Tampere University

Cecylia Borek

COMPARATIVE EVALUATION OF LLM-BASED APPROACHES TO CHATBOT CREATION

Implementing a Death Doula Chatbot

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
January 2024

Abstract

Cecylia Borek: COMPARATIVE EVALUATION OF LLM-BASED APPROACHES TO CHATBOT CREATION - Implementing a Death Doula Chatbot
Master's thesis
Tampere University
Master's Degree Programme in Software, Web & Cloud
January 2024

The recent emergence of Large Language Models (LLMs) significantly improved the performance of AI conversational assistants. Creating task-specific chatbots became more accessible with techniques such as Retrieval Augmented Generation (RAG) and fine-tuning. Despite this, the area of creating LLM-based chatbots remains relatively unexplored.

This thesis aimed to compare various approaches to creating LLM-based assistants. Additionally, the objective was to compare the impact of RAG and fine-tuning on the style of responses generated by the model. Finally, this thesis investigated if LLMs are suitable for creating a personal and trustworthy death doula chatbot.

The study encompassed the implementation of a death doula chatbot with four different approaches and an evaluation from the developer and end-user perspective.

The thesis results revealed that open-source approaches, such as fine-tuning Llama2 or using a vector database to build custom RAG, offer the highest level of security, flexibility and control. On the other hand, using well-known platforms, such as OpenAI, is easier and requires no specialised knowledge. Fine-tuning is a better approach for creating chatbots with personality and combining a fine-tuned model with RAG yields the best results. Finally, it was proved that using a fine-tuned LLM, augmented with RAG, holds the potential for creating a personal death doula chatbot with LLMs.

Keywords: Large Language Models, LLM, conversational AI, chatbot, death doula, end-of-life care, personalized chatbot, fine-tuning, Retrieval Augmented Generation

The originality of this thesis has been checked using the Turnitin Originality Check service.

Preface

This thesis has been developed between October 2023 and January 2024, in the realm of the Erasmus Traineship programme at the University of Salzburg, Austria.

I would like to thank my supervisor at University of Salzburg, prof. Wolfgang Pree, for welcoming me to the university, suggesting the area of research and providing guidance throughout implementation and writing of the thesis. I would also like to thank dr. Pia Niemelä, my supervisor from Tampere University, for offering advice on writing the thesis and the formalities related to it.

Furthermore, many thanks to Symon Braun Freck, who contributed her expertise in thanatology and end-of-life care to the project. Her insights made the implementation and evaluation of a death doula possible.

Special words of appreciation go to my family and my boyfriend for their constant support throughout the process of writing this thesis and my entire course of studies. This journey has been a great adventure, and I am excited about what lies ahead.

Salzburg, Austria, 31 January 2024

Cecylia Borek

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

LLM Large Language Model.

LLMs Large Language Models.

LoRA Low-Rank Adaptation.

NLP Natural Language Processing.

PEFT Parameter-Efficient Fine-tuning.

RAG Retrieval Augmented Generation.

UX User Experience.

Glossary

API Application Programming Interface, a software interface facilitating communication between multiple programs.

JSON JavaScript Object Notation, open standard file and data interchange format utilizing human-readable text to store and transmit data objects, comprising attribute–value pairs and arrays.

llama Large Language Model Meta AI, a family of large language models released by Meta AI starting in February 2023.

SDK Software development kit, a package of development tools, including a compiler, debugger, and sometimes a software framework, designed to streamline the creation of applications.

SLA Service level lgreement, a contract between a service provider and a customer where aspects of the service, such as quality, availability, and responsibilities, are mutually agreed upon..

SSO Single sign-on, an authentication method enabling a user to log in with a single ID across multiple related, independent software systems.

thanatology The scientific study of death, encompassing mechanisms, forensic aspects, and broader psychological and social dimensions associated with mortality.

1 Introduction

1.1 Motivation

The origins of machine learning and neural networks date back to the 1950s when Alan Turing discussed the notion of machines learning from experience [45], Arthur Samuel developed a computer program for playing checkers and coined the term ”machine learning” [42], and Frank Rosenblatt developed an artificial neuron with the ability to learn: the Mark I Perceptron [41].

Since then, machine learning, and deep learning in particular, has achieved great results, often outperforming humans, in areas such as Natural Language Processing (NLP), computer vision, medicine, biology, image generation, playing games, robotics, and many others [14].

In recent months, Large Language Models (LLMs), such as ChatGPT, BERT [7], or LLaMa [44], have achieved state-of-the-art results in the domain of NLP and have revolutionized many other domains. Those models, based on the transformer architecture, have been trained on web-scale generic language data and can understand natural language instructions and generate human-style responses.

Due to their ability to respond in a human-like fashion, LLMs have made significant advancements in the conversational AI field, greatly improving the performance of chatbots and virtual assistants [47]. It has also become much easier to build specialized chatbots by fine-tuning pre-trained generic large language models to downstream tasks [8] or using Retrieval Augmented Generation (RAG) to enhance the generic models with field-specific knowledge [21].

The emergence of new techniques for designing custom conversational assistants has opened the door to a variety of methods and services dedicated to creating chatbots based on LLMs [9] [32] [17] [29] [34] [30]. Many developers, some unfamiliar with LLMs, now face a diverse array of tools as they undertake the challenge of constructing an AI chatbot.

Moreover, LLMs unlock substantial potential for crafting personalized, informed, and reliable chatbots, a realm that remains relatively unexplored thus far.

The objective of this thesis is to assess different methodologies accessible to developers in the creation of specialized chatbots. Through the evaluation of factors such as implementation simplicity, costs, and customization options, the aim is to empower developers to make more informed and pragmatic decisions when selecting an approach to develop chatbots, focusing on representative and popular choices or prevalent categories in the field.

Additionally, this thesis aims to assess the distinct impacts that fine-tuning and

RAG have on the conversational style of the generated chatbot.

The additional objective of the thesis is to implement a death doula chatbot based on LLM. A death doula is a trained professional who provides emotional, spiritual, and practical support to individuals and their families as they navigate the end-of-life process [25]. Given the novelty of end-of-life care as a field and the delicacy surrounding the subject matter, harnessing the potential of LLMs to construct specialized and reliable death doula chatbots holds significant promise.

1.2 Research Questions

Primary research question (RQ1)

How do different approaches to creating AI conversational assistants such as:

- Fine-tuning an open-source llama model on a private machine using Hugging Face libraries,
- Fine-tuning a GPT-3.5 model using the OpenAI API,
- Building a knowledge-enhanced assistant using the Fixie.ai platform,
- Implementing a knowledge-enhanced assistant using the Weaviate vector database,

compare in terms of implementation ease, customization capabilities, and overall?

Secondary research question (RQ2)

How does the conversational style of an AI assistant differ when employing fine-tuning as opposed to retrieval augmented generation?

Tertiary research question (RQ3)

Can Large Language Models be leveraged to create trustworthy and personal death doula chatbots?

1.3 Scope of Work

The scope of this thesis involves the evaluation of different approaches to creating Large Language Model (LLM) based chatbots, grounded on the implementation of an end-of-life care assistant.

Chapter 2 encompasses an exploration of key terms and concepts within the realms of Natural Language Processing, Large Language Models, AI assistants, fine-tuning, and Retrieval Augmented Generation.

Chapter 3 outlines the methodologies employed in this thesis to assess various techniques for developing chatbots, considering both the developer and end-user perspectives. Evaluation from the developer's standpoint involves scrutinizing factors such as ease of implementation, associated costs, scalability, flexibility, security, and other pertinent considerations. On the other hand, the examination from the end-user perspective centres on the performance of the generated assistants. This includes a qualitative evaluation conducted by an expert in thanatology, complemented by a quantitative analysis utilizing BERTScore.

Chapter 4 presents implementation details of all four approaches used to create a death doula chatbot.

Chapter 5 presents the evaluation outcomes of the four distinct methods employed in the development of a death doula chatbot within the scope of this thesis.

Chapter 6 displays the implications of this study. It presents developers with guidelines to assist them in selecting an appropriate method for crafting chatbots based on LLMs. It discusses the impacts of retrieval augmented generation versus fine-tuning on conversational style and tone of an AI assistant. It further explores the impact of various models and methodologies on the efficacy, helpfulness, and suitability of generated assistants. Ultimately, the conclusions highlight considerations for establishing reliable assistants, including those in sensitive roles such as a death doula, through the utilization of LLMs.

Finally, Chapter 7 summarizes the work by presenting answers to research questions stated in Section 1.2, mentions the limitations of this thesis and proposes areas for further research.

2 Background and Related Work

This chapter provides a foundational understanding of key theoretical concepts and explores the existing advancements in the fields of Natural Language Processing (NLP), Large Language Models (LLMs), AI assistants, fine-tuning methodologies, and Retrieval Augmented Generation (RAG).

2.1 Natural Language Processing and Language Models

Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) that focuses on the interaction between computers and human language. It involves the development of algorithms and computational models to enable machines to understand, interpret, and generate human language in a way that is both meaningful and contextually appropriate. NLP encompasses various tasks such as language comprehension, sentiment analysis, language translation, and speech recognition, aiming to bridge the gap between human communication and machine understanding. [18]

Language models are machine learning models within the field of NLP that are designed to understand, generate, and predict human language patterns. They produce human-like language by determining the next most likely word, based on the preceding sequence of words. For example, assuming the following sentence:

```
It's a cruel ____
```

the model will try to predict the next word by calculating the probability of different words occurring in the place of the underscores:

```
summer 10%
world 8.9%
reality 7.7%
joke 7.3%
fate 6.9%
twist 6.2%
game 5.7%
```

and selecting the most probable one. This simple example assumes the model operates on words, however, most language models operate on tokens. A token is a basic unit of text and can be a word, a part of a word, or even a single character. Words and sentences are split into tokens in a process called tokenization, which is specific to the language model.

Large Language Models (LLMs), currently the most advanced language models, have been trained on vast, web-scale amounts of data and learned billions of parameters during the training. While early language models were able to predict the

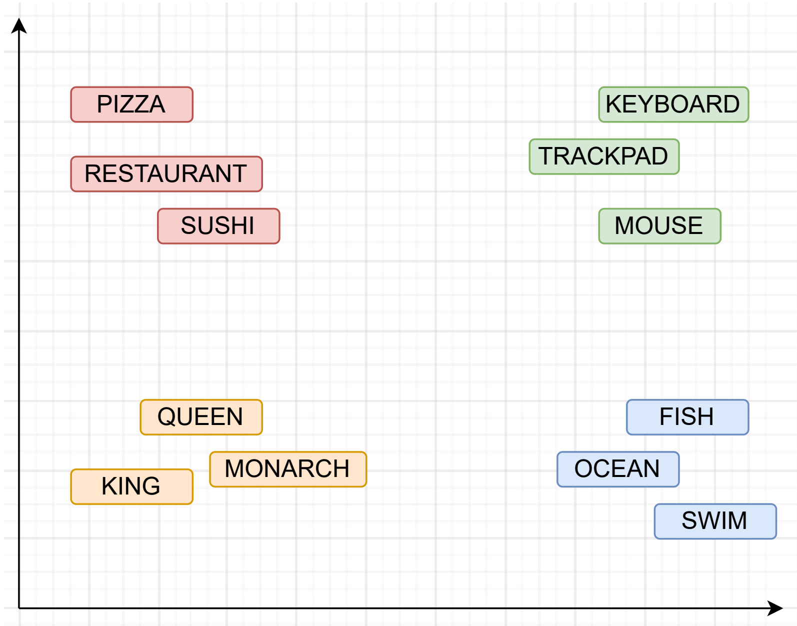

Figure 2.1 *A visualisation of a two-dimensional embedding space. Words with similar meanings are close to each other, while words semantically different are more distant.*

next word in a sentence, LLMs can predict whole sentences and paragraphs. LLMs achieve general language understanding and can generalize to different tasks they have not been trained on.

