Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i Datateknologi	Vårsemesteret 2024 Åpen

Forfatter(e): Aaron Skiftun Dalbak, Anders Josef Drønen Tungland

Fagansvarlig: Tom Ryen

Veileder: Leander Nikolaus Jehl

Tittel: Implement an AI course chatbot

Studiepoeng: 20

Emneord: AI, Discord, Petals, Python, Ollama	Sidetall: 63 Stavanger 15. mai 2024

Acknoweldgements

We would like to thank our supervisor Leander Jehl for his many ideas and help with writing this thesis. We also thank Jostein Hagen Lindhom for providing useful information and help in regards to the Unix system.

Contents

Content **i**

Summary **vi**

1 Introduction **1**

1.1 Background and motivation 1

1.2 Objectives 2

1.3 Approach and Contributions 3

1.4 Outline 3

2 Artificial intelligence **5**

2.1 Introduction to AI 5

2.2 Large language models 6

2.3 Generative AI 7

2.4 The language of AI 8

3 The Petals system 9

3.1 Motivation behind Petals 9

3.2 The Petals system 10

3.3 Choice of language model 12

3.4 Generating Tokens 12

3.5 How we use Petals 13

3.6 Petals interest drop 14

3.7 Petals need contributors 14

3.8 An alternative to public swarm 15

3.9 Training limitations 15

4 Implement a Discord bot 17

4.1 Discord 17

4.2 Chatbot walk through 18

4.2.1 Initiate a conversation 19

4.2.2 Chatbot response 20

4.3 Implement the bot 21

4.3.1 Discord developer portal 21

4.3.2 Discord interface 22

4.3.3 Petals interface 25

4.3.4 How we integrate the interfaces 26
4.4 Chatbot prefix 27
4.5 Ollama 28
4.5.1 Ollama interface 30
4.6 Added important features 31
4.6.1 Conversation context 32
4.6.2 Performance tracker 34
4.6.3 Rating logger 36
4.7 Giving the chatbot context 37

5 Performance results 39

5.1 How we evaluate the chatbot 39
5.2 Token generation results 40
5.3 Efficiency 42
5.3.1 Variation for Petals 44
5.4 Running Ollama on the Unix system 45
5.5 Gorina7 impact on Petals and Ollama 46
5.6 Comparing parameters 47
5.7 Comparing parameters results 47
5.8 Feedback from students 50

5.9 Feedback Results 50

6 Discussion **53**

6.1 Petals potential 53

6.1.1 Petals drawbacks 54

6.2 Final thoughts 55

6.2.1 What we cold have done differently 55

6.3 Llama 3 56

7 Conclusion **57**

Bibliografi **63**

Vedlegg **63**

Summary

In this paper, we look into the potential of using Petals to run large language models. Petals is a system for running large language models shared across several computers in a peer-to-peer fashion. We explore how to integrate Petals with Discord to create an AI course chatbot. We test Petals's performance of this task through comparing it with Ollama, a framework for running large language models on your own hardware. We deploy the chatbot in the Discord server for web programming students and encourage everyone to test it and provide feedback on their preferred answer. Feedback from students helps us evaluate how suitable the Petals system is for an interactive chatbot designed for the web programming course at the University of Stavanger.

We conclude that Petals in its current state is not suited for use as a course Discord chatbot because of its slow speed, challenging framework, and inability to give it course specific information.

Chapter 1

Introduction

1.1 Background and motivation

In recent years, AI chatbots have become increasingly popular. Some of these chatbots, such as OpenAI's ChatGPT and Google's Gemini, are especially popular among students and teachers [1]. A recent study shows that a significant number of students use ChatGPT frequently, and almost 75 percent of students found the chatbot very useful for assignments, projects, and exam preparations [1].

As previously mentioned, there are many free chatbots that you can find on the internet. These chatbots are very knowledgeable about information that can be found online, but lack the ability to answer questions that cannot be found on the Internet. This is one of the reasons we want to create a chatbot that we can tailor to answer questions about the web programming course taught at the University of Stavanger.

Successfully creating such a chatbot offers students a useful tool that can be used to ask questions about assignment deadlines, exam dates, and various topics that are taught throughout the course.

We believe that building a chatbot using the system known as Petals is a decent alternative to proprietary options, as it is free and relatively simple to get started with. More importantly, Petals offers the illusion of running large language models locally without the need for expensive equipment, which makes it accessible for both us and the university.

On the other hand, using Petals may prove difficult as there are some limitations in regards to how much we can customize the chatbot. We seek to create a chatbot that has a feature that popular chatbots do not have. A chatbot able to respond to questions about the web programming course in addition to more general questions would make it a unique and valuable alternative. The most important aspect of the chatbot is its ability to gather all course-relevant information in one place, where students can easily find the answers they are looking for.

1.2 Objectives

The main focus of this project is to integrate the Petals system with Dicord in order to create a useful chatbot for future students. We want to further investigate the following questions:

- How can we combine Petals and Discord to create a chatbot?
- How do we evaluate the usability and performance of the chatbot?
- Can we further evalute the chatbot by implementing it in the web programming Discord server and receive feedback from students?
- Can we provide the chatbot with course-related information such as topics taught throughout the course and exam details?
- Does the bot prove useful compared to already existing chatbots?

The desired outcome is a chatbot that can read messages sent from students in a Discord server and respond to course-related questions in addition to general questions not necessarily targeted towards the course.

1.3 Approach and Contributions

Our primary focus is on integrating Petals with Discord to create a chatbot. We put effort into evaluating it by investigating its usability, performance, and quality.

The evaluation process involves measuring some core performance metrics for both Petals and Ollama, such as tokens per second and initial time. We calculate these metrics by logging data from each conversation with the chatbot to a separate file. This includes the number of tokens generated and the time taken to do so. Additionally, we use Ollama to determine the difference between a 7B version and a 70B version of the same language model. This information is helpful because the 7B version can run on most high-end consumer laptops while the 70B model requires a system like Petals to run. If we determine that the 7B model is sufficient, we do not need to explore a system like Petals to create our chatbot, as we can run the chatbot locally on a laptop.

Furthermore, we deploy our chatbot in the Discord server for web programming students at the University of Stavanger, where they can test it and provide feedback. By combining the feedback from students with the metrics that we calculate, we can determine if Petals is a suitable option to integrate with Discord.

Based on our evaluation process, we discovered that while Petals has potential as a decent alternative, it is currently too slow to be used effectively as a system for a chatbot intended for students at the University of Stavanger.

1.4 Outline

The remainder of this thesis is structured as follows:

Chapter 2 covers what we consider to be the most significant information about artificial intelligence. This chapter serves as motivation behind why we believe the topics of this paper are highly relevant for the future. We introduce some fundamental terms that are important, such as tokens, language models, and generative AI.

Chapter 3 introduces Petals by explaining the motivation behind why it was created and how the system works. We discuss how we integrate the Petals system into our chatbot implementation. Finally, we address a recent decrease in community intrest and how it affects the performance.

Chapter 4 presents our solution for integrating the Petals system with Discord to create a chatbot. We provide a step-by-step demonstration of a typical conversation and highlight the key features we implemented to enhance the user experience and overall quality. Additionally, we discuss an alternative system for running large language models called Ollama, which aids in evaluating the Petals chatbot.

Chapter 5 covers the evaluation of our chatbot. We present the results obtained from data we have collected and feedback received from students who tested the chatbot. Moreover, we discuss running our chatbot powered by Ollama on the Unix system installed at the University of Stavanger.

Chapter 6 reflects the potential we believe Petals has and the impact the newly released large language model Llama 3 could have on our chatbot. We discuss some things we would have done differently and highlight changes we would make if we had more time.

Chapter 7 concludes our thesis by summarizing our work.

Chapter 2

Artificial intelligence

The following chapter provides a brief overview of some of the most important concepts, including AI, generative AI, and tokens. We introduce some fundamental information about AI to further illustrate our motivation for this thesis and why we believe it is relevant for the future. This chapter also talks about a crucial aspect of AI known as language models and how they are used in generative AI to create powerful tools. We will also mention some innovative areas where generative AI has been implemented, such as in the educational field.

2.1 Introduction to AI

The concept of artificial intelligence (AI) has been a topic of discussion for several decades but has gained immense popularity in recent years. Especially in fields such as computer science, medicine, cybernetics, and engineering [2]. There does not exist one universal definition of AI, but one understanding is that it's related to the mimicking of intelligence and human behavior, largely through computers.

2.2 Large language models

Upon deciding to implement a chatbot, it is essential to discuss which large language model (LLM) the chatbot should use, as it will have a major impact on performance, behavior, and overall quality. Large language models are deep learning models that have been trained on extraordinarily large amounts of data. Furthermore, LLMs are essentially very large pre-trained language models (PLM) where the increase in data size improves its ability to perform complex tasks. LLMs are typically built using a transformer architecture that includes self-attention mechanisms that significantly improve the ability to learn by weighing the importance of all parts in a sequence [3], [4].

Thus far, we have referred to language models as large. This is an indication as to how much data has been used to train the model, also known as the model size. The separation of a small, medium, or large language model is not uniformly set to a specific size, but are typically trained over an extended period of time [5]. Language models with a larger model size tend to perform better but quickly become more complex and difficult to maintain. In addition, large language models demand high-end, expensive hardware that students and enthusiasts typically do not have, making it more difficult for interested parties to research topics such as LLMs.

Large language models offer several powerful capabilities, including the ability to have a conversation, perform language translation, write poems and motivational speeches, and summarize large documents [6]. Today, the most advanced language models are capable of locating information about practically any topic that has publicly available information. Another great example illustrating the power of AI and large language models is GitHub Copilot, a useful tool to increase productivity and efficiency in software development that was recently developed using a pre-trained language model created by OpenAI. Recent surveys show that more than fifty thousand companies have adopted GitHub Copilot as an active tool during development of software [7].

How language models are trained may vary, but most of them share the ability to be fine-tuned to meet specific needs. This is another reason why we believe AI is incredibly powerful, as it can be trained to be extremely knowledgeable about any specific topic. Examples include, but are not limited to, the ability to be particularly knowledgeable about software development such as GitHub Copilot. There are also models that are trained to be exceptional at math, medicine, languages, and much more.

2.3 Generative AI

Generative AI is a subset of artificial intelligence and is particularly relevant for chatbots that communicate with people. This subset consists of a set of technologies that are able to generate new content based on already existing data, such as text, audio, or images. Generative AI is typically trained using very large amounts of data, and this is the information that is used to generate new content [8].

This kind of AI is often trained using supervised learning where the model is fed content created by humans with detailed labels that describe its purpose. In this way, generative AI learns patterns and human behavior, and this will improve its ability to generate content similar to what it was fed to begin with.

A recent example of generative AI includes one of the most known tools for learning new languages, Duolingo. A popular application used by millions of people announced in March 2023 the release of Duolingo Max, which combines generative AI and education to form an even more powerful tool to help you learn a new language. The Duolingo team believes that it will make learning easier, as AI offers highly personalized language lessons that are affordable to everyone. This was achieved using the popular and powerful model known as GPT-4, developed by OpenAI [9].

Typically, businesses integrate a generative AI chatbot on the company website to assist customers with various inquiries. The Norwegian postal company Posten has a chatbot where you can ask questions and receive a response accordingly [10]. We include examples like Duolingo and Posten to illustrate the extreme relevancy of generative AI, and the recent growth will most likely continue to increase and become an even bigger part of the future.

2.4 The language of AI

Large language models use a tokenization scheme that divides a sequence of words into smaller pieces, often referred to as tokens [11]. A token is typically a word or a special character, such as a comma or exclamation mark. In some cases, if a particular word is long, it is divided into several tokens.

A language model often has a tokenizer that reads the input and divides the sequence of text into appropriate-sized tokens. There are several types of tokenizers that have different methodologies for splitting sentences into tokens, and the method used may vary depending on the system you use to run the language model [11].

Tokens also serve as an important metric when measuring performance. Exactly what performance entails depends on the application. In some cases, a detailed and accurate response is more important than speed.

