticism prompting techniques from the literature, which mostly criticize the output in order to improve the output. In case of this thesis, the idea is to criticize the input with respect to the output in order to improve the input, i.e. the prompt. This approach appears only once in the *The Prompt Report* paper, namely at ProTeGi [Schulhoff et al. 2024].

In a nutshell, the approach consists of the following three steps: 1. Test an initial prompt on a subset of data. 2. Feed a random data point, its output, its ground truth, and the first prompt into the model and request that the first prompt should be improved. 3. Insert the new prompt together with the old system directive from step 1 into the

model and instruct it to write a new system directive for the new prompt. Formally, the agent is very similar to the ProTeGi process as described in subsection 3.3.4:

$$\mathcal{M}(\mathbf{P_1}) = \mathbf{A_1}, \qquad \mathbf{P_1} = \langle \mathbf{S_1}, \mathbf{D_1}, \mathbf{C} \rangle, \tag{5.3}$$

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{D_1}, \mathbf{C_e}, \mathbf{A_{1e}}, \mathbf{G_e}, \mathbf{E}, \mathbf{D_2} \rangle, \tag{5.4}$$

$$\mathcal{M}(\mathbf{P_3}) = \mathbf{A_3}, \qquad \mathbf{P_3} = \langle \mathbf{A_2}, \mathbf{S_1}, \mathbf{D_3} \rangle, \tag{5.5}$$

where $\mathcal{M}$ represents the model, $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ represent the input, and $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$ represent the corresponding output. $\mathbf{D_1}$, $\mathbf{D_2}$, $\mathbf{D_3}$ is the directive for the corresponding step and $\mathbf{C}$ is the actual patient-content from the dataset, i.e. one admission note. $\mathbf{D_i}$ is the user directive for prompt $\mathbf{P_i}$ and $\mathbf{S_1}$ is the system directive for prompt $\mathbf{P_1}$. Every prompt has a system directive, but only $\mathbf{S_1}$ is to be improved and thus the others are suppressed to keep it simple. $\mathbf{A_1}$ contains the predictions for an admission note, $\mathbf{A_2}$ is the new created user directive (will be $\mathbf{D_1}$ in the next iteration), and $\mathbf{A_3}$ contains the new system directive ($\mathbf{S_1}$ in the next iteration). $\mathbf{P_2}$ additionally includes $\mathbf{C_e}$, $\mathbf{G_e}$, and $\mathbf{E}$, where the index $_e$ refers to the fact that these values belong to exactly one example. $\mathbf{C_e}$ is a randomly selected admission note, $\mathbf{A_{1e}}$ is the prediction for this note, $\mathbf{G_e}$ is the ground truth, and $\mathbf{E}$ is the evaluation of the hole subset of data.

5.3 Metrics

In order to measure the performance of the models quantitatively, a well-thought-out metric must be developed. A common metric for classification problems is accuracy, which is based on the true positive (TP) and true negative (TN) predicted labels.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP (True Positive)	FN (False Negative)
Actual Negative	FP (False Positive)	TN (True Negative)

Table 5.1: Confusion matrix of classification problems showing TP, TN, FP, and FN.

To clarify what TP and TN mean, Table 5.1 shows the conventional confusion matrix,

which is well established and part of general knowledge in ML, therefore no source is given here.

Predicted positive (P) are all ICD-10 codes predicted by the model, and predicted negative (N) are all codes that were not predicted: N = all_labels - P. It becomes clear that the TN value are very large. Accuracy is therefore not suitable for the given multi-classification because the number of labels (classes) is too large at approximately 6k labels (see subsection 4.1.1). The problem becomes evident when examining how accuracy is calculated:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.6}$$

If the model now predicts ten ICD codes, then the number of correctly unassigned labels (TN) is at least TN = all_labels - 10. In this case, the accuracy would be 99.8% if all ten ICD-10 codes are false positives (FP). Therefore, accuracy is not a suitable metric, as a large number of decimal digits would have to be considered and the statement would not be very clear.

However, there are two other metrics, namely precision and recall, which solve this problem. They do not include TN in the calculation and are a good measure of which labels were predicted correctly. Precision also refers to the total number of predicted labels (denominator TP + FP) and recall to the total number of ground truth labels (denominator TP + FN) of the data point (diseases in the admission note).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.7}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.8}$$

Assuming that the ground truth provides 10 ICD codes for an admission note and the model predicts 20 diseases, the precision would be at most 50% if all 10 true diseases are included in the 20 predictions. Recall, on the other hand, would be 100% in this case, as all diseases in the ground truth were also predicted. On the other hand, with 5 predicted diseases and 10 true diseases, precision would be very high, 100% if all predictions are correct, and recall would be lower, i.e. 50% in this case.

This could lead to a problem when evaluating the prompts. A prompt that predicts many diseases has high recall and low precision, while a prompt that predicts few diseases has high precision and low recall. This also applies if both prompts have the same number of correct predictions (TP is equal), only the total number of predictions is different.

Therefore, a harmonic mean of precision and recall is chosen for comparing prompts, namely the F_1 score. In later work, a weighted Fb score could also be chosen if it is determined that precision or recall should be weighted more heavily. However, since this requires medical insights and this work focuses on the data science aspect of the medical task, only the F_1 score was used.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.9}$$

The evaluation for the experiments should therefore be designed to compare the predicted codes with the true codes. This allows the true predictions (TP), false predictions (FP), and false non-predictions (FN) to be calculated. The respective precision and recall of a data point (admission note) is then determined from this. The F_1 score can thus be calculated for this data point. Since a prompt is not only evaluated on a single data point, but on datasets such as the val dataset (8k data points), the mean of all F_1 scores is finally calculated. This allows the prompts to be compared with each other on the same data.

5.4 Summary

The challenges addressed by the thesis were examined in this chapter and methods for solving these challenges were proposed. Three concrete hypotheses were also formulated for each of the two categories, single-step and multi-step methods, and approaches were presented for experimentally testing these hypotheses. The hypotheses are designed to assert the suitability of the literature approaches to the given problem (dataset and task). Based on this, the experimental setups are described methodically. Finally, different metrics were discussed and three metrics were selected, precision recall and F_1, which should meet the requirements of the dataset.

In addition, two approaches are discussed that are not part of the experiments (DSPy and task decomposition). Nevertheless, these approaches were implemented on a small scale and quickly rejected. This trial and error process is implicitly part of the search for approaches that culminate in the self-developed agent. The structure of the agent

was then methodically outlined and compared with the approaches in the literature. This chapter is closely related to the related work chapter, as the method from the literature has already been presented there.

Chapter 6

Implementation

Three pipelines were implemented in python to perform the planned experiments. These pipelines are explained in detail in the following chapter. No python code is shown, as the agent consists of thousands of lines of code and a discussion of this code would go beyond the limits of this thesis. However, the important prompts will be shown in detail, as they are crucial for reproducibility. Furthermore, this chapter will deal with implementation difficulties, e.g. parsing outputs, applying metrics, and tracking the parameters used.

6.1 Single-Step Prompting

6.1.1 Prompting techniques

Creating prompt templates is straightforward when following the zero-shot CoT approach described in the literature. A user directive, which may contain a CoT trigger sentence, is used, leaving a gap for the corresponding data point. Additionally, a system directive is provided to better prepare the model for a specific task.

Role	**Content**
System	You are a helpful medical assistant.
User	Here is a patient's admission note. Please provide the most likely diagnoses in the form of ICD-10 codes: {}

Table 6.1: Simplest prompt template named *zero-shot*. The admission note will be inserted in {}.

In the simplest case, where the trigger sentences are taken from the literature, the

prompt template consists of a simple system and user directive, whereby the user directive is appended to the trigger sentence. All four entirely zero-shot CoT prompt templates can be found in the appendix (see subsection A.2.1), where they are referred to as *four_diff_cot*.

Method	**Trigger Sentences**
Zero-shot CoT	Let's think step by step.
Plan-and-Solve	Let's first understand the problem and devise a plan to solve the problem. Then, let's carry out the plan and solve the problem step by step.
Thread-of-Thought	Walk me through this context in manageable parts step by step, summarizing and analyzing as we go.

Table 6.2: Zero-shot CoT approaches from the literature and their trigger phrases.

6.1.2 Prompt engineering and meta prompting

In the case of manual prompt engineering and meta prompting, the system directive and user directive are expanded into a plan to be followed. Steps are specified, which can be sub-tasks, for example, that are then to be executed sequentially by the model. This could be called a kind of guided chain of thought or task decomposition with fixed sub-tasks.

To illustrate this, Table 6.3 shows the system directives for the four manually built prompts and one prompt template designed with meta prompting. The complete prompt templates can be found in the appendix (see subsection A.2.2), referred to as *four_short*.

For the manually crafted prompts, an attempt was made to gradually increase the complexity of the prompts. Table Table 6.3 shows how the system part becomes longer and more detailed (full best-meta prompt in appendix subsection A.2.3). The same applies to the user directive of these manual prompts. The complexity of the prompts produced by meta prompting was also supposed to increase, but this was not entirely achieved. The GPT-4o model used for this did not always respond as desired and introduced or omitted aspects that were not directly requested. However, meta prompting is still considered a recommended method because it reduces the human factor. Even though the prompts generated by meta prompting are difficult to reproduce exactly, meta prompting can be used by anyone and does not require domain-specific knowledge.

Finally, it should be mentioned that meta prompting, i.e. in this case asking GPT-4o

Prompt template	System directive
Zero-shot prompt	You are a helpful medical assistant.
Manual prompt 1	You are an experienced clinical assistant in the emergency department.
Manual prompt 2	You are an experienced clinical assistant in the emergency department with expert knowledge of ICD-10 coding.
Manual prompt 3	You are a senior emergency medicine clinician and expert in clinical coding using ICD-10.
Best-meta prompt	You are a clinically trained AI medical coder with extensive knowledge of ICD-10 codes, clinical terminology, and diagnostic criteria. Your task is to carefully read and analyze a patient's hospital note (e.g. discharge summary, admission note, or clinical progress note) and accurately extract all relevant final diagnoses in the form of:- ICD-10 codes - Corresponding diagnosis names You must follow the instructions and rules below with high precision.

Table 6.3: Four manual crafted prompt templates and one crafted by meta prompting with their system instruction.

to write a sophisticated prompt, is also a challenge. It is not enough to give GPT-4o the zero-shot prompt and request that it should improve it. Many attempts have been made to produce high-performing prompts with GPT-4o. In the process, a concept of prompts was found that is rarely discussed in research, the so-called "God prompts" [Rajbahadur et al. 2024, p. 3]. However, there are a lot of discussions about these God prompts in many coding blogs and other forums. GPT-4o also seems to be familiar with this term and was thus able to create the best-meta prompt, one of the best prompts from this thesis. The prompt that was passed to GPT for this purpose and produced the best meta prompt can be found in the appendix (see subsection A.2.3). The concept of god prompts was not explored further due not being scientifically grounded, but it seems to lead in the direction of a table of thought, whereby the input of the model, rather than the output, consists of a kind of markdown table. However, further work could explore this concept in detail.

6.1.3 Pipeline Data

For a single data point, i.e. in this case for a patient, it is very easy to apply the prompt techniques from the literature. Combine the prompt template and the patient admission note and enter the resulting prompt into any decoder model. For the dataset used in this thesis, with over 8k admission notes, manual prompting would not be realistic and would be very time-consuming (hardly scalable). Therefore, a pipeline was built for the experiments in this thesis, called the pipeline data to distinguish it from the other pipelines. The pipeline receives a dataset with admission notes and a dataset with prompt templates. In most cases, the largest dataset, *val.parquet*, containing 8k admission notes was used, together with a json file containing four different prompt templates to be tested against each other. The pipeline returns a dataset that contains both the outputs of the model and the evaluation of these outputs. Therefore, the pipeline can be divided into three stages for clarity: Data preprocessing, model prediction generation, and model output evaluation.

Data preprocessing

First, the data (dataset and prompts) is loaded. Then, prompt templates and data points are merged. A chat template is applied to the 8k prompts and stored as a list of strings, with each string containing a prompt. Chat template refers to a method of the Hugging Face tokenizer, which adds model-specific tokens to communicate roles such as *system* or *user* to the model (see Table 6.1). In addition, all parameters selected by the user are retrieved via arg parser and loaded into a data class to make them immutable.

Model prediction generation

First, a local model is set up. The list of prompts is then loaded and fed into the model. The model generates an output for each data point, with vllm optimizing this process as much as possible (vllm scales very well). The outputs are stored in the dataset, which thus contains the admission notes, the input prompts for each admission note, and the outputs for each input prompt. This facilitates the evaluation and traceability of the experimental results.

Model output evaluation

The final step of the pipeline is a type of data post-processing. First, the ICD codes are extracted (parsed) from the model output. This is done using a regex pattern that matches ICD-10 codes. Then, these predicted ICD-10 codes are compared with the codes from the ground truth (the label of each data point). This allows the calculation of the metrics precision and recall. The F_1 score is then calculated from these two. In addition, a simple metric based on the F_1 score is introduced to facilitate the interpretation of the quantitative results. The metric $F_1 = 0$ indicates that the F_1 score for this data point is 0. This means that no single disease was detected, neither primary nor secondary diagnosis. In addition, further properties were measured for each prompt on a dataset: The length of the prompt and the output (in digits), the time of the prompt on the dataset, and the average number of codes generated. The simple evaluation allows the prompt to be ranked immediately after the run, but is only intended to provide an overview. A more detailed evaluation can be carried out manually at a later stage. To ensure the reproducibility of the experiment, metadata is stored. The metadata (stored as json) contains all arguments of the run, such as model and temperature, as well as details about the data, such as how many prompts and the size of the dataset. To monitor the progress of the experiment, a logger was built that indicates when each step of the pipeline is reached and how long it takes to complete. The automated documentation of agent runs is described in detail in section 6.5.

6.2 Multi-Step Prompting

6.2.1 Prompt Templates

As with the single-step methods, the prompts in multi-step methods consist of prompt templates that are to be improved by the agent. As an example, consider the prompt template of the zero-shot prompt (see subsection 6.1.1). It becomes clear what the goal of the automatic prompt engineering agent is: It should interactively rewrite the system part and the user part of the prompt template. After each rewrite, the prompt is tested on a sub-dataset. It is assumed that this will result in prompts that have a higher score than the initial prompt. Please note that this chapter and the following chapters deal exclusively with prompt templates, which are abbreviated as prompts. A prompt therefore no longer

refers to a single input (template + data point), but always to the template, which can then be combined with a dataset of, for example, 100 data points to form exactly 100 prompts. However, it is easier to read and clearer to refer only to prompts, as long as it is clarified that these are prompt templates.