2.1.1 Word Embeddings

When modelling and processing human language, it is important to capture the semantic representations of the words. A pivotal advancement in that realm was the incorporation of word embeddings. A word embedding is a vector representation of a word in a continuous vector space, which captures its semantic relationship to other words. In a well-trained embedding space, similar words are positioned close to each other.[18] For example, in a good embedding, ”king” and ”queen” would be nearby, indicating their semantic similarity, while ”king” and ”mouse” would be more distant. A visualisation of a two-dimensional embedding space can be seen in Figure 2.1

One of the biggest advantages of embeddings over previously used techniques, was the representation of words as short, dense vectors, instead of long and sparse ones. Embeddings’ dimensions range from 50-1000, while older techniques used vectors with dimensions of the vocabulary size. Dense vectors proved to work better for all NLP tasks than sparse vectors did. [18]

Word embeddings are created in a training process, during which a machine learning model learns vector representations for words based on their contextual usage in a large corpus of text. One of the first models for creating word embeddings was the Word2Vec model [31], introduced in 2013 by researchers at Google. Word2Vec model produces static embeddings - a single word will always be mapped to the same vector. More recently, BERT[7] representations have been used to learn dynamic, contextual embeddings. This means that depending on the context in which the word is used, it will be represented with a different vector.

2.1.2 The Transformer Architecture

One of the main breakthroughs in language modelling which lies at the core of all modern state-of-the-art LLMs, was the introduction of the transformer architecture and the mechanism of self-attention [46] in 2017.

Self-attention mechanism enables models to weigh the importance of different elements in a sequence by measuring the similarity between elements and determining how much attention each element should give to others. This allows the model to capture intricate relationships and dependencies across the entire input sequence, enhancing its ability to understand and generate coherent and context-aware outputs. In the case of NLP, the self-attention mechanism can model relationships between all words in a sentence.

The transformer architecture, at its core, leverages self-attention mechanisms to process input sequences. Unlike traditional sequential models, the transformer processes all elements in a sequence simultaneously, thus allowing it to handle long-range dependencies efficiently. It employs an encoder-decoder structure, with the encoder specializing in input representation and the decoder responsible for generating output. The transformer is highly parallelizable, making it computationally efficient, and it excels at handling input sequences of varying lengths. This architecture serves as the foundation for state-of-the-art models in machine translation, text generation, and many other NLP tasks.

2.1.3 Large Language Models

Two of the first models to leverage the transformer architecture were BERT (Bidirectional Encoder Representations from Transformers) [7] and GPT (Generative Pretrained Transformer) [39]. Developed by Google, BERT is primarily a representation model, used to understand the context of words in search queries. It is pre-trained on a large corpus of text and then fine-tuned for specific tasks. On the other hand, GPT, developed by OpenAI, is both a representation and a generative model. It is capable of generating coherent, contextual, and natural-sounding responses, making it suitable for a wide range of tasks, from generating creative content to answering questions in a helpful and informative way.

The surge in language modelling advancements has fueled a competitive landscape among tech firms, each striving to develop more sophisticated LLMs. Consequently, a plethora of models has emerged, offering diverse options for consideration.

OpenAI has continued to advance the GPT series, with GPT-3.5 being released in 2022 and GPT-4 on March 14, 2023. On November 30, 2022, OpenAI released ChatGPT - a chatbot based on the GPT-3.5 model, fine-tuned for conversational applications. It gained significant attention from the general public and became the fastest-growing consumer application in history [16]. At the time of writing this thesis, ChatGPT is constructed based on either GPT-3.5 or GPT-4.

Another noteworthy family of large language models is LLaMA (Large Language Model Meta AI) [44] and its updated version LLaMA 2 [43], released by Meta AI in 2023. Unlike GPT, LLaMA 2 models are open source - the trained model weights, inference-enabling code, training-enabling code, and fine-tuning enabling code might be downloaded and used for free for research and commercial use[1]. This makes it particularly useful for developers wanting to run, train or fine-tune the models on their infrastructure for free.

Google's first family of pre-trained generative models was LaMDA (Language Model for Dialogue Applications), announced in 2021. Its successors, PaLM (Pathways Language Model) [36], was released in March 2023, and PaLM 2 was announced in May 2023. In December 2023, Google introduced Gemini [38] - a family of models which are inherently multimodal, capable of seamlessly integrating and processing diverse forms of information, including audio, images, videos, codebases, and text in multiple languages. Bard is a chatbot similar to OpenAI's ChatGPT, which allows to interact with Google models.

Anthropic[2], an AI safety and research company, has also developed a series of LLMs. The first model, Claude, was introduced in 2023. It was followed by Claude 2, which was announced on July 11, 2023. Claude 2 offered improved performance, and longer responses, and was accessible via API as well as a new public-facing beta website. Anthropic prioritizes the universal benefits and safety of AI by implementing safety measures and interpretability features in Claude, accompanied by a detailed policy addressing potential misuse and emphasizing ethical and responsible AI usage.

2.1.4 Large Language Models Glossary

Several terms are often used in regards to LLMs. This subsection presents some of the most popular ones and their definitions.

[1] Following the license agreement available at https://github.com/facebookresearch/llama/blob/main/LICENSE.

[2] `https://www.anthropic.com/`

- **Prompts, prompting, and prompt engineering.** A prompt is the natural language input that is provided to the LLM. Thus, prompting is the action of querying a LLM with natural language. Prompt engineering is a set of techniques, which try to obtain the best possible results for a specific task, by modifying the input prompt. [26]

- **System Prompt.** A system prompt is a special prompt that is always attached to the model input and gives the model general instructions for the tasks it should perform[3], e.g., "You are a helpful assistant. Respond with short sentences.".

- **Hallucination.** LLMs have been trained on vast amounts of data and possess broad general knowledge. However, sometimes they present incorrect information as factual. This phenomenon is often referred to as hallucination. [4]

- **Inference.** The term inference refers to the process of using the pre-trained model to generate specific outputs for given inputs. It is often used to distinguish this process from the training phase.

- **Temperature.** A temperature of the model is a real numerical value from $0 - 1$, which determines how random the output of the LLM will be. The lowest temperature 0 means that the model always chooses the next most probable token, while higher temperatures introduce more randomness, which results in more diverse and creative answers[4].

- **Grounding.** Grounding is the process of supplementing the models with use-case-specific information that is pertinent but not inherently part of their pre-trained knowledge, to ensure more accurate and contextually relevant language generation. Retrieval Augmented Generation (RAG, see Section 2.5) is often used to ground the models.

2.2 Conversational AI and Large Language Models

Conversational AI stands as a specialized field within Artificial Intelligence, harnessing the capabilities of NLP and machine learning to develop speech or text-based assistants and chatbots capable of engaging users in a manner that closely mimics human interaction. These conversational AI assistants, due to their natural language interfaces, have garnered widespread adoption across diverse sectors, including healthcare, customer service, and education. [19]

[3] `https://www.promptingguide.ai/introduction/basics`
[4] `https://www.promptingguide.ai/introduction/settings`

Large Language Models are ideal for crafting conversational assistants as they possess the ability to comprehend human language and instructions, seamlessly generating text in a manner that closely resembles human communication. They can be utilized to create general-purpose chatbots such as ChatGPT, however, more often they are used to create task and field-specific chatbots. To date, the medical sector is one of the main domains where LLMs have been applied to craft knowledgeable and trustworthy chatbots [22] [5] [24].

Multiple strategies exist for developing conversational assistants based on LLMs:

- **Using the model as is with specified system prompts.** This approach, often the most economical and straightforward, involves deploying the model as it comes and adding task-specific instructions to the system prompt. While cost-effective, it proves sufficient for many basic applications.
- **Fine-tuning.** Fine-tuning is particularly valuable when there is a need to influence the style, tone or format of responses [5]. A description of fine-tuning and its applications for LLMs can be found in section 2.3.
- **Retrieval augmented generation (RAG).** Tailored for knowledge-intensive tasks, RAG is adept at providing up-to-date information and leveraging domain-specific knowledge extracted from vast datasets. A detailed description of RAG can be found in Section 2.5.

It is important to note that these approaches are not mutually exclusive and can be combined to meet specific requirements.

2.3 Fine-Tuning

Fine-tuning, a common transfer learning practice in machine learning, involves the adjustment of pre-trained models to better suit specific tasks or domains [11]. In the context of Large Language Models, fine-tuning refers to the process of adapting a pre-trained model to perform specific Natural Language Processing tasks with a relatively small amount of task-specific training data. Large Language Models, such as BERT or GPT, are initially trained on extensive text corpora and acquire a broad understanding of language. [7] [3] [39] Fine-tuning, however, customizes the model's knowledge for particular applications, like text classification or language generation, by updating its parameters based on task-specific data. This process leverages the model's pre-existing linguistic knowledge and tailors it to excel in domain-specific or task-specific challenges, making it a versatile and powerful tool for various NLP tasks.

[5] https://platform.openai.com/docs/guides/fine-tuning/when-to-use-fine-tuning

2.4 Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-tuning (PEFT) is a set of different techniques that allow for efficient fine-tuning of Large Language Models to downstream tasks. Full fine-tuning requires loading all the model parameters and the respective gradients for each parameter during the tuning process. Additionally, a tuned copy of the whole model needs to be stored for each downstream task. With the growing size of the LLMs (7B - 70B parameters for Llama 2 [43], 175B for GPT-3 [3], and 540B for PaLM [33]), this becomes very expensive and often unattainable. PEFT approaches adapt only a small percentage of the model's parameters, freezing the rest. As a result, fine-tuning can be performed on consumer hardware for a significantly lower cost than full fine-tuning. Despite this, PEFT approaches yield results comparable to full fine-tuning. Additionally, using PEFT allows for better portability of the models. The fine-tuned weights are small and are added on top of the base pre-trained LLM, which may be reused for different downstream tasks. [8]

This section gives a short description of different PEFT approaches available today.

2.4.1 LoRA: Low-Rank Adaptation

The Low-Rank Adaptation (LoRA) technique [15], reduces the number of trainable parameters by using a matrix rank decomposition. A rank of a matrix M is the number of linearly independent rows or columns of a matrix [1] and is denoted as $rank(M)$. Let's assume a pre-trained model, with pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$ and updates to its weights for a downstream task stored in the update matrix $\Delta W \in \mathbb{R}^{d \times k}$. Because adapting a model to a specific task only changes a few features, the authors of the LoRA technique assume that the update matrix ΔW can be a low-rank matrix. This means the update matrix will have many linearly dependent columns that may be removed without losing any information:

$$rank(\Delta W_{d \times k}) \ll min(d, k)$$

The LoRA method uses rank decomposition to reduce the rank of the update matrix. It represents the update matrix ΔW as a product of two low-rank matrices: $A_{r \times k}$ and $B_{d \times r}$, where $r \ll min(d, k)$. During fine-tuning, only A and B are trained, while the pre-trained weight matrix W_0 is frozen. The fine-tuned model weights are achieved by merging W_0 with $\Delta W = BA$. A graphical representation of the LoRA technique can be seen in Figure 2.2.