In our case, we need a chatbot that uses a language model that can generate tokens efficiently because Discord is an interactive application and we believe students have certain expectations in regards to speed. Such applications are expected to work fast, and the generated output is expected to be relatively accurate and appropriately long. We strive to strike a balance between the amount of time spent generating tokens and the quality of the generated output. By measuring how many tokens our chatbot generates in addition to the amount of time needed to do so, we evaluate its performance and make changes accordingly.

Many language models have a limit as to how many tokens can be received as input and how many tokens can be generated as output. This number may vary from a few hundred to several thousand, depending on the language model used. OpenAI is the developer of some of the most popular language models in the world and has created ChatGPT-3.5 with support for up to 16,000 tokens and ChatGPT-4-turbo with support for 128,000 tokens [12].

Chapter 3

The Petals system

The following chapter introduces Petals and provides an overview of the system. Importantly, how Petals is capable of creating the illusion of running a large language model locally on any computer. We illustrate how we use Petals to run LLMs, allowing us to generate tokens. Lastly, we discuss the current state of Petals, performance and some limitations we have encountered. The main source of information in this chapter is this Petals paper [13] published in 2022. We have also gathered various information from this Google research notebook [14] and the GitHub repository [15] created by the developers.

3.1 Motivation behind Petals

Petals is a part of an AI project created by the BigScience research workshop [14]. BigScience is a collective of more than 1000 researchers from approximately 60 countries worldwide [14]. The researchers share the passion to create a large multinational, decentralized platform for language models and are investigating alternative ways of running large language models.

During the writing of the Petals paper [13], the creators of Petals noticed a recent interest in alternative ways to run large language models without the need for very expensive equipment like hardware components. One important reason is that such hardware is unavailable for many researchers, thereby making it difficult for researchers to continue the exploration of language models. Secondly, lack of access to such hardware slows the learning curve related to language models.

The Petals paper discusses two recent alternatives to running large language models by either using an API or via RAM offloading. The Petals creators conclude that using an API is not flexible enough as most of the model is controlled by the developer. This means that using an API might make it impossible to customize the language model file, change the language model, tune the model, or add additional information about topics as context.

Furthermore, according to the Petals paper, offloading is inherently slow, rendering the solution unviable. These are key motivating factors as to why Petals was created and designed to be an alternative solution to run large language models by using the computational power of many. Petals attempts to eliminate the need for expensive hardware allowing researchers to continue exploring the realm of language models.

3.2 The Petals system

Petals is a system that enables individuals to take advantage of the hardware components of others to run large language models through a peer-to-peer (P2P) structure [13]. *Peer-to-peer (P2P) is a decentralized communications model in which each party has the same capabilities and either party can initiate a communication session [16].* In other words, any individual can run a large language model on their computer by initiating an inference session that, in Petals terminology, means generating tokens on the fly as a response to human input [17].

Petals take advantage of the P2P structure by creating a network of computers where each computer is considered a node [13]. Several nodes make up a network that, in Petals terminology, is known as a public swarm. Running a large language model is often referred to as a task and is divided into smaller parts and distributed amongst the needed number of nodes where the computations and processing are done remotely, excluding the need for complex and expensive hardware to run large language models. Petals takes advantage of the computational power you get when you combine many nodes in a network of computers. This unique approach is what we consider the core of the Petals system and is the key feature that separates this solution from other available options.

Each computer contributing to the Petals system will run as a server, a client, or both. The servers are responsible for hosting a small portion of the model layers, known as a transformer block [13]. The servers are also responsible for handling requests sent by the clients, while the client is responsible for creating a pipeline of consecutive servers to run the inference, that combined, makes up the entire model.

Figure 3.1 illustrates a simple example in which multiple clients share several GPUs from individuals dedicating their hardware to the public swarm. Each layer is responsible for a portion of the task that together makes up the entire model.

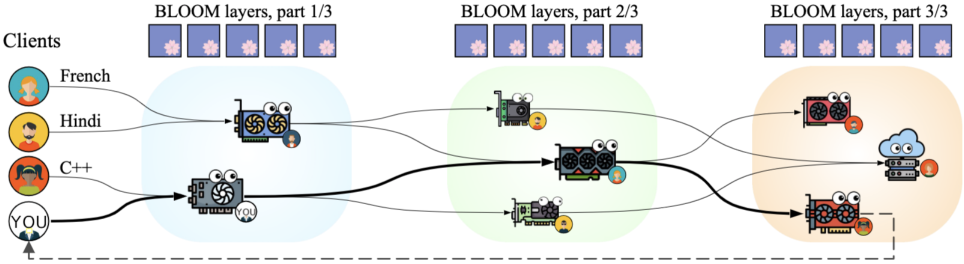


Figure 3.1: illustrates a group of clients connected to multiple servers that dedicate their hardware resources to the public. These clients use several servers to run any large language model supported by Petals [13].

The illustration shown in Figure 3.1 provides an excellent visual representation of how the Petals system utilizes the resources of multiple servers to run large language models supported by Petals without the need for expensive equipment. Although the example may be somewhat simplified, it effectively illustrates how a large language model can be run by clients. The task of running a LLM is divided into multiple layers, which are shared among multiple servers that use their respective GPUs to run each layer. In the case of this demonstration, the final layer is run on the third GPU, which returns the output after processing the last layer. In the case of our chatbot, the output that is returned after running the last layer is the set of tokens that are generated as a response to questions asked by students in a Discord server.

3.3 Choice of language model

The previous chapter introduced the concept of language models and explained how different types of AI have different use cases. Many language models are created and trained to fit a specific application, such as generating text, knowledge- or fact-based generation, code generation, or all-purpose. One of the many important objectives of this project is to create an AI chatbot with the main functionality of having conversations with students. Therefore, it is important that the chatbot is optimized for interacting with humans. As we are working on this paper, Stable Beluga 2 (70B), a fine-tuned version of the large language model known as Llama 2 (70B) [17] is the only LLM with active contributors in the public swarm [18]. This is likely due to the fact that this language model is well suited for interactive applications. For these reasons, we use the language model Stable Beluga 2 (70B) through Petals to create our chatbot.

3.4 Generating Tokens

The Petals system provides various options for generating tokens through different generation methods. The default method for Petals is called greedy [17], which compared to other algorithms is easy to implement. This algorithm follows the approach of selecting the best possible option at every given moment. [19]. In other words, each time a token is generated, the algorithm chooses the best option available at that time, without considering the long-term consequences. Although it is simple to implement, it has the drawback of focusing only on the short-term, which may negatively affect the final outcome. This is because it does not take into account the complete answer it is generating [19], but makes decisions on a per-token basis.

Other popular generation methods directly supported by the Petals system are beam searching, top-p, and top-k sampling [20]. You can choose between different generation methods by adding the necessary arguments as illustrated in Code 3.1 below.

```
outputs = model.generate(**inputs, num_beams=5, ...
    max_new_tokens=50) # Beam search
outputs = model.generate(**inputs, do_sample=True, ...
    num_beams=1, max_new_tokens=100) #Multinomial sampling
```

Code 3.1: Illustrates two examples of token generation methods we can use known as beam searching and multinomial sampling. Both examples are from the folowing article [21].

Finally, the Petals system provides the ability to implement custom methods for generating tokens. This feature is more advanced as it requires a deeper understanding of how such methods work. However, the ability to implement your own generation method can be very beneficial. For instance, it is possible to create algorithms that prohibit swear words or other words that we do not want to be included in the chatbot's responses [17].

3.5 How we use Petals

We write several scripts in the Python programming language which incorporate a combination of skeleton code provided by the developers along with our own code. These scripts are responsible for the functioning of our chatbot. Our code is structured in such a way that it consists of one main file, one file defining the Petals system, and another defining the Ollama system. All parts of our code can be found in our GitHub repository[1] [22].

For the Petals system, we download the required package from the Petals package website [23]. This package includes all the necessary components that we import into our script, following the instructions provided by the Petals developers on the above-mentioned website. This enables us to select the large language model to use, establish a connection to the P2P network, and run the model as if it were on our local machines.

---

[1] All parts of our code can be found here: `https://github.com/AJ-A-dev/discord-bot-petals`

3.6 Petals interest drop

As previously mentioned, evaluating Petals is one of the objectives of this paper. When looking at the performance of the Petals system, we separate between several factors including speed and communication delay [13]. The process of establishing the initial connection is the most time-consuming portion of the entire system. According to the developers, Petals is particularly sensitive to network latency in addition to intercontinental delays [13].

During the writing of the Petals paper [13], the developers concluded based on testing that the expected token speed was 6 tokens per second. These tests were performed under slightly unrealistic conditions, as the implementation consisted of three expensive, non-consumer-grade NVIDIA A100 80GB graphics processing units. Additional tests were performed using more realistic scenarios, such as running inference using 14 real servers spread across Europe and North America. The result of a more realistic approach concluded that the expected performance was cut in half, resulting in approximately 3 tokens per second [13].

We have been in contact with one of the Petals developers through Discord, who stated that there has been a significant decrease in performance as of late. This is mainly due to the fact that there are more people taking advantage of Petals and using it for their own purposes compared to how many people are actively contributing their GPU to increase performance for others. There is a growing imbalance between the supply of hardware (people dedicating GPUs) and the demand of people using the little available resources.

3.7 Petals need contributors

Petals is built using a community-run approach and is therefore completely dependent on the contributions of others. As previously mentioned, there has been a drop in the number of active contributors, which means less hardware to distribute various tasks to in addition to a higher load on a per-server basis. This is becoming an even bigger issue as Petals is growing in popularity in regards to new people testing it for their own purposes. We are experiencing a growing imbalance between active clients and available servers. The consequence of the above-mentioned problems is that the number of tokens generated per second is significantly reduced. Furthermore, the time it takes to find a chain of nodes and establish a connection that makes up the model to run inference takes a long time, which is not ideal in an interactive setting like a chatbot.

3.8 An alternative to public swarm

The Petals system offers another feature known as a private swarm, and this feature allows you to set up your own network of nodes. This is a great option that eliminates potential security-related risks, such as infected or malicious nodes that may exist in a public swarm. This also allows individuals to use Petals in combination with sensitive data, where they do not need worry about said data being tracked or manipulated [24].

The private swarm essentially functions in the same fashion as a public swarm, with the obvious difference that all the nodes in the network are trusted devices. However, this feature is a bit more technical and requires a greater understanding to set up compared with getting started with the public swarm.

3.9 Training limitations

In this chapter, we have established that Petals was created with the purpose of using a P2P network comprised of remote computers for running large language models. However, there is a drawback to this approach. Since the public swarm is shared among all users, there are certain limitations when it comes to customizing the model to meet specific needs. As we are not running the language model on our own machine, we are unable to modify a model file or provide it with a lot of information about a particular topic. This is similar to using a language model with an API, which does not offer much flexibility as the developer controls most of the configuration.

3.9 Training limitations

A large language model without the ability to fine-tune, tailor, or customize will in many cases not be very useful. Petals offers the option to use several parameter-efficient fine-tuning methods to make the model suitable for most tasks [14].

According to the Petals paper [13], fine-tuning in a distributed network such as Petals allows the client to use custom parameters while the original parameters used for the servers stay intact. In other words, any individual can run the model with changed parameters without interfering with other clients in the public swarm.

Chapter 4

Implement a Discord bot

In this chapter, we present our solution to integrate Petals with Discord. We provide a walk-through of a typical conversation between a student and the chatbot. Additionally, we present some features we implement, such as using a prefix to give the chatbot the ability to have previous conversations as context. We add this feature to increase the usability and user experience as this will allow students to ask follow-up questions. Lastly, we discuss Ollama and how we use it to give the chatbot exam details as context. Providing the chatbot with such information will dramatically increase its purpose as a tool for web programming students.

4.1 Discord

Discord is an application created by Jason Citron and Stan Vishnevskiy with the sole purpose of offering an easy way to communicate with friends while playing video games [25]. The founders released the application in May 2015, but it first became popular a few years later. Most commonly, users download Discord locally on their computer, tablet, or smartphone, but it is also possible to use the application in a web browser.

