Mikael Törni

Large Language Model Prompt Engineering for Software Development

Bachelor's thesis

Bachelor of Engineering

Degree Programme in Game Programming

2025

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä	Mikael Törni
Työn nimi	Suurien kielimallien kehotesuunnittelu ohjelmistokehitystä varten
Toimeksiantaja	Xamk Game Studios
Vuosi	2025
Sivut	58 sivua
Työn ohjaaja	Niina Mässeli

TIIVISTELMÄ

Suuria kielimalleja voidaan käyttää tekstin tuottamiseen syöttämällä kehotteita, jotka ohjaavat niiden käyttäytymistä. Yksi merkittävimmistä käyttötavoista näille malleille on ohjelmistokehitys. Uusimmat kielimallit, kuten GPT-4 ja Claude 3.5 Sonnet ovat osoittautuneet hyvin päteviksi koodaajiksi. Suurten kielimallien huolimaton käyttö ohjelmistokehityksessä voi kuitenkin johtaa siihen, että projektin koodikannasta tulee hyvin vaikeasti ylläpidettävä.

Opinnäytetyö keskittyy ratkaisemaan tämän ongelman luomalla räätälöityjä kehotteita ohjelmistokehitystä varten. Ne auttavat ohjelmistokehitysprosessia kielimalleja hyödynnettäessä. Tämän avulla koodin struktuuri pysyy huomattavasti ylläpidettävämpänä. Sen lisäksi myös asianmukaiset virheenkorjauslinjat lisätään automaattisesti koodiin, jotta ongelmatilanteiden ilmestyessä sekä ihminen että kielimalli ymmärtävät paremmin, mikä koodissa menee pieleen.

Työn viitekehystä avataan tutkimalla neuroverkkojen historiaa ja että miten ne kehittyivät nykyajan kielimalleiksi. Työtä varten tutkitaan, miten nämä mallit toimivat, ja niiden avainkäsitteet selitetään omassa kappaleessaan. Työ myös tutkii monia eri tekniikoita, joita voidaan hyödyntää kehotesuunnittelussa. Lisäksi selvitetään, miten käyttäjä- ja järjestelmäkehotteet eroavat toisistaan käytännöllisesti. Tämän jälkeen parhaat tekniikat valitaan käytettäväksi työtä varten. Miten nämä tekniikat toimivat selitetään niiden omassa osiossaan auki. Eri kielimallien suorituskykytestejä tutkitaan, ja tämän tiedon avulla tehdään johtopäätös, mitä tullaan käyttämään työn toteuttamisessa.

Työn päätteeksi luodaan moniagenttijärjestelmä ohjelmistokehitystä varten. Tämä ohjelma ottaa syötteeksi käyttäjän pyynnön, tekee suunnitelman monella eri vaiheella, ja sen jälkeen tuottaa ohjelman käyttäen eri agentteja, jotka ovat erikoistunut suorittamaan oman tehtävänsä yhteistyössä muiden agenttien kanssa omassa ryhmäkeskustelussaan.

Kehotteet, jotka kehitettiin tutkimusta varten, arvioitiin käyttämällä tähän tarkoitettua testausjärjestelmää. Järjestelmäkehotteet, jotka kehitettiin tutkimusta varten, joita yksinkertaiset koodaustehtävä käyttäjä kehotteet hyödyntävät, ajettiin tutkimusta kehitetyn järjestelmäkehotteiden kanssa testauskehikon läpi. Kehotteiden toimivuus varmistettiin onnistuneilla testeillä, joten tutkimuksen päämäärät suoritettiin onnistuneesti.

Asiasanat: suuri kielimalli, kehotesuunnittelu, tokeni, väliaikaismuisti, päättelykyky, ohjelmistokehitys, ohjelmointirajapinta, käyttäjäkehote, järjestelmäkehote, moniagenttijärjestelmä

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author	Mikael Törni
Thesis title	Large Language Model Prompt Engineering for Software Development
Commissioned by	Xamk Game Studios
Time	2025
Pages	58 pages
Supervisor	Niina Mässeli

ABSTRACT

Large Language Models (LLMs) can be used to generate text content by entering prompts to direct their behaviour. One of the most prominent use cases for these models is software development. Since the release of models such as GPT-4 and Claude 3.5 Sonnet, LLMs have become very proficient in following instructions provided in natural language to write code for the user. However, using LLMs for software development can cause the codebase to become quickly unmaintainable.

The objective of this study was to create tailored prompts for software engineering purposes. These prompts help software engineering process when utilizing LLMs, by maintaining the readability of the codebase by writing better structured code. Additionally, debugging lines are automatically added to the code. This way, when issues arise with the code, both humans and the LLM can understand the problem better.

The theoretical framework of the thesis comprises the history of the neural networks and their evolution into LLMs. The operating principles and key concepts of these models are explored and explained. The study examines various prompt engineering techniques, including differences between user and system prompts. The techniques that were selected for this study are explained in more detail. Additionally, benchmark data was gathered on various LLMs to determine the most suitable ones for this study.

As a result of the study, a multi-agent system for software development was created. This program receives a user query, creates a plan with multiple subtasks, and produces a program utilizing multiple different agents that are specialized in accomplishing their own tasks in collaboration with the other agents in a group conversation.

The system prompts that were developed during the study went through an evaluation process within a testing framework, which were tested with simple user prompts related to various coding tasks. With the tests evaluated as a success, the prompts were proven to function as intended.

Keywords: large language model, prompt engineering, token, context window, reasoning, software development, application programming interface, user prompt, system prompt, multi-agent system

CONTENT

1 INTRODUCTION

Prompting plays an essential part in communicating and directing the behavior of Large Language Models (LLMs) that are based on machine learning technology. Prompts serve as inputs or queries that users can provide to the model to receive a certain response from it. Effective prompting is essential for achieving desired outcomes with LLMs. Prompt engineering requires selecting the right words, phrases, symbols, and formats to generate high-quality content. (Microsoft 2023).

The original GPT-4 model by OpenAI can outperform programming tutors on some tasks. Research has shown that in some situations, depending on the task, a human tutor was on par with or even beaten by the LLM. (Singla 2023). This shows that even the older models such as the GPT-4 can be used for programming. However, prompting still plays a crucial role in helping LLMs produce the desired result for certain programming tasks.

This study proposes a multi-agent system that, with proper prompt engineering, is capable of building software projects from the ground up. Developing software with LLMs, especially without proper prompting, can cause them to generate code that becomes unmaintainable over time. The system optimizes the readability of the code. This way, the project remains maintainable for both LLMs and humans alike.

2 RESEARCH DESIGN

This section discusses research design, including the purpose of the thesis, research questions, research approach methods, and gathering of the materials.

2.1 Purpose of this thesis

When LLMs are used to build software without proper prompt engineering, the code generated might not be what the user expects. Even if the code generated by the LLM executes successfully, the codebase can easily become difficult to maintain over the project's lifespan when using LLM-generated code. The purpose of the study is to examine various prompt

engineering techniques and address issues typically associated with prompt engineering. This allows LLMs to be used efficiently for software development while reducing the effect of their downsides.

The objective of the thesis is to develop a multi-agent system that utilizes Microsoft's AutoGen framework. By utilizing prompt engineering and Python tool development, multiple agents that have varying abilities can act as developers can form a team that can build projects from scratch.

In order to attempt to achieve the highest possible productivity gains with LLMs, the multi-agent system uses its own reasoning to form a comprehensive plan to achieve the goal set by a simple user query. This significantly reduces the amount of input needed from the user to perform the desired task.

While a multi-agent system may not be ideal for editing complex projects, a specialized prompt will be developed just for coding purposes. This prompt will be separated from the agentic system's parts such as tool call partitions, so that it can also be used on its own. Provided that the user of this prompt utilizes proper software such as Cursor to index the project's codebase (Cursor 2024a), the quality of the code should significantly improve, solving the original problem of the study.

2.2 Research questions

The following research questions were formed in order to achieve the objectives of this study:

1. How can prompting be utilized for software development in a way that allows the codebase to remain maintainable?
2. In which way can multi-agent systems be utilized to build software?
3. How can writing and reading logs be utilized for debugging with LLMs to solve problems in code?

2.3 Research approach

With limited experience that the author has in prompt engineering, the subject is first approached by studying different prompting techniques. This is followed

by experimentation to build task-specific prompts to answer the research questions.

Prompts go through an evaluation process to ensure that they produce the desired output. A framework called promptfoo is used for this purpose to measure the performance of the prompts. While it is impossible to achieve 100% accuracy even with the most modern models, especially for the most challenging software engineering purposes, the study aims to explore techniques and build the most optimal prompts.

Python version 3.12.4 is used during the implementation phase of the study. In order to ensure reproducibility of the study, the dependencies of packages are listed in the requirements.txt file located at the root of the multi_agentic_system_for_programming folder, located in a project that is linked in Appendix 1.

2.4 Gathering materials and performing analysis

The conceptual basis of this study is related to LLMs' functions in general and different prompt engineering techniques. Documentation such as Anthropic's and OpenAI's Application Programming Interface (API) are used to learn about different parameters and other constraints of the models. Microsoft's AutoGen documentation is examined regarding the utilization of multi-agent systems.

Primary data is obtained through testing and observation. The information gathered will be analyzed and then used for implementation to see how it further affects the generation capabilities of the model. Version control system Git will be utilized to save the state of the project over any proven improvement when developing prompts and the software related to them.

The process of prompt engineering will be done to iteratively improve functionality of the system towards higher accuracy on various user queries that will be designed to test the system. Promptfoo testing framework will be utilized to ensure reliability of the prompts with a model selected for each specific task. Each of the prompts will have their own assertion criteria which a

LLM evaluates if the final output of the prompt passes the test. Once an accuracy of 100% is achieved on a select prompt, a human evaluation will be also performed on the outputs of the LLM. After this, the prompt will be integrated into the multi-agent system for further testing.

3 FROM MACHINE LEARNING INTO LARGE LANGUAGE MODELS

3.1 Neural Networks

The Hopfield network model that utilizes recurrences was introduced in 1982 by John Hopfield. It was one of the earliest examples of Recurrent Neural Networks (RNNs). Instead of being error-based, a Hopfield network was an energy-based network because its properties derived from a global energy function. Instead of having feed-forward connections, each unit was recurrent and directionally connected to other units. All the units receive input and send outputs to every other connected unit, as shown in Figure 1. (Caceres 2020.)

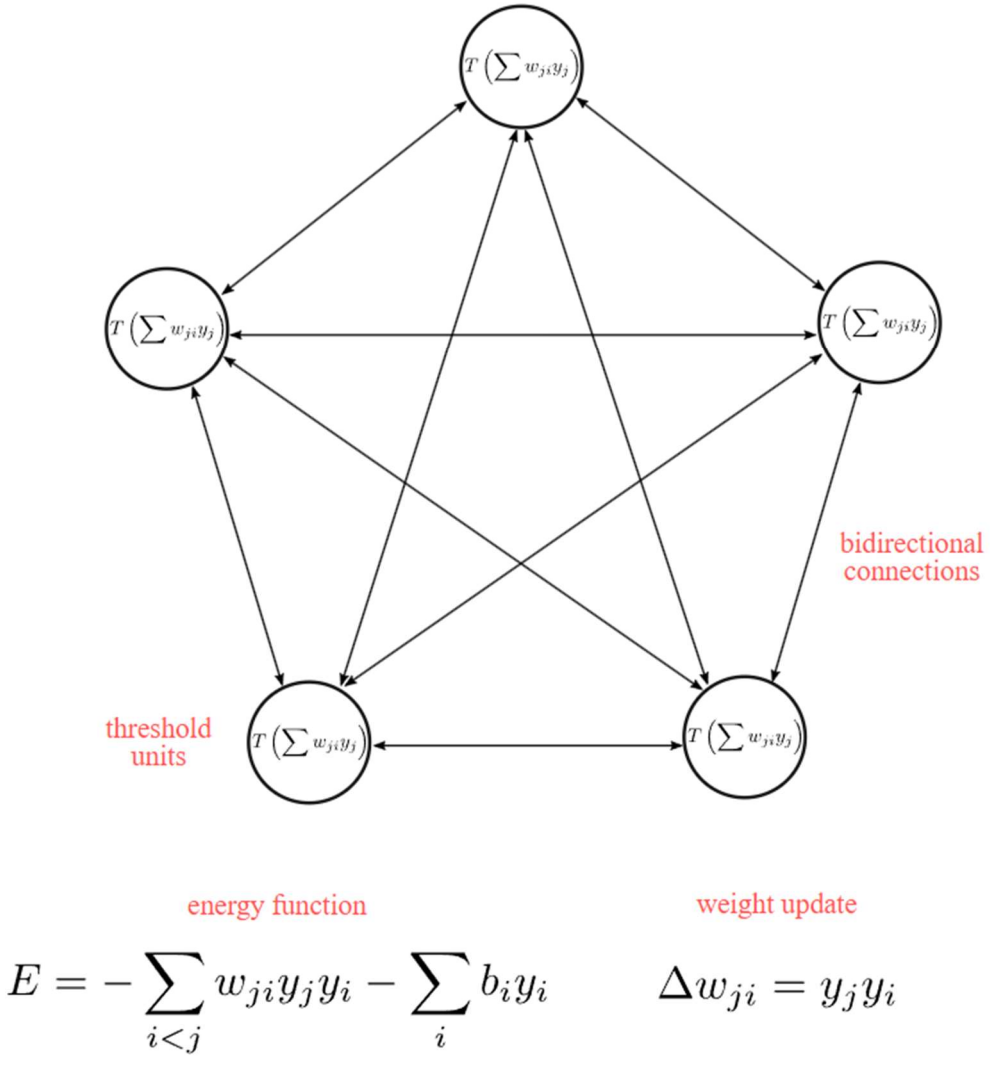


Figure 1. Hopfield Network.