6.2.2 APE Agent

The agent consists mainly of three parts, which are executed once sequentially in each iteration. The parts themselves consist of pipelines, whereby the first pipeline has already been described in subsection 6.1.3:

1. Pipeline Data
2. Pipeline Task 1
3. Pipeline Task 2

Pipeline Task 1

The objective of pipeline task 1 is to rewrite the user prompt or, more precisely, the user directive. For this purpose, the model is presented with a data point together with a brief evaluation. The task of the model is to critique the prediction of the data point and, based on this critique, rewrite the prompt (user directive). To ensure this, in addition to the information mentioned, a system and user directive must also be formulated for this step. Before the complete user directive can be presented, the structure of the prompt for pipeline task 1 will be reviewed (see subsection 5.2.3) and the individual components will be described in detail. The structure is as follows:

$$\mathcal{M}(\mathbf{P_2}) = \mathbf{A_2}, \qquad \mathbf{P_2} = \langle \mathbf{S_2}, \mathbf{D_1}, \mathbf{C_e}, \mathbf{A_{1e}}, \mathbf{G_e}, \mathbf{E}, \mathbf{D_2} \rangle,$$

with system directive $\mathbf{S_2}$ and user directives $\mathbf{D_1}$ and $\mathbf{D_2}$. $\mathbf{C_e}$ is simply one random admission note, $\mathbf{A_{1e}}$ is the predicted output and $\mathbf{G_e}$ is the ground truth of that admission note as given in the dataset. In contrast $\mathbf{E}$ is the mean evaluation of the entire subset of data.

First, let's consider the directives of task 1 prompt $\mathbf{S_2}$ and $\mathbf{D_2}$. Both can be specified when using the agent, i.e. they are taken from a prompt dataset named *one_task.json*.

Role	Content
System ($\mathbf{S_2}$)	You are a helpful prompt engineer.
User ($\mathbf{D_2}$)	Please examine the medical prediction you made. Look at your diagnoses and compare them with the ground truth. What stands out to you, and which diagnoses did you not see and why? Try to understand which parts of the admission note belong to the ground truth diagnoses. Go through this analysis step by step. After the analysis, rewrite the user prompt in such a way that you expect better results for the next prediction. Keep in mind that you should not include any patient-specific information in the prompt. The prompt is designed to apply to many different admission notes and patients. Keep the user prompt general.
User (add to $\mathbf{D_2}$)	Think through this step by step and then give a final answer. Write your final answer between two tags starting with {*start_tag*} then your final answer and ending with {*end_tag*}.

Table 6.4: Prompt template for task 1 with its directives $\mathbf{S_2}$ and $\mathbf{D_2}$. Additionally a sentence is added to $\mathbf{D_2}$ to make the output parseable.

However, $\mathbf{D_2}$ is then extended with the following sentence (see Table 6.4) to make the important part of the output parseable, i.e. the new prompt. This sentence is hard-coded in the agent and cannot be changed. But the tags, {*start_tag*} and {*end_tag*} can be specified in *one_task.json* when using the agent. It is possible that other tags also work well or even better, but in random checks, no errors were found with these tags, which is why they were not investigated further. The default for these tags is shown in Table 6.5.

Tag name	Tag value
start_tag	[[## new_prompt ##]]
end_tag	[[## completed ##]]

Table 6.5: Default parsing tags of pipeline task 1. The words are arbitrary, but the characters seem to be adopted very well by the model.

Secondly, evaluation $\mathbf{E}$ should be considered, which provides the model with information about the test of the prompt across the entire subset of data. $\mathbf{E}$ is a string that is included in the specified position in prompt $\mathbf{P_2}$. $\mathbf{E}$ is composed of the following sub-strings

shown in Table 6.6, whereby a description and the corresponding value are provided in each sub-string.

Description	**Value**
Precision:	$\{mean\_precision\}$
Recall:	$\{mean\_recall\}$
F_1:	$\{mean\_f1\}$
Prompt length:	$\{prompt\_len\}$
Output length:	$\{mean\_output\_len\}$
Mean number codes predicted:	$\{mean\_codes\_pred\}$

Table 6.6: Sub-strings that are combined in the specified order to form E. $\{value\}$ is a placeholder in which the calculated values from the evaluation are inserted.

The F_1 score is the main metric that is considered in quantitative evaluations, i.e. the best prompt is always the one with the highest F_1 score. However, the model is also given other metrics such as mean precision or the average number of predicted codes to provide the model indications of the weaknesses in the current prompt. Precision and number of predicted codes are highly correlated, which will be discussed in section 6.3.

Overall, the idea behind the prompt is mainly to improve, or rewrite, the user directive $\mathbf{D_1}$ from the first prompt $\mathbf{P_1}$. It is assumed that this improvement is more successful if the model receives the additional information mentioned above as a basis for generating a chain of thought before the new prompt is generated, i.e. $\mathbf{D_1}$ is rewritten.

The prompt $\mathbf{P_2}$ composed in this way is then fed into the decoder model and the raw output is stored. The final step in the pipeline is a data post-process, which consists of further processing the raw output of the model. The tags mentioned are used to extract the substring between the tags using a simple rex pattern. This sub-string will serve as the new $\mathbf{D_1}$ in the next iteration. To do this, another sentence must be added that contains the placeholder for the admission note. This sentence states simply: “Here is the patient’s admission note: {}”. After that, the string can be stored as the new D1 and will be loaded in the next iteration, as the initial prompt was before.

Pipeline Task 2

The idea behind pipeline task 2 is to improve or rewrite the system directive $\mathbf{S_1}$ of the actual medical prompt $\mathbf{P_1}$. The prompt $\mathbf{P_3}$ required for this has already been briefly

described in subsection 5.2.3 and will be explained in detail here. As with the previous pipeline, prompt $\mathbf{P_3}$ consists of several parts that are combined in the following order:

$$\mathcal{M}(\mathbf{P_3}) = \mathbf{A_3}, \qquad \mathbf{P_3} = \langle \mathbf{S_3}, \mathbf{A_2}, \mathbf{S_1}, \mathbf{D_3} \rangle, \tag{6.1}$$

where $\mathbf{S_3}$ and $\mathbf{D_3}$ are the system and user directives of prompt $\mathbf{P_3}$ and thus contain the instructions for this step. $\mathbf{A_2}$ represents the output of the second step and will serve as $\mathbf{D_1}$ in the next iteration. $\mathbf{A_3}$ is then the output of this step, that will serve as the new $\mathbf{S_1}$ (in the next iteration). Please note that $\mathbf{A_2}$ not only contains the new prompt ($\mathbf{D_1}$), but also includes a chain of thought, among other things. The new prompt ($\mathbf{D_1}$) is parsed via rex pattern and then inserted into prompt $\mathbf{P_3}$. This step therefore involves $\mathbf{A_2'} \subset \mathbf{A_2}$, which has been omitted from the notation for a clear overview.

Considering the system and user directives of $\mathbf{P_3}$, it becomes clear what the task of the pipeline is and how it should be executed.

Role	**Content**
System ($\mathbf{S_3}$)	You are a helpful prompt engineer.
User ($\mathbf{D_3}$)	You have just designed a new user prompt to fulfill the given medical diagnosis task. Here is the new user prompt: {*user_prompt*} Now write a suitable system prompt for your new user prompt. Here is the old system prompt as an example: {*old_system_prompt*} Please rewrite this system prompt without repeating any information from the user prompt. Note that you should not include any patient-specific information. The system prompt should be short, concise, and general.
User (add to $\mathbf{D_3}$)	Think through this step by step and then give a final answer. Write your final answer between two tags starting with {*start_tag*} then your final answer and ending with {*end_tag*}.

Table 6.7: Prompt template for task 2 with its directives $\mathbf{S_3}$ and $\mathbf{D_3}$. In addition, the variables $\mathbf{A_2'}$ ({*user_prompt*}) and $\mathbf{S_1}$ ({*old_system_prompt*}) are inserted.

As before in pipeline task 1, directive $\mathbf{D_3}$ is extended by a sentence that is intended to enable the parsing of the new prompt (new system directive). $\mathbf{D_3}$ and $\mathbf{S_3}$ can again be

specified before the agent is used, but the appended parsing sentence is hard-coded. The tags, on the other hand, can be freely assigned, whereby the characters "[[##" work well and are therefore recommended. The default tags are shown in Table 6.8.

Tag name	**Tag value**
start_tag	[[## new_system_prompt ##]]
end_tag	[[## completed ##]]

Table 6.8: Default parsing tags of pipeline task 2. The words are arbitrary, but the characters seem to be adopted very well by the model.

Comparing string $\mathbf{D_3}$ (see Table 6.7) with Equation 6.1, it becomes evident that $\mathbf{A'_2}$ and $\mathbf{S_1}$ do not appear before or after the directive, but are inserted in $\mathbf{D_3}$. However, this is more complicated to formalize and has been omitted, as the intention of the notation is conciseness rather than preciseness.

The last step is to extract the new prompt from $\mathbf{A_3}$, in this case the system directive $\mathbf{S_1}$. The tags are used to parse the new prompt ($\mathbf{S_1}$), similar to the previous pipeline. Finally, the new system directive can be saved and will be loaded in the next iteration as before the initial prompt, whereby pipeline task 1 has rewritten $\mathbf{D_1}$ and pipeline task 2 has reformulated $\mathbf{S_1}$.

6.3 Metrics

The metrics from section 5.3 need to be implemented and integrated into the pipelines. One aspect that should be highlighted here is the extraction of the predicted codes from the model's output. The authors of the Clinibench paper generated a JSON file with diagnoses in text form and mapped these text diagnoses to ICD-10 codes, e.g. via embeddings [Grundmann et al. 2025]. The approach in this thesis is simplified, as newer models with greater capabilities were used, namely Qwen2.5 and Qwen3. The model is instructed to provide a list of ICD-10 codes at the end of the output, which comprise the diseases. This procedure facilitates implementation and could be easily reproduced for future work with appropriately powerful models. Comparing the scores from ChliniBench and this thesis, it can be assumed that the loss of performance due to mapping to the ICD code from the model itself is only minor.

A regex pattern (regular expression) is used to extract the ICD-10 codes from the

model's output. A simple regex pattern written manually was unable to filter all variations of ICD-10 codes, which led to problems in early experiments. However, it turned out that GPT-4o (via ChatGPT) is capable of writing very sophisticated regex patterns. This pattern searches e.g. for matching substrings that begin with one or more letters and contain a point or no point:

```
pattern = "\b [A-Z][0-9]{2}(?:\.[A-Z0-9]{1,4}|[A-Z0-9]{1,4})?\b "
```

A test on a few ICD-10 codes (synthetic data) did not reveal any errors, which is why this pattern was not further tested. Nevertheless, larger systematic test data could be created if such a procedure is used in published research papers. However, both the outputs of the model (predicted_output) and the parsed ICD-10 codes (predicted_codes) are included in the data from the experiments, which allows to check whether all ICD-10 codes were extracted correctly.

After ensuring that the ICD-10 codes are in the same format, i.e. the point has been removed, the calculation of the metrics is straightforward. The predicted codes, a list of strings, are converted into a set, just like the ground truth. For short codes, the predicted codes are limited to the first 4 characters. This allows to calculate the intersection of the sets, which is directly the TP.

$$\text{Precision} = \frac{TP}{TP + FP}, \quad TP + FP = |\text{predicted\_codes}| \tag{6.2}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \quad TP + FN = |\text{ground\_truth}| \tag{6.3}$$

The denominators of precision and recall (see Equation 6.3) are simply the number of predicted codes (precision) and the number of ground truth codes (recall). From these, the F_1 score can finally be calculated, as shown in Equation 5.9. This F_1 score is then calculated for each data point, whereby the mean of these individual F_1 scores is used for a prompt on a dataset. Please note that precision refers to the number of generated codes, while recall considers the number of ground truth codes. This difference was discussed in section 5.3 and results for prompts with many predicted_codes to have low precision but high recall, and vice versa for prompts with few predicted_codes.

6.4 Model hyperparameter

As with most ML algorithms, language models also have hyperparameters, which are usually referred to as sample parameters. One example of this is the temperature. This controls the probability distribution of the next token generated, i.e. a low temperature corresponds to a sharp distribution and a high temperature to a flatter distribution. At a high temperature, tokens are selected with a higher variance, which leads to more diversity in the generated text (for multiple runs and the same settings). A temperature of zero (greedy decoding) would make the output exactly reproducible, while a temperature greater than zero can cause the output to vary from run to run. However, the developers of the models usually recommend a temperature setting greater than zero, especially when generating chains of thought.

In order to select a uniform and comparable setting for different models, the same sample parameters were used in all experiments. Early experiments showed that these sample parameters have a much smaller impact on the performance of prompts than the prompts themselves. In addition, robustness with respect to the parameters was observed in terms of the difference in scores between the prompts. In other words, high-performing prompts always had higher scores than low-performing prompts, regardless of whether they had 1-3% lower absolute F_1 scores due to temperature or not.

Description	**Value**
temperature	0.6
max_tokens	10,000
top_p	0.95
top_k	20
enable_thinking	true
continue_final_message	false
add_generation_prompt	true

Table 6.9: All model sample parameters used in the experiments. These parameters are the same for all experiments in the evaluation of this thesis.

The sample parameters listed in Table 6.9 were applied to all experiments to ensure a uniform and comparable basis for different models. For example, the ideal temperature for each model could be found experimentally, and an ideal sample parameter could be

set for each model (generation and size). However, there is no guarantee that different prompts with varying parameters will show different performance, i.e. in the worst case the order of the prompts scores could change as a result. Once a prompt and a model have been determined, e.g. for a medical AI application, more suitable sample parameters could be found using grid search methods. However, this adjustment requires many additional experiments and could significantly increase computational costs. Therefore, it was not performed in this thesis.

Finally, it should be mentioned that the parameters were not treated equally during implementation. Temperature and max_tokens, which are the maximum number of tokens in the response, were specified as variables in the script and can thus be selected by the user when running the agent or only the data pipeline. For top_k and top_n, the default is always set and cannot be selected by the user, but this can be easily changed. Both parameters also control the token selection from the probability distribution. Since these are set to the model publishers default, they will not be discussed further here.

The parameters enable_thinking, continue_final_message, and add_generation_prompt are not technically model sample parameters but belong to the prompt. They are used in the prompt chat template and thus change the string that the chat template outputs. This works as already demonstrated with enable_thinking: Tokens (words in the string) are inserted or omitted to instruct the model what to do (see subsection 4.2.1). However, the parameters were kept the same for all experiments, so their influence does not need to be discussed in detail here. A detailed discussion of how a model can be set up locally or on the BHT cluster would have to be more extensive and would exceed the limits of this work. Like other computer science skills, this is assumed to be known by the author of this thesis.

6.5 Documentation of the experiments

A series of experiments were planned and later run on the BHT cluster. As mentioned in subsection 5.0.1, these experiments should be reproducible and transferable. In addition, cluster resources are limited, and care was needed to reduce computational costs, e.g. by not repeating experiments unnecessarily because configurations were not documented. To meet these requirements, a comprehensive logging and reporting system was introduced. It should help to monitor and document the experiments.