The platform has become a widely popular tool for communication between friends, students, and for work-related purposes. Today, there are numerous servers dedicated to different hobbies and interests where people can join and talk with each other. Discord is offered in 30 different languages and has approximately 150 million monthly active users and roughly 19 million active servers per week [25]. Throughout our studies at the University of Stavanger, we have used Discord as a tool to communicate with our fellow students and teachers. For many students, including ourselves, it has been a helpful platform to find group partners and seek help with assignments.

Using Discord, the course responsible can create a Discord server which students can join by clicking on a link shared by the creator. This has become a standard throughout our studies, including courses such as web programming. Discord provides a platform for students to ask questions about the course while instructors can share relevant information such as assignments, deadlines, course details and information about the exams. In our opinion, Discord has certainly proven to be an effective tool for communication between lecturers and students.

To enhance the experience, Discord offers the opportunity to make your own custom applications for the platform. These applications allow us to customize our chatbot by specifying permissions and roles, giving it a name and profile image. Typically, bots are implemented in Discord servers to perform various automated tasks, such as responding to frequently asked questions and playing music. This unique feature is what we use to create our very own chatbot.

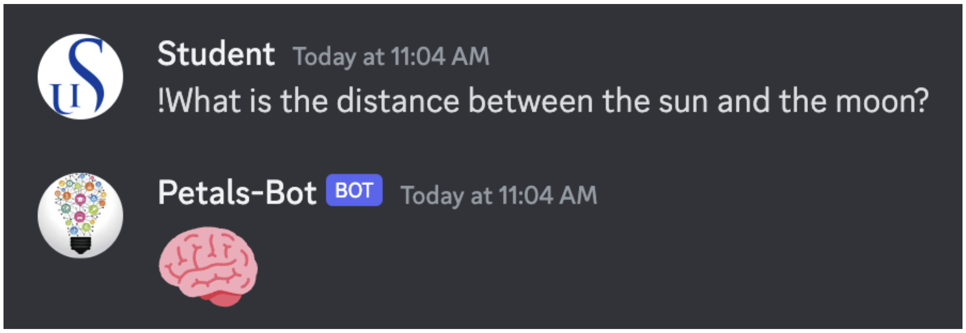
4.2 Chatbot walk through

This section provides a demonstration of a typical conversation between a student and our chatbot. We show how we initiate a conversation and how the chatbot replies by updating the initial response continuously with new tokens as they are generated.

4.2.1 Initiate a conversation

To start a conversation with the chatbot, we have to begin by writing an exclamation point followed by a question. We implement this to ensure that the bot only generates responses to messages that are intended for the chatbot. Not adding the exclamation point creates chaos as the bot spams the server by responding to all messages sent in the server.

Alternatively, we can create a dedicated channel within the server for chatbot conversations and manually remove the chatbot from all other channels. However, this is a more tedious and time-consuming task. Figure 4.1 below illustrates how to initiate a conversation with the chatbot.



Student Today at 11:04 AM

!What is the distance between the sun and the moon?

Petals-Bot BOT Today at 11:04 AM

🧠

Figure 4.1: Shows how we initiate a conversation with the chatbot by using an exclamation mark followed by a question. The Petals chatbot automatically responds with an emoji while the connection is established.

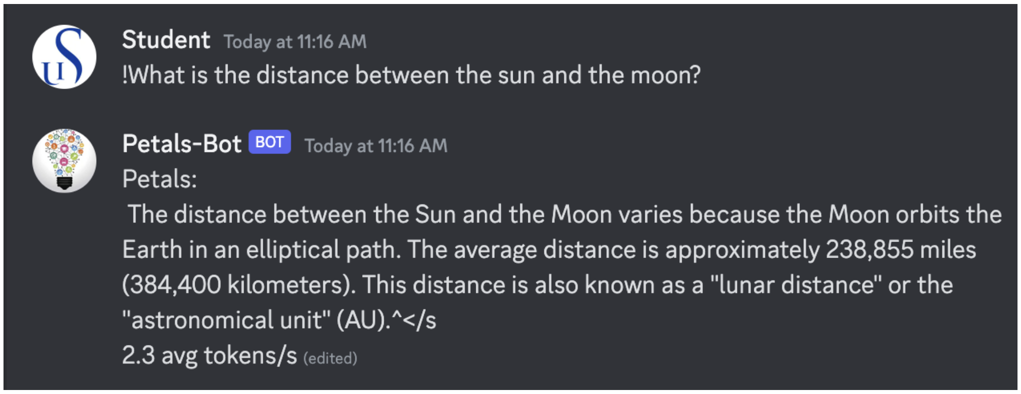
Note that the initial response sent by the bot is an emoji of a brain, indicating that the bot is thinking. We add this simply as a visual representation that the bot needs some time to read and process the question before it starts responding by continuously generating tokens. When a question is asked in the Discord server, the chatbot needs to establish a connection to the P2P network of servers to find the necessary components to run the large language model. The script located in our GitHub repository[1] reads and sends the text of any message as input to the language model. The model then generates output based on the contents of the input message.

---

[1] All parts of our code can be found here: `https://github.com/AJ-A-dev/discord-bot-petals`

4.2.2 Chatbot response

After a suitable route has been found and the connection has been established, the bot will start generating tokens and respond by sending the first few tokens generated. This is achieved by editing the previously sent message containing an emoji with the newly generated content. Figure 4.2 shows how the message changes from an emoji to a response when the first tokens have been generated.



Student Today at 11:16 AM
!What is the distance between the sun and the moon?

Petals-Bot BOT Today at 11:16 AM
Petals:
The distance between the Sun and the Moon varies because the Moon orbits the Earth in an elliptical path. The average distance is approximately 238,855 miles (384,400 kilometers). This distance is also known as a "lunar distance" or the "astronomical unit" (AU).^</s
2.3 avg tokens/s (edited)

Figure 4.2: Illustrates how the chatbot responds to a question. The chatbot continuously edits the original message as new tokens are generated. The last line reports how many tokens were generated per second.

Note that there is only one message sent in response to the question. We implemented a feature for updating the message continuously as tokens were generated, but ended up removing it because it was too much of a bottleneck when it came to speed.

Using Petals, we have not managed to find a good solution to dynamically control the length of the response that is generated. The number of tokens the AI bot generates generally depends on how much information it has and deems relevant. This can be problematic as the chatbot sometimes writes very long messages and continues to generate tokens. We have made it such that the petals bot stops generating on special characters such as </s> and /n. This is the only way we have found to stop petals from generating too long.

Furthermore, the bot can be forcefully stopped by entering the /breakall command, which will terminate the token generation immediately for all conversations.

It is possible for multiple people to have conversations with the bot, but this has a significant impact on the user experience as it directly affects other users. When a new person initiates a conversation, the bot will temporarily stop all generation until it has started generating tokens related to the latest conversation, and then continue generating text for all active conversations. This happens because every conversation using Petals needs to establish a connection to the nodes in the network. In other words, each time a new conversation is initiated the chatbot stops generating tokens for all active conversations until the connection has been established for the latest conversation.

Finally, each response ends by displaying the number of tokens that particular response had on a per-second basis. This number is calculated by logging the time difference from the first token generated to the last token generated. Furthermore, this logged number is added to a list and divided by the total number of tokens generated. Chapter 5 provides more information on the metrics used to measure performance.

4.3 Implement the bot

In this section, we explain how we integrate Petals and Discord to create the chatbot. The packages provided by the developers of Petals and Discord play a crucial role in establishing communication between the two. Importantly, we explain the Discord and Petals interfaces and how they work together to form the chatbot.

4.3.1 Discord developer portal

The developers of Discord have created a dedicated web-based developer portal to build and customize bots. To initiate the bot creation process, users are required to create an application through the online portal, which grants access to a variety of settings. This allows the owner to specify bot permissions and customize general information such as the bot name, description, and pictures. Upon the creation of a bot, a unique identifier, known as a token, is generated. This token is important as it serves as an exclusive identifier for legitimate Discord bots, facilitating authentication and interaction with the Discord API. Importantly, this token must never be disclosed to the public. It would then be necessary to regenerate it [26].

4.3.2 Discord interface

The Discord library (also called an API wrapper) for Python offers great support for creating your own bot. The library is object-oriented, easy to use, and based on listening for events [27]. In addition, this package does make it easier for us to integrate the Petals client with Discord by eliminating the need to set up custom API endpoints. Moreover, the API wrapper has several great built-in functions, shown in Code 4.1 below.

```
1 client.get_channel(channel_id)
2 message.content
3 message.edit(content=new_message)
4 message.channel.send
5 client.run(secret_token)
```

Code 4.1: A few of the most important built-in functions that we use to communicate between Discord and Petals. These functions allows us to read, edit and send messages to the originating Discord channel by fetching the Discord channel ID.

We use the first built-in function (line 1, Code 4.1) to easily fetch the channel ID from which the originating message was sent from. We use the second command (line 2, Code 4.1) to read the contents of any message sent by a user, which we store in a variable and pass as a parameter to the Petals client. This parameter contains the question sent by a student and is the input that the Petals client receives and uses to generate a response. The third function (line 3, Code 4.1) allows to easily edit a previously sent message. This is also shown in Figure 4.1 and Figure 4.2 where the chatbot edits the message by replacing the brain emoji with the generated output. The forth line (line 4, Code 4.1) shows how we send tokens generated by

Petals to the Discord server. Lastly, the fifth line (line 5, Code 4.1) shows how we run the Discord client by passing the secret token as the parameter.

The communication between Discord and the chatbot works by using websockets to know about the events happening in the Discord server. Events include actions such as when someone writes a message in the Discord server or reacts to a message using an emoji. This system is called the Gateway API [28]. Websocket is an API that allows a client to send messages to a server and receive event-driven responses [29]. Figure 4.3 shows how the chatbot communicates with the Discord API.

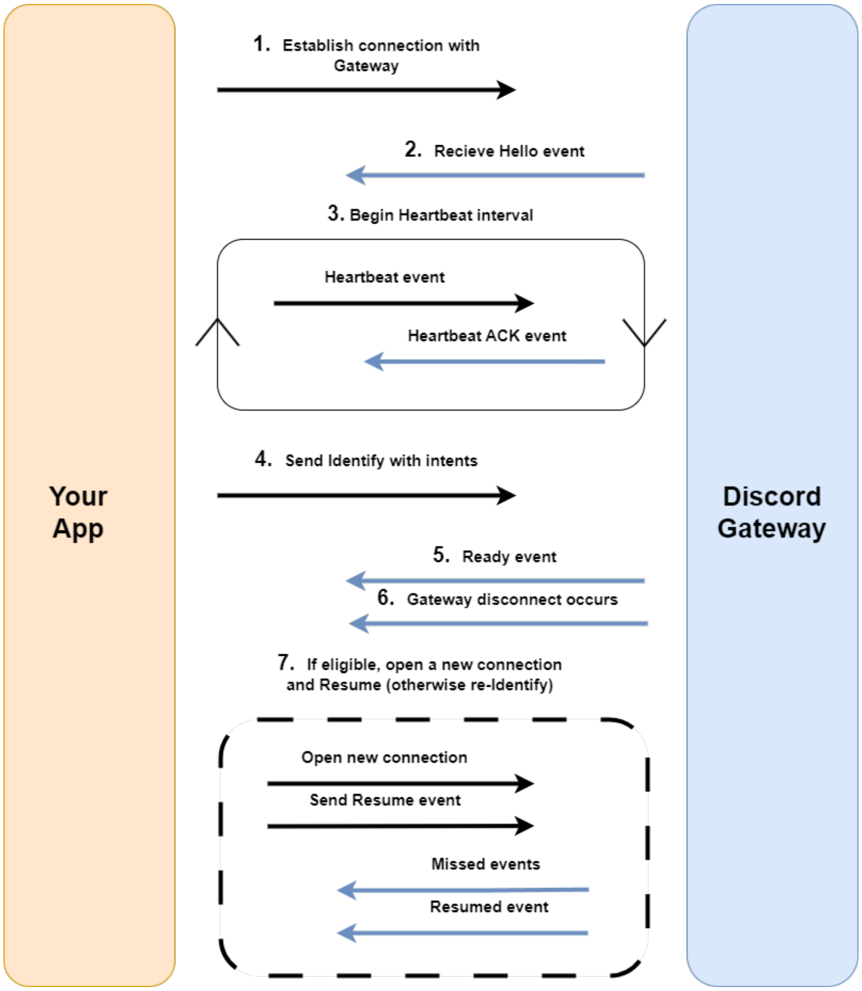

Your App
Discord Gateway
1. Establish connection with Gateway
2. Recieve Hello event
3. Begin Heartbeat interval
Heartbeat event
Heartbeat ACK event
4. Send Identify with intents
5. Ready event
6. Gateway disconnect occurs
7. If eligible, open a new connection and Resume (otherwise re-Identify)
Open new connection
Send Resume event
Missed events
Resumed event

Figure 4.3: Illustrates how the application (Discord chatbot) interacts with the Gateway API. The chatbot starts by connecting to the gateway API, which then responds with a hello event. After this, heartbeats are sent at regular intervals to maintain the connection. [28].