The LoRA method has several practical advantages over full fine-tuning and other parameter- and compute-efficient model adaptation methods:

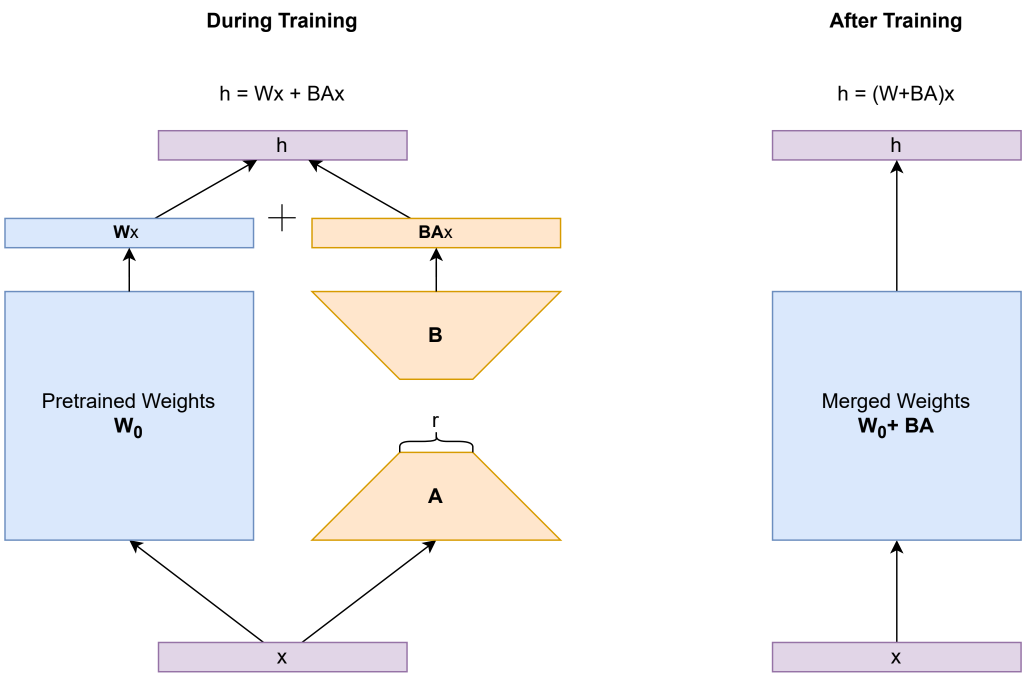

Figure 2.2 *Representation of the LoRA technique. During training, the pre-trained weights of the model W_0 are frozen, and only A and B are trained. After the training, W_0 and $\Delta W = BA$ are merged.*

- **Fast and space-efficient fine-tuning.** LoRA method only trains matrices A and B, therefore only $r \cdot (d + k)$ parameters need to be trained as opposed to $d \cdot k$ during full fine-tuning. Since $r \ll min(d, k)$, this is a significant improvement.

- **No additional latency during inference.** The weights of the fine-tuned model can be simply calculated and stored as $W = W_0 + \Delta W$, and the inference can be run as normal. This is an advantage over one of the other strategies for efficient fine-tuning of LLMs: adding adapter layers to the base model, described in Subsection 2.4.3.

- **Easy switching between tasks.** The original weight matrix W_0 can be recovered by subtracting BA and new $B'A'$ for different downstream tasks can be added instead. This operation is very quick and cost-efficient.

- **Efficacy comparable to or better than full fine-tuning.** The authors of the LoRA paper have shown that the model fine-tuned with the LoRA technique yields better or comparable results to fully fine-tuned models on different NLP benchmarks.

2.4.2 Prompt Tuning

Prompt tuning is a family of parameter-efficient fine-tuning techniques [23] [27] [20] inspired by prompting. Prompting uses natural language instructions to query a language model and guide it into the correct execution of a specific task, without the need to change the model's parameters [3] [26]. The basic concept of prompt tuning assumes changing the input prompt to achieve a better output. For example, for a text summarization task, different prompts might be tried:

```
1) Summarize the following text: {text}.
2) Text: {text} | Summary: {summary}.
3) The summary of the following text {text} is ...
```

and evaluated based on the results they yield. The technique illustrated with the above example is known as *hard prompt tuning* - the discrete input tokens, i.e. words and sentences are varied. However, because the language models are inherently continuous, in the context of optimization, it is unattainable to reach the optimal outcome using discrete natural prompts [28].

Soft prompt tuning [20] [27], rather than optimizing discrete prompt, introduces a trainable embedding vector P, that is prepended to the input sequence of the model. During the training process, P is trained via backpropagation to maximize the likelihood of producing a desired output for a specific downstream task. After the training, the learned vector P is prepended to each input, before being fed to the base model.

Another flavour of prompt-tuning is *prefix-tuning* [23] [28]. It also introduces a trainable tensor of continuous values, called prefix. However, instead of prepending it just to the embedded input, the prefix is prepended at every transformer layer.

The prompt-tuning methods provide advantages of parameter-efficient fine-tuning:

- **More efficient than full fine-tuning.** Prompt-tuning only adapts the parameters of the prompt or prefix vector, which is 0.1% of task-specific parameters in the case of prefix-tuning [23] and 0.01% task-specific parameters in the case of prompt-tuning [20]. Therefore the tuning process is more time-, resource- and cost-efficient than full fine-tuning.

- **Modularity and flexibility.** The prompt vectors for downstream tasks are lightweight and are prepended to each input of the base, pre-trained model. Therefore it is easy to transfer the task-specific prompts between different machines. It is also efficient to run multiple specialized models at once, using one base model and switching only the small prompt tensors.

- **Performance comparable to full fine-tuning.** The authors of the methods

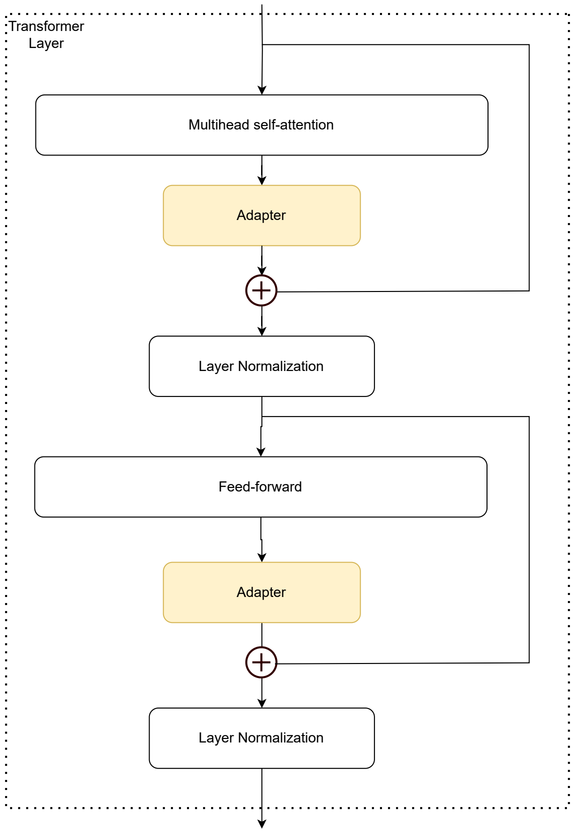

Figure 2.3 *One layer in the transformer architecture with two added adapter modules.*

show that prompt-tuning techniques achieve results comparable to full fine-tuning. [23] [27] [20]

2.4.3 Adapter Tuning

The adapter tuning method [13], adds new adapter modules between the layers of the pre-trained transformer network. In the transformer architecture, each layer contains two main sub-layers: an attention layer and a feed-forward layer. In adapter tuning, a new sub-layer, i.e. adapter, is added after each of the original sub-layers of the transformer, as shown in Figure 2.3. During the fine-tuning process, only the parameters of the adapters are changed, while the base of the model remains fixed.

Similarly to other PEFT methods, adapter tuning is significantly more efficient than full fine-tuning, training only 3.6% of the parameters per downstream task. The authors of the method show that a BERT model fine-tuned with the adapters method achieves performance comparable to a fully fine-tuned BERT model. [13]

2.5 Retrieval Augmented Generation

Another approach for improving LLMs performance in domain-specific tasks is Retrieval Augmented Generation (RAG) [21] [2]. General-purpose language models

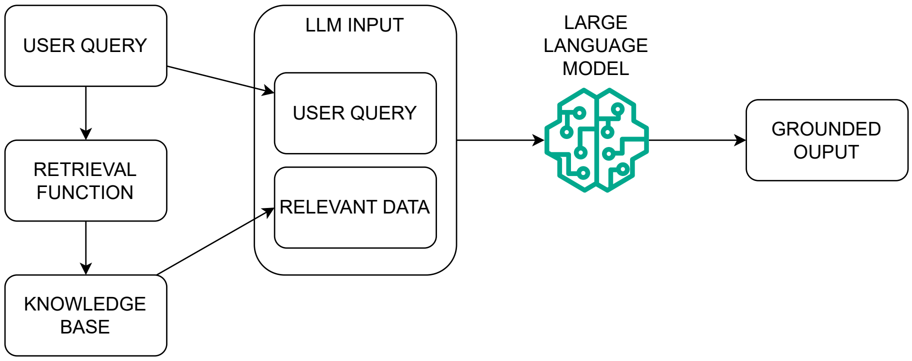

Figure 2.4 *Simple illustration of the retrieval augmented generation flow with a large language model.*

have been trained on large corpora of data, however, they often lack the knowledge of specific fields, users' or companies' private data, or information about recent events. LLMs also tend to hallucinate: they present fictitious information as facts. To overcome these limitations, especially for knowledge-intensive tasks, the RAG framework was introduced. It combines an information retrieval component with a text generator model, such as LLM. The information retrieval part of a RAG system finds relevant information for a given user query in a knowledge base, such as a database, a collection of articles, Wikipedia, etc. That information is then appended to the original user query and fed as input to the LLM. Therefore, the model has context and knowledge to generate grounded and truthful output. The depiction of the RAG flow can be seen in Figure 2.4.

The retrieval part of a RAG system plays a crucial: it should manage large knowledge bases and select relevant information efficiently. The retrieval system first divides all the documents from the knowledge base into small pieces, called chunks, and calculates a vector embedding for each of the chunks. Those embeddings are then usually stored in a vector database, which allows for efficient retrieval. To find information relevant to a given user query, the system calculates an embedding of the query and then calculates the similarity between it and the embeddings of all the chunks in the database. Various similarity metrics can be used, with cosine similarity being a common choice. Based on the computed similarities, the system ranks the chunks in the database in descending order of relevance to the query. Top-k chunks with the highest similarity scores are selected and supplied as input to the generation model. [21]

2.6 Comparison between RAG and Fine-tuning

As of January 2024, there exists little scientific research comparing the performance of RAG and fine-tuning when creating personalised and trustworthy chatbots. The authors of *Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs*

[35] assessed RAG and fine-tuning across a spectrum of knowledge-intensive tasks spanning diverse subjects. Their results demonstrated that, although fine-tuning exhibits certain enhancements, RAG consistently surpasses it in performance, particularly concerning pre-existing knowledge encountered during the training phase and entirely novel information. However, it is important to note that the study did not focus on ”personality”, stylistic nuances of responses or the trustworthiness of the created models. Additionally, the study did not focus specifically on creating LLM-based chatbots.

2.7 Tools

With the rising popularity of LLMs, there are many tools, techniques and methods which allow for creating LLMs-based assistants, using RAG or fine-tuning. The various tools offer different levels of control, and customization, and differ in how easy they are to be used. The following is a non-exhaustive overview of some of the existing methods for creating AI solutions based on LLMs using fine-tuning and RAG.

2.7.1 Fine-tuning Llama2 with Hugging Face Libraries

Fine-tuning of the open-source Llama2 model is not limited to any platform or implementation. However, the most popular option is to fine-tune Llama2 using Hugging Face's libraries. Hugging Face[6] is a company that specializes in creating tools for constructing machine learning applications. The company is particularly renowned for its transformers library, designed for natural language processing applications. Additionally, Hugging Face provides a platform enabling users to exchange machine learning models and datasets, offering a dedicated space for showcasing their projects. Hugging Face offers an implementation of the LoRA technique which can be used for fine-tuning Llama2. All LLama2 models are hosted on Hugging Face's Hub and can be downloaded from there.

2.7.2 OpenAI Platform

The OpenAI Developer Platform[7] allows developers to build and use different AI solutions that are based on GPT models. It offers a web-based graphical interface as well as an API. OpenAI Platform allows for fine-tuning GPT models and creating RAG-based solutions. The platform contains tutorials, documentation, and a developer forum. To use the developer platform, one needs to create an account and obtain an API access token. It is also necessary to add some credit to the account

[6] `https://huggingface.co/`

[7] `https://platform.openai.com/overview`

as all the solutions are paid with a pay-as-you-use schema. OpenAI also provides custom Python and Node.js client libraries which grant convenient access to the OpenAI REST API.

2.7.3 Fixie.ai

Fixie.ai[8] is a cloud-hosted Platform-as-a-Service for building applications which use large language models. It provides support for using OpenAI, Anthropic, Llama2 or custom models. The platform offers a ready-to-use, customizable RAG solution. Creating assistants and solutions on Fixie.ai can be done either through the web interface, a command-line interface, or a set of JavaScript interfaces. To use the Fixie.ai platform, one is required to set up an account and pick one of the monthly subscriptions, of which the basic one is free. Fixie provides documentation of its solutions and an online community chat for users and developers.