Backpropagation is an algorithm that enables an efficient training of multi-layer neural networks. This is achieved by adjusting the weights of the network based on prediction errors. While concepts of backpropagation exist since the 1960s, only the paper “Learning Representations by Back-Propagating Errors” published in 1986 brought the usage of the algorithm into prominence. (codewave 2024.)

One of the authors of the aforementioned paper was Geoffrey Hinton (codewave 2024), who later, together with John Hopfield won the Nobel Prize in Physics “for foundational discoveries and inventions that enable machine learning with artificial neural networks”. (The Nobel Prize 2024.) Many experts around the world argued that the inventions by these pioneers were a significant step in the field of machine learning, which eventually led into development of LLMs in 2010s through advances in deep learning (SMC Spain 2024.)

3.2 Evolution from Neural Networks into a Transformer

Launched in 2013, the Word2vec model was a significant milestone for the development of various Natural Language Processing (NLP) systems. It computes continuous vector presentations of words from very large datasets which are measured in a word similarity task. This method allowed measuring relationships between different words with greater accuracy than ever before. (Mikolov 2013.)

Table 1. Word pair examples in SemanticSyntactic Word Relationship test set.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

The performance of Deep Neural Networks (DNNs) was improved in 2014 with the introduction of multilayered Long Short-Term Memory (LSTM) method. It is capable of mapping the input sequence of a neural network into a vector of a fixed dimensionality. This vector was then used to decode the target sequence by another LSTM. This greatly increased performance on tasks such as translation from English to French. (Sutskever 2014.) The introduction of multilayered LSTM was a significant improvement over the original LSTM method invented back in 1997 (Hochreiter 1997).

Unsupervised pre-training using unlabeled data was first introduced by Google's researchers Andrew M. Dai and Quoc V. Le in 2015. It involves approaches to predicting the next part of the sequence utilizing an autoencoder. These two algorithms form a "pretraining" step for a later supervised sequence learning algorithm. (Andrew Dai 2015.)

In 2017, researchers at Google proposed a paper named "Attention Is All You Need". This paper introduced a new simple network architecture for machine learning, called the Transformer. This architecture is based solely on the attention mechanisms, thus dispensing recurrence and convolutions entirely from the neural network. The Transformer outperformed the existing models

on translation tasks while requiring significantly less training time. (Vaswani 2017.)

3.3 Generative Pre-trained Transformer

In 2018, OpenAI introduced the first Generative Pre-trained Transformer, GPT-1. This model was built on Google's previous work, combining the existing ideas of Unsupervised pre-training and Transformer. (OpenAI 2018.) This section examines the training of GPTs and their operation in terms of handling tokens and explores the use of a window as a short-term memory for GPTs.

3.3.1 Token

Tokens are units that are utilized by LLMs to decompose text. These can be entire words, subword segments, or even individual characters. (Microsoft 2024a.) According to the documentation by Anthropic and OpenAI, 100 tokens approximately equal 75 words (Anthropic 2023a; OpenAI 2024a).

The LLM analyses different semantic relationships between tokens, such as frequency of usage together and context. Tokenization is the first step in training of the LLM. It uses these relationships and patterns to generate a sequence of output tokens. (Microsoft 2024a.)

During the training of the LLM, after tokenization, it assigns an ID to each unique token in the model. For example, in the sentence "I heard a dog bark loudly at a cat", tokenization method is used which can assign token IDs as follows, quoting straight from Microsoft's documentation:

- I (1)
- heard (2)
- a (3)
- dog (4)
- bark (5)
- loudly (6)
- at (7)
- a (the "a" token is already assigned an ID of 3)
- cat (8)

(Microsoft 2024a)

By assigning these IDs, a sequence of numbers can be formed out of the text. The example sentence represented above would be formed into a sequence of “[1, 2, 3, 4, 5, 6, 7, 3, 8]”. Another sentence, such as ”I heard a cat” would be represented as “[1, 2, 3, 8]”. Words such as ”meow” and ”run” could be assigned as the next IDs, 9 and 10 as the training continues. (Microsoft 2024a.)

Embeddings are multi-valued numeric vectors used to represent the semantic relationships between the token ID sequences. Each of the tokens gets assigned an embedding based on how commonly it is used together or in similar contexts with other tokens. After the model has finished training, it can calculate an embedding for a text that contains multiple tokens. The text is tokenized, and based on the learned embeddings, an overall embeddings value is calculated. (Microsoft 2024a.)

During generation of the text, vector value is predicted for the next token in the sequence by the LLM. The model works by calculating multiple vectors by using various elements of the previous tokens' embeddings. This way, the LLM can evaluate all the potential tokens from these vectors and select the most probable one to continue the sequence with. (Microsoft 2024a.)

LLMs often use token-based pricing in their APIs. The cost of each request is calculated depending on the number of input and output tokens. The price of the input and output tokens are usually different. (Microsoft 2024a.) For instance, the Claude 3.5 Sonnet’s price is set at $3 per million input tokens and $15 per million output tokens (Anthropic 2024a).

3.3.2 Data, training and scaling of the Large Language Models

Data preprocessing filters the data in terms of its quality. Data duplication can affect the model’s performance and increase data memorization, so deduplication of the data is a crucial process in the preprocessing phase. Private data collected through web sources, such as names, addresses and phone numbers are filtered. (Naveed 2024.)

Tokenization is an essential pre-processing step for the training process of a model. It is replicated on multiple devices where the data batch will get divided through a concept called Data Parallelism. At the end of each training iteration, the weights of the model are synchronized across all devices which are involved in its training. Tensor Parallelism shards the computation across all devices while Pipeline Parallelism shards the model layers within the training units. (Naveed 2024.)

Scaling laws refer to the optimal combination of dataset size, model parameters and computational resources. These will predict improvement in the model performance. In the past, it was shown that larger models are more important than big data for better model performance. (Naveed 2024.) However, in a paper released by two researchers at Massachusetts Institute of Technology in September 2024, it was proven that increasing the scale of the model does not necessarily enhance performance (Boopathy 2024).

In a paper released in June 2024, potential constraints were investigated on LLM scaling posed by the availability of public human generated text data. Findings indicated that between 2026 and 2032, if the current LLM development trends continued, models would be trained on datasets approximately equal in size to the available stock of public human text data. This implies that the scaling of the models might eventually stop completely as soon as all available data is processed and that the scaling may already be slowing down. Possible solutions to this problem could be synthetic data generation, data efficiency improvements, transferring learning from data rich domains. (Villalobos 2024.) Various AI labs have started signing deals to receive high-quality content that is not available for free on the public internet. For instance, in May 2024, OpenAI signed a deal with News Corp valued at $250 million, granting access to the content of various news organizations for the purposes of training their LLMs. (McDade 2024.)

3.3.3 Context window

The term context window refers to the amount of text that a LLM can reference when generating new text. It is represented as a “working memory” for the model. (Anthropic 2024b.) During the beginning of a chat session, the

context window is clear. The first data to be entered comprise the user and system prompt. After that, the tokens generated by the model will be stored in the context window for a later reference.

Having an expanded context window gives the ability for LLMs to handle complex tasks more efficiently, especially when considering a wider scope of data. These scenarios include tasks that are related to long-form content creation, in-depth analysis of extensive documents, and answering complex questions from multiple sources by synthesizing information. (Rossum 2024.)

The Needle in a Haystack Test is a way to evaluate an LLM's ability to recall information from its context window. The goal of this test is to assess the LLM's ability to identify and utilize specific pieces of information amidst a vast amount of data located in the context window. (Dhinakaran 2024.) Figure 2 shows recall accuracy of the original Claude 3.5 Sonnet model released in June:

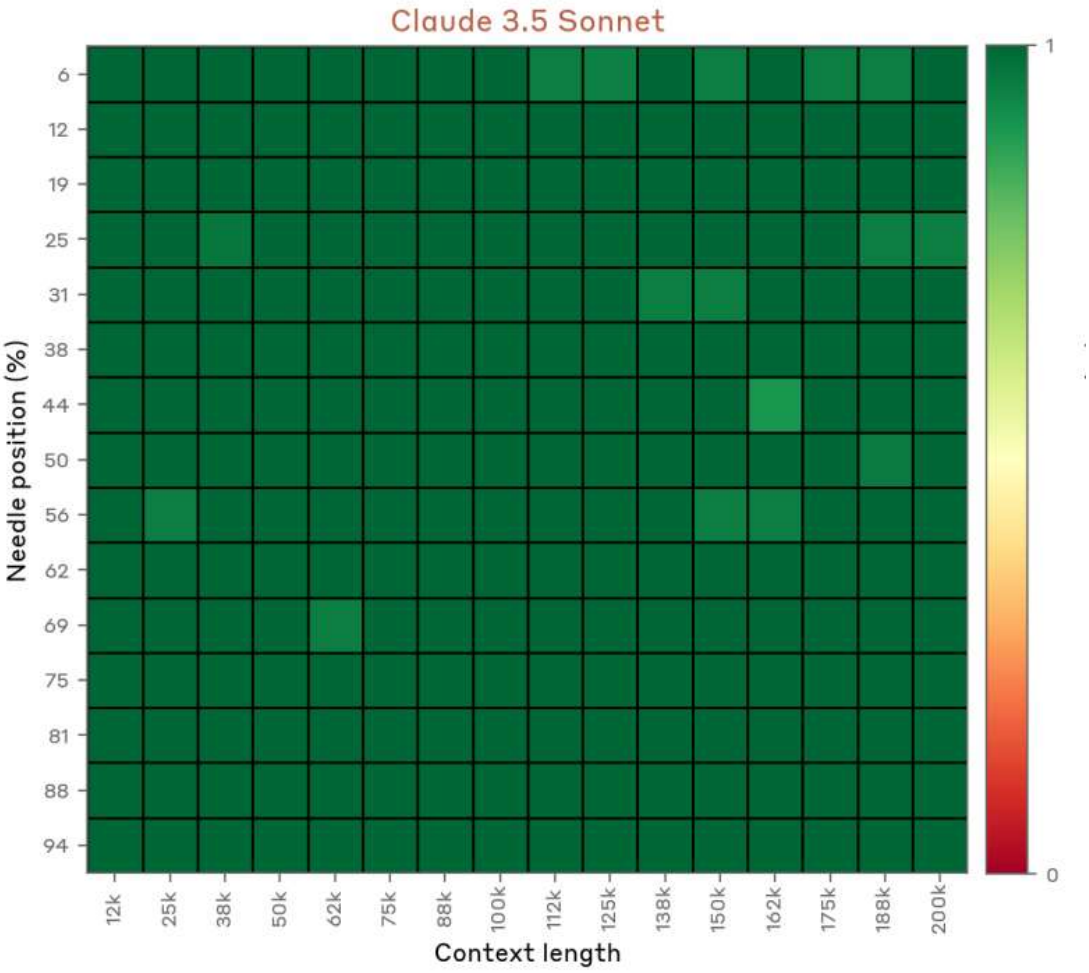


Figure 2. Needle In A Haystack plot for the original Claude 3.5 Sonnet, which achieves near perfect recall accuracy.

3.4 Scaling parameter count of the GPT architecture

OpenAI released its GPT-2 model with 774 million parameters in August 2019 (Askell 2019). Comparing this model to the GPT-1 which had 117 million parameters, this was quite a significant leap in terms of scalability (hgs 2023). In November 2019, OpenAI published a model with 1.5 billion parameters (OpenAI 2019). GPT-2 was enhanced through a training technique known as Modified Objective Training. This method refines the ability to generate coherent and contextually relevant responses by the model. (hgs 2023.)

The first research paper on GPT-3 was published In May 2020. Unlike the previous models published by OpenAI, GPT-3 was not released as open-source. The model was scaled up to 175 billion parameters, which is over 100 times what GPT-2 had. The model showed strong capabilities of handling data in zero-shot situations where it was not given examples for the task it had to complete. On one-shot and few-shot situations, where the model was given one or a few examples, it showed drastic improvements as the parameter count went up, as shown in Figure 3. (Brown 2020.)

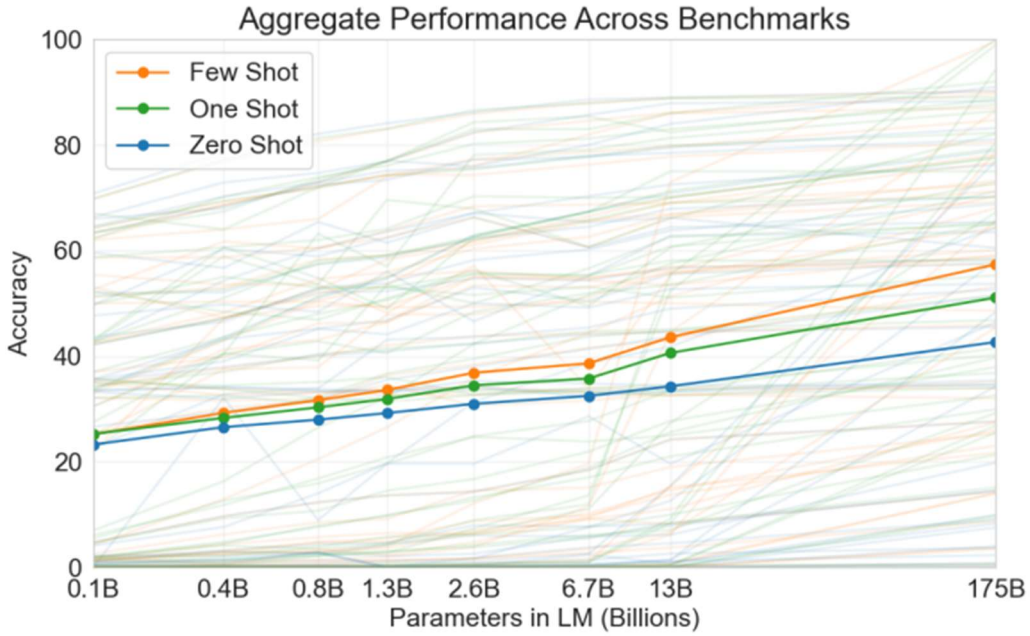


Figure 3. Aggregate performance across benchmarks for GPT-3 utilizing different parameter sizes with Zero, One, and Few Shot examples.