Folder or File	Description
`2025-09-22-02-27-49/`	Root folder of the run named by its date
`data/`	Contains the dataset with the experimental results
`logs/`	Contains the logs of the run and a meta data json
`prompts/`	Contains all prompts used or produced by the run
`report.txt`	Summary report of the evaluation of experimental run

Table 6.10: Experiment folder automatically created by the agent for every run.

Table 6.10 shows the structure of the folders that are created for each run of the agent. First, the date and exact the time of the agent run are used as the root folder. This root folder then contains the `data`, `logs`, and `prompts` folders. The `data` folder contains the results of the experiments, `logs` contains the logs of the experiments, and `prompts` contains the prompts used for that run. In addition, a `report.txt` file can be found in the root folder, which shows a brief evaluation of the experiment. The `logs` folder contains a log file showing the printouts during the run. It also contains a `meta.json` file showing all parameters for this run in JSON format. Please consider the files report.txt, pipeline.log, and meta.json in the appendix as examples (see section A.3). In addition, there is a `time.json` file showing the duration time for each iteration. The same applies to the prompt folder, which of course stores the prompt dataset used or simply the initial prompt. In addition, the new prompt is appended to this file for each iteration. The prompts also contain the instructions for pipeline task 1 and pipeline task 2 in separate JSON files.

This detailed documentation is intended to ensure that all aspects of the agent run are saved, for example, to repeat the run or to track which parameters were used. With such detailed documentation, some values may appear twice and thus be redundant. However, this is only intended to show an attempt at documentation and could be further refined in sophisticated agents.

6.6 Summary

To carry out the planned experiments, a total of three pipelines were implemented in python. The first pipeline, pipeline data, is used to perform all single-step experiments. For the second part, the multi-step methods, an agent was built that also contains the

pipeline data and two additional pipelines that rewrite the current prompt. The implementation does not show any python code, but focuses on the approaches and prompts used. This is to enable a comparable implementation to be reproduced, whereby the exact code is arbitrary. In contrast, the prompts used must be exactly the same in order to reproduce the results. A crucial aspect in the implementation of such a multi-step framework was the parsing of certain parts of the output, such as the new prompt. For this purpose, the exact tags used, sometimes also referred to as flags, were given. This method is rarely discussed in the literature, but seems to occur in almost all multi-step frameworks.

Furthermore, the metrics used were discussed, as well as the parsing of the ICD-10 codes. The hyperparameters used in the models (sample parameters) that are necessary to reproduce the experiments were also described. Finally, the documentation of the experiments was discussed, including the folder structure created by each agent run. It has been shown that detailed documentation is useful for evaluating the experiments later and for finding possible errors in the experiments or producing experiments with slightly modified parameters.

Chapter 7

Evaluation and Discussion

This chapter deals with the evaluation of the experiments described so far. In addition, findings are discussed and interpreted, i.e. possible reasons are given, which of course are speculative. To further discuss the hypotheses from section 5.1, corresponding experimental data are presented. However, only some of the experiments can be presented here, as approximately 2k prompts were tested in different settings. Presenting all experiments from the last five months would exceed the scope of this thesis. Nevertheless, the omitted parts can be seen as helpful guidance for the results presented here.

7.1 Single-Step Prompting

7.1.1 Zero-shot CoT prompt techniques

The three prompt techniques mentioned in subsection 6.1.1 for achieving CoT in a zero-shot setting can now be compared using the six qwen models. Since the performance of the models increases significantly with their size, the models were sorted by size rather than by generation. The F_1 score was used to measure the performance of each prompt template across the models. Each point in the plot shows the mean F_1 score of a prompt template on a model across the rounded 8k admission notes of the val dataset. Thus, each model is fed with 32k prompts, which adds up to a total of approximately 200k prompts for all six models. It is assumed that this leads to the measured data better approximating the actual distributions and scores of the prompt templates.

Figure 7.1 shows that the performance increases significantly with the generation and size of the model. It could be concluded that prompts can only be compared with reference

to their exact model (in terms of size and generation). Furthermore, no clear order of the prompt templates used is observable. Since the lines intersect, no prompt is consistently better or worse than the other prompts across the models. If we also take into account uncertainty, e.g. due to temperature, no clear difference between the literature approaches is identifiable. Furthermore, the zero-shot prompt is almost as good as the zero-shot CoT prompt templates with trigger sentences. This clearly rejects **H2** (see section 5.1).

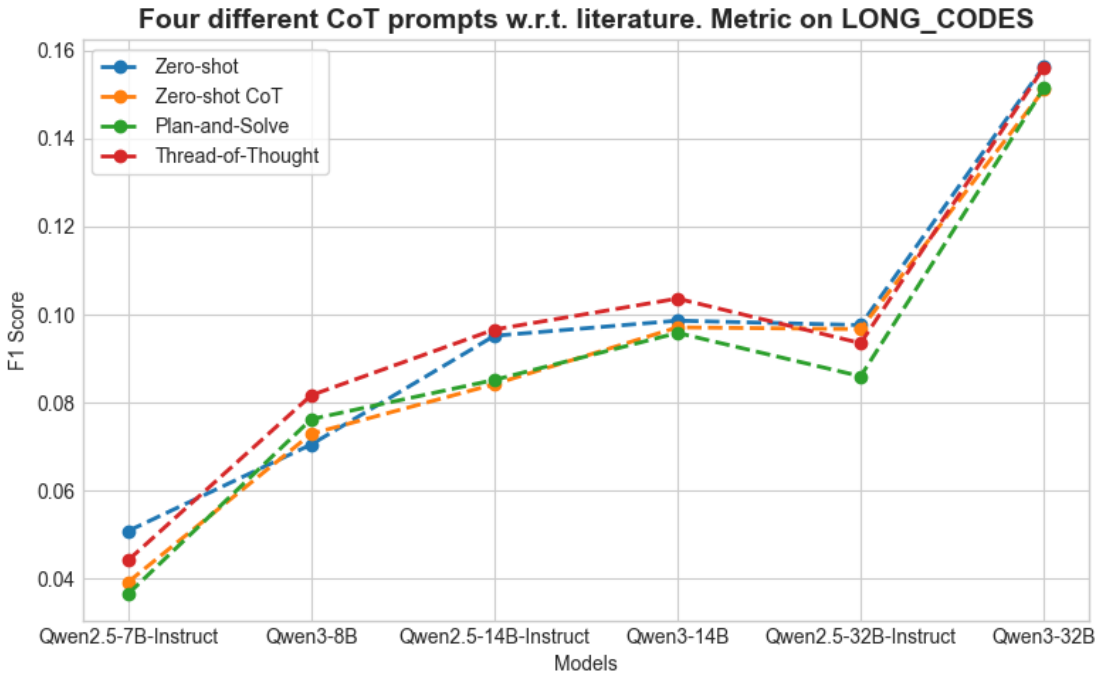

Figure 7.1: Four different prompt templates with approaches from the literature on 8k val dataset. Zero-shot contains no trigger sentence. The plot shows the differences between the models and that no prompt performs consistently, even in comparison to the other prompts.

In contrast, **H1** will be examined in more detail in the following, as no clear quantitative difference has yet been found between prompts with and without CoT. However, distinguishing between CoT and no CoT is problematic with the selected models, as all selected models were trained to generate chains of thought in post-training (see Table 4.1).

7.1.2 Prompt engineering and meta prompting

Prompt engineering

Since no clear differences between the zero-shot CoT prompt techniques from the literature could be observed, three additional prompts were written manually (see subsection 6.1.2) and compared with the standard zero-shot prompt. In this comparison in Figure 7.2, the lines also overlap. This means that even the manually written prompts, which contain significantly more instructions, i.e. longer directives, do not perform measurably better than the *zero-shot* prompt. In fact, the manual_p_1 prompt performed worse than the zero-shot prompt over all models, which was of course not the intention. This demonstrates the difficulty that more sophisticated prompts can be worse than simple ones and that the performance of a prompt is hardly estimable before testing it on a dataset.

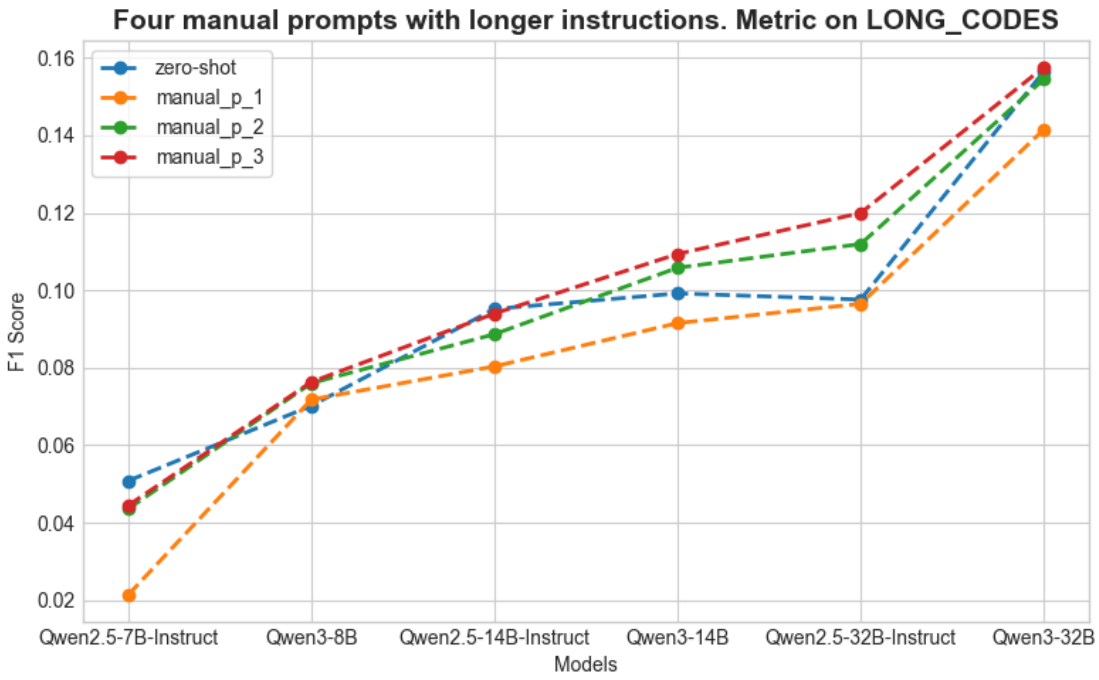

Figure 7.2: Three different prompt templates written manually and the zero-shot prompt for comparison on 8k val dataset. Even manually written prompts do not perform consistently across models. And the order of prompts with respect to their score does not remain the same.

A comparison of the 7B and 32B models from the Qwen2.5-Instruct generation indicates that the same prompt templates perform differently on different models, even on models of the same generation, i.e. they were trained in the same way. In this series

of experiments, it is still difficult to make statements about the capabilities of prompt approaches, and more variations will be needed to formulate clear observations.

Meta prompting

In order to achieve increased diversity in the tested prompt templates and to reduce the human factor, prompts were created using GPT-4o. GPT-4o was instructed to make the prompts longer, more specific, and clearly structured. This resulted in a prompt length of more than 2k digits, which is 10 times longer than the standard zero-shot prompt. Finally, a clear difference in performance could be achieved (see Figure 7.3). The three

Figure 7.3: Four different meta prompts and the zero-shot prompt for comparison on 8k val dataset. Unlike previous prompts, meta prompts seem to perform consistently across models. The best-meta prompt clearly stands out.

meta_p_i prompts are similar in structure to the *best-meta* prompt, but show no clear improvement over the *zero-shot* prompt. The *best-meta* prompt, on the other hand, consistently performs better than all previously tested prompts, which partially supports **H3**. In contrast, other meta prompts were unable to demonstrate this, which is why **H3** is only partially supported.

The best-meta prompt could be further examined for its characteristics. But, on the

one hand, it is difficult, to clearly establish a correlation between structure and wording to the performance. On the other hand, the goal is not to describe particularly good prompts found by chance, but to demonstrate the process of finding such prompts.

7.1.3 All CoT prompts single step

So far, 11 different prompts have been tested on a large dataset of 8k admission notes, with the zero-shot prompt occurring twice (12 runs in total). This double testing was done intentionally to get an idea of the variance in terms of temperature. Surprisingly, this temperature-related variance is very low in the models used. A greater variance (1-2%) was observed on smaller subsets of the large dataset, but this seems to be neutralized by the average over 8k data.

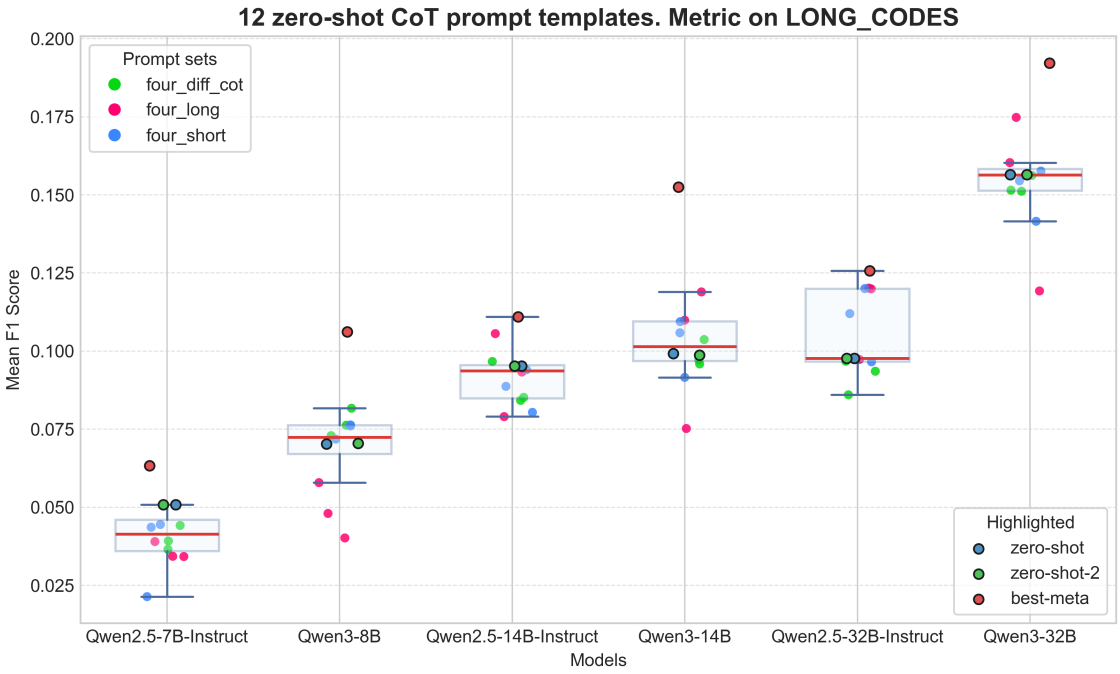

Figure 7.4: Three prompt sets, each containing four prompts, on the 8k val dataset. The zero-shot prompt and the best-meta prompt are highlighted. Zero-shot 2 is the same prompt as zero-shot and shows the temperature-related variance. The box plot shows a clear trend across generations and model sizes. The best-meta prompt and the Qwen3-32B model are significantly superior.