The chatbot starts by trying to establish a connection to the Discord gateway (1). The gateway responds with a hello event (2). Once the initial connection is established, the chatbot regularly sends heartbeat intervals (3) to maintain the connection. To validate its legitimacy, the chatbot must send an identity event (4) with its secret token. When the token is validated and the chatbot is successfully identified, the gateway API responds with a ready event (5). A disconnect may happen (6). If a disconnect occurs the bot will open a new connection, send a resume event which Discord responds to by sending the missed events and the resumed event (7) [28].

4.3.3 Petals interface

The Petals library enables us to use input (such as messages sent by students in Discord), use any supported language model to generate output and return it. Code 4.2 shows how we run inference and the most important built-in functions we use from the library. All parts of our code can be found in our GitHub repository[2].

```
with self.model.inference_session(max_length=512) as sess:
    prefix = f"{context}Human: {prompt}\nFriendly AI:"
    prefix = self.tokenizer(prefix, ...
        return_tensors="pt")["input_ids"]
        while self.client.generation:
            outputs = self.model.generate(prefix, ...
                max_new_tokens=1, session=sess,
                                do_sample=True, ...
                                    temperature=0.9, ...
                                    top_p=0.6)
            result += ...
                self.tokenizer.decode([self.fake_token, ...
                outputs[0, -1].item()])[1:]
```

Code 4.2: Shows an overview of the built-in functions and how we use them to generate tokens. Importantly, we have a prefix that contains the input, an output variable that stores the generated output, and the result variable that stores the final generated output after decoding.

---

[2] All parts of our code can be found here: `https://github.com/AJ-A-dev/discord-bot-petals`

The Petals library, as shown in Code 4.2, enables us to use a prefix to store a student's question. This prefix is used as a parameter to generate a response based on the input. The response is generated by calling the generate function (Line 5). Petals generate tokens continuously and appends the output to a result variable (Line 7). When the model is finished generating the final output is stored in the results variable.

4.3.4 How we integrate the interfaces

In Figure 4.4 below we see an illustration of the data-flow between our chatbot, Discord and Petals. When a student sends a question our bot receives an On_message event from Discord which contains the question asked (1). The bot then sends a placeholder to the channel where the original message was sent from (2) to illustrate that it have received the question. After, the bot will extract the content from the message and send it to Petals (3) where Petals will start looking for a path through servers containing the whole language model of Llama2-70B. The output will then be sent back to the bot as Petals is generating (4). Finally it will be sent back to Discord and replace the placeholder message (5).

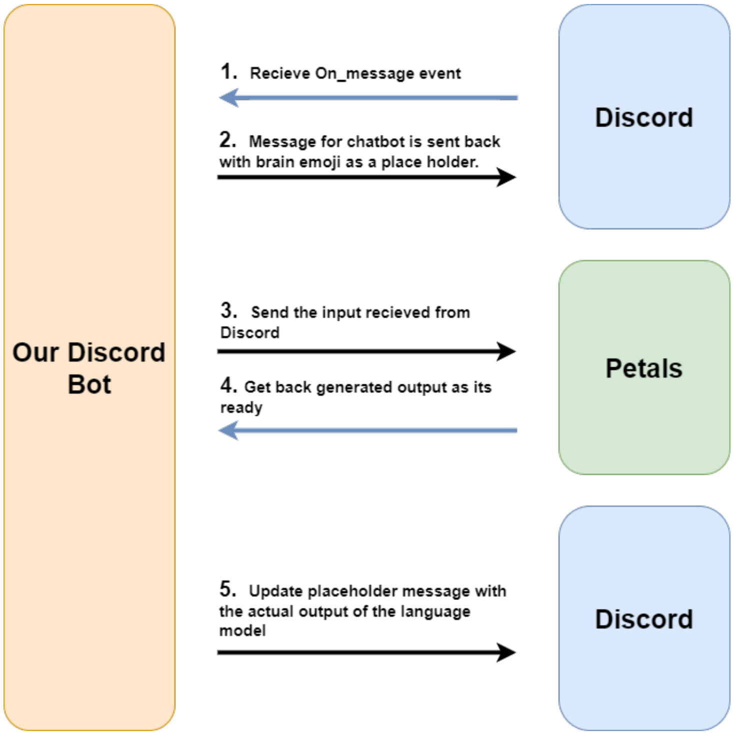

Our Discord Bot
1. Recieve On_message event
2. Message for chatbot is sent back with brain emoji as a place holder.
Discord
3. Send the input recieved from Discord
4. Get back generated output as its ready
Petals
5. Update placeholder message with the actual output of the language model
Discord

Figure 4.4: Demonstrates the flow between the chatbot, Discord, and Petals. Importantly, how the Discord API listens to events such as when a message is sent in a server or when a student reacts to an existing message.

4.4 Chatbot prefix

We have experienced through testing that it is important to give the bot a prefix that describes both the purpose of the bot and its intended behavior. We achieve this by writing a descriptive prefix such as friendly AI, as shown in Code 4.3 below.

We can instruct the bot to have a certain behavior that also adds additional characteristics that may be desired. In this case, by instructing the bot to be friendly, we experience a kind bot that greets you and asks if there is anything the bot can help with. Adding this prefix is quite useful, as it dramatically impacts the behavior of the bot and allows us to tailor the be-

```
with model.inference_session(max_length=512) as sess:
    prefix = f"{context}Human: {inputs_discord}\nFriendly AI:"
    prefix = tokenizer(prefix, ...
        return_tensors="pt")["input_ids"]
```

Code 4.3: Illustrates how Petals can be used to give a descriptive prefix to a chatbot, which helps the chatbot understand its behavior. For instance, we can instruct the chatbot to be a "friendly AI". This is an effective solution that allows us to customize the chatbot according to our needs.

havior to our needs. We want an AI chatbot that is kind, helpful, and polite when talking to students, which is why it is important to use a prefix to further assist the bot by instructing it how to behave. Furthermore, during testing, we noticed that not using a prefix has a significant impact on the type of language and overall behavior of the bot. When not giving the bot a prefix, we experienced some cases where the bot generated inappropriate content, such as profanity or adult content, that most certainly does not belong in a chatbot intended for students.

4.5 Ollama

We believe that Petals has promise, and the solution to running large language models by using a network of computers is quite clever. However, we have encountered some issues while working on our chatbot using the Petals framework.

One of the main problems we faced was the incompatibility of different Python versions with the Petals client. While the client works with an older version of Python, updating to a newer version resulted in several issues. We have spent a considerable amount of time deleting and reinstalling Petals packages to make it work on the latest version of Python. Additionally, after installing the necessary packages, running our Petals script on a new computer was quite challenging. This is problematic when we attempt to run the chatbot on other computers than our laptops.

Furthermore, the Petals framework has limited documentation and has been challenging to work with. We also found that the speed of Petals was not up to our expectations, which is a motivation for us to explore alternative ways of running large language models.

We want to give the bot course-specific information in order to make it more targeted towards the web programming course and make it more useful in that aspect compared to already existing chatbots. We think this feature will add value to our chatbot and make it a valuable resource as this is a feature other available chatbots do not have. In other words, we want to give the bot context to questions students have that are closely related to the course, such as assignment deadlines and exam information. The petals client does not seem to support this feature and we have not found a good solution to integrate course information as context. Therefore, we introduce Ollama as an alternative to running large language models using the same language model (Llama2), with greater support for this feature to be implemented.

Ollama is an API server and a centralized hub for running large language models on your own hardware [30]. It is a system that shares some similarities with Petals in regards to making it easy to run large language models locally. However, unlike Petals, Ollama runs all parts of the model locally using the available GPU and memory resources. This is a larger and more popular system with over 20,000 community members. Ollama offers great support for a heap of different language models, including Llama2 which is the model we use with the Petals client. Ollama offers various packages that can be downloaded in the same fashion as Petals, but you can also run language models using a terminal like Bash or Windows Powershell. This is simple and only requires a few commands to get started. By running a language model using a terminal, it will run on localhost:11434 by default [31].

Ollama also offers to interact with models using an application programming interface (API) such as OpenAI, allowing you to efficiently send requests and receive responses in JSON format [32]. Generally, Ollama offers greater flexibility compared to Petals.

4.5.1 Ollama interface

When we communicate with the Ollama REST API, we create an HTTP client and send a POST request. The following URL is to Ollama set up at the University of Stavanger:

```
curl https://canvas.xini.no/api/create -d '{
  "name": "llama7B_with_context",
  "modelfile" : "$(<modelfile)"
}'
```

Code 4.4: This is the API-endpoint we use to create a model with a modelfile where you can have system information.

```
curl https://canvas.xini.no/api/generate -d '{
  "model": "llama7B_with_context",
  "prompt": "Why is the sky blue?"
}'
```

Code 4.5: Here is an example on the api-endpoint we use to communicate with the language model. The generation will return in JSON format for each generation.

```
        async with aiohttp.ClientSession() as session:
            async with ...
                session.post(f'{self.api_url}/api/generate', ...
                headers=headers, data=data) as response:
                if response.status == 200:
                    result = ""
                    async for line in response.content:
                        decoded_line = ...
                            line.decode('utf-8').strip()
                        if decoded_line:
                            message_data = ...
                                json.loads(decoded_line)
                            result += ...
                                message_data.get("response", ...
                                "")
```

Code 4.6: Here is our async function for creating a http request where we send the same type of request as in Code 4.5.

In Code 4.6 we create a POST request with the aiohttp library. We then send the input from our question to where we run Ollama. Then we store the responses we receive as JSONs in a variable called result if the response status is OK.

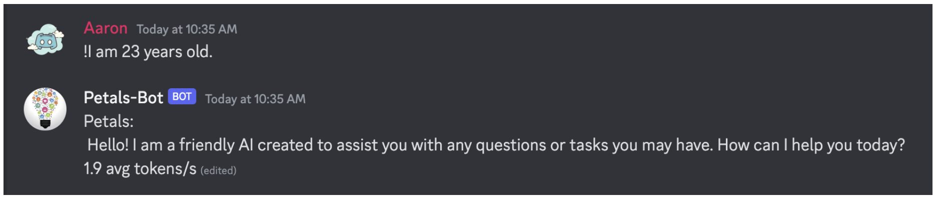
4.6 Added important features

We implement a new feature to enhance the functionality and user experience of the chatbot. This feature allows the chatbot to have context in conversations, which means it can remember previously exchanged messages between the bot and the student. We use this feature by clicking the reply button on a previously sent message to follow up on a question. If the reply button is not utilized, the chatbot will not take into account previously sent messages by default. To ask a follow-up question and provide the chatbot with context it is crucial to use the reply functionality. We refer to this feature as *conversation context* and achieve it by using a prefix, similar to the way chatbot behavior is described in section 4.5. The prefix contains the previously sent messages between the chatbot and the student. The next time a question is asked using the reply button in Discord the conversation history is added as a prefix and sent along with the new question. The prefix allows the chatbot to consider all previously discussed information before generating a new response. The prefix is also additive, which means the prefix grows in size by adding more content as the conversation continues.