2.7.4 Vector Databases

Creating a knowledge-enhanced assistant can be done by implementing a custom RAG system, using a vector database. A vector database is a specialized type of database designed for the efficient storage, indexing, and retrieval of vector embeddings. Therefore, it is often used in RAG solutions to efficiently store the embeddings of the knowledge pieces and retrieve the ones relevant to a given query by performing a similarity search. Some of the most popular vector databases at the time of writing this thesis are Pinecone[9], Milvus[10], Weaviate[11], Chroma[12], and Qdrant[13]

As of January 2024, to the best of the author's knowledge, no scientific papers have been published that comprehensively compare various tools for developing AI conversational assistants.

2.8 Death Doula

A death doula, also known as an end-of-life doula, is a supportive companion who offers holistic care to individuals facing terminal illness or nearing death. Beyond medical assistance, they provide emotional, spiritual, and practical support, helping

[8] `https://www.fixie.ai/`
[9] `https://www.pinecone.io/`
[10] `https://milvus.io/`
[11] `https://weaviate.io/`
[12] `https://www.trychroma.com/`
[13] `https://qdrant.tech/`

individuals and their loved ones make informed decisions in a compassionate environment. Death doulas normalize conversations about death, promoting increased communication and empowering individuals to plan for their end-of-life wishes. Their role extends to light grief support for family and loved ones. [25]

As of January 2024, to the best of the author's knowledge, no attempts, whether scientific or non-scientific, have been made to develop a death doula chatbot using LLM.

3 Research Methodology

To assess various methods and approaches in developing AI conversational assistants, a chatbot specializing in end-of-life support, akin to a death doula, was implemented utilizing diverse available methodologies. The subsequent evaluation scrutinized these methods from two distinct standpoints:

1. Developer perspective: This involved an assessment of the techniques and services employed in constructing the AI conversational assistant.
2. End user perspective: This focused on evaluating the conversational style and efficacy of the assistant in assisting.

The aim of the evaluation from the developer's perspective was to answer the primary research question (RQ1). The insights gained from the end user perspective allowed us to answer the secondary and tertiary research questions (RQ2 and RQ3).

This Chapter describes in detail the methods used to evaluate both of the perspectives.

3.1 Development services and frameworks

For assessing different approaches and services for creating AI conversational assistants, human evaluation was performed by the author of the thesis. The evaluation was based on the experience gained while implementing assistants with different methods for the thesis. During the assessment, the following categories were considered:

1. **Ease of Implementation**
   - How user-friendly is the development environment?
   - Are there clear documentation and tutorials available?
2. **Customization Capabilities**
   - How much customization of the fine-tuning process is possible?
   - To what extent can the chatbot be customized to meet specific requirements?
   - Is there support for integrating custom code or external modules?
3. **Scalability**

- How scalable is the solution to handle increased user interactions?
- Does it provide options for scaling up or down based on demand?

4. **Development Cost**
   - What is the pricing model of the service/framework?
   - Are there any hidden costs, and how does the cost scale with usage?
5. **Training and Fine-Tuning**
   - How easy is it to train and fine-tune the chatbot for specific tasks?
6. **Performance Metrics**
   - What are the response time and latency of the chatbot?
7. **RAG (Retrieval-Augmented Generation)**
   - Does the service/framework support retrieval-augmented generation for more contextually relevant responses?
8. **Community and Support**
   - Is there an active community around the service/framework?
   - What level of support is provided by the developers or the platform?
9. **Privacy and Security**
   - What security features are in place to protect user data?
   - Does the solution comply with relevant data protection and privacy regulations?

Each of the methods used for creating a conversational assistant was given a numerical assessment on a scale from $1-3$ (1 - adequate, 2 - good, 3 - excellent), and a written assessment, in each category. The selected methods were not examined in isolation; rather, they were compared against their alternatives.

In conclusion, a mean score was computed from the numerical assessments, consolidating them into a singular rating for each method. Nevertheless, it is essential to note that this consolidated score does not represent an absolute outcome, as certain categories may hold varying degrees of relevance in distinct use cases.

For the *Performance Metric* and *Development Costs* categories, mean latency and mean cost per request were calculated to provide better insights. The calculations were made based on 200 requests sent to the assistants and the received responses. The requests contained prompts from the test dataset, which was the same for all the methods. The latency of a single request was calculated as the time difference between sending the request and receiving the complete response, time measurements were done using Python's `time.time()` [1] function. Requests that time-out or resulted in errors were repeated. The mean latency was calculated as the arithmetic mean of single measurements, using panda's dataframe `mean()` [2] function. The mean costs were determined by dividing the total cost incurred during the testing session by the number of requests sent within that session. In addition to calculating the mean latency and cost, an average token count for both the prompt and assistant responses was also computed. These metrics play a crucial role in the overall measurements, as the cost and latency are significantly influenced by the number of tokens transmitted and received by the assistant.

All evaluations presented in this thesis are based on the state of the methodologies as of late 2023. Given the rapidly evolving nature of the field, the specific details may change, but the general principles underlying the evaluations should persist.

3.2 Conversational style and helpfulness

To comprehensively assess the conversational style and helpfulness of models generated through various techniques, a thorough evaluation has been conducted, encompassing both qualitative and quantitative analyses.

3.2.1 Qualitative Assessment

For the qualitative assessment of the death doula chatbots, an expert review was performed. Expert review is a common User Experience (UX) evaluation method, during which a professional with domain-specific expertise inspects the system to find its strengths and problems [12]. Although the method was originally meant for assessing a user interface, it can be easily adapted to evaluate the conversational style, content accuracy, cultural appropriateness, overall effectiveness, and trustworthiness of a death doula chatbot.

Moreover, human evaluation is frequently regarded as the benchmark for assessing models in natural language generation, particularly dialogue models [43]. The expert review method is especially relevant for assessing the death doula chatbot because of the sensitivity of the topic. Since the target users of the death doula chatbot are people facing death or grieving, it would be hard and unethical to perform user testing of a potentially insensitive system with people who are already emotionally burdened.

[1] https://docs.python.org/3/library/time.html#time.time

[2] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

Additionally, an expert's involvement in evaluating the system offers the advantage of their specialized knowledge in the field. For instance, an expert in end-of-life care contributes unique insights, ensuring accuracy, compassion, and alignment with the nuances of the subject matter.

For reviewing the death doula chatbot, an expert with a Bachelor of Arts in Thanatology and a master's degree in Visual Anthropology was chosen. Their unique academic background, coupled with the active promotion of death positivity and awareness, makes them well-qualified. The expert has authored articles on death-related topics and conducted interviews with death doulas and specialists, offering valuable insights into end-of-life care and grief support.

The expert was asked to perform testing of different versions of the death doula chatbot implemented within the scope of this thesis. During testing, the specialist interacted with the chatbot by asking different questions related to end-of-life topics. Afterwards, the expert wrote a short, free-form review of each evaluated model.

3.2.2 Quantitative Assessment

For the quantitative assessment of the created models, BERTScore was used. BERTScore is a text generation evaluation metric that automatically assesses the quality of the generated text. Similar to conventional text generation evaluation metrics, it calculates a similarity score for every token in the generated sentence in comparison to the reference sentence. Unlike traditional metrics that rely on exact matches, BERTScore leverages pre-trained contextual embeddings from the BERT model and matches words in candidate and reference sentences by their meaning. It has been demonstrated to exhibit a correlation with human assessment at both the sentence and system levels. The outputs of BERTScore are precision, recall, and F1 score values, which are common machine learning metrics, helpful in evaluating language generation tasks.[48] In all three metrics, a higher score means a closer match to the reference.

Because it correlated with human judgment and evaluation based on semantics, BERTScore was a suitable choice for assessing the created death doula models.

A test set containing 200 questions and reference answers, not seen previously by the models, was utilized for the evaluation. The reference answers set the benchmark for the models. BERTScore was used to compare the reference answers with answers generated by different models. Apart from the fine-tuned and RAG-augmented models, the performance of the base models was also evaluated and compared. An implementation of BERTScore[3] from the authors of the BERTScore paper was used.

[3] https://github.com/Tiiiger/bert_score

The hash code of the model used for calculating the scores is `roberta-large_L17_no-idf_version=0.3.12(hug_trans=4.35.2)`.

4 Large Language Model Fine-Tuning and Adaptation

Within the scope of this thesis, four distinct approaches to constructing AI conversational assistants were implemented and subsequently subjected to comparison. These approaches encompassed:

1. Fine-tuning a Llama-2-7b-chat model locally with Hugging Face libraries and hosting the fine-tuned model on a private server.
2. Fine-tuning a gpt-3.5-turbo model through the OpenAI API.
3. Building a knowledge-enhanced assistant using the Fixie.ai platform.
4. Implementing a knowledge-enhanced assistant using the Weaviate vector database.

For all the methods, the same system prompt, developed by a prompt engineer with a background in thanatology, was used.

This chapter describes the implementation details of each approach used to create a death doula chatbot.

4.1 Fine-tuning Llama-2-7b-chat model

The Llama-2-7b-chat model is a 7B parameters model from the Llama 2 family, which has already been optimized for dialogue applications using Reinforcement Learning from Human Feedback (RLHF) by Meta AI. [43] Having only 7B parameters, Llama-2-7b-chat is the smallest model in the family, enabling its fine-tuning and inference on consumer hardware. Despite the small size, the 7B parameters model achieves satisfactory results on standard academic benchmarks, especially Commonsense Reasoning, Reading Comprehension, and Multi-task Language Understanding [43], which are crucial for a conversational assistant.

This approach has been chosen as one of the most popular and most accessible approaches to creating LLM based assistants. Being open-source and offering small-sized models, Llama can be fine-tuned by almost anyone on consumer-grade hardware.

4.1.1 Tools and Libraries Used

- **Python and Jupyter Notebook.** Jupyter Notebook[1] is an open-source interactive web application that allows one to create and share documents that contain live code, equations, visualizations, and narrative text. It supports various programming languages, including Python, R, and Julia. Jupyter Notebooks are favoured in machine learning for their interactive code execution, seamless integration with data science libraries, and the ability to combine code with rich documentation. Therefore it was selected as the platform for fine-tuning llama. Python was selected as the de facto standard for machine learning projects.

- **Google Colab.** Colab is a hosted Jupyter Notebook platform that demands no initial setup and offers complimentary access to computational assets, including GPUs and CPUs. It is often used in machine learning, data science, and educational pursuits [40]. The fine-tuning of the chatbot was performed in Colab Jupyter Notebook on NVIDIA Tesla T4 GPU with 16GB RAM, which is available for free in Google Colab.

- **Hugging Face.** During the fine-tuning process, the implementation of the LoRA technique from Hugging Face's `peft` [2] library was used. Furthermore, we used Hugging Face's `transformers` [3], `datasets` [4], `trl` [5], and `accelerate` [6] libraries and downloaded the base llama-2-7b-chat model already converted to be compatible with Hugging Face's `transformers` library from Hugging Face's Hub.

- **bitsandbytes.** `bitsandbytes` [7] is a library that enables quantization of the models: reducing the precision of numerical values in a model (e.g., from 32-bit floating-point numbers to 8-bit integers), while maintaining acceptable accuracy of the model. With `bitsandbytes`, quantization methods can be integrated into the Hugging Face's `transformers` library, allowing for loading, inferring and fine-tuning pre-trained models in 4-bit or 8-bit precision. During fine-tuning the chatbot, `bitsandbytes` was used to train the model in 8-bit precision.