3.5 Reinforcement learning from human feedback and ChatGPT

In the paper published in March 2022 by OpenAI, the company stated that making language models larger does not inherently make them better at following the user's intent. Using GPT-3 with a dataset of rankings of the model outputs, InstructGPT was fine-tuned utilizing a technique called reinforcement learning from human feedback (RLHF). In human evaluations, the RLHF models with a parameter count of 1.3B were preferred to the output of GPT-3 with 175B parameter count. (Leike 2022.)

The goal of RLHF is to align models for responses that are meant for humans to be read. This is achieved by making the model follow a broad class of written instructions. The technique uses human preferences as a rewards signal during the fine-tuning process. (Leike 2022.)

In November 2022, OpenAI released a model called ChatGPT which is a sibling model of InstructGPT. It interacts with the user in a conversational way, allowing the user to consult the model with follow-up questions and making it possible for the AI to admit its mistakes, to challenge incorrect premises, and reject inappropriate requests. The model has slight differences compared to InstructGPT, namely in the data collection setup. ChatGPT is a fine-tuned model from the GPT-3.5 series. (OpenAI 2022.)

3.6 GPT-4 Series of models

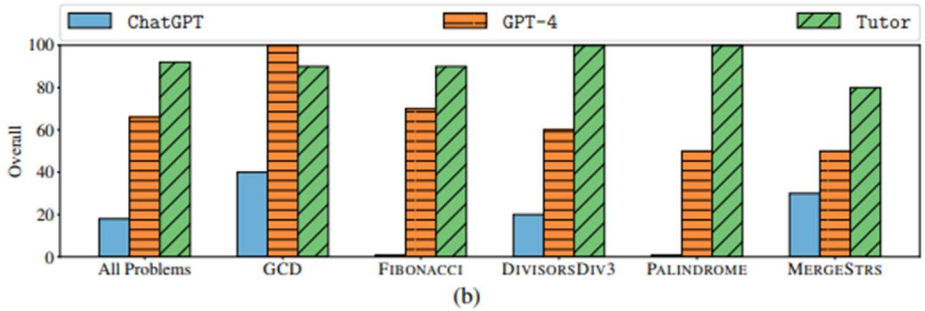
OpenAI released the GPT-4 model on 14 March 2023. The model is multimodal, meaning that it can accept images and text as the user input. While GPT-3.5 scores within the bottom 10% of bar exam takers, GPT-4 scores in the top 10%. Due to its broader general knowledge and problem-solving abilities, GPT-4 can tackle problems with greater accuracy. (OpenAI 2023.) The parameter count of this model has never been disclosed to the public by official sources.

It has been shown that GPT-4 outperforms ChatGPT 3.5 in most programming tasks. According to a study, in some tasks such as hint generation situations involving the greatest common divisor, GPT-4 even beat the human tutor, as

shown in Figure 4. (Singla 2023.)

Method	(Hint, Explanation)	Hint					Explanation
	Overall	HOverall	HCorrect	HInformative	HConceal	HComprehensible	ECorrect
ChatGPT	18.0 (6.0)	22.0 (6.0)	50.0 (10.0)	38.0 (6.0)	38.0 (10.0)	94.0 (2.0)	40.0 (8.0)
GPT-4	66.0 (10.0)	70.0 (10.0)	74.0 (10.0)	74.0 (10.0)	72.0 (8.0)	96.0 (4.0)	70.0 (10.0)
Tutor	92.0 (4.0)	92.0 (4.0)	94.0 (6.0)	94.0 (6.0)	94.0 (6.0)	98.0 (2.0)	94.0 (6.0)

(a)



(b)

Figure 4. ChatGPT, GPT-4 and a human tutor compared in a hint generation scenario.

The study did not specify which one was faster at completing each of the tasks. (Singla 2023.) However, according to the Artificial Analysis leaderboard which measures the speeds of various LLMs (Artificial Analysis 2024), the conclusion was made that LLMs outperform humans by a huge margin on writing tasks such as coding. This means that if the code works as intended, efficiency gains in completing some specific coding tasks with the help of LLMs will be tremendous. While this does not guarantee that the code is absolutely impeccable, this study focuses on mitigating impacts from cases such as these.

GPT-4o (“omni”) was first released on 13 May 2024. It offers similar performance as OpenAI’s earlier GPT-4 Turbo model on the processing of English and code, but providers significant improvement on the processing of text written in other languages than English. It is also much faster and 50% cheaper in the API. (OpenAI 2024b.)

Later that year, on July 18 GPT-4o Mini was released by OpenAI. It outperforms their older GPT-3.5 Turbo model while being approximately 17 times cheaper than the GPT-4o model released in August. GPT-4o Mini scores 82% on Massive Multitask Language Understanding (MMLU) evaluation benchmark. (OpenAI 2024c.)

3.7 Claude models by Anthropic

On 14 March 2023 Anthropic released their first Claude model, although with limited access on the API that could be requested. This model can help with use cases such as writing, summarization, Q&A, coding and more. It was released in two versions: Claude and Claude Instant, with each having a context window of 9000 tokens. The regular version of Claude offered State of the Art level performance, while the Instant model was a lighter, less expensive, and much faster option. (Anthropic 2023b.)

The first versions of Claude and Claude Instant were later upgraded to versions 1.1, and 1.2 while regular Claude got upgraded to version 1.3 also. These upgrades brought the context window up to 100 000 tokens. (Anthropic 2023a.) The models have been retired since the beginning of November 2024 and are no longer usable in Anthropic's API (Anthropic 2024c).

Claude 2 was released on 11 July 2023 by Anthropic. The model offered improved performance over its predecessor, Claude 1.3, getting 76.5% on the multiple-choice Bar exam, up from 73%, while providing the same 100K context window. Anthropic also released a beta chat experience on their site for users in US and UK, claude.ai. On 21 November Claude 2.1 came out with a context window of 200K tokens, with 2x of reduction in hallucination rates. With this release, Anthropic also introduced tools for operations such as retrieving information from private knowledge bases and searching over web sources. (Anthropic 2023d.)

On 4 March 2024 Anthropic released their next generation LLM, Claude 3. The release included three State of the Art models: Claude 3 Haiku, Claude 3 Sonnet, and Claude 3 Opus. Each successive model offers increasingly better performance, allowing users to select the model that fits their project's needs in terms of optimal balance of intelligence, speed and cost. (Anthropic 2024d.) Figure 5 demonstrates cost and intelligence levels of the different Claude 3

models.

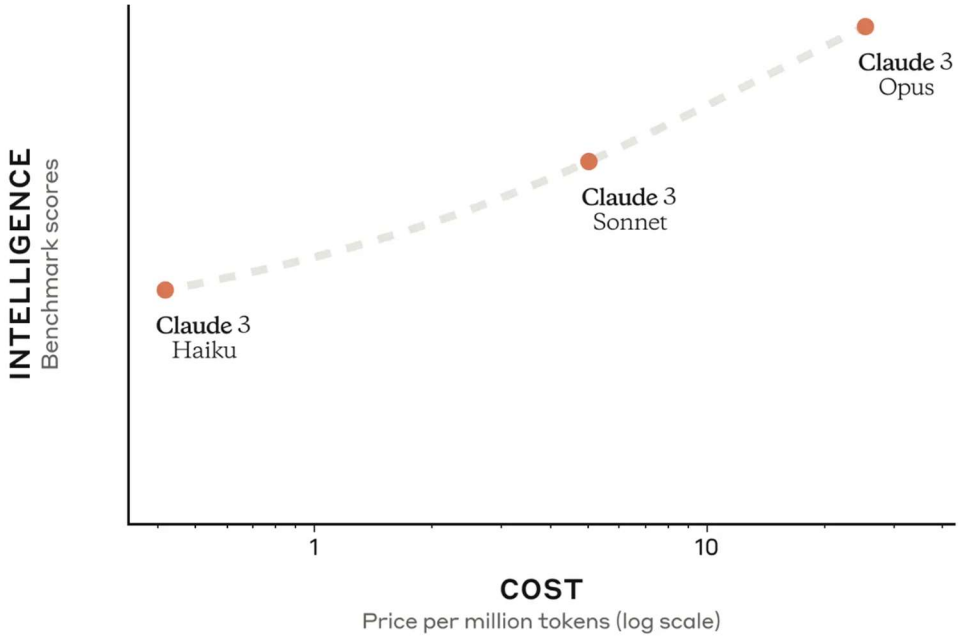


Figure 5. Claude 3 model family.

Comparing Anthropic's and OpenAI's State of the Art models at that time, Claude 3 Opus rivaled gpt-4-0125-preview model on many benchmarks, and even beat it at some of them (OpenAI 2024d), while Sonnet and Haiku were released with a good price-to-performance price ratio (Anthropic 2024d; OpenAI 2024e).

Table 2. Comparing Anthropic's Claude 3 models to the gpt-4-0125-preview model by OpenAI. Data for the table was combined from three different sources. Generated using ChatGPT.

Model	MMLU	GPQA	MATH	HumanEval	MGSM	DROP (F1, 3-shot)	Input Cost ($/1M tokens)	Output Cost ($/1M tokens)
gpt-4-0125-preview	85.4	41.4	64.5	86.6	85.1	81.5	$10.00	$30.00
Claude 3 Opus	86.8	50.4	60.1	84.9	90.7	83.1	$15.00	$75.00
Claude 3 Sonnet	79.0	40.4	43.1	73.0	83.5	78.9	$3.00	$15.00
Claude 3 Haiku	75.2	33.3	38.9	75.9	75.1	78.4	$0.25	$1.25

3.8 Claude 3.5 Sonnet's coding abilities

Anthropic released their Claude 3.5 Sonnet on 21 June 2024. With the speed and the cost of their older mid-tier model, the Claude 3.5 Sonnet beats Claude 3 Opus and other models by various AI companies at many benchmarks, including at coding. (Anthropic 2024e.)

	Claude 3.5 Sonnet	Claude 3 Opus	GPT-4o	Gemini 1.5 Pro	Llama-400b (early snapshot)
Graduate level reasoning *GPQA, Diamond*	**59.4%*** 0-shot CoT	50.4% 0-shot CoT	53.6% 0-shot CoT	—	—
Undergraduate level knowledge *MMLU*	**88.7%**** 5-shot 88.3% 0-shot CoT	86.8% 5-shot 85.7% 0-shot CoT	— 88.7% 0-shot CoT	85.9% 5-shot —	86.1% 5-shot —
Code *HumanEval*	**92.0%** 0-shot	84.9% 0-shot	90.2% 0-shot	84.1% 0-shot	84.1% 0-shot
Multilingual math *MGSM*	**91.6%** 0-shot CoT	90.7% 0-shot CoT	90.5% 0-shot CoT	87.5% 8-shot	—
Reasoning over text *DROP, F1 score*	**87.1** 3-shot	83.1 3-shot	83.4 3-shot	74.9 Variable shots	83.5 3-shot Pre-trained model
Mixed evaluations *BIG-Bench-Hard*	**93.1%** 3-shot CoT	86.8% 3-shot CoT	—	89.2% 3-shot CoT	85.3% 3-shot CoT Pre-trained model
Math problem-solving *MATH*	71.1% 0-shot CoT	60.1% 0-shot CoT	**76.6%** 0-shot CoT	67.7% 4-shot	57.8% 4-shot CoT
Grade school math *GSM8K*	**96.4%** 0-shot CoT	95.0% 0-shot CoT	—	90.8% 11-shot	94.1% 8-shot CoT

* Claude 3.5 Sonnet scores 67.2% on 5-shot CoT GPQA with maj@32
** Claude 3.5 Sonnet scores 90.4% on MMLU with 5-shot CoT prompting

Figure 6. Claude 3.5 Sonnet benchmarks compared to the other models.

On 22 October 2024, Anthropic released an upgraded Claude 3.5 Sonnet model, named claude-3-5-sonnet-20241022 in their API. While maintaining 200K context window, it has increased reasoning capabilities alongside scoring 93.7% at the HumanEval coding benchmark, up from 92.0% from the claude-3-5-sonnet-20240620 model released earlier in June. (Anthropic 2024f.)

According to unofficial benchmarks such as LiveBench, average coding score increased from 60.85 to 67.13 (LiveBench 2024). Aider LLM Leaderboards showed an increase from 77.4% to 84.2% on code editing tasks. On the code

refactoring leaderboard, the increase was even larger, from 64% to 92.1% compared to the old model. (aider 2024a.)

4 PROMPT ENGINEERING

4.1 System and user prompts

The LLM can be provided with a system prompt to give a role for it. This technique is known as role prompting. By providing the LLM with a system prompt, it can boost the model's performance significantly in complex scenarios such as legal analysis or financial modeling. While also improving its ability to focus, the LLM can generate content with a tailored tone, depending on its system prompt. (Anthropic 2024g.)

LLMs cannot properly operate with the system prompt only. User prompt acts as the query for the model to process to accomplish the main goal set by the user. (Anthropic 2024g.) User prompt can provide the exact context that makes the model focus on this one specific task which needs to be solved. With software that utilizes LLMs to act only as a chatbot, system prompts do not generally remain unchanged during the session. The model can be chained with multiple different user prompts to achieve the end goal of the task. (OpenAI 2024f.)