Figure 7.4 shows three different prompt sets that were examined in more detail in the previous sections. The $four\_diff\_cot$ set refers to the literature approaches from Table 6.2, the $four\_short$ set contains the manually created prompts from Table 6.3, and

the *four_long* set contains the meta prompts (see subsection 6.1.2). The orange line, known as the median, shows that the Qwen3 models consistently perform better than the previous generation, which was to be expected.

As mentioned, the meta prompts are very different from the other sets in terms of their length. It could be assumed that the length of the prompts improves reasoning, i.e. the chains of thought. However, if we compare two models of the same generation, e.g. 8B and 32B of the Qwen3 family, this idea is immediately rejected. The length of the prompt alone does not seem to offer any clear advantages. Only the best-meta prompt consistently performs better than all others, while the remaining meta prompts are sometimes worse than the short manual prompts (*four_short*), especially on small models like Qwen3-8B. While the best-meta clearly stands out, there is also a recognizable meta prompt that performs worst in almost every model. A qualitative analysis of both prompts could be carried out. However, this is not only difficult, as already mentioned, but also problematic, since with 2k long prompts, many aspects, i.e. words and phrases, could make the difference. If one wanted to conduct such an analysis reliably, many more prompts would have to be produced in the respective scheme in order to make clear observations and verify influences. This is avoided, and instead an analysis of the experiments already executed is intended.

The experimental data contains a large amount of information that cannot all be discussed in this thesis. However, some interesting aspects that may influence the capabilities of reasoning will be highlighted. An important factor is the length of the output, as this indicates a long or short chain of thought.

Figure 7.5 shows the output length of the different models for the respective prompt sets. The first clear observation is that the Qwen3 models generate significantly longer outputs than their predecessors. This occurs regardless of the prompts, which limits the influence of the prompts on the output length. Furthermore, two different perspectives can be analyzed: First, the influence of the prompt on the output length. Second, the influence of the output length on performance. Intuitively, one might expect that a longer prompt would also lead to a longer chain of thought in the output. The experimental data demonstrate the opposite: For all models, especially the Qwen3 generation, the output for the long meta prompts (four_long) is shorter than for the other prompt sets. The best-meta prompt in particular seems to lead to very short responses. The shortest prompts from four_diff_cot are in the middle, so they have a medium output length in comparison.

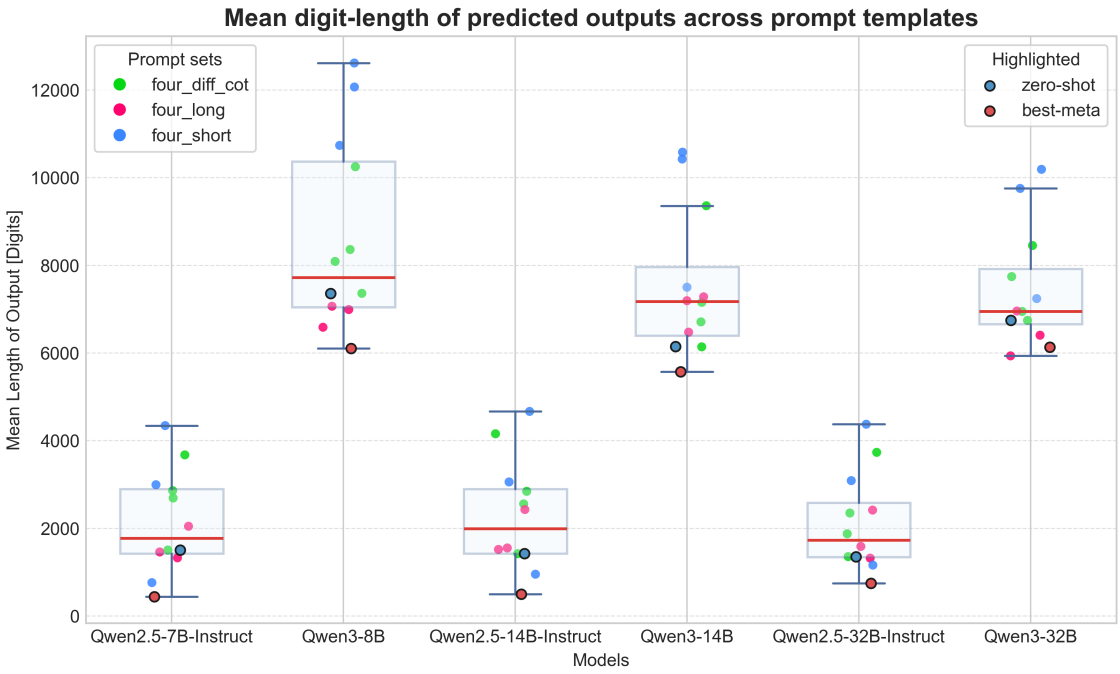

Figure 7.5: Three prompt sets, each containing four prompts, on the 8k val dataset and their mean output length. The Qwen3 models show significantly longer output compared to the Qwen2.5 models. This indicates a longer reasoning part in the response.

The slightly longer prompts from four_short, on the other hand, seem to generate the longest outputs, but this does not lead to a clear performance increase (see Figure 7.4).

The author of this work is very careful with interpretations, as the various models sometimes react unexpectedly, i.e. an aspect seems to apply to one model but not to another. Furthermore, such interpretations are not an exact science, but merely a human attempt to explain the trends that have been identified. In this field of research the models are often humanized and attributed with intentions or behaviors that otherwise only occur in humans and not in machines.

Nevertheless, an interpretation should be provided here as an example to illustrate the output length tendency to readers and highlight it for later work. The Qwen2.5 models show a very short output for the best-meta prompt. Nevertheless, its performance is by far the best. An attempt to explain (interpretation) could be as follows: The model receives the prompt as input and is trained to continue writing this input. Even though post-training trains the model to respond rather than just supplement, the prompt has a major influence on the generated response. All tokens in the prompt are taken into

account, while the response tokens are generated sequentially. Disregarding the post-training influence, one could say that the model does not care whether a reasoning step occurs in the prompt or in the output; the prompt and output are treated equally while the last tokens containing the solution are generated.

In other words, one might assume that reasoning can already be performed in the prompt, thereby shortening the output. If one understands the prompt as an instruction and the output as execution, this behavior is difficult to explain. In contrast, if one changes perspective and treats both input and output as context that helps the model generate the last tokens, i.e. the solution, it is almost evident that reasoning can also take place in the prompt. This would also explain why multi-step prompting is so successful and will take up the largest part of research from 2023-2024 onwards. The model can process a task across many prompts, incorporating the insights from the last output into the next prompt. This could lead to a paradigm that treats reasoning in the prompt and in the output similarly and draws corresponding advantages from this insight.

7.2 Multi-Step Prompting

7.3 Agent Runs

After the agent has been developed, it can be determined whether the previously used zero-shot and best-meta prompts can be improved by the agent. Before the agent can be used, the model must be specified. It seems reasonable to use the latest and best model (here Qwen3-32B). In order to test smaller models as well, the Qwen3-8B model was also used. Figure 7.6 shows how the 8B and 32B models iteratively improve the two prompts. Please note that the val dataset with 8k admission notes was not used here, but rather a subset with only 500 data points (called subval_4).

The plot shows the two prompts on two models, i.e. four agent runs. The dashed line shows the F_1 mean at iteration 0. This is the score of the initial prompt on the subset before the prompt is improved and thus an indicator if the prompt has been improved.

Figure 7.6 illustrates that the prompts are mostly improved or that better prompts were found, which **H4** confirms. However, it is also noticeable that the score does not increase iteratively but fluctuates from iteration to iteration. This could indicate that it is actually a type of Monte Carlo search and not a gradient method. The results

largely confirm **H5**, see section 5.2. At this stage, the assumption of a genuine gradient method can be considered to be questionable. In a Monte Carlo search, one would expect a fluctuating change in the score, as random elements are selected from a set. With a gradient method, one would expect a positive improvement (positive slope of the curve) until some asymptotic behavior occurs, e.g. because the prompt can no longer be improved (global maximum) or because the prompt is the best in its neighborhood (local maximum). Neither of these seems to be the case, which indicates a random method, referred to as Monte Carlo search in the literature.

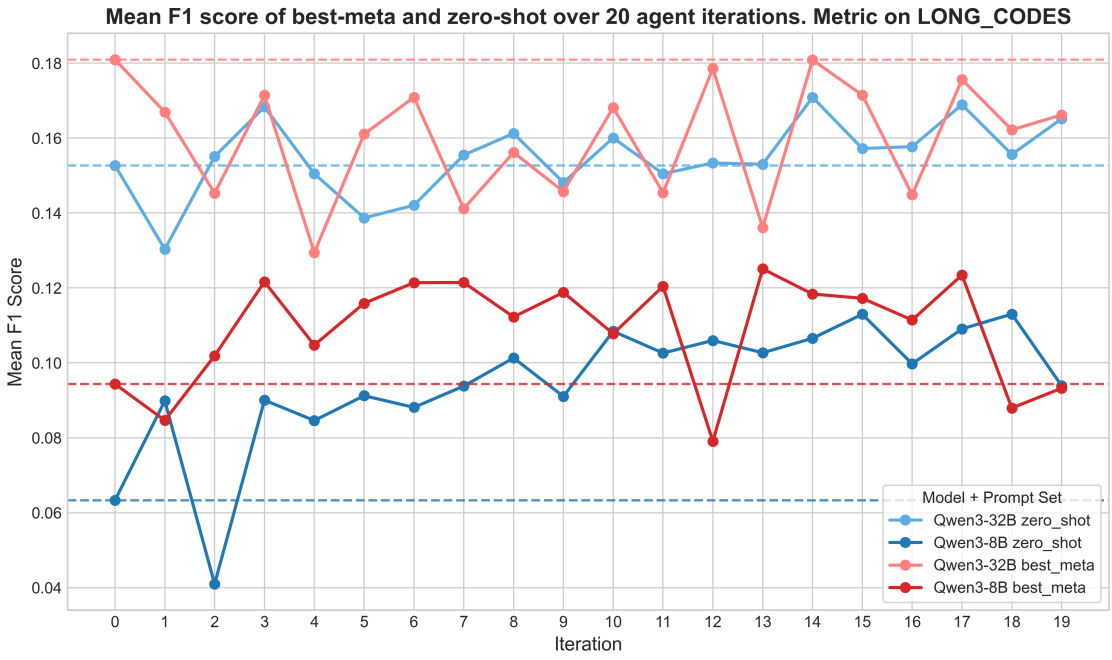

Figure 7.6: Two prompts on two models with their mean F_1 score on the subset subval_4. The dashed line shows the F_1 score before improving the prompt. Both initial prompts show improvement by the agent on both models. Except for the best-meta prompt on Qwen3-32B. This model seems to have problems further optimizing the best-meta prompt.

Furthermore, **H6** can neither be confirmed nor completely rejected, as both cases have been observed. Figure 7.6 shows that for the 8B model, the zero-shot prompt was improved to the level that it can outperform the best-meta prompt. For the 32B model, this is not the case; on the contrary, the zero-shot prompt could not even reach the score of the initial best-meta prompt after the agent's optimization. The agent is therefore not robust to model changes, and it was observed that a high-scoring initial prompt leads to better prompts than a low-scoring initial prompt. In fact, **H6** must be partially rejected

because the agent is not robust to the initial prompt.

If the behavior were asymptotic, increasing the number of iterations would only be useful to a limited extent. But if the selection of prompts is random (Monte Carlo), then the procedure would benefit from higher scaling (more iterations). To test this, an even smaller dataset was selected, called subval_3, which contains only 100 of the 8k admission notes included in the large dataset. Figure 7.7 shows only the improvement of the best-meta prompt on three models of the Qwen3 generation (8B, 14B, 32B). It is clear to see that the smaller models 8B and 14B show a clear improvement, while the largest model (32B) has difficulty improving the prompt. There may be several reasons for this, some of which will be discussed in the following.

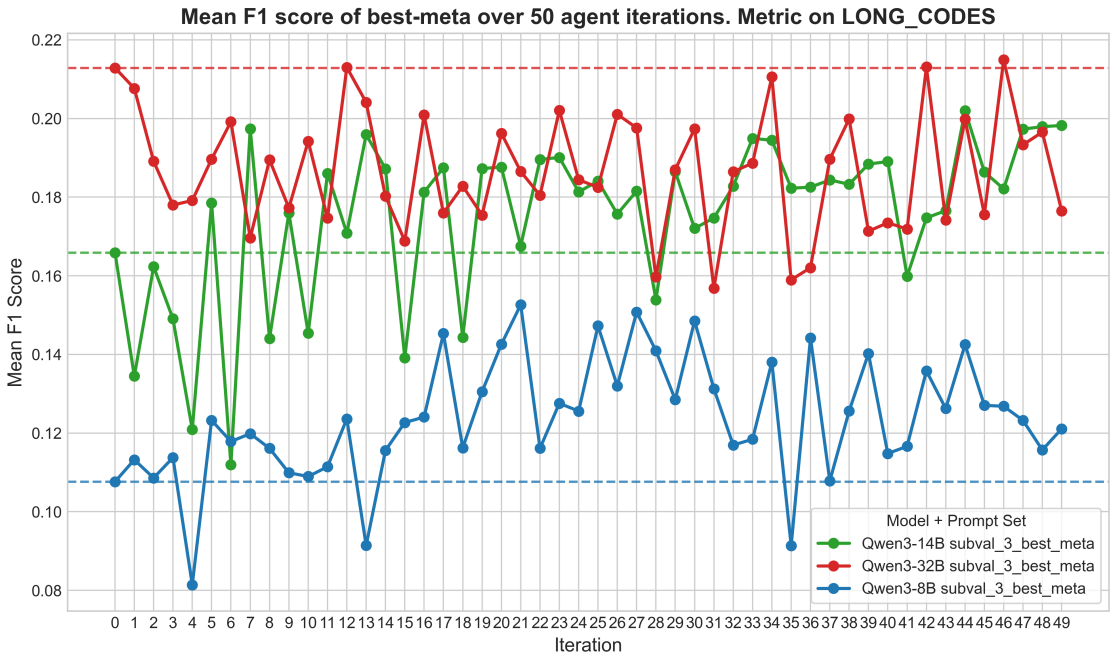

Figure 7.7: Prompt best-meta on three models with corresponding mean F_1 score on the subset subval_3. The dashed line shows the F_1 score before improving the prompt. Qwen3-8B and Qwen3-14B significantly improve the best-meta prompt, resulting in previously unreached scores. Qwen3-32B still seems to have problems optimizing the best-meta prompt.

Firstly, it can be observed that the prompt score is sometimes increased and sometimes decreased, as expected. Apart from that, the models (i.e. their score lines) do not show any clear similarities. A random seed was set for the experiment, thus the examples for self-criticize shown per iteration are always the same. One might assume that some

patients (used as a self-criticize example) lead to better prompts and others to worse ones. However, this must be partially rejected, as the lines do not always peak at the same iteration.