---

[1] `https://jupyter.org/`
[2] `https://github.com/huggingface/peft`
[3] `https://github.com/huggingface/transformers`
[4] `https://github.com/huggingface/datasets`
[5] `https://github.com/huggingface/trl`
[6] `https://github.com/huggingface/accelerate`
[7] `https://github.com/TimDettmers/bitsandbytes`

4.1.2 Data Collection and Preparation

To fine-tune a chat model, a dataset of high-quality question-answer pairs is needed. For fine-tuning the death doula chatbot, we meticulously curated a dataset containing 1000 sample questions concerning mortality, funerals, grief, and the death doula role along with the corresponding death doula's answers. The dataset was created with the help of a thanatology expert.

The question-answer dataset had to be converted into a format compatible with the Llama-2-7b-chat model. The authors of Llama 2 used the following template for the chat models:

```
<s>[INST] <<SYS>>
{{ system_prompt }}
<</SYS>>

{{ user_message }} [/INST]
```

Therefore, all the question-answer pairs were converted to follow the template. Table 4.1 shows a few samples from the converted dataset.

After converting the dataset into the correct format, it was split into random train and test sets with ratio 8:2, using the `train_test_split`[8] method from the Python machine learning library `scikit-learn`. Finally, the train set was loaded into Hugging Face's `Dataset` type from the `datasets` library, which processes the data and prepares it for training in a deep learning model with `transformers`.

4.1.3 Loading the model

The base llama-2-7b-chat model was downloaded from Hugging Face's hub, in a format already compatible with Hugging Face's transformers library. The model was loaded in 8-bit precision, by providing the `load_in_8bit` argument of the `from_pretrained` method. Loading the model in quarter-precision reduces the memory requirement to only 1/4th of the model size while introducing no performance degradation for inference [6]. Later, an 8-bit optimizer was used to fine-tune the model in 8-bit precision, which significantly reduces memory requirements. Apart from the model, the tokenizer was also loaded. The code for loading the model and the tokenizer can be seen in Listing 4.1

```python
import torch
from transformers import LlamaForCausalLM, LlamaTokenizer

model_id = "meta-llama/Llama-2-7b-chat-hf"
```

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

1	<s>[INST] My mom is dying and I'm so afraid. I don't know where to start [/INST] I'm so sorry to hear about your mom. It's completely understandable to feel afraid during this time. Remember, you're not alone in this. Have you considered seeking support from a local hospice or death doula who can guide you through the process and provide comfort and assistance? Is there anything specific you're feeling unsure about or would like assistance with? </s>
2	<s>[INST] How do I write a living will [/INST] I'm here to help you with writing a living will, also known as an advance healthcare directive. It's an important document to outline your healthcare wishes if you become unable to communicate or make decisions. Here's a basic outline to get you started: 1. Begin with your personal information: Full name, date of birth, and contact details. 2. State your intent: Clearly state that the purpose of the document is to express your wishes regarding medical treatment in case you can't speak for yourself. 3. Appoint a healthcare proxy: Choose someone you trust to make healthcare decisions on your behalf. Make sure they are willing to take on this responsibility. 4. Specify your medical treatment preferences: You can outline your preferences for life-sustaining treatments such as resuscitation, mechanical ventilation, tube feeding, and pain management. 5. Include your organ donation preferences: Indicate if you want to donate your organs and specify any restrictions or preferences you may have. 6. Consider other end-of-life decisions: You can express your wishes about palliative care, hospice care, and funeral arrangements. 7. Make it legal: Consult with a lawyer or use a reputable online resource to make sure your living will is legally valid in your region. Remember, it's important to consult with a legal professional to ensure your living will comply with local laws and regulations. Have you thought about who you would like to appoint as your healthcare proxy? </s>
...	...

Table 4.1 *Two samples from the converted dataset, which was used for fine-tuning the death doula chatbot.*

```
tokenizer = LlamaTokenizer.from_pretrained(model_id)
model = LlamaForCausalLM.from_pretrained(
    model_id,
    load_in_8bit=True,
    device_map='auto',
    torch_dtype=torch.float16
)
```

Program 4.1 *Loading the base model and tokenizer with Hugging Face's transformers library.*

4.1.4 LoRA Configuration

The Llama model was fine-tuned with the parameter-efficient technique, LoRA, described in Subsection 2.4.1. There are several configurable parameters, which control how LoRA is applied to the base model:

- **Rank of Decomposition**. As described in Subsection 2.4.1, LoRA represents the weights update matrix ΔW by a low-rank decomposition BA, where B is a $d \times r$ matrix and A is a $r \times k$ matrix. The rank of the decomposition r, $r \ll min(d, k)$, is one of the configurable parameters, with the default value of 8.

- **LoRA Scaling Factor**. Alpha is a LoRA scaling factor by which the weight update matrix ΔW is scaled.

- **Dropout Probability for LoRA Layers**. Dropout helps prevent overfitting by randomly turning off some neurons with the `lora_dropout` probability during training. This means certain neurons don't contribute temporarily in the forward pass, and they don't get updated during the backward pass.

- **Modules to Apply LoRA to**. The `target_modules` specifies which of the model's modules (e.g., attention blocks) to fine-tune with LoRA.

In the `peft` library, these parameters are set with `LoraConfig`. The parameters selected for fine-tuning the chatbot can be seen in Listing 4.2.

```
from peft import LoraConfig

peft_config = LoraConfig(
    task_type=TaskType.CAUSAL_LM,
    inference_mode=False,
    r=8,
    lora_alpha=32,
    lora_dropout=0.05,
    target_modules=["q_proj", "v_proj"]
)
```

Program 4.2 *LoRA cofiguration used for fine-tuning the death doula chatbot.*

4.1.5 Fine Tuning

Fine-tuning was performed with the `SFTTrainer` from the Hugging Face's `trl` library. `trl` is a full-stack library built on top of the `transformers` library which provides tools to train transformer language models. The `SFTTrainer` is a light wrapper around the `transformers Trainer`, meant for easily fine-tuning language models on custom datasets.

The pre-trained model was fine-tuned for two epochs with the LoRA configuration specified in Subsection 4.1.4, on the dataset described in Subsection 4.1.2. Additional parameters used for the training are depicted in Table 4.2.

Parameter	Value
learning rate	$1e^{-4}$
number of training epochs	2
batch size	2
gradient accumulation steps	2

Table 4.2 *Training parameters used during fine-tuning the chatbot.*

The training took around 34 minutes on an NVIDIA T4 GPU with 16GB of RAM. Because LoRA was used, only around 0.06% of all model parameters were trained during the process of fine-tuning.

4.1.6 Saving the Fine Tuned Model

After the training, the model was saved with the `save_pretrained` method of the `LlamaForCausalLM` model. This method saves the LoRA checkpoint, which contains parameters learned during the fine-tuning process, and in the case of the death doula chatbot was only 16MB.

4.1.7 Running Inference

To serve and run inference on the fine-tuned model, it had to be loaded back from the file it was saved to. Because the saved LoRA checkpoint, also called the LoRA adapter, contains only a small portion of parameters learned during training, first the base Llama model needed to be loaded. It was done using the same method as when loading the model for fine-tuning. Afterwards, the `PeftModel` from the Hugging Face's `peft` library was initialized. The `PeftModel` is created by attaching the LoRA weights learned during training to the base llama model.

Hugging Face's `peft` and `transformers` libraries provide different methods for loading and using peft models effectively. Multiple adapters can be added to the base model with the `add_adapter` method and later specific adapter can be selected for inference with the `set_adapter` method. This is very convenient as it allows one to easily switch between the small adapters, while the base model stays the same. Adapters can be also removed from the base model with the `delete_adapter` method. It is also possible to disable the adapters and run inference on the base model by calling the `disable_adapters` method.

The inference on the fine-tuned model is run the same as on the base model. It is done by first tokenizing the prompt, and then calling the `generate` method of the model, with the tokenized prompt as input. The code used for loading the fine-tuned model and running inference on it can be seen in Listing 4.3.

```
doula_adapter = "./tuned-doula-llama"
base_model_name = "meta-llama/Llama-2-7b-chat-hf"

base_model = AutoModelForCausalLM.from_pretrained(
    base_model_name,
    device_map="auto",
    load_in_8bit=True,
    torch_dtype=torch.float16
)
model = PeftModel.from_pretrained(base_model, doula_adapter)
tokenizer = AutoTokenizer.from_pretrained(base_model_name)


model_input = tokenizer(
    str(request.prompt),
    return_tensors="pt"
).to("cuda")

model.eval()
with torch.no_grad():
    output = tokenizer.decode(
        model.generate(
            **model_input,
            max_new_tokens=512)[0],
        skip_special_tokens=True
    )
```

Program 4.3 *Loading and running inference on the fine tuned chatbot.*

4.2 Fine-tuning gpt-3.5-turbo model

GPT-3.5 (Generative Pre-trained Transformer 3.5) is a family of models developed by OpenAI and first released in March 2022. The gpt-3.5-turbo is, according to the authors, the most capable and cost-efficient model from the GPT-3.5 series, and has been optimized for chat usage. From GPT-3 onward, OpenAI's models are no longer open source, which means they can only be used through OpenAI's API. Additionally, details of the models, such as the number of parameters, are not publicly known. In August 2023 OpenaAI made fine-tuning for GPT-3.5 Turbo available [37].

Fine-tuning gpt-3.5-turbo on the OpenAI Platform has been chosen as one of the most popular ways for creating LLM based chatbots. Offering a graphical user interface and a level of abstraction, it is also an easier approach than fine-tuning Llama.

4.2.1 Tools Used

- **OpenAI Platform.** Described in Subsection 2.7.2.
- **OpenAI Python API library.** Described in Subsection 2.7.2.

4.2.2 Data Collection and Preparation

For fine-tuning the gpt-3.5-turbo model, the same dataset which was used for fine-tuning the llama model was utilized. The dataset was split into random train and test sets with ratio 8:2, using the `train_test_split`[9] method from the Python `sklearn` library. Afterwards, both sets were formatted so that each example followed the convention used in the Chat Completion API, which is also required for fine-tuning. The format can be seen in Listing 4.4. Finally, both the training and testing sets were uploaded to the OpenAI Platform through the Files API, with the Python `openai` library.

```
{
    "messages":
        [
            {"role": "system", "content": <system_msg>},
            {"role": "user", "content": <user_msg>},
            {"role": "assistant", "content": <assistant_msg>}
        ]
}
```

Program 4.4 *OpenAI chat completion format used for the training dataset.*

4.2.3 Fine-tuning

To fine-tune a model, a fine-tuning job was created through the OpenAI Fine-tuning API with the Python `openai` library. The id of the file uploaded in the previous step was provided as the training file and `gpt-3.5-turbo-1106` was selected as the base model. When creating a fine-tuning job, several hyperparameters which will be used for tuning can be selected. Those include batch size, scaling factor for the learning rate, and number of epochs to train the model for. To create the death doula, the number of epochs was specified as 2.

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

4.2.4 Using the Fine-tuned Model

The fine-tuned model was accessed after training for testing and usage through the Chat API, with the Python `openai` library. Given a list of previous messages, the Chat endpoint responds with the model's response.

4.3 Fixie.ai Agent

Fixie.ai, though the smallest and least widely adopted method among those implemented in this thesis, was deliberately chosen for its unique market share and approach distinct from its more popular counterparts. Unlike other methods, Fixie.ai does not create proprietary language models but instead supports well-established ones like GPT series, LLama, Claude, and custom models. This characteristic minimizes dependence on a single model or company. Furthermore, Fixie.ai stands out by providing a ready-to-use Retrieval Augmented Generation (RAG) solution and facilitating effortless embedding of created assistants into web applications.

4.3.1 Tools used

- **Fixie.ai Web Interface.** Described in Subsection 2.7.3.
- **Fixie.ai Command Line Interface.** Described in Subsection 2.7.3.

4.3.2 Data Collection and Preparation

To facilitate creating assistants with RAG, the Fixie Platform provides *Document Collections*. A Fixie Document Collection is a set of static documents, which can be accessed by the assistant to better respond to the users. The Fixie Platform handles parsing, chunking, storing and retrieving the documents added to the Document Collection, requiring the developer to only upload the documents and point the assistant to the created collection.