In this study, the coder agent utilizes a system prompt that optimizes its behavior for building software. This way, every user query that the model receives to achieve the task does not need to include things that make the agent build software in an optimal way. Instead, the user can focus on writing a proper query that describes the task without, for example, reminding the model to refactor the code into smaller functions on every single message.

4.2 Prompting techniques

Basic prompting is a method of directly querying the model without any additional context, examples or utilizing any of the other prompt techniques. While this method works for use cases such as building a simple snake game, it is not recommended for more complex tasks, due to increased risk of misinterpreting the user’s request due to lack of information.

Prompting techniques such as providing context and examples, while writing in clear and concise language are necessary for more advanced tasks. In some cases, methods such as Chain-of-Thought (CoT) have been proven to boost the LLM's performance by a huge margin, while providing information on the thought process of the model for debugging purposes.

New prompting techniques are continuously discovered, and they are often released with new publications of research papers. While this thesis does not introduce any new techniques, it utilizes the ones already discovered for software engineering purposes. This section describes various prompting techniques that were used in the study.

4.2.1 Writing in clear and concise language while giving enough context

It is important to design prompts to be clear and easy to understand. Ambiguous or confusing prompts can cause the model to give inaccurate or irrelevant responses. Prompts should be written to be concise and to the point, as lengthy prompts can be overwhelming and confusing for the model. (Dae-Kyoo 2023.)

A good prompt is aligned with the task at hand. This is achieved by using language and structure that clearly indicates the nature of the task to the model. Clearly stating the requirements that the model must follow is important for it to produce the correct content. (Shen 2024.)

Providing the model with the motivational context of the task gives it rationale for the problem at hand and explains why solving it is important. Explaining the structure and key ideas to the LLM is crucial. This describes the fundamental contextual ideas, that the prompt pattern provides to the LLM. (White 2023.)

4.2.2 Providing examples

Zero-shot means that no demonstration is given to the model at all during the prompting process. One-shot means that an example is given to the model,

but only once. Few-shot means multiple demonstrations are being given to the model in a prompt. (Brown 2020.)

It has been proven that while zero-shot performance increases steadily with model size, few-shot performance improves more rapidly. This demonstrates the ability for larger models to excel at in-context learning. (Brown 2020.) Figure 7 demonstrates how few-shot prompting works in its simplest form:

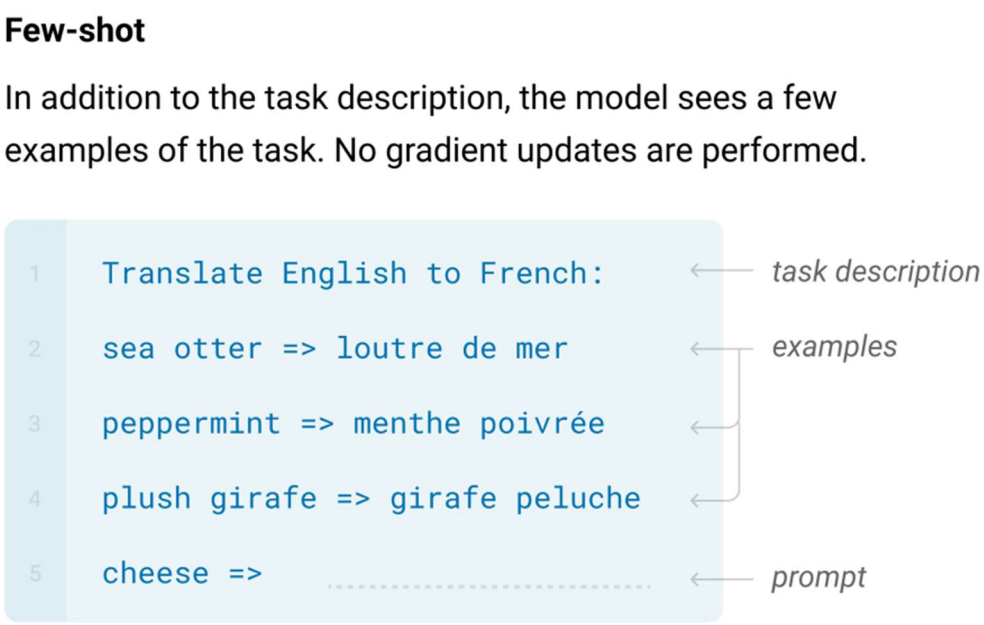


Figure 7. Few-shot translation prompt example.

While these kinds of examples for tasks such as translation do not need to be provided anymore for the modern models, specific few-shot prompting instructions for more complex tasks can still be very beneficial to produce the best output out of the model. This could be, for example, a demonstration of what the model will receive as its data, how it should be handled, and what the output should be. Multiple cases of this format should be provided to the model so it can succeed in the task with very high accuracy.

4.2.3 Prompt formats

According to Anthropic, XML tagging should be used as the default prompt format for their Claude series of the models. The tags provide clarity by clearly separating the prompt into parts, so it is more structured. This allows the writer to easily find, add, remove or modify parts of the prompts without having to

rewrite everything. The format also helps to reduce errors in the LLM by misinterpretation. (Anthropic 2024h.)

Table 3. Performance of different prompting formats.

Format	TabFact	HybridQA	SQA	Feverous	ToTTo
	Acc	Acc	Acc	Acc	BLEU-4
NL + Sep	70.26%	45.02%	70.41%	75.15%	**12.70%**
Markdown	68.40%	45.88%	66.59%	71.88%	8.57%
JSON	68.04%	42.40%	70.39%	73.84%	8.82%
XML	70.00%	47.20%	70.74%	73.14%	8.82%
HTML	**71.33%**	**47.29%**	**71.31%**	**75.20%**	12.30%
GPT-4 w/ HTML	78.40%	56.68%	75.35%	83.21%	20.12%

As shown in Table 3, HTML provided the highest scores on these tests. However, as it is very tedious to write compared to formats such as Markdown and XML, the small gains might not be worth it.

Writing prompts in Markdown language can be greatly beneficial in terms of formatting and ease of writing for the text. Markdown language formatting includes usage of markers such as H1 (#), H2 (##), bullet points (-), creating more organized text. While this format is the most readable to a human, benchmarks such as the one above, show that the format performs the worst, even outpaced by the natural language with separators. (Sui 2024.)

Although not confirmed by OpenAI explicitly in their documentation, according to various unofficial sources, GPT series models seem to prefer markdown as their prompt format (krishnanrohit 2024; elder_plinius 2024; Every 2024). In addition, OpenAI's cookbook has examples that utilize markdown language as well (OpenAI 2024g).

4.2.4 Chain-of-Thought

The goal of Chain-of-Thought (CoT) prompting is to make the LLM generate a set of intermediate reasoning steps that will produce the definitive answer for a given problem (Wei 2023). This is demonstrated in Figure 8 below.

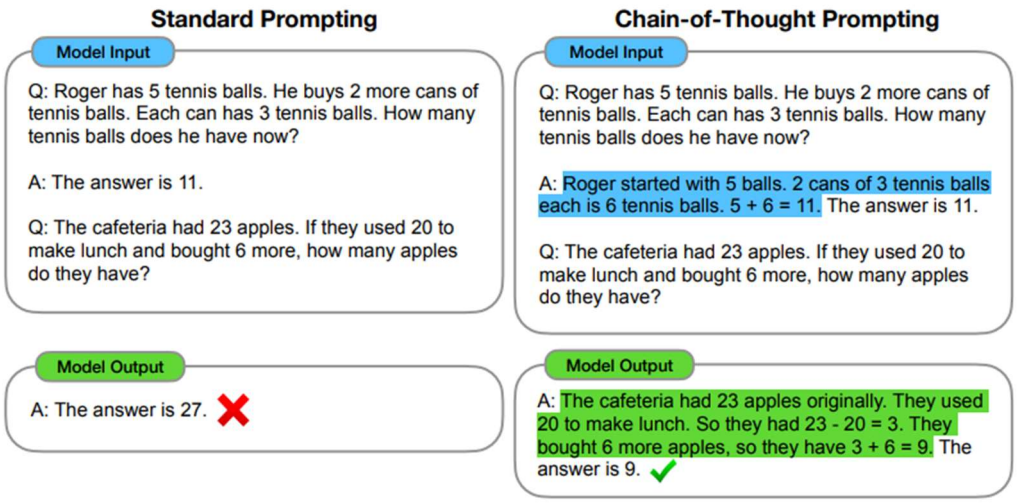


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Figure 8. Chain-of-Thought prompting example.

In order to get the model to produce this output, it must be instructed to think step-by-step (Wei 2023). It has been observed that sometimes modern LLMs can generate reasoning steps on their own without explicit prompting, especially when they determine that given prompt is a logic problem to solve.

According to a new paper released in late October 2024, it was found out that CoT only substantially helps with problems that require mathematical, logical or algorithmic reasoning. While CoT primarily helps with the execution step that performs the computation and symbolic manipulation, it falls short on what LLMs can do with tool augmentation. Models prompted utilizing CoT can generate executable formal solution plans and then execute those plans better than just direct answering. Using the LLM to generate a solution plan and then proceed to using an external symbolic solver to solve it outperforms CoT in both steps of these tasks. (Sprague 2024.)

For more advanced prompt engineering, using CoT might be necessary. If the answer that the model produces is incorrect, its steps show how it ended up in the conclusion will be visible. This way, the prompt will be easier to debug by observing where the reasoning goes wrong during the thinking steps. After the reasoning fault has been spotted, prompt can be adjusted to be run again with the fixed instructions.

4.2.5 Meta-Prompting

Meta-prompting is a scaffolding technique designed to enhance the functionality of LLMs by transforming a single model into a multi-faceted conductor which excels at managing and integrating multiple independent queries. Meta-prompting guides the LLM to break down complex tasks into smaller, more manageable subtasks by employing high-level instructions. Tailored instruction instances that are distinct from each other will then handle each of these subtasks. (Suzgun 2024.)

“Expert” instances will then handle these subtasks. Each of these have their own tailored instructions for the tasks that they need to execute. A single LLM is made to act as an orchestrator by this collaborative prompting approach. (Suzgun 2024.) While the paper mentions that these instances are operated under a single LLM, in the study the meta-prompting method was used with a multi-agent system that runs multiple instances different models that have their own system prompts.

By the task-agnostic and zero-shot nature of meta-prompting, user interaction is greatly simplified by obviating the need for detailed, task-specific instructions. Research by Suzgun and Kalai demonstrated seamless integration of external tools such as the Python interpreter into the meta-prompting framework. The technique was tested with the GPT-4 while augmenting it with the Python interpreter, and results surpassed other prompting methods (Suzgun 2024), as shown in Figure 9.

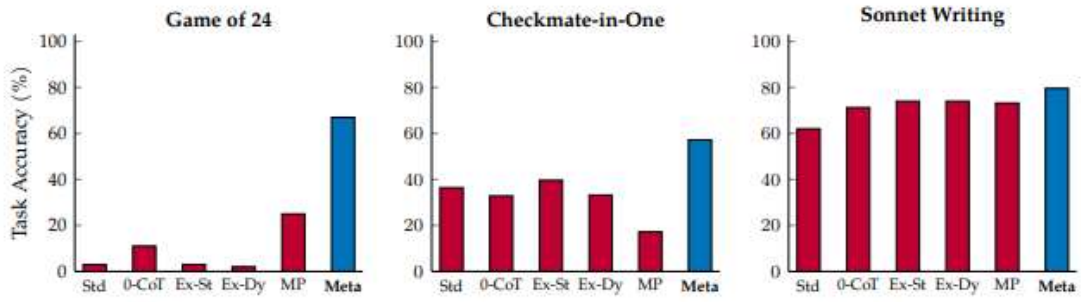


Figure 9. Meta-prompting performance compared to other prompting techniques.

5 LARGE LANGUAGE MODEL USAGE IN SOFTWARE ENGINEERING

This section explains how to choose the right model for each of their tasks and setting the right API parameters for optimal performance in the context of software engineering. It also covers the model limitations and different types of LLM applications for software engineering.

5.1 Choosing the right model for coding

LiveBench and aider LLM code benchmark measure performance of different LLMs. They utilize various Python code snippets to evaluate how LLMs perform in various coding tasks. (LiveBench 2024; aider 2024b.)

According to LiveBench, the new Claude 3.5 Sonnet was the highest scoring LLM on the “Coding Average” test set (LiveBench 2024). This is shown in Table 4.

Table 4. The new Claude 3.5 Sonnet compared to other models on the LiveBench site.

Model	Global Average	Reasoning Average	Coding Average	Mathematics Average	Data Analysis Average	Language Average	IF Average
claude-3-5-sonnet-20241022	60.33	58.67	67.13	51.28	52.78	58.09	74.05
claude-3-5-sonnet-20240620	59.80	58.67	60.85	53.32	56.74	56.94	72.30
qwen2.5-coder-32b-instruct	48.15	47.33	56.85	45.95	43.44	27.02	68.31
dracarys2-72b-instruct	51.81	42.67	56.64	50.58	49.14	37.15	74.69
qwen2.5-72b-instruct	52.89	46.00	56.56	52.36	48.79	38.12	75.50
gemini-exp-1114	56.97	54.67	52.36	54.92	57.49	44.92	77.45
gpt-4o-2024-08-06	56.03	54.67	51.44	48.21	52.89	54.37	74.58

Table 5. Aider's code editing benchmark shows how the new Claude 3.5 sonnet outperforms other models.

Code editing leaderboard

Aider's code editing benchmark asks the LLM to edit python source files to complete 133 small coding exercises from Exercism. This measures the LLM's coding ability, and whether it can write new code that integrates into existing code. The model also has to successfully apply all its changes to the source file without human intervention.