Nevertheless, this sometimes but rarely appears to be the case, see iteration 44. Such a behavior could be explained as follows: Assume that there are admission notes that lead to better prompts because, for example, they represent a good average of all patients. Then the score for these admission notes should improve. However, for each iteration, the model receives not only one patient, but also the output (i.e. chain of thought + predicted diseases) and the best previous prompt (+ evaluation). This output and the best previous prompt determine a large part of the improvement prompt ($\mathbf{D_2}$, see subsection 6.2.2). It follows that the same admission note does not lead to the same improvement prompt ($\mathbf{D_2}$), but only a (smaller) part of the improvement prompt is the same. However, this improvement prompt generates the new prompt for the new iteration, which is why it is not surprising that the iterations do not proceed in the same way (large fluctuation in the improvement prompt).

Secondly, it can be observed that after a peak (good prompt), the score decreases in the next iteration. The explanation for this could be similar to the first observation. The new prompt ($\mathbf{D_1}$) is generated by the improvement prompt ($\mathbf{D_2}$). It is assumed that only certain combinations of admission note (patient), predicted output (for this patient), and best previous prompt lead to good new prompts. Such a good combination may occur only rarely, which is why it is more likely that a good new prompt will be followed by a bad one. If, as already discussed, this is a Monte Carlo search, then the agent's task is to influence the set from which random selections are made so that it contains fewer bad prompts and more good prompts. This was attempted with the agent structure, e.g. by adding an evaluation and corresponding directives such as criticize the output and use this criticism to write a better prompt. In further work, this consideration could also serve to further improve the agent.

7.3.1 Best prompts found

The primary question regarding the agent is whether the prompts found so far, whether handwritten or meta prompting, can be further improved. For this purpose, three prompts were selected from all runs of the agent that had very good scores on the subsets of the data.

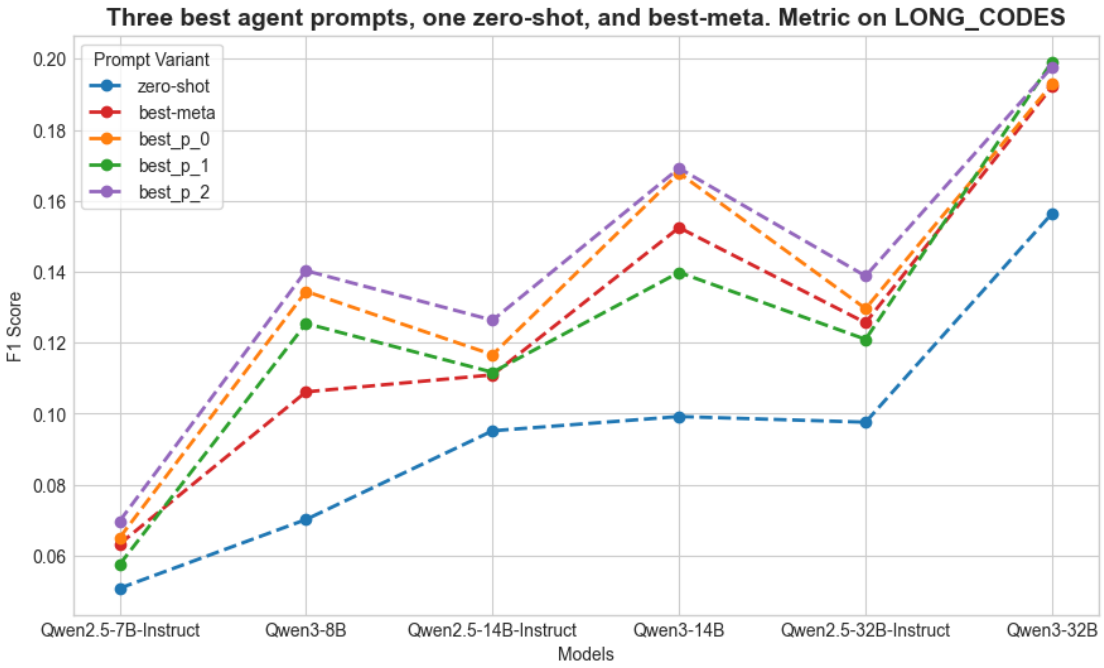

Figure 7.8: Three different prompt produced by the agent and the zero-shot and best-meta prompt for comparison on 8k val dataset. The zero-shot prompt significantly outperformed all models. The best meta prompt was also surpassed, although by a smaller margin.

These must now be tested on the large dataset in order to compare them with the single-step prompts. This is because different datasets also have different scores, whereby a fluctuation of 1-3% was observed.

Figure 7.8 shows the three best prompts found by the agent on the large val dataset and on all six models. For comparison, the zero-shot prompt and the best-meta prompt were added. The prompts best_p_1 and best_p_2 were found by Qwen3-32B, and the prompt best_p_0 by Qwen3-8B. Please consider best_p_2 in the appendix as an example of the prompts found by the agent (see subsection A.2.4). It is surprising that the 8B model was also able to find a prompt that performs that well.

Finally **H1** can be confirmed (see section 5.1). Although the zero-shot prompt also generates a chain of thought, as the models are trained to do so, this chain of thought was not specifically inspired by the prompt. For all other prompts, a chain of thought was triggered and partially guided. This specialization of the prompts to the task (and the dataset) led to a significant improvement in performance. **H1** assumed exactly that: CoT prompt methods lead to an increase in performance. This could therefore be demonstrated

within the experimental configuration, i.e. for the models and datasets used.

Furthermore, the best-meta prompt could also be improved for all models. Even though best-meta prompt seems to be really good, i.e. it could be really close to the best possible prompts, the prompt was still outperformed. Thus **H4** from section 5.2 could be accepted. Although the effectiveness of the agent depends on the model, most models show a performance increase of 2-4% between the best-meta and the best agent prompt. Nevertheless, performance advantage depends highly on the model and the 32B model benefits least from the improvement. However, since Qwen3-32B was the model with the best overall performance, this issue needs to be further discussed (see subsection 7.3.2).

All evaluations shown in the thesis were consistently performed on the long codes (see section 4.1). Here, the short code evaluation, which was frequently considered at CliniBench (with regard to the encoders), should also be shown for the best agent prompts and for the comparison prompts zero-shot and best-meta [Grundmann et al. 2025].

Prompt	Precision	Recall	F1	Prompt len	Output len	Codes Pred
zero-shot	22.83	21.56	20.81	226	6742	12
best-meta	**30.37**	23.25	24.81	2802	6135	10
best_p_0	28.03	24.69	24.82	**4375**	7056	11
best_p_1	28.37	25.38	**25.28**	2758	7014	12
best_p_2	27.85	**25.64**	25.10	1268	**9758**	12

Table 7.1: Evaluation results of the best agent prompts and the zero-shot and best-meta prompt for comparison on 8k val dataset. Qwen3-32B was used and the metric applies to short_codes. Bold indicates the maximum within a column and len is the length of the prompt or output measured in digits.

On the one hand, Table 7.1 shows more information such as prompt length, output length, and mean predicted codes. The prompts predicted between 10 and 12 codes on average, whereas the ground truth of the dataset has an average of 12.7 codes. It is interesting to see that the best prompts converge on this average, which is described in more detail in subsection 7.3.2. On the other hand, it shows that the ratio of prompts (score order) depends on the metric. There seem to be prompts that perform better on precision and others that perform better on recall.

7.3.2 Limitations of the Agent

The advantages and disadvantages of the agent will be highlighted in this subsection. An attempt will be made to explain why the Qwen3-32B model was not able to improve the prompts as well as the other models. Two potential explanations will be considered, although there are many other possible reasons. Like all other attempts at explanation in this thesis, these are interpretations and therefore speculative. First, the distribution of prompts can be considered in terms of their scores, with the assumption that very good prompts are occurring less frequently. Second, the number of predicted ICD codes can be considered, as these have a major impact on precision and recall (and thus F_1). It will be shown that Qwen3-32B tends to produce too many codes, which significantly decreases precision.

The first assumption has already been outlined with the Monte Carlo search. It consists of the assumption that the prompts are not uniformly distributed with respect to their score, but rather follow a normal distribution.

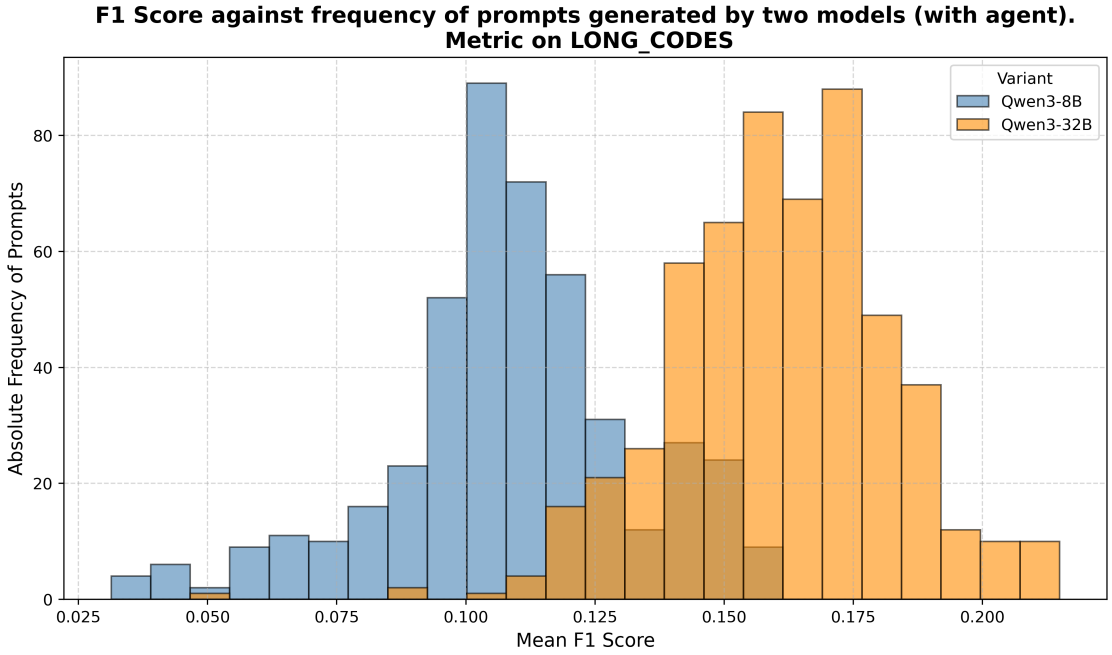

Figure 7.9: The histogram illustrates the frequency of the mean F_1 scores for all approximately 1.1k prompts generated by the agent with Qwen3 8B and 32B. The prompts were found and tested on different subsets. For both models, the agent appears to produce prompts that are approximately normally distributed.

This would result in the set of all prompts from which the agent selects (Monte Carlo search paradigm) containing many prompts with medium scores and few with very good or very poor scores.

Figure 7.9 shows that all prompts produced by the agent are indeed approximately normally distributed (about 1.1k prompts). Assuming that the agent actually selects one randomly from the subset of all possible prompts, say those with a score greater than 0, i.e. those that actually produce ICD codes. Then the distribution of all agent prompts would approximate the real distribution and thus demonstrate the normal distribution of the underlying set. In this case, it would suggest that very good prompts are rarely selected because they are rare. However, this assumption is unlikely. It is possible, for example, that prompts exist whose score is twice as high as the agent's best prompts, but the agent does not find them. Since this cannot be verified experimentally and cannot be ruled out theoretically, the assumption that this is the actual distribution should be rejected from a scientific point of view.

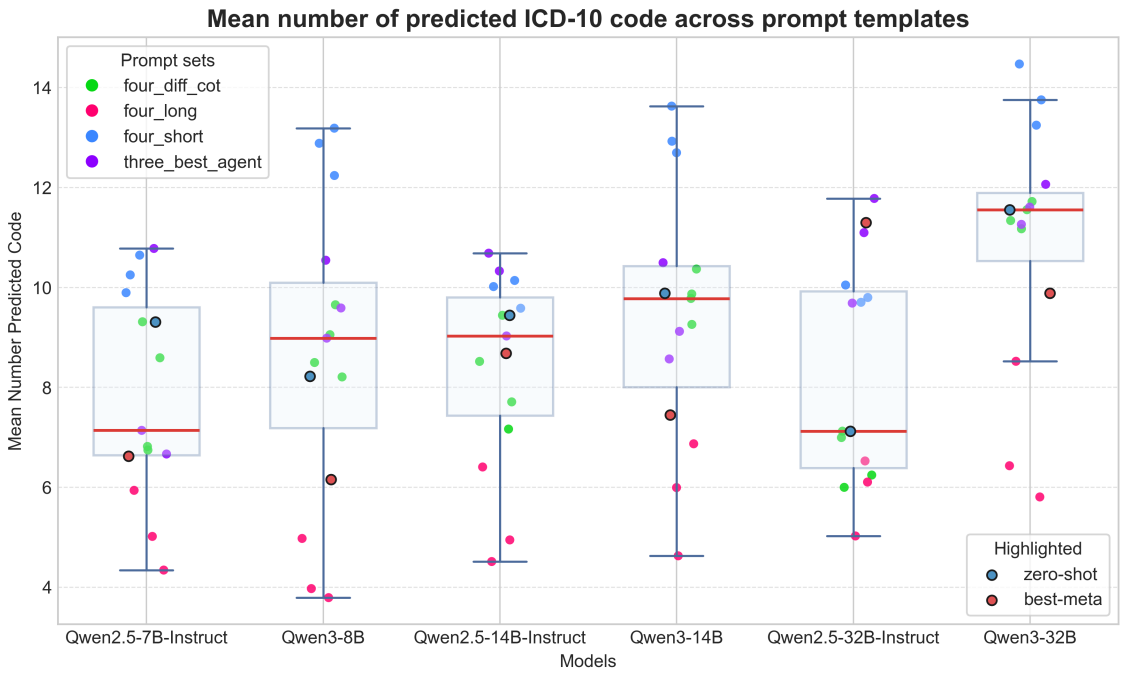

Figure 7.10: Four prompt sets, each containing three to four prompts, on the 8k val dataset and their mean number of predicted codes. The number of predicted codes appears to be relatively constant across different models per prompt set. The Qwen3 models, especially 32B, tend to produce more codes, at least for some prompts.

What can be clearly observed, though, is that the set from which the agent chooses

is approximately normally distributed. At least under the experimental configurations, i.e. the model, the model hyperparameters, the dataset, and the initial prompt. It can be concluded that applying the agent repeatedly also produces more very good prompts.

The second assumption is that Qwen3-32B tends to produce more ICD codes than the other models. To verify this, the data from the single step section was used again and expanded to include the best three agent prompts. Figure 7.10 clearly shows that Qwen3-32B produces more ICD codes on average than the other models. The appendix also contains an example showing the automatic evaluation txt file after each agent run (see section A.3). This file is available for each experiment in the data attached to the thesis. It clearly shows that Qwen3-32B produces far too many codes for this dataset.