The following subsection demonstrates how to use this feature in Discord. Additionally, it shows the difference in responses when using the context feature versus not using it.

4.6.1 Conversation context

We start the conversation by using an exclamation mark followed by some information we want the chatbot to remember later in the conversation, as shown in Figure 4.5.

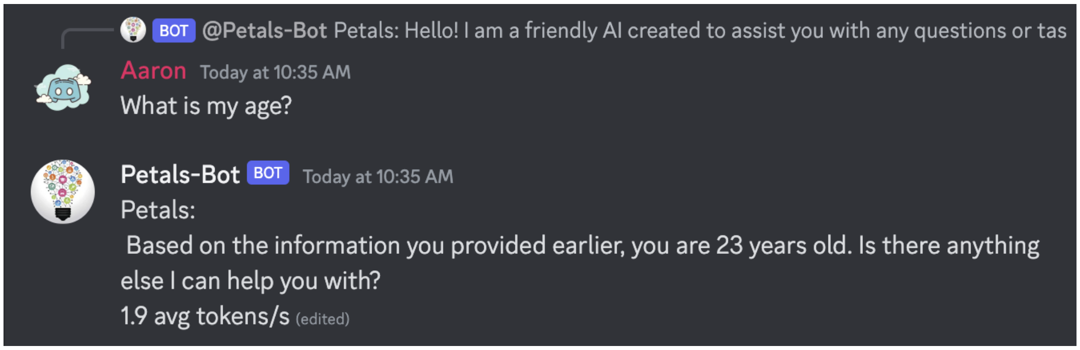


Aaron Today at 10:35 AM
!I am 23 years old.

Petals-Bot BOT Today at 10:35 AM
Petals:
Hello! I am a friendly AI created to assist you with any questions or tasks you may have. How can I help you today?
1.9 avg tokens/s (edited)

Figure 4.5: Shows an example where we initiate a conversation by informing the chatbot about my age.

The chatbot generates a generic, but friendly response, as instructed by its descriptive prefix. We continue the conversation by asking the chatbot about my age, as illustrated in Figure 4.6. We use the reply button in Discord to give the chatbot context about the beginning of the conversation.



BOT @Petals-Bot Petals: Hello! I am a friendly AI created to assist you with any questions or tas

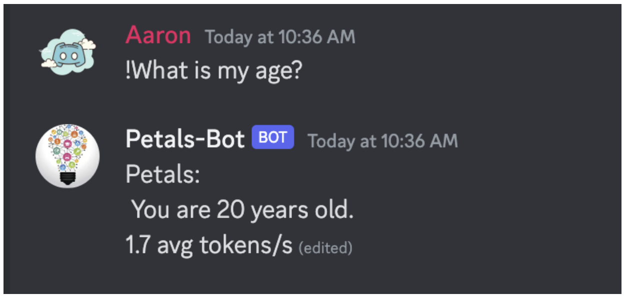
Aaron Today at 10:35 AM
What is my age?

Petals-Bot BOT Today at 10:35 AM
Petals:
Based on the information you provided earlier, you are 23 years old. Is there anything else I can help you with?
1.9 avg tokens/s (edited)

Figure 4.6: We test the chatbot's ability to answer questions related to previously given information using the reply function in Discord. Note that the chatbot provides the correct age.

The chatbot is able to accurately respond to the question by generation the same information we provided earlier in the conversation. This works as expected because the age we provided the chatbot with earlier in the conversation is stored as a prefix and used in the generation process.

Figure 4.7 demonstrates how we ask the same question without using the reply button in Discord. This is further indicated by using an exclamation point, followed by a question. This tells the chatbot that the new message is about some other topic and is a seperate conversation.



Aaron Today at 10:36 AM

!What is my age?

Petals-Bot BOT Today at 10:36 AM

Petals:

You are 20 years old.

1.7 avg tokens/s (edited)

Figure 4.7: We ask the same question as in Figure 4.6, but without using the reply button in Discord. Here, the chatbot has no idea what the correct answer is and compensates by making up a number.

As shown in Figure 4.7, the bot suggests that my age is 20, which is incorrect based on the information we provided the chatbot with previously in the conversation. The chatbot has no clue what the right answer is and therefore makes a random guess. The reason the chatbot does not know the answer is that it did not recieve a prefix with any information about my age as we did not use the reply button.

```
def give_context_to_chatbot(self, message) -> str:
    obj = self.return_conversation_object(message)
    if obj: #if it found an object
        tmp_str = "context:\n"
        for conversation in obj.conversations:
            tmp_str += "Friendly AI" if ...
                conversation["is_robot"] else "Human"
            tmp_str += f": {conversation['content']}\n"
        return tmp_str
    return "" # returns an empty string if there is no context
```

Code 4.7: This is the most important function in the message history handler. It makes up what the bot sees so it knows the context of the conversation. Its called if you reply to a Discord message.

Code 4.7 returns the string that is added as input sent to Petals, it looks for previous messages stored in memory and creates a chat log based on the previous messages for that specific conversation.

We looked into the possibility of using a prefix to provide the chatbot with context about the course, including details from exercise slides and exam information. However, there is a limit in regards to the number of input tokens we can use, which is 4096 tokens, as set by the large language model, Llama2 [33]. While we could provide some course information, it would not be possible to include all the details from presentation slides, exercises, and other sources. The information combined would exceed the limit of the accepted number of tokens as input.

4.6.2 Performance tracker

In order to evaluate if the Petals system is a viable option for the web programming course, we implement some logic that logs data to a separate file. This data includes the date the data was logged, the language model that was used, the time it took to establish a route, and the number of tokens generated per second.

```
1               tracker = PerformanceTracker()
2               tracker.get_∆_and_from_initial_time()
3               tracker.get_average_token_speed()
4               performance_logger.info("petals, tokens/s %s, ...
                    initial time %ss", tracker.average_speed, ...
                    tracker.pathfinding_time)
```

Code 4.8: This code snippet shows the essential part of the logging system. We initialize the tracker by calling the function which starts counting. After the route has been established by the Petals system the tracker calls the 2nd line to update the time. This way we know the time spent by the Petals system to establish the connection.

Code 4.8 above illustrates how we have calculated the results we present in Chapter 5. We have a performance tracker that we call at the beginning of the script. This tracker is first updated after the Petals system has found and established a connection to the other nodes in the network. When this is achieved, the tracker time is updated. This allows us to calculate what we refer to as the *initial time* spent before generating the tokens. The last line in Code 4.8 shows how we log the data to a separate file.

4.6.3 Rating logger

Discord offers the ability to react to messages by using emojis, such as a thumb's up or thumb's down. We take advantage of this feature by creating a small script that registers these emojis by logging each reaction to a separate file. Each message has a unique id, a score that is initially set to zero, and this score will increment by one when a student reacts to a message by using a thumbs up emoji.

Code 4.9 shows the logic we implement in order to know which answers the students prefers. The script only checks if the emoji is equal to a thumb's up emoji. If students use any other emojis, the script does not register this as a rating. Upon registering that a student clicked the thumb's up emoji, the script finds the question ID and checks if the message originated from Petals or Ollama. There is a variable that keeps track of the score for both systems. Each time a student reacts with a thumb's up emoji the score is incremented by one.

```
        if reaction.emoji == "thumbs-up-emoji":
            qa = ...
                answer_rater.find_question_answer_by_id(reaction.message.id)
            if qa:
                # print("found right", reaction, ...
                    reaction.message.id)
                # print(qa)
                if reaction.message.id == qa.ollama_id:
                    qa.ollama_likes += 1
                else:
                    qa.petals_likes += 1
                # print(f"likes on ...
                    ({reaction.message.id}): Ollama: ...
                    {qa.ollama_likes}, Petals ...
                    {qa.petals_likes}")
                answer_rater.store(qa)
```

Code 4.9: Shows how our rating system works. The script starts by checking if the reaction to a answer is a thumb's up emoji. If true, it will find which question the rating belongs to by searching for the message ID. Upon finding the correct ID, it increments the score by one.

4.7 Giving the chatbot context

The Ollama framework has great support for customizing the model by allowing us to create a custom model file. According to the Ollama documentation, the model file is defined as *the blueprint to create and share models with Ollama [34].* In other words, we can change the model to be more familiar with the web programming course by giving it course-related information regarding the exam project.

We believe this feature is essential, as we established earlier in this thesis that there are several free and popular language models online. Therefore, we need our chatbot to have certain qualities that ChatGPT or Gemini does not offer to make it a more attractive option.

Code 4.10 below demonstrates how we create a custom model file that reads a markdown file that is stored locally on the computer that runs the program. Essentially, this script reads the contents of any specified markdown file and adds the content to the global model file for both the 7B and 70B versions of the language model. This way, we have the general language model with the added knowledge of topics related to what the markdown file contains.

This script shown in Code 4.10 creates a language model that has knowledge about the exam regarding the web programming course, making it possible for the chatbot to answer questions students have about the exam without creating as much false information. Furthermore, this is an important feature, as we have noticed through testing that the chatbot tends to make up information when it does not know the right answer. Without the markdown file as context, the chatbot will answer questions about the exam by creating false, misleading, or very generic information that may not be true or useful. This is another reason why including a custom markdown file is important.

```
def read_and_add(f):
    global modelfile_70B, modelfile_7B
    text = f.read()
    modelfile_7B += text
    modelfile_70B += text

# they have different header
with open("create_context_for_ollama/start_7B_modelfile", ...
    "r") as f:
    modelfile_7B = f.read()
with open("create_context_for_ollama/start_70B_modelfile", ...
    "r") as f:
    modelfile_70B = f.read()

modelfile_7B += data[0].page_content
modelfile_70B += data[0].page_content

# end part of file
with open("create_context_for_ollama/base_end", "r") as f:
    read_and_add(f)

response = client.create(model='llama7B_with_context', ...
    modelfile=modelfile_7B)
response = client.create(model='llama70B_with_context', ...
    modelfile=modelfile_70B)
```

Code 4.10: The code snippet shows how we automate the creation of the custom model file. It reads the markdown file with the information related to the course and adds the contents to the already existing more general model file for both the 7B and 70B models. Lastly, the script creates the models we have customized to fit our needs.

This is created for effectivity and make its easy to work with two files when it comes to changing the content of the modelfile like the info of the exam and keeping the parts that needs to be different in the two modelfiles.

Chapter 5

Performance results

This chapter presents how we evaluate the chatbot and discusses the results we have gathered. We also introduce running Ollama on the Unix machines at the University, which consists of high-end equipment that we use to compare the Llama2 model using 7B and 70B parameters. Finally, we present the feedback we have received from students who have tested our chatbot.

5.1 How we evaluate the chatbot

We believe that in order to evaluate the performance of the AI chatbot we have created, we need to divide the evaluation criteria into two main categories. First, we evaluate the performance of the chatbot by strictly looking at the data we have collected by conversing with the chatbot over time. This includes gathering the data we have logged and calculating some helpful performance indicators related to token generation speed. Secondly, we evaluate the overall quality of the chatbot, which includes the length and correctness of the answers from the chatbot in addition to the overall user experience.

Measuring the quality of the response from the chatbot is difficult, as this is subjective and there is not necessarily one right answer. Students most likely have different opinions of what a satisfactory response looks like. However, we attempt to evaluate the quality by receiving feedback from students using our rating system introduced in Chapter 4. In April 2024, we add the chatbot to the web programming course taught at the University of Stavanger, along with a description of how the chatbot works. When students use the chatbot, they are advised to rate their preferred answer.

5.2 Token generation results

As previously mentioned, we have logged data in a separate file containing information about the initial time it takes Petals to establish a connection, the number of tokens generated, and the time spent generating tokens. Using this data, we can calculate an important performance metric, which is the average tokens generated per second for both Petals and Ollama. Figure 5.1 presents the results based on the data we have collected. The red color represents Petals, and the green color represents Ollama.