Model	Percent completed correctly	Percent using correct edit format	Command	Edit format
claude-3-5-sonnet-20241022	84.2%	99.2%	`aider --model anthropic/claude-3-5-sonnet-20241022`	diff
o1-preview	79.7%	93.2%	`aider --model o1-preview`	diff
claude-3.5-sonnet-20240620	77.4%	99.2%	`aider --model claude-3.5-sonnet-20240620`	diff
claude-3-5-haiku-20241022	75.2%	95.5%	`aider --model anthropic/claude-3-5-haiku-20241022`	diff
Qwen2.5-Coder-32B-Instruct (whole)	73.7%	100.0%	`aider --model openai/Qwen2.5-Coder-32B-Instruct`	whole
DeepSeek Coder V2 0724 (deprecated)	72.9%	97.7%	`aider --model deepseek/deepseek-coder`	diff
gpt-4o-2024-05-13	72.9%	96.2%	`aider`	diff

Table 6. The new Claude 3.5 Sonnet's performance in aider code refactoring benchmark.

Code refactoring leaderboard

Aider's refactoring benchmark asks the LLM to refactor 89 large methods from large python classes. This is a more challenging benchmark, which tests the model's ability to output long chunks of code without skipping sections or making mistakes. It was developed to provoke and measure GPT-4 Turbo's "lazy coding" habit.

The refactoring benchmark requires a large context window to work with large source files. Therefore, results are available for fewer models.

Model	Percent completed correctly	Percent using correct edit format	Command	Edit format
claude-3-5-sonnet-20241022	92.1%	91.0%	`aider --sonnet`	diff
o1-preview	75.3%	57.3%	`aider --model o1-preview`	diff
claude-3-opus-20240229	72.3%	79.5%	`aider --opus`	diff
claude-3.5-sonnet-20240620	64.0%	76.4%	`aider --sonnet`	diff
gpt-4o	62.9%	53.9%	`aider`	diff
gpt-4-1106-preview	50.6%	39.3%	`aider --model gpt-4-1106-preview`	udiff
gpt-4o-2024-08-06	49.4%	89.9%	`aider --model openai/gpt-4o-2024-08-06`	diff

While in the aider LLM leaderboards on the "Percent completed correctly" column, Claude 3.5 Sonnet outperformed any other LLM, in both code editing and refactoring leaderboards (aider 2024a.), as was shown in Table 5 and 6.

Due to its coding performance, as proven by these benchmarks, the model was chosen to be used in the study to reduce cases where the LLM was the cause of not being able to complete the programming task.

5.2 Controlling model behavior with temperature and max tokens parameters

This section explains how temperature and max tokens parameter can be used to control the model behavior to generate better output. Parameters such as top_p and top_k are available in the API (Anthropic 2024i) but are irrelevant to be optimized with the scope of this study.

Randomness of the LLMs output can be controlled by a parameter called temperature. Lower value of this makes the model more deterministic, while higher values will make it more probabilistic. (Anthropic 2024i.) While this value can range from 0 to 2, for this study a temperature of 0 was chosen to guarantee that the model stays deterministic during the completion of its tasks.

Max tokens parameter defines how many tokens can be generated in a chat completion. This limit is 16,384 tokens for the gpt-4o-mini-2024-07-18 model (OpenAI 2024h), and 8192 for the claude-3-5-sonnet-20241022 model (Anthropic 2024a). In order to maximize the tokens that the model can generate on one go, it was set to maximum for each of the models utilized in this study.

5.3 Understanding the model and API limitations

As mentioned in the previous chapter, the LLM can only output a certain number of tokens for one request. While it is unlikely that with simple prompts this limit will be reached, the model can be made to generate as many tokens as possible with prompt engineering. Thus, exceptionally large code generation tasks will have to be split into parts to ensure that this limit is not reached.

Having a larger context window means that the LLM can store more of the codebase in its short-term memory, providing better generation capabilities and answers for questions regarding the project. The context window size is 128 000 tokens for the gpt-4o-mini-2024-07-18 model (OpenAI 2024h), and 200 000 for the claude-3-5-sonnet-20241022 model (Anthropic 2024a).

Although context windows are reaching hundreds of thousands (Anthropic 2024a) or even millions of tokens at the moment of publication of the thesis (Google 2024), they can be still a limitation for huge codebases. If the limit is reached, it is wise to upload only the needed context to complete the coding task required. The study aims to implement a system that continuously

refactors the codebase. This way, the LLM will understand it easier, and is able to operate better with lower context window limits.

Hallucinations in LLMs happen when the models generate content that is false, made up or inconsistent according to their training data. This happens when the model attempts to create coherent responses by filling the gaps with plausible sounding but incorrect information. (Banerjee 2024.) In terms of code generation, hallucination can make the model produce code that just outright does not work. If the data is missing from its dataset, with the context given, the model will try to guess to the best of its ability to fill in the gaps. In this case, the code might still work if the model guesses right.

Rate limits are set in place to an API to manage capacity and protect it against misuse. On Anthropic's API, limits are defined by usage tier where each tier is associated with a different limit. These are defined by the credit purchase amounts by the customer. For enterprise use cases, the API provider can be contacted to unlock higher limits. (Anthropic 2024j.)

5.4 AI Copilots, Coding Assistants and Software Engineering tools

In the video, "AI Coding Tool Breakdown: AI Copilots vs AI Coding Assistants vs AI Software Engineers", IndyDevDan compared different types of tools for software engineering that utilize LLMs. These tools are AI Copilots, Coding Assistants and Software Engineers. While each of them has their own advantages and disadvantages, autonomy of the system increases towards the Software Engineer multi-agent system. (IndyDevDan 2024.) This is shown in Figure 10.

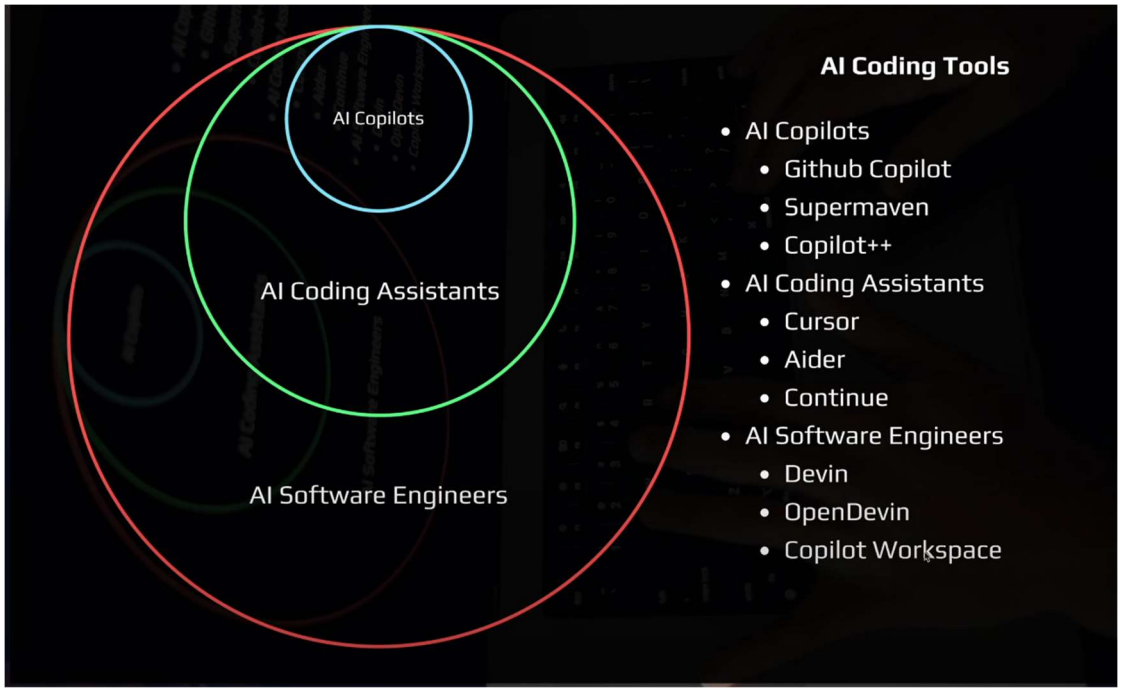


Figure 10. AI Copilots, Coding Assistants, and Software Engineers.

As the developer writes code on to the editor, GitHub Copilot suggests code completions. Copilot is available as a plugin in IDEs such as VS Code. Other alternative Copilot tools are available such as Tabnine and OpenAI's Codex. (Wermelinger 2024.) Prompt engineering for Copilots was not included in this study. While it is possible to write natural language within a comment to make GitHub Copilot suggest code to accomplish the goal (GitHub 2024), this is not very practical, especially with editors such as Cursor, since they edit code by pressing a hotkey (Cursor 2024b). This method can utilize a specific, more powerful LLM that excels at code generation tasks instead of the autocomplete models, such as what Cursor Tab uses (Fridman 2024).

Especially for larger codebases, utilizing software like Cursor might be necessary for development purposes. This program has features such as loading the relevant parts of the codebase onto the LLM's context window, so it can produce better answers for the code generation. Additionally, to the chat feature with context that the program provides, other features such as inline code generation and edits are available too. (Cursor 2024c.) These, and other features included in the software, will significantly increase the effectiveness of using LLMs for development purposes.

While regular LLM usage can accomplish basic coding tasks, multi-agent system or “AI Software Engineers” can build entire projects from scratch or edit existing ones. These systems can have different agents such as coder and project manager, while each of them has their own system prompts that make them excel at the specific task they are delegated to.

6 CODER SYSTEM PROMPT FOR SOFTWARE DEVELOPMENT

The purpose of the coder system prompt is to optimize the process of writing the code with LLMs. The new Claude 3.5 Sonnet Model for executing this task was chosen. Prompts for Anthropic’s models were written in the XML format for optimal performance.

Due to its extraordinary coding abilities, out of the models that the author had access to, the new Claude 3.5 Sonnet was chosen to be used in the study for the coder prompt. This model was used for all the research in terms of prompting research for programming.

Before writing the code, model performs step-by-step reasoning using CoT by writing a numbered list of what it should do. After this, the model will write the code, based on that reasoning chain. Additionally, the requirements.txt file will be generated if the project has any dependencies on external python libraries. Instructions to run the program will be presented last.

The LLM was instructed to write clean code that is functional and easy to understand. With this paradigm, the functions of the code are split into many manageable ones. Each of the functions have a docstring comment on top, explaining what they do, what parameters they have, and which data they return upon calling them.

Additionally, each of the generated files have a docstring comment explaining what their purpose is and what functions they have in them. It is also explained how to run the program via command line interface provided that the file is entry point of it. This way, as the codebase is built with this system prompt stays clean and organized, while making it easier for humans and LLMs to understand.

Error handling is included by default in the function while using this prompt to generate the code. Additionally, each of the functions will have debugging logs in them, what parameters they are taking in and what is in the output before returning the result. Utilizing the inbuilt logging library in Python, these debug logs will print their severity level, and in which function they are executed. This way the model can recognize easier where error is happening. Because of this, debugging with LLMs becomes way easier when problems arise with the computer-generated code.

This prompt does not include writing test cases, as it was only added into the multi-agent system as an instruction by the project manager agent. In order to avoid overengineering prompts to make them stay focused on their initial task, such as this one was supposed to write code only, it is possible that separate prompt for this use case could be created. However, in the current implementation the coder prompt just completes the subtask assigned to it in the multi-agent system.

The final prompt was tested in the Cursor IDE and confirmed to be working for building software in Python. The agent system's codebase was refactored using this prompt. Other than unintentionally altering some agent parameters, the prompt flawlessly inserted appropriate docstrings and logging into each file. Without this prompt, IDE users would need to prompt the LLM each time to make these changes or implement them separately.

7 UTILIZING PROMPTFOO FOR THE EVALUATION OF THE PROMPTS

7.1 Promptfoo testing framework

Promptfoo is an open-source command line interface application and library for evaluating LLM applications. While it also can be used for LLM security scans, the framework was used to build reliable prompts to be used in this study. Promptfoo produces matrix views that lets the user evaluate outputs

across many prompts. (promptfoo 2024a).