To build a death doula Fixie Agent with RAG, a compilation of articles, papers, interview transcripts, and websites related to the topics of mortality, funerals, grief, and the death doula role was collected. The creation of the document collection was supported by a thantology expert. The collection consisted of 22 PDF and text documents, which amounted to 44 megabytes, 5 transcripts of interviews with end-of-life care specialists of a total size equal to 580 kilobytes, and 4 links to websites. Because the Fixie Platform accepts PFDs, text documents, and links to websites as data sources in Document Collection, no further preparations were needed.

A Data Collection for the death doula agent was created with the Fixie CLI. The documents and links were uploaded to the Fixie platform and added to the Document Collection.

4.3.3 Building a Fixie Agent

A Fixie Agent was built using the Fixie Web Console, which requires no code to build a simple agent. The Document Collection created in the previous step was selected as a knowledge base to ground the agent's responses. Additionally, a system prompt developed by a thantology and prompt engineering specialist, the same as used in other approaches, was provided for the agent. The agent was based on the GPT 4 Turbo Preview model, which at the time of writing this thesis in late 2023, is the most advanced model of the OpenAI's GPT series.

4.3.4 Accessing a Fixie Agent

The Fixie Platform offers a simple user interface for each agent which allows one to interact with it and test it. A created agent can also be accessed and tested through calls to an API.

4.4 Creating RAG Assistant with Weaviate

To create the assistant with custom RAG, a vector database Weaviate has been used for the retrieval part and a fine-tuned gpt-3.5-turbo, described in Section 4.2, for the generation part.

Creating a custom RAG solution has been selected for the comparison, as it offers more flexibility and customization compared to pre-made solutions from services like Fixie.ai. Moreover, given the rise of Large Language Models, a myriad of vector databases specializing in indexing embeddings and conducting similarity searches has emerged. Hence, evaluating the construction of an assistant using one of these databases holds merit and warrants exploration.

Because the implementation of the generation model used for this approach has already been described in Section 4.2, this section focuses on the implementation of the retrieval part with Weaviate.

4.4.1 Tools Used

- **Weaviate.** Weaviate[10] is an open source vector database. It offers storing and retrieving vector embeddings and data objects, pre-built modules for popular machine learning models, client libraries in Python, JavaScript and GO, and keyword, vector, and hybrid search functionality.

- **LangChain.** LangChain[11] is a framework aiming to simplify the development of applications utilizing Large Language Models. Within this approach, it was

[10] `https://weaviate.io/`
[11] `https://www.langchain.com/`

used to split the documents into chunks, which can be later indexed in the vector database.

4.4.2 Data Collection and Preparation

For creating a custom RAG solution with Weaviate, a set of 5 transcripts from interviews with end-of-life care specialists, the same which were used with Fixie.ai, totalling 580 kilobytes, was used. The transcripts have been divided into 1000 characters long chunks with langchain's `RecursiveCharacterTextSplitter`[12].

4.4.3 Weviate configuration

Weaviate has been set up in a docker container, using the official docker image. OpenAI's ada-002 has been selected as the model to calculate the embeddings for the chunks. Cosine similarity has been chosen as the similarity search function. Finally, all the chunks created in the previous step have been added to the Weaviate schema. The process of calculating embeddings with OpenAI's model and indexing the chunks is done automatically by Weaviate.

4.4.4 Retrieval Function

The retrieval function used the Weaviate built-in similarity search and limited the number of return results to 3 and the maximum cosine distance to 0.18. The prompt sent to the gpt-3.5-turbo model was constructed based on the retrieval function result. If the retrieval function returned relevant chunks, they were appended to the initial system prompt and the model was instructed to use them to generate answers. If no relevant chunks were found (i.e., there were no chunks within a cosine distance smaller than 0.18 from the user prompt), the original prompt was used.

[12] `https://api.python.langchain.com/en/latest/text_splitter/langchain.text_splitter.RecursiveCharacterTextSplitter.html#`

5 Results

The effectiveness of all four approaches employed to develop a death doula chatbot in the context of this thesis was assessed through the methodologies outlined in Chapter 3. This chapter presents the results of the evaluation process.

5.1 Development services and frameworks

The evaluation from the developer's standpoint involved a thorough assessment of different facets of the services employed in the creation of the death doula chatbot. The results of the evaluation have been split into two parts: the numerical assessment presented in Subsection 5.1.1, giving an overview of the methods evaluation side by side, and the written assessment in Subsection 5.1.2, providing a more detailed evaluation of each method.

5.1.1 Numerical Assessment

The numerical assessments of services and methods employed within the scope of this thesis are presented in Table 5.1. The assessments are on a scale from $1-3$ (1 - adequate, 2 - good, 3 - excellent).

	Fine-tuning Llama	OpenAI API	Fixie.ai	Custom RAG with Weaviate
Ease of implementation	1	3	2	2
Customization Capabilities	3	2	3	3
Scalability	2	2	2	2
Development Cost	3	1	1	2
Training and Fine-Tuning	3	3	1	3
Performance Metrics	2	3	2	3
RAG	2	2	3	3
Community and Support	2	2	3	2
Privacy and Security	3	2	1	3
Mean	**2.3**	**2.2**	**2.1**	**2.5**

Table 5.1 *Numerical assessment of services by category; 1 - adequate, 2 - good, 3 - excellent.*

5.1.2 Written Assessment

Fine tuning Llama

1. **Ease of Implementation.** Compared to all the methods used for creating a conversational agent within the scope of this thesis, the process of fine-tuning a llama model on a personal device emerged as the most intricate, demanding the biggest amount of implementation and knowledge about LLMs, fine-tuning methodologies and parameter-efficient fine-tuning techniques. The libraries selected for fine-tuning offered reasonable documentation, tutorials and examples. However, considering fine-tuning is a new rapidly evolving area, the documentation was not always in-depth. Additionally, there were many alternative solutions and methods for performing the same actions, which were hard to compare and choose from.

2. **Customization Capabilities.** Within the methods employed in the scope of this thesis, fine-tuning and hosting a llama model on a local setup offered the most freedom and customization. Firstly, the fine-tuning method was not restricted and to be chosen by the developer. The implementation of the fine-tuning method and all its parameters were also customizable. This method was also not bound to any platform, giving the freedom to select the most suitable one for training and deployment. Inevitably, it was easy to integrate the model with custom code and functions. However, most of the libraries used for training and running inference offered support for Python only.

3. **Scalability.** The fine-tuned model can be hosted on any platform, therefore there are no restrictions on its scalability. However, it is worth noting that efficient scaling requires correct implementation of the inference by the developer.

4. **Development Cost.** Due to the open-source nature of the llama models, they incur no direct cost for both research and commercial use. However, it is essential to acknowledge the ancillary costs associated with the infrastructure required for model training and subsequent hosting. While fine-tuning can be performed in a free version of Google Colab, as demonstrated in the context of this thesis, the deployment and execution of inferences on the fine-tuned model are typically bound to the costs of the server. Moreover, scaling up or improving the performance of the solution, introduces supplementary financial implications. Additionally, fine-tuning one of the larger Llama models would require investment into the training infrastructure.

5. **Training and Fine-Tuning.** Fine-tuning is the primary objective of this

method, therefore there are no significant obstacles and the process is flexible.

6. **Performance Metrics.** With the fine-tuned llama model, the performance depends mostly on the platform selected for training and hosting. Therefore, it can be adjusted accordingly, to satisfy specific requirements. Nevertheless, the performance is also influenced by the implementation, thus requiring the developer to be knowledgeable and efficiently deploy the model.

7. **RAG (Retrieval-Augmented Generation).** Since the fine-tuned llama model can be hosted on any platform and incorporated into custom code, it is possible to combine it with retrieval augmented generation. However, the whole RAG system needs to be implemented from scratch by the developer.

8. **Community and Support.** Both the open-source llama models and the libraries used for fine-tuning and inferring the model have gained significant interest in recent times and therefore have an active community around them. Additionally, Hugging Face provides a forum through which developers can ask questions and share solutions.

9. **Privacy and Security.** Fine-tuning the open source LLM model on custom data provides the highest level of control and privacy, which is especially important when dealing with sensitive user data. However, while there is much flexibility and control over the processed data and the model itself, there is also a lot of responsibility on the developer to make the solution secure and compliant with privacy standards.

OpenAI Developer Platform

1. **Ease of Implementation.** Fine-tuning OpenAI models can be done either through the API or through the graphical interface. Both methods proved to be easy and intuitive. For working with the API, OpenAI provides client libraries, which supply convenient access to the OpenAI REST API from Python and JavaScript code. The API and libraries offer a level of abstraction for the fine-tuning process, thus no special knowledge of fine-tuning techniques is required. The API endpoints are well-documented and accompanied by examples. Moreover, the graphical interface allows users to fine-tune models with no code, which makes it accessible for developers and non-developers as well.

2. **Customization Capabilities.** During fine-tuning OpenAI models, only few training parameters can be adjusted. Those include the number of epochs to fine-tune the model for, the number of examples in each training batch and

the scaling factor for the learning rate. Therefore, the method offers little control and customization capabilities compared to fine-tuning a Llama model. Additionally, the fine-tuning and hosting of the models is only possible on the OpenAI platform, which makes it platform-dependent. In terms of integrating the models with custom code and functions, the API and accompanying libraries make it easy to use the models from custom code and applications. Additionally, the Chat API allows to provide a list of custom functions with their descriptions, which the model can intelligently advise to call when necessary. The model does not call the functions, but it can generate a JSON that can be used to call the function in the application.

3. **Scalability.** Models are hosted on the OpenAI platform, and scalability is significantly more streamlined compared to on-premises solutions, as we are relieved of the burden of managing infrastructure. The adoption of a pay-as-you-go billing model further enhances flexibility, allowing effortless adjustment to fluctuating demand. It's important to note that these advantages specifically pertain to the models. For solutions utilizing the API, developers are responsible for scaling the associated servers. Additionally, OpenAI has implemented rate limits to regulate the number of requests, tokens, and images a single user can submit to the API per day and per minute.

4. **Development Cost.** The OpenAI platform offers a "Pay as you go" billing schema, which means users are charged based on the actual consumption of services or resources. In the case of the OpenAI platform, users pay per 1000 tokens (roughly equal to 750 words) sent or received from the model. Additionally, different models have different prices. The fine-tuning of the gpt-3.5-turbo model with 800 question-answer pairs resulted in a charge of around 1.22$. Any further requests sent to the fine-tuned model were charged at around 0.002$ per question-answer interaction (of average token count equal to 595.2) between the user and the model. While for the small number of interactions performed in the scope of the thesis, the costs are insignificant, when scaling the solutions, the costs will grow accordingly.

5. **Training and Fine-Tuning.** Fine-tuning was both achievable and straightforward using the fine-tuning API. However, as of the thesis writing, fine-tuning the latest GPT-4 was limited to an experimental access program. Eligible users could request access via the fine-tuning UI, but unfortunately, the author of this thesis did not meet the eligibility criteria for application.

6. **Performance Metrics.** The latency of a model's response is mostly influenced by the model used and the number of tokens generated by the model.

Compared to other solutions implemented in the scope of this thesis, the latency of OpenAI models was low and for a single request-response interaction of around 595 tokens in total (48 tokens in response) averaged to 0.774s. However, it is essential to note that Llama, described in Subsection 5.1.2, was set up on a very basic server for testing purposes, and not designed for production, rendering direct comparisons with production-level systems inappropriate. Additionally, the OpenAI API allows for enabling the `stream: true` option in a request, which causes the model to begin returning tokens promptly as they become available, as opposed to waiting for the entire sequence of tokens to be generated. This does not decrease the latency as such, however, it may improve user experience.

7. **RAG (Retrieval-Augmented Generation).** OpenAI offers an Assistants API, through which an assistant with RAG can be created. The entire RAG system is implemented by OpenAI, requiring developers only to upload the documents upon which the Assistant should base its responses. Despite the documentation hinting at the possibility of constructing an assistant from a fine-tuned model, at the current stage of writing this thesis, the Assistants API does not support the creation of assistants from fine-tuned models. Nevertheless, once this functionality becomes available, creating an Assistant based on a fine-tuned model, enhanced with RAG, is poised to be effortlessly achievable with minimal or no coding required.