This reveals a clear weakness of the agent. If a model cannot improve the initial prompt, it is only presented with this best (initial) prompt to improve. In other words, the agent has landed in a local maximum and cannot get out of it. In earlier versions of the agent, it was not the best prompt that was used for improvement, but the last prompt. This led to greater fluctuations in performance, but less getting trapped in local maxima. However, a local minimum could be problematic here, i.e. the current prompt is so bad that it can no longer lead to good prompts. Metaphorically, the agent has forgotten the initial prompt.

Please note that local extrema does not refer to a gradient here, nor is a loss function considered that could be minimized. The metaphorical local minimum refers to the (initial) prompt that is to be improved. If this prompt performs very poorly, for example because it does not request ICD codes, this could lead to all subsequent prompts performing equally poorly (if the last prompt is improved). If, on the other hand, the previous best prompt is always improved, it may be that no better prompt is found after 50 iterations. In this case, the agent would receive the same prompt to improve with each iteration (50 times). This has been metaphorically referred to as a local maximum because the prompt leads to a set from which a new prompt is taken that does not contain a better prompt. However, it is possible that a better prompt is included in another Monte Carlo search set, but the agent does not find it because it is biased by the initial prompt.

Finally, let's consider again the relationship between predicted codes and the performance (F_1 score) of the prompts. Figure 7.11 shows all prompts from the four prompt sets that have already been discussed. This time, the average number of codes was plotted against the F_1 score of the prompt. In addition, each prompt is identifiable because the

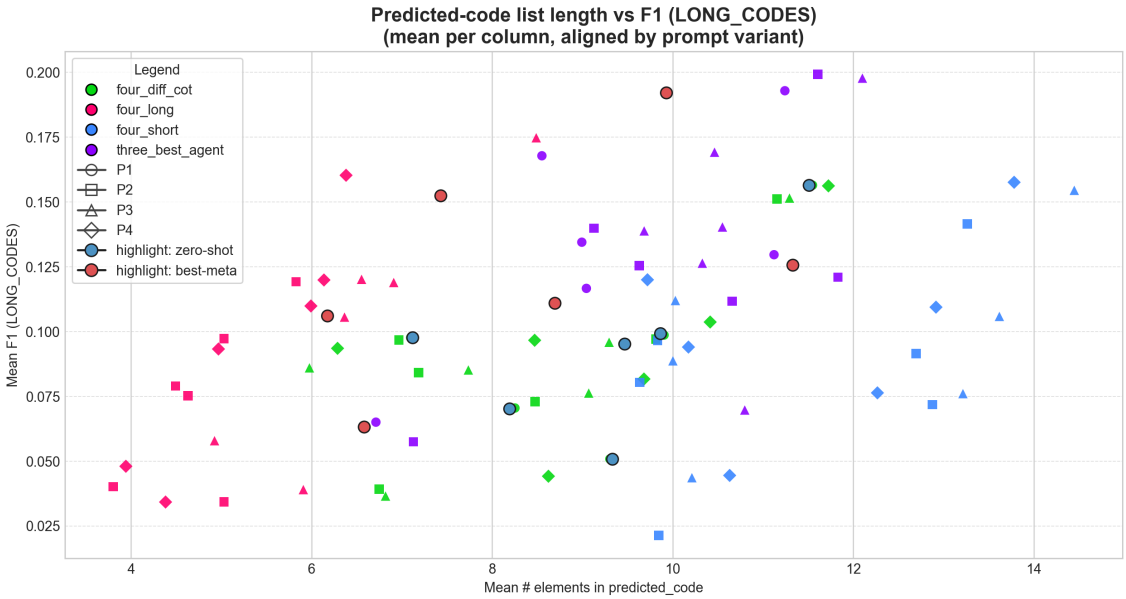

Figure 7.11: Four prompt sets, each containing three to four prompts, on the 8k val dataset. The mean number of predicted codes is plotted against their mean F_1 score. The scatter plot shows a peak between 10 and 12 predicted codes. With more or fewer codes, performance appears to decrease. Furthermore, a broad clustering of the prompt sets can be observed.

symbols point to the respective index of the prompt set. Please note that each prompt occurs six times, as six different models were used (see e.g. best-meta prompt). Although the models respond differently to the prompts, it can be observed that the respective prompt sets (colors) form a kind of cluster. Each prompt set seems to trigger a certain number of predicted codes, with a model-dependent variance. It is also interesting to see that the best agent prompts are close to the ideal 12.7 codes. This ideal is the actual average of the codes per admission note. It is not surprising that prompts close to this ideal have good precision and recall scores. However, it is remarkable that the agent finds this without being told the actual code count of the dataset.

7.4 Summary

The evaluation of the experiments showed that prompts perform better when they lead to an adequate reasoning (chain of thought). As in the previous chapters, the evaluation was also divided into single-step and multi-step prompting. The single-step CoT methods, i.e. zero-shot CoT methods from the literature, did not show any clear improvements. Even

the manually written prompts could not outperform the zero-shot default prompt. Only the meta-prompts created with GPT-4o showed a clear performance advantage. However, this analysis allows the first three hypotheses for single-step methods to be evaluated:

H1: CoT prompting improves the performance of the models tested on the given dataset. **H1** could be confirmed, given that the CoT prompts are optimized. In this case, the prompts that induce reasoning perform significantly better than those that do not request reasoning (see Figure 7.8).

H2: There are zero-shot CoT prompt templates from the literature that show the high performance across different models, i.e. they are robust to model changes. **H2** had to be rejected, because no CoT literature prompt provided a clear performance advantage. In addition, these literature prompts did not have a consistent score across different models and could not outperform the default zero-shot prompt (see subsection 7.1.1).

H3: Manual prompt engineering or meta prompting can adapt the zero-shot CoT prompt to the task and the dataset. This allows the previous tested prompt templates to be outperformed. **H3** was partially supported, because the best-meta prompt found by GPT-4o outperformed all previous CoT prompts. The prompt was adapted to the task and thus provided significant performance advantages. On the other hand, neither the manually crafted prompts nor the other meta prompts were significantly better than the zero-shot default prompt (see section 7.1.2).

In the multi-step sections, an evaluation of the agent was discussed. In most cases, the agent was able to improve the initial prompts. It was shown that the zero-shot prompt and the best-meta prompt given as initial prompts could be improved by different models (Qwen3-8B, Qwen3-14B, Qwen3-32B). Finally, three high-performing prompts found by the agent were selected and compared with the zero-shot and best-meta prompt. Across all six models, the zero-shot prompt was outperformed by 2-6%, and in some cases performance was almost doubled. The best-meta prompt was already very good, but could also be outperformed by 2-3%. These observations could then be used to validate the three multi-step hypotheses:

H4: Automatic prompt engineering (APE) can further improve manually crafted prompts or the earlier meta prompts and thus outperform previous methods. **H4**

could be confirmed, as the previous prompts were clearly improved by the agent. Even though the improvement depends on the model and the initial prompt, significant increases were observed in most cases (see section 7.3).

H5: The method used, APE with self-criticism, is not a gradient method, as it does not show continuous improvement (increasing performance for every iteration) and should be described as a random method (e.g. Monte Carlo Search). **H5** could be supported, as no evidence of a gradient method was found. However, it seems reasonable to refer to a random selection method, such as Monte Carlo search (see section 7.3).

H6: The agent is able to replaces manual prompt engineering completely, i.e. it is robust against the initial prompt. The agent is therefore capable of producing a high-scoring prompt from a low-scoring prompt. **H6** must be partially rejected because the agent is not robust to the initial prompt. A high-performing initial prompt leads to better generated prompts than a low-performing one. However, since the initial zero-shot prompt sometimes led to generated prompts that perform better than the best-meta prompt (on the same model), H6 cannot be completely rejected. It is assumed that more sophisticated agents can always achieve this (see Figure 7.6).

Furthermore, an evaluation table was shown, which is available for each individual experiment. This shows the evaluation on the short codes and enables direct comparison with the CliniBench paper. Furthermore, it was discussed whether this is a Monte Carlo search or a gradient method. The gradient paradigm was rejected and the Monte Carlo search paradigm was considered likely. This is also supported by an approximately normal distribution of all prompts found by the agent. When discussing the limitations of the agent, it was observed that Qwen3-32B tends to produce too many codes. The other Qwen3 models appear to have similar problems. However, the Qwen3-32B model seems to be particularly biased in this direction.

Finally, a figure of all prompts performed on the val dataset was shown, plotting the mean number of predicted codes against their F1 score. The prompt sets were observed to form clusters and to show minimal robustness of their average number of predicted codes across all models.

Overall, it was shown that prompts requesting specific reasoning adapted to the task perform better than general prompts. The thesis is intended as an approach to this task adaptation and presents an automated framework (agent) that can perform this task adaptation through self-criticism and meta prompting.

Chapter 8

Conclusion and Future Work

8.1 Summary

The object of the thesis was to examine the reasoning capabilities of language models on a clinical, i.e. medical, task or dataset. Reasoning here mainly refers to the generation of a chain of thought by the model. With respect to the literature and the evolution of language models, it is evident that since late 2024 (or early 2025) all new models have been learning chain of thought generation as default during training. In other words, they are trained to generate chains of thought. It does not seem reasonable to suppress this chain of thought for the given task. Instead, the question arises whether there exist so-called CoT prompts that can positively influence the generated chains of thought.

It was demonstrated that prompts that guide or explicitly ask for chains of thought can clearly outperform more general prompts. In other words, with CoT prompts specifically designed for the task, performance could be doubled in some cases and increased by about a third for the best model (Qwen3-32B). However, it also became apparent that prompts with long CoT instructions, i.e. directives that request chains of thought explicitly, can also have a negative impact on performance. Several prompts were created that perform poorer than the default zero-shot prompt. In fact, this default prompt is close to the median of the 12 crafted prompts (see Figure 7.4).

As a result, this allowed to answer the first research **RQ1** question, which was: Can prompts that induce task-specific reasoning outperform general CoT prompts, i.e. does this approach lead to sophisticated reasoning capabilities and thus better clinical outcome prediction? Even though not all prompts requesting task-specific reasoning showed high

performance, this approach enabled to find prompts (e.g. best-meta) that significantly outperformed all previous prompts.

Based on this finding, a multi-step framework, referred to as an agent, was developed that automatically generates and tests prompts. It became clear that not every generated prompt is better than the initial prompt. Nevertheless, a large number of prompts were found that clearly outperform the initial prompt. The approach is therefore to generate a large number of prompts using a random method in order to filter out the best ones. It appeared to be problematic to identify promising prompts before they were tested. The only way to ensure that these prompts are indeed superior to the initial prompt is to test them on a large dataset.

The agent developed in this thesis achieved certain successes, but should be considered to be a showcase within the limits of a master thesis. A sophisticated agent for commercial use would require further development and refinement, which can hardly be accomplished by one individual within a few months. Nevertheless, the prototype agent in this thesis was able to show that it is possible to create such an agent using relatively basic methods such as tag-supported output parsing. The agent was thus able to produce the expected results in terms of prompt optimization. Further work may show whether this method can also be applied in other situations and on a larger scale of medical tasks (or other task areas).

Overall, this also provided an answer to the second research question **RQ2**, which was: Can the process of writing and optimizing a task-specific prompt be automized, i.e. can a multi-step framework perform task-specific prompt engineering at a human level or better? With newer language models (from 2025), the process could be fully automated. It was demonstrated that the framework (agent) used generated prompts that were significantly better than all manually created prompts and even clearly outperformed those created by GPT-4o.

The advantages and disadvantages of such an agent approach were also discussed, such as getting stuck in local maxima or minima. In addition, some of these weaknesses can be mitigated in further work. Other limitations or challenges may apply to all agents. This work has mainly focused on improving a zero-shot prompt. In order to initiate the improvement of such a prompt in a multi-step framework, further instructions are required. However, these instructions could themselves be optimized again. This creates a recursive problem in which each optimization of an instruction itself requires a directive,

and this directive could be further optimized. Like the combination problem mentioned above (with few shots and context), this problem is difficult to solve because it recursively refers back to itself and cannot be completely avoided.

A clear advantage of such automated approaches, i.e. APE agents or multi-step frameworks, is to generate a large number of prompts. As already discussed, the score of these prompts appears to be approximately normally distributed, which means that few poor and few high quality prompts are found, and many average ones. Nevertheless, by scaling the method, a large number of prompts with high scores can be found. If a kind of meta-analysis of prompts is to be carried out, then such an APE approach can deliver a large number of prompts that can then be further investigated.

8.2 Future Work

Based on these capabilities of automated frameworks, two future scientific approaches can be outlined. Assuming that a large number of good prompts for a specific task (dataset) have already been found. First, these high-scoring prompts can be analyzed for semantic similarity. This could lead to the development of a method that filters generated prompts without testing them by quantitatively determining their similarity to the very high scoring prompts (embedding distance).

Second, for an important task with a labeled dataset, a large number of high performing prompts can be generated. It is possible that in a medical context, prompts not only perform well or poorly overall, but also perform individually for each patient, i.e. their diseases and the associated medical domain. A method could be discussed in which a large number of prompts are created for a task, such as clinical predictions , and the prompt that matches the corresponding patient and their symptoms is selected. This method would be very similar to a RAG approach, except that instead of retrieving similar patients as few-shots, similar prompts would be retrieved. This would enable the prompt to be patient-specific rather than task-specific only, and it is anticipated that such a method would further improve performance per patient or admission note. Both approaches for future work require a large number of prompts, especially high-performing prompts. Automated processes appear necessary to find these, and agents such as the one presented could be assigned to this task.

8.3 Conclusion

In summary, this work is located in the field of in-context learning. This implies that models are not further trained or fine-tuned, but rather directly utilized, i.e. prompted, to perform tasks. The thesis was able to show that the success of in-context learning approaches depends heavily on the prompt used. According to recent research, the field of in-context learning seems to be evolving in a direction where the prompt is adapted to the task. Automated approaches seem to be suitable for this task-specific adaptation (prompt engineering).

Since only zero-shot approaches for in-context learning were further investigated in this thesis, the question remains whether few-shot learning can outperform zero-shot approaches. However, the generation of examples (few shots) containing chains of thought is a completely different topic, and it has already been discussed that the selection of these few shots itself requires a complex process. In terms of computational costs, this is in competition with the prompt engineering of the directive (zero-shot approach). However, it cannot be excluded that such few-shot prompts may outperform the zero-shot prompts shown here. Future work will be required to examine the capabilities of such few-shot prompts and to compare them with zero-shot prompting.

In conclusion, it is hoped that this work could contribute to the field of research by investigating the developability and applicability of such agents. The agent is only a showcase and can be mostly disregarded. But the thesis has been shown that it is possible for one person (a master student) to develop such a prototype agent within two months. It is assumed that research groups consisting of many qualified individuals can develop much larger and more sophisticated agents. The advantage of this thesis is to demonstrate that it is relatively straightforward to develop a minimal agent and that the approaches used here can be easily implemented by experts in the field of NLP.