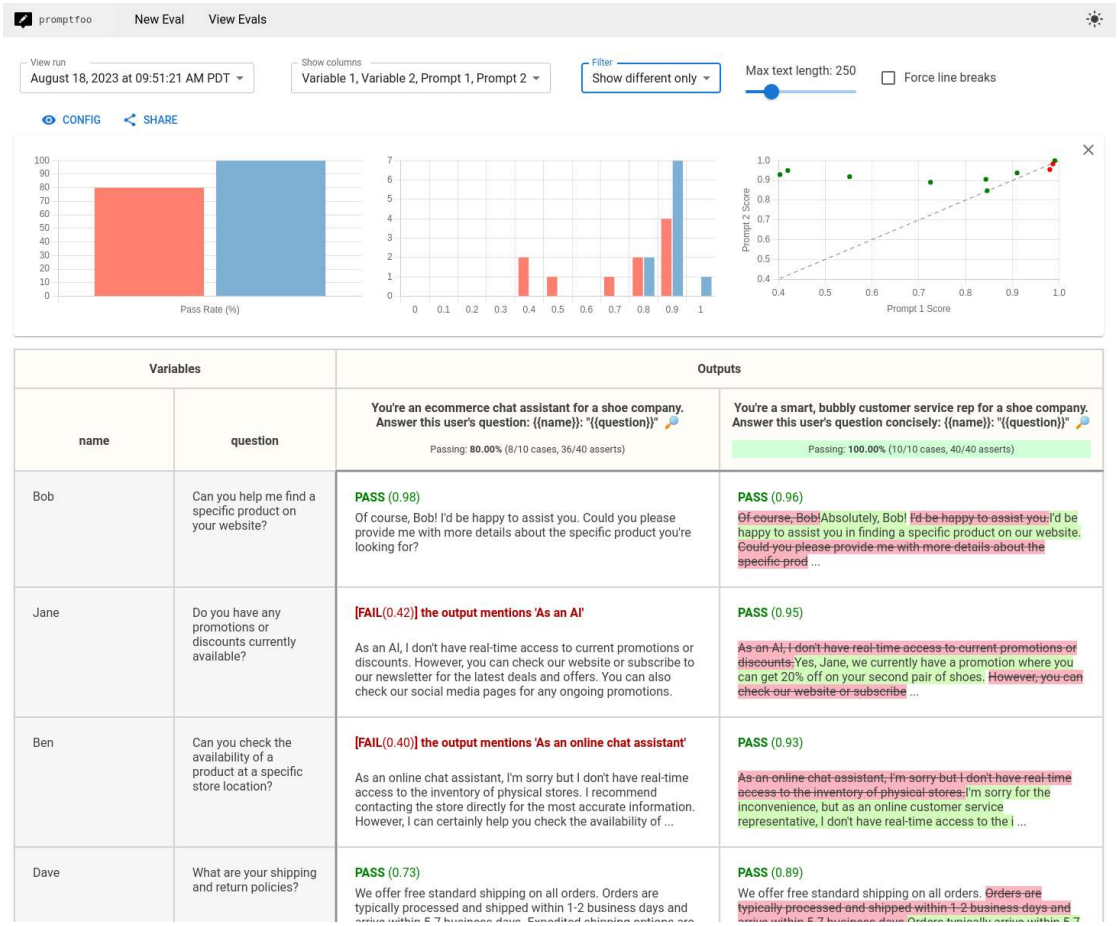


Figure 11. Promptfoo test run evaluation report.

Many different LLMs can be configured to be tested with the framework. YAML language configuration files must be created specifying the models. (promptfoo 2024b.). Prompts and assertion criteria to be used in the test variables can be specified also, by adding their file location (promptfoo 2024c). These variables can also be accessed by reading them from a vector database with a Python script (promptfoo 2024d).

The evaluation metrics can be deterministic, meaning that for example if a certain word appears in the output of a test case, the test could fail if a threshold defined in the config is reached. LLM grading of the output can also be used, with the possibility of having multiple different graders doing this task. (promptfoo 2024e.)

7.2 Evaluation of the project's prompts

Ten different user prompts that contained simple Python programming challenges were written to test the different prompts developed during the study. For the evaluation configuration, it was set in a way that one test equals running with these prompts. During the development of the system prompts, this was very beneficial since some user prompts would constantly cause issues.

Since LLMs are probabilistic when producing output, it is important to test that the prompts do not fail their evaluations, especially if they are meant for production grade systems. The prompts developed during the study were tested to ensure that they will pass their evaluations. Assertions were written from the prompts to help reach the 100% evaluation score by the grader model. After the max accuracy was reached according to the grading model, final evaluation outputs were checked manually by the author to ensure that they completed the task for sure.

Additionally, the user prompts intended for building various programs were tested without the coder system prompt. However, the assertion prompt, which was designed specifically as a system prompt for the coder, was utilized in this test set. As expected, all tests failed because they didn't meet the criteria of the assertion prompt intended exclusively for the coder system prompt. This confirmed that the assertion prompt was functioning correctly.

With custom scripts made for the project, code can be extracted from the JSON files that the testing framework produces. Since the coder and project manager prompts produce more consistent LLM outputs, these scripts will only succeed on the tests without such formatting issues. For the project manager agent prompt, an additional script was created to validate the XML output format, so that it reliably works with the agentic system. All the instructions for reproducing these steps are found in the instructions_for_running_evals.md file.

Comparing the results of the two different test sets, the one with the coder system prompt cost more than twice as much as the set without it. This was

because the coder prompt consistently wrote significantly more content to the file across the hundred tests in each test set. Since input tokens are much cheaper, even though their count was nearly five times higher in the system prompt test set, the costs didn't scale up proportionally.

During testing, rate limits were hit utilizing Anthropic's API on the new Claude 3.5 Sonnet model, due to enormous amounts of tokens produced at once, causing some of the evaluations to fail completely. Limits of the API at the first tier were at 50 requests, 40,000 input and 8000 output tokens per minute (Anthropic 2024j). Because of this, the author had to make sure that the evaluations are being run one at a time with no concurrency. With these limits, when running more than 25 tests in a row, even delay between the tests had to be added to make sure that the evaluations did not fail because of the API limitations. OpenAI's models were running fine with a batch of hundreds of tests with a concurrency of 15 without delay. Due to some internal errors in the Anthropic API, some final tests of the coding prompt failed because of this. In the end it was judged that one test failing didn't disprove the functionality of the tested prompts, since the one test set had 100 cases in them. All the information regarding running the evaluations for the project is included in the documentation of the files in the GitHub repository which is linked in Appendix 1.

8 MULTI-AGENT SYSTEM FOR SOFTWARE DEVELOPMENT

8.1 Microsoft's AutoGen framework

AutoGen is a unified multi-agent conversation framework which offers high-level abstraction using foundation models. It enables the creation of capable, customizable and conversable agents which integrate LLMs, tools and humans with an automated agent chat. With this, the chat can be automated amongst multiple capable agents, making them collectively perform tasks autonomously or with human feedback. (Microsoft 2024b.)

Every agent in the AutoGen framework is Conversable, meaning that they can send and receive messages from other agents to initiate or continue conversations. The following classes can be inherited from the ConversableAgent:

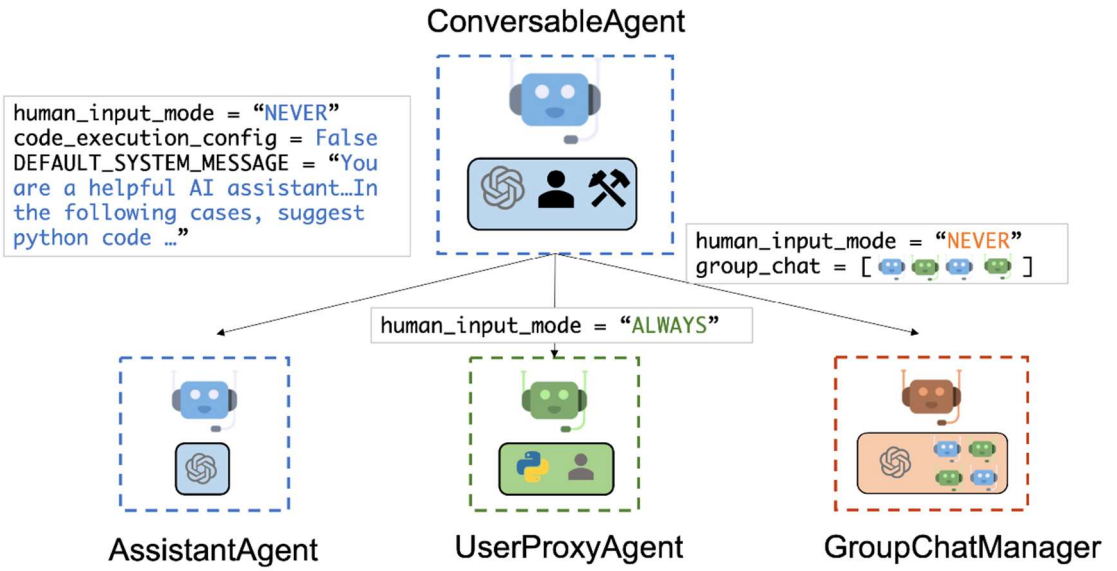


Figure 12. Different agent types in the AutoGen framework.

While UserProxyAgent is designed to be a proxy agent which solicits human inputs and can execute code, AssistantAgent is designed to act as an AI assistant that will use LLMs by default but does not require human input or code execution. GroupChatManager is a chat manager agent that manages a group chat of multiple different agents. (Microsoft 2024c.)

A code executor component can be utilized by the agents. It takes input messages, identifies code blocks, performs the execution and then outputs messages with the results (Microsoft 2024d). However, controlling what code the agent writes can be challenging. This is where tool usage comes in handy, which are predefined Python functions (Microsoft 2024e.) Rather than just being able to generate text, agents will be able to interact with the outside world by utilizing tool calling.

8.2 Multi-agent system for Software Engineering

One of the goals of this thesis was to research ways to shorten the required user input to achieve the desired goal in programming tasks. With the use of meta-prompting, a system was created to construct its own reasoning based on a user query to formulate a plan which achieves the user's intent. The AutoGen framework described earlier was utilized to build the system.

SWE-bench evaluates LLMs' abilities to resolve real-world GitHub issues automatically. The dataset contains 2,294 Issue-Pull Request pairs from 12 popular Python repositories. (SWE-bench 2024.) Most of the programs on the leaderboard, such as OpenHands (OpenHands 2024), are multi-agent systems (SWE-bench 2024). This proves that the most capable programming systems utilize multi-agentic workflows to achieve the best results in these leaderboards.

Two different models were used for the implementation of the system. Coder and the debugger agent utilized the new Claude 3.5 Sonnet model that is according to many benchmarks the top one. Other agents utilized the GPT-4o Mini model due to its decent reasoning skills and cost effectiveness for purposes such as generating the full project plan. According to Artificial Analysis leaderboard Google's new Gemini 1.5 Flash is faster and two times as cheap with about the same Quality Index (Artificial Analysis 2024). This model was attempted to be integrated into the project, but it did not work with AutoGen's tool calling at the time of the implementation.

The system introduces agents that work together to build the software. These agents are the project manager, coder, tester, and debugger. Project manager creates a plan based on the user's input to the system, assigns each agent its tasks through the creation of an XML table. Coder writes the program based on that plan with unit tests. After that, the tester agent will run the tests on it, print the output logs. If there are no errors, the session will be terminated as the program is deemed complete for the human to review and use. However, if errors appear then the debugger agent steps in, analyses the log and provides suggestions for the coder agent to fix. The loop goes on until the issue is resolved, or a certain variable limit defined in the code is reached.

8.2.1 Project setup and the parameters

Before attempting to run the system, it is recommended to create a new Conda environment with Python version 3.12.4. Instructions to set up the project are in the README.md file of the folder "multi-agent_system_for_programming", located on the root of the project. The program entry point file called "main.py" is in located in this folder. After

installing the dependencies with the “requirements.txt” file in the same folder on to the python environment, this program can be run with “python main.py”.

The program supports four different parameters which are: Source location, project name, user prompt, and the reset feature. Source location defines the location where the project will be built to. Project name defines name of the folder which will be created inside the source location. User prompt takes a custom query for the system to process and to build off from. Entering “file:[Define user prompt path here]” parses the contents of a file on to the system. If a folder already exists in the defined source location with the same project name, then using the reset parameter can be beneficial since currently this system does not support editing of the projects.

The program can be run without any additional parameters. In this case, it will default to building a simple calculator application, which is created on to a “src/calculator” folder next to the “main.py” file. The “src” folder is in the. gitignore file of the version control system by default.

8.2.2 Taking in the prompt input and generating the project plan

During the execution of the “main” function in the project, project manager agent receives the user prompt entered to the system and through custom instructions that were provided in its system prompt, it creates a plan based on that input. With a tool call, a custom XML based file is created which will be used to determine which agents to use with their order in the coming chat session.

Utilization of multiple agents will allow them each to have their own instructions. Each of the agents can focus on their own tasks only with their specialized system prompt. For instance, it was noticed during the implementation of the system that the project manager agent tried to assign a separate task for handling errors for the coder to do. In order to fix this issue, it was explicitly mentioned in the project manager’s system prompt that this agent should not do that. Instead, this was added to the coder’s instructions to handle this on each of the cases of generating the code. This way, the system becomes more efficient in its operation.

The XML file generated by the project manager will be parsed after the session is done. This is done with a custom class that handles all XML interactions in the project. Tasks are placed onto an array that will be looped through, utilizing the other agents in a separate session.

This means that the agents in the building phase of the program, which are in a different session will not be able to see the project manager's reasoning chain at all. Rather, the system constructs the list by parsing the XML file. Upon the startup of the second session, the builder agents in it will see a list in a plain text, consisting of items parsed from the XML file that was generated by the project manager agent and parsed by the predefined Python class.

8.2.3 Building phase of the program from the project plan

With the plan laid out in a comprehensive list, the coder agent can focus on working on small tasks assigned by the project manager. This agent is optimized for writing clean and commented code with debug lines ready at the beginning and end of the function. The coder agent will also write unit tests for the tester where logging is implemented for delivering the results. Then if problems arise during the testing phase of the system, the debugger agent can understand the error better and provide suggestions on how to fix this issue. Utilizing a tool call, this agent exports all the written code, the actual classes built for the program, and a separate file containing all the test cases.

During the testing phase, the tester agent uses a specific tool to execute the code by reading the file that has all the test cases written by the coder agent. The code is structured in a way that running it on that file will test all the functions that were written by the coder agent. Results are printed and if everything passes, the tester agent will terminate the session as the program executed as expected.

However, if the code execution stops due to an error or the test case does not return what it is supposed to do, then the debugger agent will analyze the log output and provide insight for the coder agent. Using this information, the problem will be attempted to be solved.

Since the tester agent is basically just a tool caller with appropriate data, an original testing prompt was not introduced in the prompt engineering section of the study. The purpose of the agent is to help create a smooth transition between the coder agent providing the code and the debugging agent providing solutions when issues arise. This removes the need to have testing instructions on the coder agent itself, while saving costs on utilizing cheaper GPT-4o mini model that is also faster.