8. **Community and Support.** At the time of writing this thesis, OpenAI was one of the most popular solutions for fine-tuning and using large language models. Therefore, it gathers a large community of developers and enthusiasts. The OpenAI Platform features a developer forum where developers can ask questions, exchange tips and solutions, and receive responses from the OpenAI team. Moreover, all the solutions offered by OpenAI are well documented and accompanied by illustrative examples.

9. **Privacy and Security.** OpenAI commits to not use any data submitted through the API Platform for training. Additionally, data at rest is encrypted with AES-256 and the data in transit with TLS 1.2+ - both widely adopted industry-standard encryption protocols. Nonetheless, hosting a private model, such as Llama, on a private server provides an added layer of control, security, and privacy.

Fixie.ai Platform

1. **Ease of Implementation.** The Fixie Platform offers several ways for developers to build AI agents and solutions based on them. Building simple agents can be done either through the web interface, with no code, or through the API with minimal code. Nonetheless, the documentation lacks comprehensive detail and, at times, proves to be ambiguous. In the context of this thesis, handling the Fixie API proved to be challenging. The response generated by the conversation API was a highly intricate data structure, lacking valid JSON format and accompanied by inadequate documentation.

2. **Customization Capabilities.** Fixie agents are constructed using the open-source AJ.JSX framework, providing the flexibility to host agents on-premises. With comprehensive API support and an SDK, extending and integrating agents into custom applications is possible. These agents are also easily embeddable into web applications, and Fixie offers a repository with a basic solution for reference. Furthermore, Fixie provides support for various large language models, including Llama 2 models, GPT-3.5, GPT-4 models, Claude 2, and Claude Instant.

3. **Scalability.** Fixie agents can be hosted on the Fixie Platform, which as Platofrm-as-a-Service offers scalable resources. The monthly billing schema offered by Fixie.ai allows for scaling to accommodate varying levels of demand. However, it is less flexible than a pay-as-you-go model, implemented by OpenAI. Fixie Agents can also be hosted on-premises, requiring the developer and system administrator to handle scaling. Finally, the scalability of the solutions built with Fixie also depends on the underlying LLMs used.

4. **Development Cost.** Fixie.ai offers 4 different monthly subscription plans. The basic plan is free and allows for using 800,000 tokens per Month (which is approximated to 100 Messages per month), an unlimited number of Agents and document ingestions for RAG, and community support. Higher tiers, for $99 and $499 per month, offer 2,000,000 tokens (approx. 500 messages) and 10,000,000 tokens (approx. 3000 messages) per month respectively. They also include higher support through emails and dedicated Slack channels. The highest tier offers a custom number of tokens per month for a custom price, support for SLA, SSO and custom models. While the usage of the Fixie.ai platform in the scope of the thesis was performed as part of the free plan, any bigger project would require one of the higher tiers.

5. **Training and Fine-Tuning.** Fixie Platform does not support fine-tuning of the models.

6. **Performance Metrics.** The average latency of the Fixie agent based on GPT-4 Turbo was 14s per single response. This was the mean latency of 200 requests which contained prompts of an average token count equal to 543.695 and yielded assistant responses of an average token count equal to 74.135. It is important to note that a substantial part of the latency is caused by the retrieval function which is not present when using only fine-tuned models in other methods. Nevertheless, the latency of the Fixie Agent was still exceptionally high.

7. **RAG (Retrieval-Augmented Generation).** Retrieval augmented generation is the principal application of Fixie.ai. Implementation is exceptionally straightforward, requiring developers to simply upload documents to a designated collection and direct the agent to the pertinent collection. The entire retrieval system is orchestrated seamlessly by the Fixie platform. Furthermore, the agent can cite the documents utilized to underpin the response, offering transparency, and can supply links for further reading to enhance the user's understanding.

8. **Community and Support.** Because Fixie.ai is the least popular method for creating AI conversational agents from all the methods implemented in this thesis, it has the smallest community and support. At the time of writing this thesis, there exist little tutorials or discussions regarding Fixie solutions. Nonetheless, Fixie has its community on Discord, a communication platform that combines text, voice, and video chat features, with around 1400 members. The developers of the platform proved to be active in the Discord community, offered support, and tried helping with potential problems.

9. **Privacy and Security.** The Fixie.ai documentation lacks explicit information on security and privacy considerations. Consequently, users should exercise utmost caution, particularly when uploading sensitive data. It's crucial to note that the level of privacy and security is also dependent on the underlying model used by the agent, such as the OpenAI model, Claude model, and others. Among the various approaches scrutinized in this thesis, Fixie stands out as requiring the highest degree of caution in terms of both security and privacy.

Custom RAG with Weaviate

1. **Ease of Implementation.** Compared to creating a knowledge-enhanced assistant with Fixie.ai, building a custom RAG solution with Weaviate emerged to be more difficult. The process of preparing, chunking and parsing the

documents for the knowledge base had to be performed by the developer. Connecting the retrieval and the generation parts of the system also had to be implemented. This method also required more architectural choices which, while offering more flexibility, might also result in more mistakes and inefficiencies. Despite this, the documentation and tutorials offered by Weavite were thorough, illustrative and helpful. Weaviate also provides Python and JavaScript client libraries, which make working with the database easy. Furthermore, database indexing and similarity search are implemented as integral parts of Weavite, requiring no additional work from the developer.

2. **Customization Capabilities.** Creating a custom RAG solution offered a lot of freedom and customization possibilities. Firstly, the model for the generation part of the solution could be freely selected. Building a custom RAG solution gives more flexibility for choosing how the documents are divided and indexed, how the similarity search is performed, and how many relevant pieces are provided to the generation model, among other considerations. The choice of the vector database is also free. Within the scope of this thesis, Weaviate has been chosen, but there were no restrictions for choosing a different database.

3. **Scalability.** The created RAG solution can be hosted on any platform, therefore there are no restrictions on its scalability. However, it is worth noting that efficient scaling requires correct implementation of the inference by the developer.

4. **Development Cost.** Weaviate database is free, therefore it incurs no direct costs. However, it is essential to acknowledge the ancillary costs associated with the infrastructure required for database hosting and creating embeddings from the knowledge pieces. Within the scope of this thesis, OpenAI's billed embedding model was used. Moreover, scaling up or improving the performance of the solution, introduces supplementary financial implications.

5. **Training and Fine-Tuning.** Since building a custom RAG solution is platform- and model-independent, there are no obstacles to combining it with a fine-tuned model.

6. **Performance Metrics.** The Weaviate database, as a vector database, is especially efficient for storing, indexing, and retrieving vector embeddings. The custom RAG solution built with Weaviate as the knowledge base and fine-tuned gpt-3.5-turbo as the generation model had an average latency of 1.5s per request of an average of 884.11 tokens per prompt and an average of 53.075 tokens per model response. It is important to remember that the performance also depends on the platform selected for hosting the vector database.

7. **RAG (Retrieval-Augmented Generation).** The primary objective of this method was building a RAG solution, therefore there are no significant obstacles and the process is flexible.

8. **Community and Support.** Weaviate offers a forum and a Slack community for developers to share ideas and help each other. More generally, building custom RAG solutions has gained popularity and there are numerous blogposts and communities sharing experiences and tutorials on that topic.

9. **Privacy and Security.** Building a custom RAG solution provides a high level of control and privacy, which is especially important when dealing with sensitive user data. However, while there is much flexibility and control over the processed data and the model itself, there is also a lot of responsibility on the developer to make the solution secure and compliant with privacy standards.

5.2 Conversational style and helpfulness

The end-user perspective evaluation encompassed the assessment of the conversational style and helpfulness exhibited by the various assistants generated through distinct methods. All outcomes were subjected to scrutiny by a thanatology expert for qualitative evaluation, while BERTScore was employed for a quantitative assessment.

5.2.1 Qualitative Results

The expert evaluation of the different versions of the death doula chatbot, created with different methods, is presented in Table 5.2. Apart from the review written by the expert, the table indicates the method used to develop the given version of the chatbot, and whether the method included RAG and fine-tuning.

Version	Method	RAG	Fine-tuning	Expert Review

V1	Fine-tuning Llama (1)	No	Yes	• *"Feels robotic, like it got answers off wikipedia* • *nothing was factually incorrect, but it was not personalized or unique* • *felt like your typical chatbot rather than something trained on specific content* • *Very long answers and never addressed me personally* • *Has no personality"*
V2	Fine-tuning gpt3.5-turbo (2)	No	Yes	• *"Extremely fast, short, and welcoming answers* • *Very spiritual and culturally focused* • *Best answer length and always gives suggestions/follow-up questions* • *It always asks 2 questions at the end of the response, hard to understand which to answer* • *The personality is a bit too overbearing, does not feel like responses a human would give, more like a spiritual advisor"*
V3	Using Fixie.ai Agent (3)	Yes	No	• *"Very repetitive, always says "remember there is no right or wrong answer"* • *Extremely non-personal, very robotic* • *Much of the information is either inaccurate or vague* • *Answers are insanely wrong and don't feel "human""*

V4	Fine-tuning gpt3.5-turbo and building custom RAG (4)	Yes	Yes	• *"Feels extremely human and answers in a nurturing way* • *More professional than V3* • *Felt respectful, honest, and the answers were always accurate to the best of my knowledge* • *Asks amazing follow-up questions and keeps the conversation going in a healthy way* • *I worry we need more guardrails so that it will suggest professional help much sooner when topics like suicide, abuse, or dangerous behavior arise* • *I am impressed with the speed and the length of the messages"*

Table 5.2 *Expert evaluation of different versions of the death doula chatbot.*

The expert preferred version 4 of the chatbot, created based on fine-tuned gpt-3.5-turbo model and custom RAG system. This was the only version within this project that employed both fine-tuning and RAG to create a chatbot.

Noting the need to establish additional guidelines for the chatbot to prompt professional assistance at an earlier stage when specific topics arise, the expert expressed optimism about the method's potential. This positive outlook reinforces the belief in the feasibility of creating a death doula based on LLM technology.

5.2.2 Quantitative Results

The BERTScore results for different models calculated based on the answers to the questions from the test dataset can be seen in Table 5.3. For all metrics, Precision, Recall and F1 Score, a higher score is better, denoting a closer match to the reference. A score of 1 would mean a perfect match.

The quantitative results suggest the fine-tuned gpt-3.5-turbo model outperforms others, with fine-tuned gpt-3.5-turbo enhanced with custom RAG to closely follow. However, it is important to note that these results are calculated based on answers to the test dataset, which was similar in style to the training dataset. Due to this limitation, the quantitative results are less comprehensive than the expert review.

Version	**Model**	**Precision**	**Recall**	**F1 Score**
-	**llama2-7b-chat**	0.833	0.889	0.860
V1	**fine-tuned llama2-7b-chat**	0.878	0.898	0.888
-	**gpt-3.5-turbo**	0.874	**0.909**	0.891
V2	**fine-tuned gpt-3.5-turbo**	**0.899**	0.895	**0.897**
V3	**Fixie.ai Agent with RAG (GPT-4-turbo preview)**	0.836	0.889	0.862
V4	**fine-tuned GPT-3.5-turbo with RAG**	0.898	0.890	0.894

Table 5.3 *BERTScore results for different models calculated based on the answers to the questions from the test dataset.*

6 Discussion

This chapter unfolds the implications of the research conducted within the scope of this thesis, offering insights into its notable findings and contributions.

6.1 Guidelines for Developers

Based on the evaluation of different methods for creating conversational assistants based on LLMs, the following guidelines for developers are proposed. These guidelines help to choose the right tool for creating LLM based chatbot, depending on the use-case, developer's priorities and needs.