The average tokens generated per second for both models are calculated using approximately 40 measurements for Petals and 40 measurements for Ollama. The logged measurements for Ollama are the results of running the large language model Llama2-7B on one MacBook Pro (2020) equipped with an Apple M1 processor and 8GB RAM and one MacBook Air (2022) with an Apple M2 processor and 8GB RAM. The measurements for Petals are obtained by running the large language model Stable Beluga 2 (70B) using the public swarm. Because the process of running the LLM is performed remotely, we do not have information regarding the hardware used, as that information is not disclosed publicly because of its peer-to-peer nature.

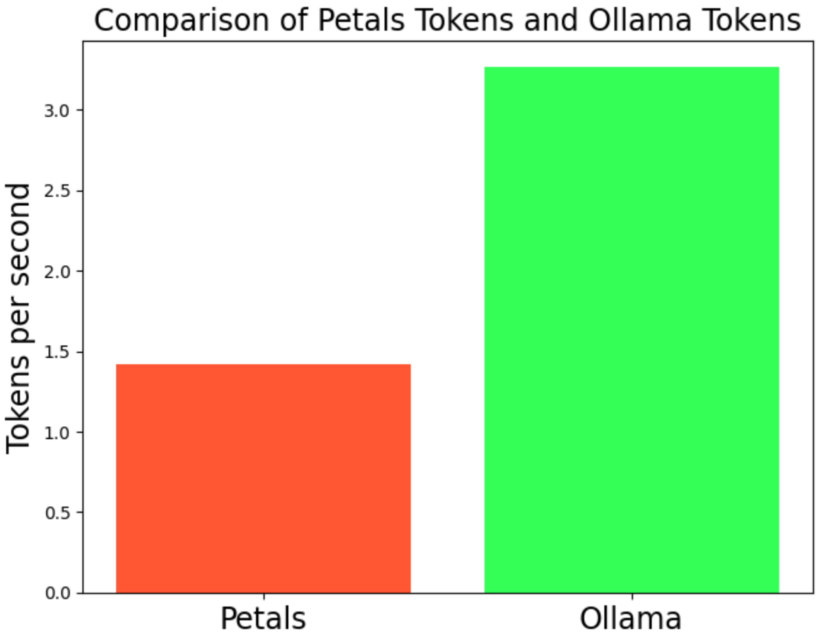


Figure 5.1: The graph shows the average tokens generated per second calculated from logged data over a short period of time. The average is calculated based on approximately 40 measurements for Petals using the LLM Stable Beluga 2 (70B) and approximately 40 measurements for Ollama using the LLM Llama2-7B. A few measurements that had very large variation have been removed.

At first glance, we can see from Figure 5.1 that Ollama is significantly better at generating more tokens per second, making it more efficient and desirable in an interactive setting such as a chatbot. The average tokens generated per second for Petals is 1.4, while Ollama generates around 3.4 tokens per second. In other words, this means that Ollama, on average, generated 2 additional tokens per second compared to Petals. This is significant as 2 tokens per second quickly add up over time and make a big difference when looking at the length and detail of the response generated by the two different systems. The response generated by Ollama is faster, includes more words, and provides a better interactive experience as a result.

5.3 Efficiency

An important quality of a good chatbot is efficiency. We want the chatbot to be able to respond to students as quickly as possible without sacrificing the quality of the response. Furthermore, we do not want the student to have to wait too long before the answer is displayed on the screen. Therefore, we must investigate the time efficiency for Petals and Ollama. We do this by comparing the time spent before tokens are generated. Admittedly, this is a slightly unfair comparison, as the two systems are built differently, and Petals is at a significant disadvantage. By default, Petals will spend more time establishing a connection before it can start generating tokens. This is because Petals needs to find enough servers in the public swarm and establish a connection.

There is no doubt that comparing Petals and Ollama using what we described above as initial time is unfair, but it is necessary to investigate the initial time the systems need before the token generation starts. After all, this has a direct impact on the user experience. We do not want students to sit and wait for a response for too long. Figure 5.2 below shows the average time spent before token generation calculated using logged data in the same fashion as in Section 5.1.

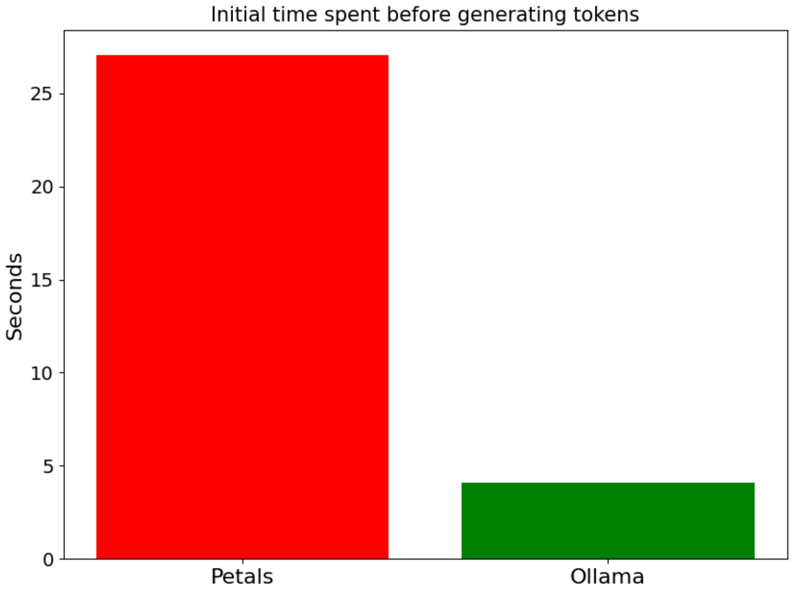


Figure 5.2: This figure shows the time (on average) spent for both systems before the first tokens are sent to the Discord server. For both Petals and Ollama, the time is calculated by finding the average and removing any extreme values. The average has been calculated using approximately 40 measurements. Note that Ollama ran on a MacBook Pro (2020) with an Apple M1 processor and 8GB RAM.

Practically speaking, the above-mentioned initial time refers to the amount of time it takes for the chatbot to register that a message has been sent in the Discord server, use the contents as input, generate a few tokens, and send these tokens. The process consists of the Petals system having to find several servers in the public swarm that, together, can accomplish the task of running the large language model. Upon finding enough servers, a route to these servers must be found, and the connection has to be established.

Figure 5.2 above shows how the Petals system, on average, needs 27 seconds to fulfill the above-mentioned steps. On the other hand, Ollama, on average, spends 4 seconds to complete its process. This means that the Petals system, on average, spends 23 additional seconds before responding to a student.

Ollama has no requirement to find a server to run the language model or find a suitable route, giving Ollama an advantage when comparing time efficiency. Ollama runs all parts of the language model locally, thereby excluding the need to establish any connections to any foreign servers.

5.3.1 Variation for Petals

A problem we encounter with Petals is the large variation in the time needed to establish a connection with the servers in the public swarm. Figure 5.3 shows a box plot of the data we have collected. We see that we have a very large gap between the smallest value (4 seconds) and the largest value (55 seconds). Consider that this is after removing additional extreme values as well. The unpredictable behavior we experience when using Petals is arguably the biggest issue with the Petals system.

The key factors that impact this time are the number of available servers, the location of these servers, and the number of active users running their language model on the servers. These are factors that we have little control over, and therefore there are no steps we can take to improve efficiency. This is the unfortunate downside of a system like Petals, as it is completely dependent on users who contribute their resources. However, we firmly believe that the Petals system is a solid concept that with more available resources has great potential.

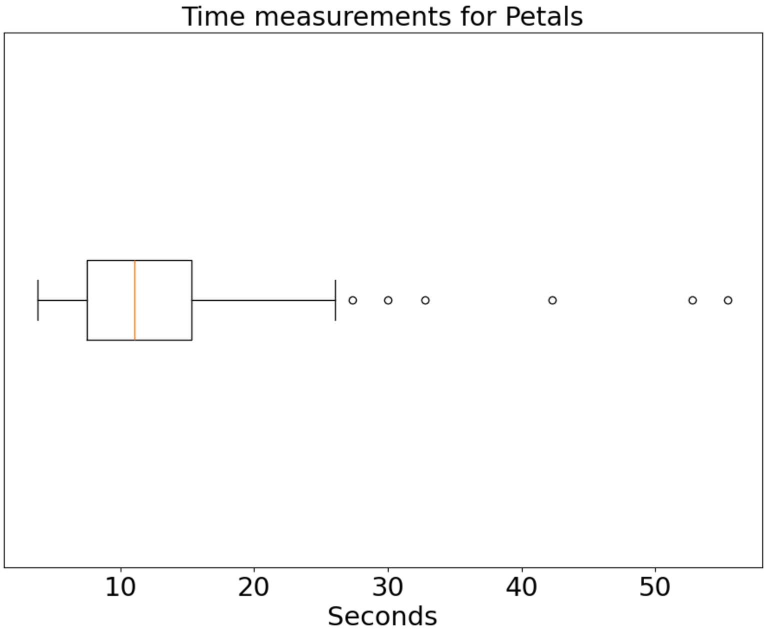


Figure 5.3: Box Plot that shows the variations we experience when running the Petals system. All measurements are related to the initial time the Petals system uses to establish a connection before generating tokens. A total of approximately 40 measurements have been used, and a few extreme valuables have been removed.

5.4 Running Ollama on the Unix system

The University of Stavanger has a system of computers and servers running the Linux Ubuntu operating system. We are able to test both Llama2-7B and Llama2-70B using one of the machines called Gorina7. This machine has the following hardware: 4x NVIDIA A100 40GB graphics cards, 2x AMD EPYC 7302 16-core processors, and 512 GB random access memory. Note that only 1 GPU is utilized during our testing.

We connect to the machine remotely and run the chatbot using more complex and powerful hardware compared to our laptops. From a user perspective, the chatbot works exactly the same, but with a significant increase in performance. A POST HTTP request is sent to an API from the Ollama framework that runs the model on Gorina7, which generates the tokens before sending a response in JSON format. The motivation behind the desire to run these models on this machine is because we want to see the difference in performance with such high-end equipment and compare it to the results we have from running the same models on consumer laptops.

5.5 Gorina7 impact on Petals and Ollama

As previously established, Gorina7 has expensive hardware that we use to test both the Petals system and the Ollama system.

Unfortunately, the hardware Gorina7 has to offer has little impact on the performance of Petals, as the Petals system still relies on remote servers in the public swarm. An option to potentially increase performance for Petals could be to dedicate a part of Gorina7 to the public swarm. We do not necessarily know the extent to which adding a machine like Gorina7 would impact the performance of Petals. However, we know the Petals system is in desperate need of more contributors, and the powerful hardware Gorina7 has to offer would increase performance significantly. As of April 2024, there are only around 4 active contributors, with one individual contributing significantly more than the rest. With this in mind, we imagine adding a machine like Gorina7 would help a lot.

In the case of Ollama, this high-end equipment has a major impact on performance. Unlike Petals, Ollama takes advantage of all the available GPU resources on a machine. Previously, during our testing, we have used laptops where the graphics are integrated into the processor. This is why we expect a very large increase in performance when running Ollama on Gorina7, as this machine has a significantly more powerful dedicated graphics unit installed.

5.6 Comparing parameters

Because running the Petals system on Gorina7 has little impact on performance, we have to make some adjustments accordingly. If the University of Stavanger were to implement our chatbot in the next web programming course, it would have to be simple to do so.

Running Petals using the current system is not a viable solution, as it is simply too slow. However, we want to know if running the 70B parameter language model that Petals allows us to run is significantly better than running Ollama using 7B parameters. This is because running 7B is a lot less resource heavy and can be run on a (high end) consumer PC. Running a simple and less resource-demanding language model like Ollama 7B is arguably easier for the university as it is simple to set up and requires little hardware resources. Thereby still making the hardware available to other students.

With this in mind, we compare the language model by running both Ollama using Llama2-7B and Ollama using Llama2-70B. We compare the large language model with two different parameters, 7B and 70B, in two ways. First, we compare them by performing our own tests where we write multiple questions and manually read the replies. We look for the most obvious difference we can spot. Examples include the length of the response we get in addition to the overall quality of the text. Secondly, we receive feedback from students by deploying the chatbot in the Discord server for the web programming course. We present the feedback we received from students at the end of this chapter.