The program keeps looping with these three agents until the issue is resolved. An integer parameter was implemented on to the code that defines how many iterations of the loop can be run during the session. The system will always try to read the context from the chat to avoid repeating the same mistakes; while also trying different approaches based on the data it received from the debugger agent. Although this can get very expensive if looped indefinitely, it is recommended not try to set too high of a value, since it is possible that LLMs cannot solve the problem and manual fix by an actual human software engineer should be implemented on to the codebase.

9 CONSIDERATIONS FOR FURTHER DEVELOPMENT

9.1 Reasoning models and their usage in software development

During the implementation phase, the author did not have access to OpenAI's new reasoning models due to the tier limitations by the API provider. According to the latest benchmarks, o1-mini and o1-preview are leading in tasks that require better reasoning capabilities (LiveBench 2024). This is shown in Figure 13:

Model	Global Average	Reasoning Average	Coding Average	Mathematics Average	Data Analysis Average	Language Average	IF Average
o1-mini-2024-09-12	56.66	72.33	48.05	59.22	54.07	40.89	65.40
o1-preview-2024-09-12	64.74	67.42	50.85	62.92	63.97	68.72	74.60
claude-3-5-sonnet-20240620	58.22	57.17	60.85	53.32	56.74	53.21	68.01
claude-3-5-sonnet-20241022	58.49	56.67	67.13	51.28	52.78	53.76	69.30
gemini-exp-1114	56.03	55.67	52.36	54.92	57.49	38.69	77.08
gpt-4o-2024-11-20	50.64	55.67	46.08	42.53	47.23	47.37	64.94

Figure 13. Reasoning capabilities benchmark of the new o1 models.

According to another benchmark that measures coding performance, LiveCodeBench, OpenAI's reasoning models outperformed the new Claude 3.5 Sonnet model on the Code Generation category. The major performance gap was evident in the results where Anthropic's model only achieved score of 5.5, whereas o1-mini achieved 31.7 on the Hard-Pass@1 results. The difference was significantly smaller on the Easy- and Medium-Pass@1, proving that the reasoning models excel at complicated tasks compared to models that are not finetuned with CoT. (LiveCodeBench 2024.)

OpenAI's reasoning model such as the o1-preview have max output token amount parameter of 32,768, while o1-mini doubles this to 65,536 (OpenAI 2024i). This is four times higher than the GPT-4o Mini model that was used in the study. With a system prompt tailored for a reasoning model as a planner, it could construct massive plans for the coder to implement, especially with comprehensive instructions by the user and this amount of output tokens.

Tokens that are produced during the reasoning process are not shown via the API. The only output that is visible through the API are the completion tokens. (OpenAI 2024i.) This could limit the debuggability of the prompts when optimizing them towards the 100% accuracy for specific tasks.

There are also limitations in these models, such as that they do not accept system prompts (OpenAI 2024i). There is a way to overcome the problem by adding the system prompt as text to the beginning of the user prompt, but this would not be very optimal. Other parameters such as the temperature and

top_p are fixed at 1, while presence_penalty and frequency_penalty are locked to 0. Tools and function calling are not supported either. These limitations could be lifted after the beta phase. (OpenAI 2024i.)

At the time of writing, open-source models such as DeepSeek-R1-Lite-Preview (DeepSeek 2024) and QwQ-32B-Preview (Qwen 2024) that do the extra reasoning steps are being released. These models will not have such limitations as OpenAI's models or hide the produced reasoning tokens. For the multi-agent system that was developed during this study, these kinds of reasoning models could be a great addition.

9.2 Improving the multi-agent system

The multi-agent system currently does not support editing projects, only building them from scratch. Upon the session termination, all the data in the context window of various models are lost. A system that would read the codebase intelligently, getting the parts of the codebase that it needs to be edited or built upon, would be recommended to be created to save input tokens, and on large projects, this would be necessary that the model's context window does not get contaminated with data that is unnecessary.

Retrieval-Augmented Generation (RAG) is a method where information is taken from external databases (Gao 2024). This method could be used for programming purposes where information such as the documentation that is not necessarily present in the training data of the model could be retrieved from other sources. If the model does not have the required data, it will take a guess and possibly hallucinate, which could result in incorrectly written code. RAG could reduce the likelihood cases.

Project management systems such as Trello could be integrated into the project. This would enhance the collaboration between humans and LLM. Automation such as this could make it possible to write cards full of information about what the system should do to edit the project. These could be moved into a column, for the system to read as a user prompt and start processing it. Also, Git integration could be implemented into the project, and a specific agent could be added to automatically commit changes.

Integrating all of these development proposals would pose significant challenges in terms of prompt engineering. It would be advisable to split tasks such as Git committing to specified agents having their custom system prompts. With the agent's speaking order modified, the agents would stay in the chat sessions that the system currently uses, gathering the necessary information and then calling custom tools to accomplish their tasks.

10 CONCLUSION

This section presents the conclusion of the study, the main parts of it and shows how the purpose of the thesis was met by answering each of the research questions.

Before the implementation phase, the concepts of user and system prompt with their differences were examined. After this, various prompting techniques were explored, and the most suitable were selected to be used in this study.

The performance of different LLMs in coding and reasoning tasks was studied. By examining benchmark information provided by various sources, the most optimal models were chosen to be used in this study. Information about parameters and their effect on the model's performance was acquired from Anthropic's and OpenAI's documentation.

During the implementation phase of the study, coding system prompt was developed to eliminate the accumulation of technical debt caused by prompting and ensure the maintainability of the codebase. CoT was used to create reasoning steps to make the model think about its final coding solution. The prompt can be used with simple queries to build software in a way that keeps codebase maintainable when using LLMs to write the code for the project. This was achieved by requesting the model explicitly to limit the size of produced functions so as not to clutter the codebase built with the prompt. Error handling is included by default and comments for describing the functions and logs are set in place to help debug the program when problems arise. This prompt was confirmed to be working in software such as Cursor, increasing the code quality while building software with such code editors.

Building upon implementation phase of the study, a multi-agent system was constructed next. While not directly related to only prompt engineering, building agentic systems requires crafting the of system prompts to bring the agents alive while providing tools for them to interact with the outside world rather than only chatting with themselves.

For the multi-agent system, a project manager agent was created that utilizes a prompting technique called meta-prompting. Upon running the program, this agent initiates the session. The system's task is to take in a user prompt, divide it to subtasks and compose a list of tasks to be done by the other agents in another session. In this, the agents will talk to each other in order to achieve the goal.

A debugging agent was created for the system to solve problems in programming utilizing LLMs. This agent analyzes the errors produced by the program using data from the debugging lines that were added earlier to by the coder prompt. By utilizing CoT, the prompt makes the model think step-by-step of what is wrong with the code that produced the error. As the debugging prompt is optimized for just analyzing the log, it will not provide the coding solution, making the agentic system flow better. Instead, the coder agent which is optimized for only writing the code by following simple instructions will do fix for the codebase.

The tester agent's purpose is to make the coder and debugger agent collaborate by closing the gap between them. If errors are not detected and the tests succeed, the session will be terminated. Since the coder agent was optimized to simply write the code, the four-agent paradigm allows it to excel in an environment where its job is to follow simple instructions generated by project manager and possibly the debugger agent.

The evaluation framework called promptfoo was used during the development phase of the prompts. Full 100% accuracy was achieved in all the tests, excluding an API error, utilizing the LLM evaluator with simple user prompts that were written for testing purposes. The test results were afterwards manually evaluated by the author to confirm that the prompts worked as

intended.

In sum, it can be deemed that the objectives of the thesis were accomplished during the implementation phase of the study. A GitHub repository containing all the prompts, the associated systems, evaluations, and instructions for reproducibility of the study is linked to Appendix 1. With an MIT license, the author grants full permission to use and build upon, if attribution is provided.

SOURCES

aider. 2024a. Aider LLM Leaderboards. WWW-document. Available: https://aider.chat/docs/leaderboards/ [Accessed 13 November 2024].

aider. 2024b. GPT code editing benchmarks. WWW-document. Available: https://aider.chat/docs/benchmarks.html [Accessed 20 November 2024].

Anthropic. 2023c. Claude 2. WWW-document. Released 11 July 2023. Available: https://www.anthropic.com/news/claude-2 [Accessed 13 November 2024].

Anthropic. 2023a. Introducing 100K Context Windows. WWW-document. Released 11 May 2023. Available: https://www.anthropic.com/news/100k-context-windows [Accessed 11 December 2024].

Anthropic. 2023b. Introducing Claude. WWW-document. Released 14 March 2023. Available: https://www.anthropic.com/news/introducing-claude [Accessed 12 November 2024].

Anthropic. 2023d. Introducing Claude 2.1. WWW-document. Released 21 November 2023. Available: https://www.anthropic.com/news/claude-2-1 [Accessed 13 November 2024].

Anthropic. 2024e. Claude 3.5 Sonnet. WWW-document. Released 21 June 2024. Available: https://www.anthropic.com/news/claude-3-5-sonnet [Accessed 13 November 2024].

Anthropic. 2024i. Create a Text Completion. Documentation. Available: https://docs.anthropic.com/en/api/complete [Accessed 30 October 2024].

Anthropic. 2024g. Giving Claude a role with a system prompt. Documentation. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/system-prompts [Accessed 14 November 2024].

Anthropic. 2024b. Glossary. Documentation. Available: https://docs.anthropic.com/en/docs/resources/glossary [Accessed 23 September 2024].

Anthropic. 2024f. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. WWW-document. Released 22 October 2024. Available: https://www.anthropic.com/news/3-5-models-and-computer-use [Accessed 13 November 2024].

Anthropic. 2024d. Introducing the next generation of Claude. WWW-document. Released 4 March 2024. Available: https://www.anthropic.com/news/claude-3-family [Accessed 13 November 2024].

Anthropic. 2024c. Model Deprecations. Documentation. Available: https://docs.anthropic.com/en/docs/resources/model-deprecations [Accessed 12 November 2024].

Anthropic. 2024a. Models. Documentation. Available: https://docs.anthropic.com/en/docs/about-claude/models [Accessed 13 November 2024].

Anthropic. 2024j. Rate limits. Documentation. Available: https://docs.anthropic.com/en/api/rate-limits [Accessed 3 December 2024].

Anthropic. 2024h. Use XML tags to structure your prompts. WWW-document. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags [Accessed 3 December 2024].

Artificial Analysis. 2024. LLM Leaderboard - Comparison of GPT-4o, Llama 3, Mistral, Gemini and over 30 models. WWW-document. Available: https://artificialanalysis.ai/leaderboards/models [Accessed 11 December 2024].

Askell, A. 2019. Release Strategies and the Social Impacts of Language Models. Research paper. PDF-document. Updated 13 November 2019. Available: https://arxiv.org/abs/1908.09203 [Accessed 10 December 2023].

Banerjee, S. LLMs Will Always Hallucinate, and We Need to Live With This. Research paper. PDF-document. Released 9 September 2024. Available: https://arxiv.org/abs/2409.05746 [Accessed 23 September 2024].

Boopathy, A. 2024. Unified Neural Network Scaling Laws and Scale-time Equivalence. Research paper. PDF-document. Released 9 September 2024. Available: https://arxiv.org/abs/2409.05782 [Accessed 27 November 2024].

Brown, T. 2020. Language Models are Few-Shot Learners. Research paper. PDF-document. Updated 22 July 2020. Available: https://arxiv.org/abs/2005.14165 [Accessed 10 December 2023].

Caceres, P. 2020. The Recurrent Neural Network. WWW-document. Available: https://com-cog-book.github.io/com-cog-book/features/recurrent-net.html [Accessed 26 November 2024].

Codewave. 2024. History and Development of Neural Networks in AI. Article. Released 21 October 2024. Available: https://codewave.com/insights/development-of-neural-networks-history/ [Accessed 26 November 2024].

Cursor. 2024c. Cmd K. Documentation. Available: https://docs.cursor.com/cmdk/overview [Accessed 19 September 2024].

Cursor. 2024a. Codebase Indexing. Documentation. Available: https://docs.cursor.com/context/codebase-indexing [Accessed 11 December 2024]

Cursor. 2024b. Tab. Documentation. Available: https://docs.cursor.com/tab/overview [Accessed 16 October 2024].

Dai, A. 2015. Semi-supervised Sequence Learning. Research paper. PDF-document. Updated 4 November 2015. Available: https://arxiv.org/abs/1511.01432 [Accessed 10 December 2023].

Dae-Kyoo, K. 2023. Prompted Software Engineering in the Era of AI Models. Research paper. PDF-document. Released 7 September 2023. Available: https://arxiv.org/abs/2311.03359 [Accessed 1 February 2024].

Dhinakaran, A. 2024. The Needle In a Haystack Test. Article. Released 15 February 2024. Available: https://towardsdatascience.com/the-needle-in-a-haystack-test-a94974c1ad38 [Accessed 7 November 2024].

DeepSeek. 2024. DeepSeek-R1-Lite-Preview is now live: unleashing supercharged reasoning power!. Documentation. Released 20 November 2024. Available: https://api-docs.deepseek.com/news/news1120 [Accessed 29 November 2024].

elder_plinius. 2024. Post 25 September 2024. X-microblog service. Status update. Available: https://x.com/elder_plinius/status/1838982899120431330/photo/4 [Accessed 3 December 2024].

Every. 2024. How to Win With Prompt Engineering - Ep. 38 with Jared Zoneraich. YouTube video. Available: https://www.youtube.com/watch?v=exzPO4hAD9s&t=1518s [Accessed 3 December 2024].

Fridman, L. 2024. Cursor Team: Future of Programming with AI | Lex Fridman Podcast #447. YouTube video. Released 6 October 2024. Available: https://www.youtube.com/watch?v=oFfVt3S51T4&t=1890s [Accessed 16 October 2024].