It is assumed that further work in this field will improve the optimization of prompts by adapting them optimally to the task and task-specific reasoning. This could leverage the capabilities of reasoning and mark a new stage in the evolution of in-context learning.

Bibliography

Aken, Betty van, Jens-Michalis Papaioannou, Manuel Mayrdorfer, Klemens Budde, Felix A. Gers, and Alexander Löser [2021]. "Clinical Outcome Prediction from Admission Notes using Self-Supervised Knowledge Integration". In: *CoRR* abs/2102.04110. arXiv: 2102.04110. URL: https://arxiv.org/abs/2102.04110.

Archiv, Startup [n.d.] *Sam Altman explains the two strategies for startups building on AI.* URL: https://www.startuparchive.org/p/sam-altman-explains-the-two-strategies-for-startups-building-on-ai. (accessed: 24.10.2025).

Bai, Jinze, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu [2023]. "Qwen Technical Report". In: *CoRR* abs/2309.16609. DOI: 10.48550/ARXIV.2309.16609. arXiv: 2309.16609. URL: https://doi.org/10.48550/arXiv.2309.16609.

Berglund, Martin and Brink van der Merwe [2023]. "Formalizing BPE Tokenization". In: *Proceedings of the 13th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2023, Famagusta, North Cyprus, 18th-19th September, 2023.* Ed. by Benedek Nagy and Rudolf Freund. Vol. 388. EPTCS, pp. 16–27. DOI: 10.4204/EPTCS.388.4. URL: https://doi.org/10.4204/EPTCS.388.4.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child,

Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei [2020]. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165. arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

Christiano, Paul F., Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei [2017]. "Deep reinforcement learning from human preferences". In: *CoRR* abs/1706.03741. arXiv: 1706.03741. URL: http://arxiv.org/abs/1706.03741.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova [2018]. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Dubey, Abhimanyu et al. [2024]. "The Llama 3 Herd of Models". In: *CoRR* abs/2407.21783. DOI: 10.48550/ARXIV.2407.21783. arXiv: 2407.21783. URL: https://doi.org/10.48550/arXiv.2407.21783.

Grundmann, Paul, Dennis Fast, Jan Frick, Thomas Steffek, Felix A. Gers, Wolfgang Nejdl, and Alexander Löser [2025]. "CliniBench: A Clinical Outcome Prediction Benchmark for Generative and Encoder-Based Language Models". In: *CoRR* abs/2509.26136. DOI: 10.48550/ARXIV.2509.26136. arXiv: 2509.26136. URL: https://doi.org/10.48550/arXiv.2509.26136.

Jaech, Aaron, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso,

Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, and Ilge Akkaya [2024]. "OpenAI o1 System Card". In: *CoRR* abs/2412.16720. DOI: 10.48550/ARXIV.2412.16720. arXiv: 2412.16720. URL: https://doi.org/10.48550/arXiv.2412.16720.

Johnson, Alistair E. W., Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J. Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, Li-wei H. Lehman, Leo A. Celi, and Roger G. Mark [Jan. 2023]. "MIMIC-IV, a freely accessible electronic health record dataset". en. In: *Scientific Data* 10.1. Publisher: Nature Publishing Group, p. 1. ISSN: 2052-4463. DOI: 10.1038/s41597-022-01899-x. URL: https://www.nature.com/articles/s41597-022-01899-x [visited on 01/07/2025].

Khattab, Omar, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts [2023]. "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines". In: *CoRR* abs/2310.03714. DOI: 10.48550/ARXIV.2310.03714. arXiv: 2310.03714. URL: https://doi.org/10.48550/arXiv.2310.03714.

Kojima, Takeshi, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa [2022]. "Large Language Models are Zero-Shot Reasoners". In: *CoRR* abs/2205.11916. DOI: 10.48550/ARXIV.2205.11916. arXiv: 2205.11916. URL: https://doi.org/10.48550/arXiv.2205.11916.

meta-llama [n.d.] *Llama-4-Scout-17B-16E: Model Card*. URL: https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E. (accessed: 21.10.2025).

OpenAI [2023]. "GPT-4 Technical Report". In: *CoRR* abs/2303.08774. DOI: 10.48550/ARXIV.2303.08774. arXiv: 2303.08774. URL: https://doi.org/10.48550/arXiv.2303.08774.

OpenAI [2025]. *GPT-5 System Card*. OpenAI. URL: https://cdn.openai.com/gpt-5-system-card.pdf.

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder,

Paul F. Christiano, Jan Leike, and Ryan Lowe [2022]. "Training language models to follow instructions with human feedback". In: *CoRR* abs/2203.02155. DOI: 10.48550/ARXIV.2203.02155. arXiv: 2203.02155. URL: https://doi.org/10.48550/arXiv.2203.02155.

Pak, Igor [1997]. "When and how n choose k". In: *Randomization Methods in Algorithm Design, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, December 12-14, 1997*. Ed. by Panos M. Pardalos, Sanguthevar Rajasekaran, and José Rolim. Vol. 43. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, pp. 191–238. DOI: 10.1090/DIMACS/043/12. URL: https://www.math.ucla.edu/~pak/papers/nk13.pdf.

Pryzant, Reid, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng [2023]. "Automatic Prompt Optimization with "Gradient Descent" and Beam Search". In: *CoRR* abs/2305.03495. DOI: 10.48550/ARXIV.2305.03495. arXiv: 2305.03495. URL: https://doi.org/10.48550/arXiv.2305.03495.

Rajbahadur, Gopi Krishnan, Gustavo Ansaldi Oliva, Dayi Lin, and Ahmed E. Hassan [2024]. "From Cool Demos to Production-Ready FMware: Core Challenges and a Technology Roadmap". In: *CoRR* abs/2410.20791. DOI: 10.48550/ARXIV.2410.20791. arXiv: 2410.20791. URL: https://doi.org/10.48550/arXiv.2410.20791.

Reynolds, Laria and Kyle McDonell [2021]. "Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm". In: *CoRR* abs/2102.07350. arXiv: 2102.07350. URL: https://arxiv.org/abs/2102.07350.

Schulhoff, Sander, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson C. Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Miserlis Hoyle, and Philip Resnik [2024]. "The Prompt Report: A Systematic Survey of Prompting Techniques". In: *CoRR* abs/2406.06608. DOI: 10.48550/ARXIV.2406.06608. arXiv: 2406.06608. URL: https://doi.org/10.48550/arXiv.2406.06608.

Sun, Kaiser and Mark Dredze [2024]. "Amuro & Char: Analyzing the Relationship between Pre-Training and Fine-Tuning of Large Language Models". In: *CoRR* abs/2408.06663.

DOI: 10.48550/ARXIV.2408.06663. arXiv: 2408.06663. URL: https://doi.org/10.48550/arXiv.2408.06663.

Wang, Lei, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim [2023]. "Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models". In: *CoRR* abs/2305.04091. DOI: 10.48550/ARXIV.2305.04091. arXiv: 2305.04091. URL: https://doi.org/10.48550/arXiv.2305.04091.

Wei, Jason, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le [2021]. "Finetuned Language Models Are Zero-Shot Learners". In: *CoRR* abs/2109.01652. arXiv: 2109.01652. URL: https://arxiv.org/abs/2109.01652.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou [2022]. "Chain of Thought Prompting Elicits Reasoning in Large Language Models". In: *CoRR* abs/2201.11903. arXiv: 2201.11903. URL: https://arxiv.org/abs/2201.11903.

Yang, An, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jian Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu [2025]. "Qwen3 Technical Report". In: *CoRR* abs/2505.09388. DOI: 10.48550/ARXIV.2505.09388. arXiv: 2505.09388. URL: https://doi.org/10.48550/arXiv.2505.09388.

Yang, An, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou,

Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan [2024]. "Qwen2 Technical Report". In: *CoRR* abs/2407.10671. DOI: 10.48550/ARXIV.2407.10671. arXiv: 2407.10671. URL: https://doi.org/10.48550/arXiv.2407.10671.

Yang, An, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu [2024]. "Qwen2.5 Technical Report". In: *CoRR* abs/2412.15115. DOI: 10.48550/ARXIV.2412.15115. arXiv: 2412.15115. URL: https://doi.org/10.48550/arXiv.2412.15115.

Zhao, Tony Z., Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh [2021]. "Calibrate Before Use: Improving Few-Shot Performance of Language Models". In: *CoRR* abs/2102.09690. arXiv: 2102.09690. URL: https://arxiv.org/abs/2102.09690.

Zhou, Denny, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed H. Chi [2022]. "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models". In: *CoRR* abs/2205.10625. DOI: 10.48550/ARXIV.2205.10625. arXiv: 2205.10625. URL: https://doi.org/10.48550/arXiv.2205.10625.

Zhou, Yongchao, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba [2022]. "Large Language Models Are Human-Level Prompt Engineers". In: *CoRR* abs/2211.01910. DOI: 10.48550/ARXIV.2211.01910. arXiv: 2211.01910. URL: https://doi.org/10.48550/arXiv.2211.01910.

Zhou, Yucheng, Xiubo Geng, Tao Shen, Chongyang Tao, Guodong Long, Jian-Guang Lou, and Jianbing Shen [2023]. "Thread of Thought Unraveling Chaotic Contexts". In: *CoRR* abs/2311.08734. DOI: 10.48550/ARXIV.2311.08734. arXiv: 2311.08734. URL: https://doi.org/10.48550/arXiv.2311.08734.

Appendix A

A.1 Quotes

> "I think fundamentally there are two strategies to build on AI right now. There's one strategy which is assume the model is not going to get better and build all of these little things on top of it. Then there's another strategy, which is to build assuming that OpenAI is going to stay on the same trajectory and the models are going to to keep getting better at the same pace. It would seem to me that 95% of the world should be betting on the latter category" [Archiv n.d.].

A.2 Prompt Sets

A.2.1 four_diff_cot

```
{
  "prompt_0": [
    {
      "role": "system",
      "content": "You are a helpful medical assistant."
    },
    {
      "role": "user",
      "content": "Here is a patient's admission note. Please provide the most
      ↪  likely diagnoses in the form of ICD-10 codes:\n\n{} \n\n"
    }
  ],

  "prompt_1": [
    {
```

```
      "role": "system",
      "content": "You are a helpful medical assistant."
    },
    {
      "role": "user",
      "content": "Here is a patient's admission note. Please provide the most
      ↪  likely diagnoses in the form of ICD-10 codes:\n\n{} \n\nLet's think
      ↪  step by step"
    }
  ],

    "prompt_2": [
    {
      "role": "system",
      "content": "You are a helpful medical assistant."
    },
    {
      "role": "user",
      "content": "Here is a patient's admission note. Please provide the most
      ↪  likely diagnoses in the form of ICD-10 codes:\n\n{} \n\nLet's first
      ↪  understand the problem and devise a plan to solve the problem. Then,
      ↪  let's carry out the plan and solve the problem step by step"
    }
  ],

    "prompt_3": [
    {
      "role": "system",
      "content": "You are a helpful medical assistant."
    },
    {
      "role": "user",
      "content": "Here is a patient's admission note. Please provide the most
      ↪  likely diagnoses in the form of ICD-10 codes:\n\n{} \n\nWalk me
      ↪  through this context in manageable parts step by step, summarizing
      ↪  and analyzing as we go"
    }
  ]
}
```

A.2.2 four_short

```
{
  "prompt_0": [
    {
      "role": "system",
      "content": "You are a helpful medical assistant."
    },
    {
      "role": "user",
      "content": "Here is a patient's admission note. Please provide the most
      ↪  likely diagnoses in the form of ICD-10 codes:\n\n{} \n\n"
    }
  ],
  "prompt_1": [
    {
      "role": "system",
      "content": "You are an experienced clinical assistant in the emergency
      ↪  department."
    },
    {
      "role": "user",
      "content": "Carefully read the following admission note and directly
      ↪  provide at least ten most likely diagnoses in the form of ICD-10
      ↪  codes. Only enter diagnoses that you consider certain based on the
      ↪  information provided: \n\n{} \n\n"
    }
  ],
  "prompt_2": [
    {
      "role": "system",
      "content": "You are an experienced clinical assistant in the emergency
      ↪  department with expert knowledge of ICD-10 coding."
    },
    {
      "role": "user",
      "content": "Analyze the following admission note step-by-step:\n1.
      ↪  Extract relevant clinical findings (symptoms, history, physical exam,
      ↪  labs, imaging).\n2. From these findings, infer at least ten likely
      ↪  diagnoses.\n3. Map each diagnosis to the most appropriate ICD-10
      ↪  code.\n\nAdmission Note:\n{}\n\nProvide your reasoning followed by a
      ↪  list of ICD-10 codes."
    }
  ],
  "prompt_3": [
    {
```

```
      "role": "system",
      "content": "You are a senior emergency medicine clinician and expert in
      ↪  clinical coding using ICD-10."
    },
    {
      "role": "user",
      "content": "Please carefully read the following admission note.\n\nFollow
      ↪  this process:\n1. Summarize key patient data: demographics, chief
      ↪  complaint, relevant history, vitals, exam findings, and test
      ↪  results.\n2. Identify and explain possible differential
      ↪  diagnoses.\n3. Select at least ten most likely final diagnoses based
      ↪  on available evidence.\n4. Assign the most appropriate ICD-10 codes
      ↪  to each final diagnosis.\n5. Justify each code assignment based on
      ↪  the text.\n\nAdmission Note:\n{}\n\nPresent your reasoning and then
      ↪  provide the ICD-10 codes."
    }
  ]
}
```

A.2.3 best-meta prompt

Author's note: Please note that this prompt created by GPT-4o was not generated after a single instruction to GPT-4o, but after a chat with various instructions. These cannot be listed in detail here, but the content of the conversation and the last two instructions can be. First, MIMIC-IV was discussed with GPT-4o. Then the model was asked to generate a sample admission note.

Prompt: "Okay, as a next step, could you please give me a real example of a patient node and the real ground truth of that patient node in ICD-10 codes and their words of the diseases?"

This was followed by the instruction that led to the generation of best-meta prompt.