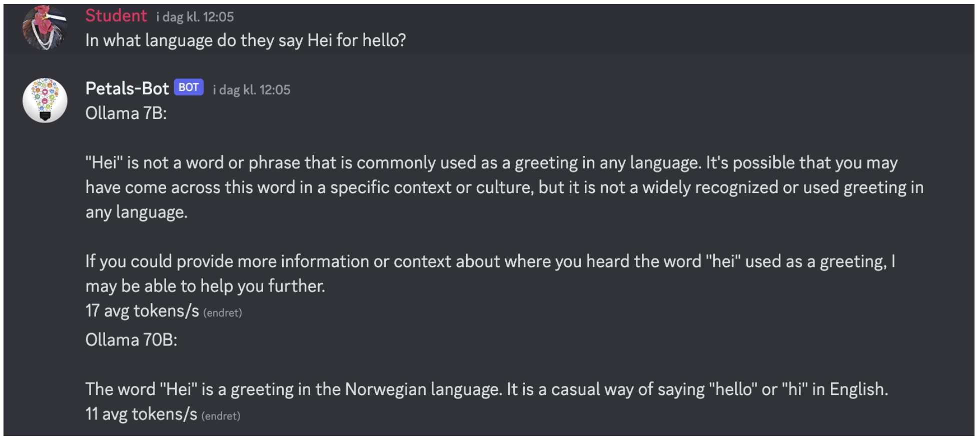
5.7 Comparing parameters results

This section presents two examples that clearly show the difference in behavior we experience when using the same model with different parameters (7B and 70B). After working with large language models for several months, we notice that the following factors vary greatly:

- **Knowledge:** how much information does the chatbot know and give as output to a question.

- **Understanding of language:** how well the bot understands other languages than English, and the ability to reply to questions in different languages.
- **Hallucinations:** does the chatbot make up information if it does not know the answer to a question.

Figure 5.4 shows the first example where we test how knowledgeable the chatbot is. We start by asking in which language the word hei means hello. The language model with 7B parameters is able to pick up the word hei but does not recognize that the word is used in any other languages. Furthermore, the chatbot comes across as quite certain that this is true by stating that this is not a word or phrase that is commonly used. The same language model with 70B parameters is able to identify the exact language where the word hei means hello.



Student i dag kl. 12:05
In what language do they say Hei for hello?

Petals-Bot BOT i dag kl. 12:05
Ollama 7B:

"Hei" is not a word or phrase that is commonly used as a greeting in any language. It's possible that you may have come across this word in a specific context or culture, but it is not a widely recognized or used greeting in any language.

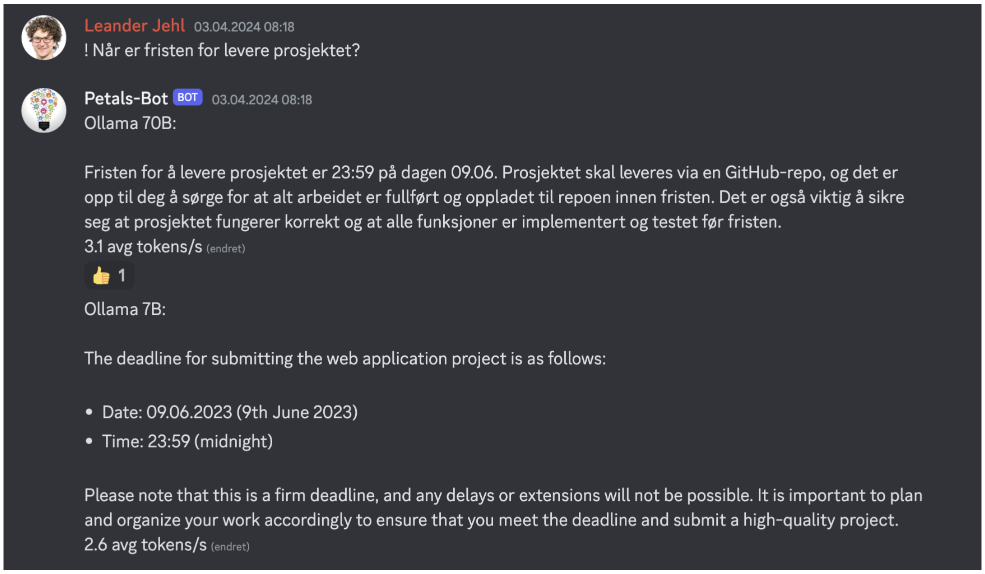
If you could provide more information or context about where you heard the word "hei" used as a greeting, I may be able to help you further.
17 avg tokens/s (endret)
Ollama 70B:

The word "Hei" is a greeting in the Norwegian language. It is a casual way of saying "hello" or "hi" in English.
11 avg tokens/s (endret)

Figure 5.4: The figure shows the difference in behavior we encounter when using the same large language model with two different parameters. The 7B parameter model does not know about any languages that use the word hei as a greeting. The 70B parameter model is able to locate the exact country and language where the word hei is commonly used as a greeting.

Knowledge and hallucinations are two factors that were previously introduced. By looking at Figure 5.4, we can see how these factors vary depending on the parameters the language model uses. The 7B parameter model hallucinates by stating that the word or phrase is not commonly used in any language. Further proving that this version of the model has no knowledge about the question and compensates by creating false information.

Figure 5.5 demonstrates another example where a question is written in Norwegian to test how well the chatbot is able to interpret and write other languages than English. First, notice how the 7B parameter version is able to understand the question but is not capable of writing the answer in the same language. Secondly, the markdown file the chatbot has been provided with does not specify the year of the exam, only the month and date. The Llama2-7B states that the exam date is June 9th, 2023, which is not entirely true. The month and date is correct, but the bot hallucinates by making up the year of the exam. It is also not able to respond to an inquiry using the same language as the question.



Leander Jehl 03.04.2024 08:18

! Når er fristen for levere prosjektet?

Petals-Bot BOT 03.04.2024 08:18

Ollama 70B:

Fristen for å levere prosjektet er 23:59 på dagen 09.06. Prosjektet skal leveres via en GitHub-repo, og det er opp til deg å sørge for at alt arbeidet er fullført og oppladet til repoen innen fristen. Det er også viktig å sikre seg at prosjektet fungerer korrekt og at alle funksjoner er implementert og testet før fristen.

3.1 avg tokens/s (endret)

👍 1

Ollama 7B:

The deadline for submitting the web application project is as follows:

- Date: 09.06.2023 (9th June 2023)
- Time: 23:59 (midnight)

Please note that this is a firm deadline, and any delays or extensions will not be possible. It is important to plan and organize your work accordingly to ensure that you meet the deadline and submit a high-quality project.

2.6 avg tokens/s (endret)

Figure 5.5: The figure shows how the two large language models respond differently to a question they do not know the answer to. The markdown file does not contain information about the year of the exam, only the day and month. The 7B model includes the current year when asked about the exam date, while the 70B model only informs about the day and month.

On the other hand, the language model with 70B parameters is not only able to understand the question asked in Norwegian, but is also capable of generating a response using the same language. Importantly, the chatbot is also able to answer the question by only specifying the month and date. In this situation, the chatbot is precise, able to interpret and write using a foreign language, and does not hallucinate by making up potentially false information. Based on these tests we conclude that the 70B version of Llama2 is a significantly better option than the 7B version of the language model.

5.8 Feedback from students

As previously mentioned, evaluating the quality in terms of how well the chatbot responds is difficult as there are no metrics we can directly compare. We cannot measure how well the chatbot answers, or how detailed or accurate the contents of the message is. Therefore, we use the Unix system to run the model with both 7B and 70B parameters on the Gorina7 machine from April 3, 2024 to April 15, 2024. The chatbot is added to the web programming Discord server, where students have the opportunity to try it out.

This gives us useful feedback from the audience the chatbot is intended for and can help us find areas in which we can improve the chatbot. Importantly, the feedback helps us evaluate if there is a significant difference in the quality produced by the language model using the 7B and 70B parameters.

Each time a student sends a message, the bot will reply with one response from the 7B model and one response from the 70B model. We encourage all students to try the chatbot by asking a few questions and rating the preferred response.

5.9 Feedback Results

The chatbot was available for students to try for approximately two weeks. Unfortunately, the level of interest shown by the students was not very significant. We expected more students to test the chatbot by asking questions, asking follow-up questions, and even testing how many questions the chatbot is capable of answering simultaneously. In total, 38 questions were asked by students, of which only 11 received a rating in the form of thumbs-up emoji. We sent a reminder in the Discord server encouraging students to rate their preferred answer a week after deploying the chatbot. Even though we have little data, there does seem to be a clear preference. Figure 5.6 below demonstrates the ratings we have received from the students.

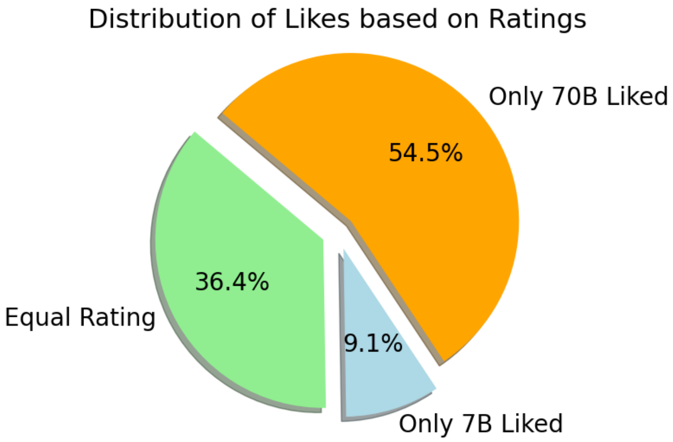


Figure 5.6: Distribution of ratings received from students. Importantly, we only have a total of 11 questions rated by students. The green pie shows the cases where the models got the same rating. The orange pie illustrates when the 70B model was rated 1 while 7B was rated 0. Lastly, the blue pie represents the questions where the 7B model received a rating of 1 while the 70B model received 0.

The results gathered from figure 5.6 is as follows:

- **Llama2-70B preference (54.5%):** A total of 38 questions were asked and answered throughout the two weeks the chatbot was available in the Discord server for students taking the web programming course at the University of Stavanger. 11 replies generated by the chatbot received a rating from students. Each reply consist of one generated answer from the 7B model and one answer generated from the 70B model. 6 out of 11 rated answers shows that Llama2-70B had the most satisfactory reply resulting in 54.5% of the total answers.

- **Llama2-7B preference (9.1%):** There was only one instance where a student preferred the reply generated by the 7B model.
- **Equal preference (36.4%):** A significant amount of answers were given the same rating by students expressing the models equally satisfactory.

The small number of ratings we received makes it difficult to conclude that one model is strictly better than the other. We do not know exactly what the result would be if for example 50 additional students tested the chatbot and gave their feedback. Therefore, with such a small dataset, we can only point out that Llama2-70B seems to be the clear favorite based on the ratings we received. However, im combination with our own tests where we compared the language model using both parameter versions we can clearly define that the 70B version is far superior.

Chapter 6

Discussion

6.1 Petals potential

After an extended period of working with Petals, we believe that the peer-to-peer structure that Petals incorporated is a better alternative than running a small language locally as it allows us to run large language models without expensive hardware. We think this is the case as we have discovered through testing that running a large language model with 70B parameters is significantly better than running a small language model on a laptop.

Allowing students to test the chatbot has in combination with performing our own tests further illustrated the high value of using a language model with 70B parameters. The 70B model is significantly better than the 7B model at interpreting and writing using different languages, and we have observed that it hallucinates a lot less. The chatbot is intended to help students in a Discord server by answering questions about the course. Therefore, it is crucial that the bot only replies to questions when it knows the answer and does not make up information where it lacks knowledge. In chapter 5, section 5.8, we concluded that the 7B model hallucinates a lot more, making it an almost unviable option. This is where Petals becomes highly relevant, as it allows us to run the 70B model without expensive and high-end equipment.