Gao, Y. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. Research paper. PDF-document. Released 27 March 2024. Available: https://arxiv.org/abs/2312.10997 [Accessed 1 December 2024].

GitHub. 2024. Getting code suggestions in your IDE with GitHub Copilot. Documentation. Available: https://docs.github.com/en/copilot/using-github-copilot/getting-code-suggestions-in-your-ide-with-github-copilot [Accessed 16 October 2024].

Google. 2024. Long context. Documentation. Updated 28 September 2024. Available: https://ai.google.dev/gemini-api/docs/long-context [Accessed 15 November 2024].

hgs. 2023. From GPT-1 to GPT-4: A Look at the Evolution of Generative AI. Article. Updated 20 September 2023. Available: https://hgs.cx/blog/from-gpt-1-to-gpt-4-a-look-at-the-evolution-of-generative-ai/ [Accessed 1 February 2024].

Hochreiter, S. 1997. LONG SHORT-TERM MEMORY. Research paper. PDF-document. Available: https://www.bioinf.jku.at/publications/older/2604.pdf [Accessed 26 November 2024].

IndyDevDan. 2024. AI Coding Tool Breakdown: AI Copilots vs AI Coding Assistants vs AI Software Engineers. YouTube video. Released 6 October 2024. Available: https://www.youtube.com/watch?v=2j_fgMPJGM0 [Accessed 16 October 2024].

krishnanrohit. 2024. Post 7 February 2024. X-microblog service. Status update. Available: https://x.com/krishnanrohit/status/1755122786014724125 [Accessed 3 December 2024].

Leike, J. 2022. Training language models to follow instructions with human feedback. Research paper. PDF-document. Updated 4 March 2022. Available: https://arxiv.org/abs/2203.02155 [Accessed 10 December 2023].

LiveBench. 2024. Dataset Card for “livebench/coding”. Hugging Face repository. Updated 22 October 2024. Available: https://huggingface.co/datasets/livebench/coding [Accessed 20 November 2024].

LiveBench. 2024. A Challenging, Contamination-Free LLM Benchmark. WWW-document. Updated 25 November 2024. Available: https://livebench.ai/ [Accessed 29 November 2024].

LiveCodeBench. 2024. Holistic and Contamination Free Evaluation of Large Language Models for Code. WWW-document. Updated 1 November 2024. Available: https://livecodebench.github.io/leaderboard.html [Accessed 1 December 2024].

Microsoft. 2023. What are prompts? Documentation. Updated 14 December 2023. Available: https://learn.microsoft.com/en-us/semantic-kernel/concepts/prompts/ [Accessed 30 November 2024].

Microsoft. 2024c. agentchat.groupchat. Documentation. Available: https://microsoft.github.io/autogen/docs/reference/agentchat/groupchat/ [Accessed 28 September 2024].

Microsoft. 2024d. Code Executors. Documentation. Available: https://microsoft.github.io/autogen/docs/tutorial/code-executors [Accessed 28 September 2024].

Microsoft. 2024b. Multi-agent Conversation Framework. Documentation. Available: https://microsoft.github.io/autogen/docs/Use-Cases/agent_chat [Accessed 28 September 2024].

Microsoft. 2024e. Tool Use. Documentation. Available: https://microsoft.github.io/autogen/docs/tutorial/tool-use [Accessed 28 September 2024].

Microsoft. 2024a. Understand tokens. Documentation. Released 20 May 2024. Available: https://learn.microsoft.com/en-us/dotnet/ai/conceptual/understanding-tokens [Accessed 7 November 2024].

Naveed, H. 2024. A Comprehensive Overview of Large Language Models. Research paper. PDF-document. Updated 17 October 2024. Available: https://arxiv.org/abs/2307.06435 [Accessed 27 November 2024].

The Nobel Prize. 2024. NOBEL PRIZES 2024. WWW-document. Available: https://www.nobelprize.org/all-nobel-prizes-2024/ [Accessed 26 November 2024].

McDade, A. 2024. What News Corp's Deal With OpenAI Says About Future of Publishing, AI. News article. Released 23 May 2024. Available: https://www.investopedia.com/newscorp-openai-deal-future-of-publishing-ai-8652856 [Accessed 27 November 2024].

Mikolov, T. 2013. Efficient Estimation of Word Representations in Vector Space. Research paper. PDF-document. Updated 7 September 2013. Available: https://arxiv.org/abs/1301.3781 [Accessed 26 November 2024].

OpenAI. 2018. Improving language understanding with unsupervised learning. WWW-document. Updated 11 June 2018. Available: https://openai.com/research/language-unsupervised [Accessed 10 December 2023].

OpenAI. 2019. GPT-2: 1.5B release. WWW-document. Updated 5 November 2019. Available: https://openai.com/research/gpt-2-1-5b-release [Accessed 10 December 2023].

OpenAI. 2022. Introducing ChatGPT. WWW-document. Updated 30 November 2022. Available: https://openai.com/blog/chatgpt [Accessed 10 December 2023].

OpenAI. 2023. GPT-4. WWW-document. Updated 14 March 2023. Available: https://openai.com/research/gpt-4 [Accessed 10 December 2023].

OpenAI. 2024g. Enhance your prompts with meta prompting. Documentation. Released 23 October 2024. Available: https://cookbook.openai.com/examples/enhance_your_prompts_with_meta_prompting [Accessed 3 December 2024].

OpenAI. 2024c. GPT-4o mini: advancing cost-efficient intelligence. WWW-document. Released 18 July 2024. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ [Accessed 12 November 2024].

OpenAI. 2024b. Hello GPT-4o. WWW-document. Released 13 May 2024. Available: https://openai.com/index/hello-gpt-4o/ [Accessed 12 November 2024].

OpenAI. 2024h. Models. Documentation. Available: https://platform.openai.com/docs/models [Accessed 29 November 2024].

OpenAI. 2024e. Pricing. WWW-document. Available: https://openai.com/api/pricing/ [Accessed 13 November 2024].

OpenAI. 2024f. Prompt engineering. Documentation. Available: https://platform.openai.com/docs/guides/prompt-engineering/strategy-test-changes-systematically [Accessed 14 November 2024].

OpenAI. 2024i. Reasoning models (Beta). Documentation. Available: https://platform.openai.com/docs/guides/reasoning [Accessed 29 November 2024].

OpenAI. 2024d. simple-evals. GitHub repository. Updated 31 October 2024. Available: https://github.com/openai/simple-evals [Accessed 13 November 2024].

OpenAI. 2024a. What are tokens and how to count them? Documentation. Available: https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them [Accessed 11 December 2024]

OpenHands. 2024. Getting Started with OpenHands. Documentation. Available: https://docs.all-hands.dev/modules/usage/getting-started [Accessed 29 November 2024].

promptfoo. 2024e. Assertions & metrics. Documentation. Available: https://www.promptfoo.dev/docs/configuration/expected-outputs/ [Accessed 30 November 2024].

promptfoo. 2024d. Configuration. Documentation. Available: https://www.promptfoo.dev/docs/configuration/guide/ [Accessed 8 December 2024].

promptfoo. 2024b. Getting started. Documentation. Available: https://www.promptfoo.dev/docs/getting-started/ [Accessed 30 November 2024].

promptfoo. 2024a. Intro. Documentation. Available: https://www.promptfoo.dev/docs/intro/ [Accessed 30 November 2024].

promptfoo. 2024c. Model-graded metrics. Documentation. Available: https://www.promptfoo.dev/docs/configuration/expected-outputs/model-graded/ [Accessed 30 November 2024].

Qwen. 2024. QwQ-32B-Preview. Hugging Face repository. Available: https://huggingface.co/Qwen/QwQ-32B-Preview [Accessed 29 November 2024].

Rossum, D. 2024. Context Windows Explained: Everything You Need to Know. Article. Updated 15 October 2024. Available: https://www.flexos.work/learn/context-window [Accessed 11 December 2024].

Shen, Z. 2024. Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4. PDF-document. Updated 18 January 2024. Available: https://arxiv.org/abs/2312.16171 [Accessed 7 December 2024].

Singla, A. 2023. Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. Research paper. PDF-document.

Updated 1 August 2023. Available: https://arxiv.org/abs/2306.17156 [Accessed 10 December 2023].

SMC Spain. 2024. Nobel Prize in Physics for Hinton and Hopfield for discovering the basis of machine learning with artificial neural networks. Article. Released 8 October 2024. Available: https://sciencemediacentre.es/en/nobel-prize-physics-hinton-and-hopfield-discovering-basis-machine-learning-artificial-neural [Accessed 26 November 2024].

Sui, Y. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. Research paper. PDF-document. Updated 17 July 2024. Available: https://arxiv.org/abs/2305.13062 [Accessed 18 September 2024].

Sutskever, I. 2014. Efficient Estimation of Word Representations in Vector Space. Research paper. PDF-document. Released 14 December 2014. Available: https://arxiv.org/abs/1409.3215 [Accessed 26 November 2024].

Suzgun, M. 2024. Meta-Prompting: Enhancing Language Models with Task-Agnostic Scaffolding. Research paper. PDF-document. Released 23 January 2024. Available: https://arxiv.org/abs/2401.12954 [Accessed 29 November 2024].

Sprague, Z. 2024. To CoT or not to CoT? Chain-of-thought helps mainly on math and symbolic reasoning. PDF-document. Updated 29 October 2024. Available: https://arxiv.org/abs/2409.12183 [Accessed 7 December 2024].

SWE-bench. 2024. Can Language Models Resolve Real-World GitHub Issues? WWW-document. Available: https://www.swebench.com/ [Accessed 27 November 2024].

Vaswani, A. 2017. Attention Is All You Need. Research paper. PDF-document. Updated 2 August 2023. Available: https://arxiv.org/abs/1706.03762 [Accessed 3 December 2023].

Villalobos, P. 2024. Will we run out of data? Limits of LLM scaling based on human-generated data. Research paper. PDF-document. Released 4 June 2024. Available: https://arxiv.org/abs/2211.04325 [Accessed 27 November 2024].

Wei, J. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Research paper. PDF-document. Updated 10 January 2023. Available: https://arxiv.org/abs/2201.11903 [Accessed 19 September 2024].

Wermelinger, M. 2023. Using GitHub Copilot to Solve Simple Programming Problems. Research paper. PDF-document. Released 3 March 2023. Available: https://dl.acm.org/doi/10.1145/3545945.3569830 [Accessed 16 October 2024].

White, J. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. PDF-document. Released 21 February 2024. Available: https://arxiv.org/abs/2302.11382 [Accessed 6 December 2024].

LIST OF FIGURES AND TABLES

Assistants vs AI Software Engineers. YouTube video. Released 6 October 2024. Available: https://www.youtube.com/watch?v=2j_fgMPJGM0 [Accessed 16 October 2024].

Figure 11. Promptfoo test run evaluation report. promptfoo. 2024. Intro. Documentation. Available: https://www.promptfoo.dev/docs/intro/ [Accessed 30 November 2024].

Figure 12. Different agent types in the AutoGen framework. Microsoft. 2024b. Multi-agent Conversation Framework. Documentation. Available: https://microsoft.github.io/autogen/docs/Use-Cases/agent_chat [Accessed 28 September 2024].

Figure 13. Reasoning capabilities benchmark of the new o1 models. LiveBench. 2024. A Challenging, Contamination-Free LLM Benchmark. WWW-document. Updated 25 November 2024. Available: https://livebench.ai/ [Accessed 29 November 2024].

Table 1. Word pair examples in SemanticSyntactic Word Relationship test set. Mikolov, T. 2013. Efficient Estimation of Word Representations in Vector Space. Research paper. PDF-document. Updated 7 September 2013. Available: https://arxiv.org/abs/1301.3781 [Accessed 26 November 2024].

Table 2. Comparing Anthropic's Claude 3 models to the gpt-4-0125-preview model by OpenAI. Data for the table was combined from three different sources. Generated using ChatGPT. Anthropic. 2024. Introducing the next generation of Claude. WWW-document. Released 4 March 2024. Available: https://www.anthropic.com/news/claude-3-family [Accessed 13 November 2024].

Table 2. Comparing Anthropic's Claude 3 models to the gpt-4-0125-preview model by OpenAI. Data for the table was combined from three different sources. Generated using ChatGPT. OpenAI. 2024. simple-evals. GitHub repository. Updated 31 October 2024. Available: https://github.com/openai/simple-evals [Accessed 13 November 2024].

Table 2. Comparing Anthropic's Claude 3 models to the gpt-4-0125-preview model by OpenAI. Data for the table was combined from three different sources. Generated using ChatGPT. OpenAI. 2024. Pricing. WWW-document. Available: https://openai.com/api/pricing/ [Accessed 13 November 2024].

Table 3. Performance of different prompting formats. Sui, Y. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. Research paper. PDF-document. Updated 17 July 2024. Available: https://arxiv.org/abs/2305.13062 [Accessed 18 September 2024].

Table 4. The new Claude 3.5 Sonnet compared to other models on the LiveBench site. LiveBench. 2024. A Challenging, Contamination-Free LLM Benchmark. WWW-document. Updated 25 November 2024. Available: https://livebench.ai/ [Accessed 29 November 2024].

Table 5. Aider’s code editing benchmark shows how the new Claude 3.5 sonnet outperforms other models. aider. 2024a. Aider LLM Leaderboards. WWW-document. Available: https://aider.chat/docs/leaderboards/ [Accessed 13 November 2024].

Table 6. The new Claude 3.5 Sonnet's performance in aider code refactoring benchmark. aider. 2024. Aider LLM Leaderboards. WWW-document. Available: https://aider.chat/docs/leaderboards/ [Accessed 13 November 2024].

APPENDIX

Appendix 1. Thesis project repository. Available:

https://github.com/mikaeltorni/prompt_engineering_for_software_development