Prompt: "Okay, so if you have this patient node and this ground truth, can you design me a very high professional prompt which included everything that is called a god prompt or something like this with different steps of rule content and warnings? And could you write me this prompt?"

```
{
  "prompt_0": [
    {
      "role": "system",
      "content": "You are a clinically trained AI medical coder with extensive
      ↪  knowledge of ICD-10 codes, clinical terminology, and diagnostic
      ↪  criteria. Your task is to carefully read and analyze a patient's
      ↪  hospital note (e.g., discharge summary, admission note, or clinical
      ↪  progress note) and accurately extract all relevant final diagnoses in
      ↪  the form of:\n\n- ICD-10 codes\n- Corresponding diagnosis
      ↪  names\n\nYou must follow the instructions and rules below with high
      ↪  precision."
    },
    {
      "role": "user",
```

```
      "content": "======================\n STEP-BY-STEP
      ↪  INSTRUCTIONS\n======================\n\n1. **Read the entire clinical
      ↪  note thoroughly**. Focus especially on sections such as:\n   - Chief
      ↪  Complaint\n   - History of Present Illness (HPI)\n   - Assessment and
      ↪  Plan\n   - Discharge Diagnoses\n   - Hospital Course\n\n2. **Identify
      ↪  all FINAL diagnoses** mentioned either explicitly (e.g., in
      ↪  \Discharge Diagnoses") or inferable based on findings, procedures,
      ↪  and medications.\n\n3. **Map each diagnosis to the most specific and
      ↪  appropriate ICD-10 code**, ensuring clinical and coding accuracy. Use
      ↪  subcodes when possible (e.g., I21.11 instead of I21).\n\n4.
      ↪  **Exclude** symptoms, test results, procedures, and provisional
      ↪  diagnoses **unless** they are explicitly confirmed as final
      ↪  diagnoses.\n\n5. Ensure **no duplication** of codes. Each code should
      ↪  be unique and only listed once.\n\n======================\n OUTPUT
      ↪  FORMAT (STRICT)\n======================\n\nRespond ONLY in the
      ↪  following format:\n\nDiagnosis Summary:\n1. [ICD-10 Code] { [Full
      ↪  diagnosis name]\n2. [ICD-10 Code] { [Full diagnosis
      ↪  name]\n...\n\nExample:\n1. I21.11 { ST elevation (STEMI) myocardial
      ↪  infarction involving right coronary artery\n2. E11.9 { Type 2
      ↪  diabetes mellitus without complications\n3. I10 { Essential (primary)
      ↪  hypertension\n4. E78.5 { Hyperlipidemia, unspecified\n\nDo not
      ↪  provide explanations, introductions, or interpretations outside of
      ↪  the list. Be concise and precise.\n\n======================\n
      ↪  WARNINGS & ERROR HANDLING\n======================\n\n-  Do NOT
      ↪  include symptoms (e.g., chest pain) as diagnoses unless confirmed in
      ↪  the final diagnosis section.\n-  Do NOT suggest possible diagnoses.
      ↪  Include only what is **clinically confirmed or clearly
      ↪  documented**.\n-  Do NOT return placeholder or unspecified codes if a
      ↪  more specific code can be confidently assigned.\n-  If the note
      ↪  includes terms like \history of X", include it **only** if it's
      ↪  relevant to the current admission or explicitly stated as an active
      ↪  condition.\n-  Use your clinical understanding to interpret indirect
      ↪  mentions (e.g., insulin therapy implies active
      ↪  diabetes).\n\n======================\n PATIENT
      ↪  NOTE\n======================\n\nAdmission
      ↪  Note:\n{}\n\n\n\n======================\n NOW RETURN THE ICD-10
      ↪  DIAGNOSIS LIST BELOW\n======================\n\n"
    }
  ]
}
```

A.2.4 best_p_2

```
{
    "prompt_2": [
        {
            "role": "system",
            "content": "You are a medical coding specialist trained to extract
            ↪  clinical findings and assign ICD-10 codes strictly based on
            ↪  explicit textual evidence. Prioritize precision, adhere to
            ↪  coding guidelines, and ensure all conclusions are grounded
            ↪  solely in the provided content, emphasizing current conditions
            ↪  over historical ones."
        },
        {
            "role": "user",
            "content": "**Revised Prompt:**  \nCarefully read the clinical note
            ↪  and identify all FINAL diagnoses, including **all
            ↪  comorbidities** listed in the medical history or explicitly
            ↪  stated as active conditions. Assign the **most specific ICD-10
            ↪  code** possible for each diagnosis (e.g., use G47.33 for
            ↪  obstructive sleep apnea instead of G70.0). Exclude symptoms,
            ↪  test results, and procedures unless confirmed as diagnoses. For
            ↪  conditions like obesity or BMI, include relevant codes if
            ↪  indirectly indicated (e.g., medications like metformin or
            ↪  documentation of weight). Ensure no duplication of codes.
            ↪  \n\nExample:  \n1. I21.11 { ST elevation (STEMI) myocardial
            ↪  infarction involving right coronary artery  \n2. E11.9 { Type 2
            ↪  diabetes mellitus without complications  \n3. I10 { Essential
            ↪  (primary) hypertension  \n4. E78.5 { Hyperlipidemia,
            ↪  unspecified \n\nHere is the patient's admission note:\n\n{}"
        }
    ]
}
```

A.3 Agent Run documentation example

A.3.1 subval_4_report.txt

Please note that the table has been shortened in width to fit on the page. However, the original characters from the txt file should be shown as an example, not a LaTeX table.

```
## Data and Model ##
Dataset: subval_4.parquet, Shape: (500, 186)
Prompt set: one_long.json, #prompts: 20
Model: Qwen/Qwen3-32B

## p_data_config ##
Temperature: 0.6
Max tokens: 10000
Role with patient note: user
Enable thinking: True
Continue final message: False
Add generation prompt: True
Improve prompt: True

## Time last prompt##
Make prompts: 0h 0m 1s
Predictions: 0h 10m 37s
Evaluation: 0h 0m 1s
Script total: 0h 10m 39s

## Evaluation ##
Evaluation dataset:subval_4.parquet,length:500,metric:LONG_CODES, mean codes GT:12.9
| Prompt | Precision| Recall |   F1   |Prompt len|Output len|  Duration |Codes Pred|
-------------------------------------------------------------------------------------
|   0    |  0.2180  | 0.1732 | 0.1809 |  2802    |   6127   | 0h 8m 39s |    11    |
|   1    |  0.1808  | 0.1769 | 0.1669 |  1526    |   7819   | 0h 9m 15s |    13    |
|   2    |  0.1335  | 0.1779 | 0.1453 |  2674    |  10166   | 0h 11m 52s|    16    |
|   3    |  0.1959  | 0.1720 | 0.1714 |  3774    |   6903   | 0h 7m 38s |    11    |
```

4	0.1056	0.1898	0.1294	1667	11731	0h 15m 38s	22
5	0.1721	0.1712	0.1610	3836	6673	0h 8m 26s	12
6	0.1876	0.1769	0.1709	3310	6472	0h 8m 8s	12
7	0.1283	0.1822	0.1412	3200	11800	0h 15m 4s	19
8	0.1496	0.1859	0.1562	3684	7537	0h 9m 18s	15
9	0.1365	0.1824	0.1457	2131	11337	0h 13m 39s	17
10	0.2362	0.1446	0.1681	3148	5752	0h 6m 39s	7
11	0.1351	0.1768	0.1454	2265	11205	0h 12m 26s	16
12	0.2274	0.1686	0.1786	3499	6384	0h 8m 0s	10
13	0.1162	0.1868	0.1361	1946	10626	0h 12m 56s	19
14	0.2163	0.1732	0.1808	2571	5231	0h 6m 57s	10
15	0.1958	0.1711	0.1715	2207	8412	0h 10m 8s	12
16	0.1444	0.1637	0.1448	2458	8278	0h 9m 25s	14
17	0.1817	0.1933	0.1756	3199	7740	0h 9m 47s	14
18	0.1697	0.1743	0.1622	3394	7116	0h 9m 12s	13
19	0.1863	0.1737	0.1662	2724	9128	0h 10m 37s	13

Evaluation dataset:subval_4.parquet,length:500,metric:SHORT_CODES,mean codes GT:12.8

Prompt	Precision	Recall	F1	Prompt len	Output len	Duration	Codes Pred
0	0.2870	0.2177	0.2319	2802	6127	0h 8m 39s	11
1	0.2452	0.2302	0.2215	1526	7819	0h 9m 15s	13
2	0.1764	0.2282	0.1890	2674	10166	0h 11m 52s	16
3	0.2639	0.2204	0.2242	3774	6903	0h 7m 38s	11
4	0.1476	0.2480	0.1763	1667	11731	0h 15m 38s	22
5	0.2329	0.2212	0.2116	3836	6673	0h 8m 26s	12
6	0.2516	0.2282	0.2246	3310	6472	0h 8m 8s	12
7	0.1821	0.2400	0.1937	3200	11800	0h 15m 4s	19
8	0.2224	0.2509	0.2226	3684	7537	0h 9m 18s	15
9	0.1924	0.2432	0.2008	2131	11337	0h 13m 39s	17
10	0.3092	0.1804	0.2133	3148	5752	0h 6m 39s	7
11	0.1841	0.2300	0.1939	2265	11205	0h 12m 26s	16
12	0.3039	0.2144	0.2322	3499	6384	0h 8m 0s	10

13	0.1638	0.2473	0.1872	1946	10626	0h 12m 56s	19
14	0.2857	0.2176	0.2314	2571	5231	0h 6m 57s	10
15	0.2595	0.2191	0.2227	2207	8412	0h 10m 8s	12
16	0.1946	0.2118	0.1913	2458	8278	0h 9m 25s	14
17	0.2505	0.2503	0.2346	3199	7740	0h 9m 47s	14
18	0.2313	0.2260	0.2144	3394	7116	0h 9m 12s	13
19	0.2476	0.2219	0.2154	2724	9128	0h 10m 37s	13

A.3.2 subval_4_run_meta.json


```
{
  "run": {
    "date": "2025-10-18",
    "started_at": "2025-10-18T19:56:59Z",
    "finished_at": "2025-10-18T19:56:59Z",
    "status": "ok"
  },
  "data": {
    "name": "subval_4.parquet",
    "rows": 500,
    "shape": [
      500,
      186
    ]
  },
  "model": {
    "name": "Qwen/Qwen3-32B"
  },
  "prompts": {
    "set": "one_long.json",
    "num_prompts": 20
  },
  "p_data_config": {
    "path_data": "/workspace/data/mimic-iv/icd-10/hosp/",
    "name_data": "subval_4.parquet",
    "path_prompt_set": "/workspace/data/prompts_agent/",
    "name_prompts_data": "one_long.json",
```

```
    "name_model": "Qwen/Qwen3-32B",
    "temperature": 0.6,
    "max_tokens": 10000,
    "role_to_add_note": "user",
    "enable_thinking": true,
    "continue_final_message": false,
    "add_generation_prompt": true,
    "type_gpu": "b200",
    "path_output": "/workspace/data/output/"
  },
  "p_task_config": {
    "name_prompt_task_1": "one_task.json",
    "name_prompt_task_2": "one_task_2.json",
    "random_seed": 42,
    "max_iterations": 20,
    "improve_prompt": true
  },
  "timings_seconds": {
    "make_prompts": 0.676,
    "predictions": 637.178,
    "evaluation": 0.538,
    "total": 638.554
  },
  "timings_hms": {
    "make_prompts": "0h 0m 1s",
    "predictions": "0h 10m 37s",
    "evaluation": "0h 0m 1s",
    "total": "0h 10m 39s"
  },
  "artifacts": {}
}
```

A.3.3 pipeline_data.log

Please note that the logs have been shortened and only the first 3 iterations are shown.
In addition, the hadm_id has been obscured.

```
2025-10-18 16:20:33,658Z | INFO | Agent: Start iteration: 0
2025-10-18 16:20:40,787Z | INFO | Agent: Start pipeline data iteration: 0
2025-10-18 16:29:21,936Z | INFO | Agent: Finished pipeline data iteration: 0,
duration: 0h 8m 41s, F1_0_Long: 0.1809
2025-10-18 16:29:22,048Z | INFO | Agent: Start pipeline task iteration: 0
2025-10-18 16:29:22,634Z | INFO | Agent: Use seed: 42, data_idx: 15570, hadm_id: xxxx
2025-10-18 16:29:39,917Z | INFO | Agent: Finished pipeline task iteration: 0,
duration: 0h 0m 18s
2025-10-18 16:29:39,946Z | INFO | Agent: Start pipeline task_2 iteration: 0
2025-10-18 16:29:51,119Z | INFO | Agent: Finished pipeline task_2 iteration: 0,
duration: 0h 0m 11s
2025-10-18 16:29:51,148Z | INFO | Agent: Start iteration: 1
2025-10-18 16:29:51,148Z | INFO | Agent: Start pipeline data iteration: 1
2025-10-18 16:39:07,247Z | INFO | Agent: Finished pipeline data iteration: 1,
duration: 0h 9m 16s, F1_0_Long: 0.1669
2025-10-18 16:39:07,376Z | INFO | Agent: Start pipeline task iteration: 1
2025-10-18 16:39:07,907Z | INFO | Agent: Use seed: 43, data_idx: 17843, hadm_id: xxxx
2025-10-18 16:39:33,528Z | INFO | Agent: Finished pipeline task iteration: 1,
duration: 0h 0m 26s
2025-10-18 16:39:33,554Z | INFO | Agent: Start pipeline task_2 iteration: 1
2025-10-18 16:39:43,658Z | INFO | Agent: Finished pipeline task_2 iteration: 1,
duration: 0h 0m 10s
2025-10-18 16:39:43,682Z | INFO | Agent: Start iteration: 2
2025-10-18 16:39:43,683Z | INFO | Agent: Start pipeline data iteration: 2
2025-10-18 16:51:38,440Z | INFO | Agent: Finished pipeline data iteration: 2,
duration: 0h 11m 55s, F1_0_Long: 0.1453
2025-10-18 16:51:38,502Z | INFO | Agent: Start pipeline task iteration: 2
2025-10-18 16:51:39,042Z | INFO | Agent: Use seed: 44, data_idx: 4913, hadm_id: xxxx
2025-10-18 16:52:03,810Z | INFO | Agent: Finished pipeline task iteration: 2,
```

```
duration: 0h 0m 25s
2025-10-18 16:52:03,838Z | INFO | Agent: Start pipeline task_2 iteration: 2
2025-10-18 16:52:14,331Z | INFO | Agent: Finished pipeline task_2 iteration: 2,
duration: 0h 0m 10s
2025-10-18 16:52:14,358Z | INFO | Agent: Start iteration: 3
2025-10-18 16:52:14,358Z | INFO | Agent: Start pipeline data iteration: 3
2025-10-18 16:59:53,562Z | INFO | Agent: Finished pipeline data iteration: 3,
duration: 0h 7m 39s, F1_0_Long: 0.1714
2025-10-18 16:59:53,637Z | INFO | Agent: Start pipeline task iteration: 3
2025-10-18 16:59:54,275Z | INFO | Agent: Use seed: 45, data_idx: 1811, hadm_id: xxxx
2025-10-18 17:00:18,111Z | INFO | Agent: Finished pipeline task iteration: 3,
duration: 0h 0m 24s
2025-10-18 17:00:18,151Z | INFO | Agent: Start pipeline task_2 iteration: 3
2025-10-18 17:00:30,911Z | INFO | Agent: Finished pipeline task_2 iteration: 3,
duration: 0h 0m 13s
2025-10-18 17:00:30,936Z | INFO | Agent: Start iteration: 4
...
```