Moreover, Petals eliminates the need for the University to use its limited resources to run a large language model. Earlier in this paper, we presented Ollama, an alternative option to run large language models using expensive hardware. While offering better performance results, it also demands that expensive hardware be available during the entire time the LLM is running. In other words, Unlike Ollama, Petals does not require the University to dedicate a portion of the available resources to run Ollama exclusively. In this way, the hardware can be used by other students.

6.1.1 Petals drawbacks

Providing the chatbot with information on course topics is challenging due to Petals' P2P structure. The system is designed in a way that prevents us from feeding the language model with a lot of data, as this would directly impact other users in the public swarm. The only option available to us is to use external systems to add the amount of data we require, as Petals only offers the opportunity to fine-tune the model, which is not what we are trying to achieve. This is why we did not use the prefix to give Petals the context about the course of web programming. It would also have required that the information would be given every time a message is sent.

One of the main objectives of this paper was to explore the possibility of integrating course-related information into our chatbot. However, due to limitations just mentioned, we concluded that it cannot be achieved without the help of additional systems, which would make it more complex and redundant. While we came across a few frameworks such as LangChain that could read files like presentation slides and exercises which could have been added to the prefix. We decided not to pursue that route as it involved technical complexities and was a time-consuming task.

Based on our experience, we found the Petals framework quite challenging to work with. While working on the project, we faced continuous issues due to working with different versions of the Python programming language, even though the developers claimed that the Petals system supports those versions. Additionally, there were problems with a new Petals version not functioning correctly. We believe that the documentation for getting started with Petals is somewhat responsible for these issues. An improvement in the areas mentioned above would significantly improve the user experience when using the Petals system.

Furthermore, Petals is pretty slow, so if it were to be used as something else than a last resort it would need to be faster, i.e. more people sharing their hardware. Also in our experience, we have found that the client can be quite hard to work with, and with a steep learning curve, so an improvement here would also make Petals a lot more viable.

6.2 Final thoughts

Upon reviewing the conversation history between the chatbot and students, we identified a few areas where we could have improved.

We noticed that some students were seeking more information related to the course, such as questions about the exam project, previous exam project examples, and solution guides. We would have liked to provide students more information regarding previous exam project examples. However, this is ultimately not something we can control as we do not have access to the necessary documents.

6.2.1 What we cold have done differently

We observed that some students had difficulty continuing the conversation with the chatbot. Specifically, they were unsure how to use the reply function to follow up on questions. To address this, we could perhaps make the reply functionality more intuitive.

Whenever the chatbot is added to a new Discord server, it is granted permission to access all active channels like any regular user. To add the chatbot to a specific channel within a server, we have to manually remove it from all other channels. This task is time-consuming and tedious, particularly when deploying the chatbot on larger servers. Although it may not affect the chatbot's performance, we believe that automating this process would be beneficial if the University decides to use the chatbot in future courses. One possible solution to this issue is to create code that removes the chatbot's access to all channels and then creates a new channel with a descriptive name dedicated to the chatbot. However, because we did not prioritize the development of this feature, we did not commit to implementing it.

6.3 Llama 3

As a final note, we would like to briefly mention the relase of Llama 3 as this might impact the performance for both Petals and Ollama.

Llama 3 was released on April 18, 2024. Currently, there is an 8B parameter version and a 70B version of the model available. The creators of Llama 3, Meta, are also working on a 140B parameter model. According to Meta, the goal of Llama 3 is to be the best open model, on par with the best proprietary ones [35]. The creators used a dataset that was seven times larger than the one used for Llama 2 [36]. This new dataset had four times more code [36]. Furthermore, doubled the context length, resulting in an 8000-token context length. These numbers look very promising and could have a significant impact on the performance of both Petals and Ollama. However, since it's new, it's too soon to tell how much of an impact it will have on these systems. Nevertheless, it might significantly increase the performance of Petals, and we would certainly investigate this if we had more time.

Chapter 7

Conclusion

In this project, we investigated Petals, introduced how the Petals system works, and the benefits and drawbacks of Petals. Furthermore, how we used the system in combination with Discord to create an AI chatbot for the web programming course at the University of Stavanger. We introduced Ollama, an alternative solution to run large language models locally. We illustrated a walk through that shows what a typical conversation looks like and how to use the most important features the bot has to offer. Moreover, presented several features we implemented to increase the user experience.

Importantly, we discussed how we evaluated both Petals and Ollama by investigating token generation speed, efficiency, and overall quality of the output generated. We also investigated the difference we experience between the 7B and 70B versions of the large language model Llama2.

We deployed our chatbot (Ollama version) in this semester's web programming course Discord server and allowed students to test the chatbot for approximately two weeks and gathered as much data as possible to aid us in evaluating the performance of the chatbot. We presented the gathered data and discussed our findings which shows that the majority of students prefer the 70B parameter version of the large language model Llama 2. This is likely because the 70B parameter version of the large language model hallucinates significantly less, has a greater understanding of languages, and generally generates more cohesive and higher-quality output.

Conclusion

Finally, we conclude our thesis by briefly looking at the objectives we introduced in the first chapter (1.2) of this paper. We asked the following questions:

1: How can we combine Petals and Discord to create a chatbot?

A: We combine Petals and Discord by installing the necessary Python packages. These packages make it possible to utilize the frameworks for both Petals and Discord. We build on these packages by writing several scripts that are located in our GitHub repository. We take advantage of the Discord interface by using an API to communicate between the Petals chatbot and Discord server.

2: How do we evaluate the usability and performance of the chatbot?

A: We evaluated the usability and performance of the chatbot by logging data we used to calculate several important metrics, including tokens per second and initial time. We further evaluated by comparing the results we got from running the chatbot locally using the Petals framework with the results we got from running the chatbot on the Unix system using the Ollama framework.

3: Can we further evalute the chatbot by implementing it in the web programming Discord server and receive feedback from students?

A: Yes, we deployed the chatbot in the web programming Discord server for roughly two weeks before gathering all the results. We evaluated which version of the language model the students preferred by generating one response for the 7B version and one for the 70B version of the model for each question. Students gave feedback through a rating system, and the preferred choice was the 70B parameter language model.

4: Can we provide the chatbot with course-related information such as topics taught throughout the course and exam details?

A: Using Ollama, we did manage to give course-specific context like information about the exam project as context. We achieved this by customizing the model file. This was however not possible with Petals because the peer-to-peer nature of the system does not allow us to make such large adjustments. Lastly, we did not find a solution to implement the entire curriculum including exercises and presentation slides as context for neither system.

5: Does the bot prove useful compared to already existing chatbots?

  A: With regards to the information about the exam project the bot provided something other chatbots can not provide. Other than that when considering the quality of answers, time it takes to generate a response with Petals and the ease of using it through Discord, we would say its less convenient than other available chatbots.

In conclusion, we have determined that while Petals can have some potential for a Discord chatbot, it is ultimately not a suitable option for a chatbot designed to answer questions about a course. The slow speed, challenging framework, and inability to feed the language model with course information make it unsuitable for this purpose.

Bibliography

[1] N. Forman, J. Udvaros, and M. S. Avornicului, “Chatgpt: A new study tool shaping the future for high school students,” *International Journal of Advanced Natural Sciences and Engineering Researches*, vol. 7, no. 4, p. 95–102, May 2023. [Online]. Available: https://as-proceeding.com/index.php/ijanser/article/view/562

[2] J. Liu, X. Kong, F. Xia, X. Bai, L. Wang, Q. Qing, and I. Lee, “Artificial intelligence in the 21st century,” *IEEE Access*, vol. 6, pp. 34 403–34 421, 2018.

[3] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, “A survey of large language models,” 2023.

[4] S. R. Choi and M. Lee, “Transformer architecture and attention mechanisms in genome data analysis: A comprehensive review,” *Biology*, vol. 12, no. 7, 2023. [Online]. Available: https://www.mdpi.com/2079-7737/12/7/1033

[5] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun, “Parameter-efficient fine-tuning of large-scale pre-trained language models,” 2023. [Online]. Available: https://doi.org/10.1038/s42256-023-00626-4

[6] S. AlZu’bi, A. Mughaid, F. Quiam, and S. Hendawi, “Exploring the capabilities and limitations of chatgpt and alternative big language models,” vol. 2, p. 28–37, Apr. 2023. [Online]. Available: https://ojs.bonviewpress.com/index.php/AIA/article/view/820

[7] GitHub, "The world's most widely adopted ai developer tool." 2024. [Online]. Available: https://github.com/features/copilot

[8] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, "Generative ai," 2024. [Online]. Available: https://doi.org/10.1007/s12599-023-00834-7

[9] D. team, "Introducing duolingo max, a learning experience powered by gpt-4," 2023. [Online]. Available: https://blog.duolingo.com/duolingo-max/

[10] "Posten," 2024. [Online]. Available: https://www.posten.no/

[11] A. Kaushal and K. Mahowald, "What do tokens know about their characters and how do they know it?" 2022. [Online]. Available: https://arxiv.org/pdf/2206.02608

[12] OpenAI, "Models," 2024. [Online]. Available: https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo

[13] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Petals: Collaborative inference and fine-tuning of large models," *arXiv preprint arXiv:2209.01188*, 2022. [Online]. Available: https://arxiv.org/abs/2209.01188

[14] BigScience, "A one-year long research workshop on large multilingual models and datasets," 2022, https://bigscience.huggingface.co/ [Accessed: 07.02.24].

[15] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "bigscience-workshop (github)," 2024. [Online]. Available: https://github.com/bigscience-workshop/petals

[16] L. Rosencrance, "What is peer-to-peer (p2p)?" 2023. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/peer-to-peer

[17] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Petals - getting started with llama 2 and stable beluga 2 (gpu colab)," 2022. [Online]. Available: https://colab.research.google.com/

drive/1uCphNY7gfAUkdDrTx21dZZwCOUDCMPw8?usp=sharing#scrollTo=02d0BDEAuUFQ

[18] Petals, "Petals health monitor." [Online]. Available: https://health.petals.dev/

[19] "Greedy algorithms," 2024. [Online]. Available: https://www.geeksforgeeks.org/greedy-algorithms/

[20] P. V. Platen, "How to generate text: using different decoding methods for language generation with transformers," 2020. [Online]. Available: https://huggingface.co/blog/how-to-generate

[21] "Text generation strategies," 2024. [Online]. Available: https://huggingface.co/docs/transformers/generation_strategies#default-text-generation-configuration

[22] "discord-bot-petals," 2024. [Online]. Available: https://github.com/AJ-A-dev/discord-bot-petals

[23] P. Developers, "petals 2.2.0.post1," 2023, https://pypi.org/project/petals/.

[24] A. Borzunov, "Security, privacy, and ai safety," 2024. [Online]. Available: https://github.com/bigscience-workshop/petals/wiki/Security,-privacy,-and-AI-safety

[25] Discord, "About discord," 2024. [Online]. Available: https://discord.com/company

[26] ——, "Building your first discord app," 2024. [Online]. Available: https://discord.com/developers/docs/getting-started

[27] "Discordpy documentation," 2024. [Online]. Available: https://discordpy.readthedocs.io/en/stable/api.html

[28] "Gateway," 2024. [Online]. Available: https://discord.com/developers/docs/topics/gateway#gateway

[29] "The websocket api (websockets)," 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[30] "Ollama github," 2024. [Online]. Available: https://github.com/ollama/ollama/tree/main

[31] langChain, “Ollama,” 2024. [Online]. Available: https://python.langchain.com/docs/integrations/llms/ollama

[32] J. Morgan, “Api,” 2024. [Online]. Available: https://github.com/ollama/ollama/blob/main/docs/api.md

[33] H. Face, “Llama2.” [Online]. Available: https://huggingface.co/docs/transformers/main/model_doc/llama2

[34] J. Morgan, “Ollama documentation,” 2024. [Online]. Available: https://github.com/ollama/ollama/tree/main/docs

[35] “Introducing meta llama 3: The most capable openly available llm to date,” 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/

[36] Meta, “Build the future of ai with meta llama 3,” 2024. [Online]. Available: https://llama.meta.com/llama3/