Flexibility, control and long-term cost-efficiency. Fine-tuning open-source Llama on a private machine is the best choice when flexibility and granular control are top priorities. Similarly, for the most flexibility and control while creating a RAG solution, building a custom RAG with one of the available open-source vector databases is the best choice. In both cases, the developer is in full control of all the data, security practices, platform choice for hosting the solution, and incorporation with other functions and modules. Fine-tuning Llama and using an open-source database are also the most cost-effective solutions long-term, as their open-source nature means they are free. It is important to note that the security of the custom RAG solution also depends on the model selected for the generation part. For the highest level of security and control, combining it with Llama is the best choice. The downside of both fine-tuning Llama and building a custom RAG solution is that they demand a more profound understanding of the underlying technology. Additionally, creating custom solutions might require more up-front investments, especially if using bigger and more efficient models.

Ease of implementation and prototyping. The OpenAI Platform stands out as the most user-friendly and easy to use. Models can be fine-tuned using the web interface, without any code. The platform provides an abstraction to the fine-tuning process, therefore no profound knowledge of the underlying techniques is required. Additionally, OpenAI offers detailed documentation of all functions and solutions. The drawbacks of choosing the OpenAI Platform are limited flexibility, and being platform- and model-dependent. In the long run, it also incurs higher costs than using open-source models. However, it requires no up-front investment in the hardware infrastructure and allows to build solutions based on the most powerful models for a very affordable price. Considering this and the additional aspect of the ease of implementation, leveraging OpenAI proves to be a favourable option when the aim is to establish a proof-of-concept.

Easy RAG and proximity to developers. Fixie.ai platform emerges as the best option when Retrieval Augment Generation (RAG) is the main use-case and additionally, the easiness of the method is important. Creating an agent with RAG on Fixie.ai is exceptionally straightforward, thanks to the platform's pre-implemented technicalities. Fixie RAG solution supports uploading documents and providing links to websites, which Fixie will crawl. Another advantage of Fixie.ai is its proximity to the platform developers. Being the smallest service of all the evaluated, it offers support directly from its developers and the possibility for users to influence their usage terms. However, offering only RAG, Fixie.ai is not the best solution when the objective is to create personal chatbots with unique styles of responses.

6.2 Accessibility of AI

Another noteworthy conclusion coming from this research is the general accessibility and ease of creating LLM based chatbots. Given the evolving nature of technologies and the sophistication required in AI chatbots, even the most demanding solution: fine-tuning Llama, proved to be surprisingly easy, accessible and effective, without necessitating specialized machine learning knowledge. Moreover, fine-tuning was possible with a relatively small dataset of 1000 question-answer pairs (800 for training, and 200 for testing). The fine-tuned models achieved good performance, displaying a unique style of conversation and personality, according to the expert. Furthermore, all the methods were very affordable, requiring no big investments. This proves that the currently available technologies allow all developers, not only machine learning specialists, to create sophisticated AI assistants tailored to specific tasks with relatively small effort, low price and using consumer hardware.

6.3 RAG vs Fine-tuning

Fine-tuning a model proves to be the optimal approach when aiming for a specific style and tone in the model's responses. This is particularly advantageous when creating a chatbot with a distinctive "personality" that exudes authenticity and instils confidence. On the other hand, retrieval augmented generation ensures that the model provides truthful responses without hallucination. This technique also allows for the enhancement of responses with dynamic data, such as statistics, location-specific information, or user-specific details.

To construct chatbots that are both personal and trustworthy, a dual strategy is necessary: tailoring responses to a specific style and incorporating truthful, personalized information. Consequently, the combination of fine-tuning and retrieval augmented generation emerges as the most effective option.

6.4 Death Doula Chatbot

The development of four distinct versions of a death doula chatbot, as explored within the confines of this thesis, highlights the varying performance outcomes achieved. While not all models demonstrated optimal performance, the two versions of the chatbot based on the fine-tuned GPT-3.5 model, one additionally augmented with external knowledge, underscore the potential feasibility of creating a reliable end-of-life care chatbot using LLMs. Both qualitative and quantitative assessments were conducted, affirming the commendable performance exhibited by these two iterations of the assistant.

In light of the global emergence of the death positivity movement and society's increasing recognition of the importance of discussing sensitive subjects like death [10], the death doula chatbot founded on LLM holds significant potential. Beyond serving individuals coping with death or grief, it extends its utility to anyone seeking to delve into conversations surrounding mortality—a topic still often regarded as taboo in contemporary society.

Furthermore, this research underscores the capability of LLMs in crafting personalized and dependable chatbots adept at handling and assisting with delicate subjects. The implications of this study extend beyond the immediate application in end-of-life conversations, offering a broader perspective on the adaptability of LLMs in fostering discussions about sensitive topics.

7 Conclusions

This chapter summarizes the work by providing answers to the research questions, discussing the limitations of the current study, and suggesting potential avenues for future research within the specified domain.

7.1 Answers to Research Questions

Primary question: How do different approaches to creating AI conversational assistants such as:

- **Fine-tuning an open-source llama model on a private machine using Hugging Face libraries,**
- **Fine-tuning a GPT-3.5 model using the OpenAI API,**
- **Building a knowledge-enhanced assistant using the Fixie.ai platform**
- **Implementing a knowledge-enhanced assistant using the Weaviate vector database**

compare in terms of implementation ease, customization capabilities, and overall?

In comparing various approaches to creating AI conversational assistants, the use of platforms such as OpenAI and Fixie.ai emerges as user-friendly and convenient, providing ease of implementation. On the contrary, building customized solutions, such as fine-tuning open-source models like Llama or implementing custom RAG with an open-source vector database (e.g., Weaviate), presents advantages in terms of greater flexibility, control and lower costs in the long term.

Secondary question: How does the conversational style of an AI assistant differ when employing fine-tuning as opposed to retrieval augmented generation?

The conversational style of an AI assistant differs notably between fine-tuning and retrieval augmented generation. Fine-tuning offers greater influence over tone and style, allowing for precise adjustments based on the use case. This flexibility makes it possible to create responses that are more personalized, humorous, formal, or tailored to specific preferences, making fine-tuning a preferred choice for nuanced control over the assistant's language.

Tertiary question: Can large language models be leveraged to create trustworthy and personal death doula chatbots?

The evaluation confirms the promising potential of utilizing LLMs to create trustworthy and personal death doula chatbots, with certain models garnering positive feedback from the expert review. Specifically, a chatbot based on the powerful GPT-3.5, fine-tuned with question-answer pairs between a death doula and a user, and augmented with knowledge from experts' conversations related to end-of-life topics achieved these commendable results. Notably, this approach demonstrated effectiveness in navigating sensitive conversations and providing interactions imbued with a comforting and individualized style, showcasing encouraging and promising outcomes.

7.2 Limitations

This study has certain limitations which need to be taken into account when interpreting the results of the thesis.

Firstly, the assessment of the services employed in crafting conversational assistants based on LLMs was subjective and carried out by a single individual. While it drew from the insights garnered through hands-on experience with the services, it is essential to acknowledge that a developer's review inherently retains a subjective nature. It should be approached with the understanding that, although valuable, it may not represent an absolute truth.

Moreover, the landscape of tools and services in the field of AI and LLMs evolves rapidly, making it challenging to replicate the exact conditions and outcomes presented in this thesis. Throughout the four-month process of writing the thesis, significant changes unfolded. New models were continuously released, libraries underwent updates, and services experienced further development, sometimes altering their operational frameworks. For instance, during the initial stages, Fixie.ai offered free usage but later introduced a monthly subscription billing plan. Furthermore, Gemini, the newest large language model from Google, was only introduced after the writing period [38].

Additionally, the use of the smallest available Llama model was necessitated by hardware constraints, and it should be acknowledged that employing a larger model might potentially enhance performance when compared to other techniques employed in this study.

Finally, the death doula was created for research purposes only. It necessitates thorough testing before its potential introduction to users to ensure that the chatbot operates without causing harm and remains free from any tendencies to hallucinate or exhibit undesirable behaviours.

7.3 Future Work

In exploring potential directions for future research, several promising areas emerge from the insights and methodologies of this thesis.

Utilization of alternative large language models. During the period of writing this thesis, several prominent large language models were released, among others Google's Gemini and Mistral's Mixtral 8x7B[1]. A great avenue for future research is the utilization of those alternative models to create conversational chatbots and comparison of their efficacy, performance and user experience with the models employed in this thesis.

Extensive testing. A crucial next step in further advancing the development of a death doula chatbot involves extensive testing of the chatbot. Given the sensitivity of the topic, an AI ethical evaluation would be required before the potential release of the chatbot.

Guardrails for the death doula. The expert evaluation of the death doula models revealed that there are certain areas, such as suicide, in which the chatbot should direct the user to a human advisor earlier. To achieve this, the development of guardrails for the chatbot stands as a crucial next step in the death doula chatbot development process.

Chatbot personalization. An intriguing prospect for future work is the personalization of the death doula chatbot by incorporating user-specific details such as medical history and place of residence. This enhancement would tailor responses for a more individualized and effective interaction.

Broadening the evaluation scope. Broadening the services and methods evaluation scope by involving a larger group of developers can contribute to a more comprehensive understanding of the usability and effectiveness of the development services utilized in this study.

[1] An open-source model released under Apache 2.0 license in December 2023, by a French start-up Mistral. Mixtral 8x7B outperforms Llama2 70B on most benchmarks, while the inference is 6x faster, and outperforms or matches the performance of GPT3.5 on most benchmarks.

References

[1] Karim M. Abadir and Jan R. Magnus. “Rank, inverse, and determinant”. In: *Matrix Algebra*. Econometric Exercises. Cambridge University Press, 2005, pp. 73–96.

[2] Meta AI. *Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models*. 2020. URL: `https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/` (visited on 11/06/2023).

[3] Tom B Brown et al. “Language Models are Few-Shot Learners”. eng. In: *arXiv.org* (2020). ISSN: 2331-8422.

[4] Alessandro Bruno et al. “Insights into Classifying and Mitigating LLMs’ Hallucinations”. eng. In: *arXiv.org* (2023). ISSN: 2331-8422.

[5] Hannah Decker et al. “Large Language Model—Based Chatbot vs Surgeon-Generated Informed Consent Documentation for Common Procedures”. eng. In: *JAMA network open* 6.10 (2023), E2336997–e2336997. ISSN: 2574-3805.

[6] Tim Dettmers et al. *LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale*. 2022. arXiv: `2208.07339 [cs.LG]`.

[7] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`. URL: `https://aclanthology.org/N19-1423` (visited on 10/03/2023).

[8] Ning Ding et al. “Parameter-efficient fine-tuning of large-scale pre-trained language models”. eng. In: *Nature machine intelligence* 5.3 (2023), pp. 220–235. ISSN: 2522-5839. (Visited on 10/26/2023).

[9] *Fixie.ai Documentation*. URL: `https://fixie.ai/docs/getting-started` (visited on 11/23/2023).

[10] The Order of the Good Death. *The Order of the Good Death - Our Story*. URL: `https://www.orderofthegooddeath.com/our-story/` (visited on 11/13/2023).

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[12] Aurora Harley. *UX Expert Reviews*. 2018. URL: https://www.nngroup.com/articles/ux-expert-reviews/ (visited on 11/24/2023).

[13] Neil Houlsby et al. "Parameter-efficient transfer learning for NLP". eng. In: *36th International Conference on Machine Learning, ICML 2019*. Vol. 2019-. 2019, pp. 4944–4953. ISBN: 9781510886988.

[14] J. Howard and S. Gugger. *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD*. O'Reilly Media, Incorporated, 2020. ISBN: 9781492045526. URL: https://books.google.no/books?id=xd6LxgEACAAJ.

[15] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. en. arXiv:2106.09685 [cs]. Oct. 2021. URL: http://arxiv.org/abs/2106.09685 (visited on 10/09/2023).

[16] Krystal Hu. *ChatGPT sets record for fastest-growing user base - analyst note*. 2023. URL: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/ (visited on 01/08/2024).

[17] *Hugging Face*. URL: https://huggingface.co/ (visited on 11/17/2023).

[18] D. Jurafsky et al. *Speech and Language Processing*. Pearson Education, 2014. ISBN: 9780133252934. URL: https://books.google.at/books?id=Cq2gBwAAQBAJ.

[19] Pradnya Kulkarni et al. "Conversational AI: An Overview of Methodologies, Applications & Future Scope". In: *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*. 2019, pp. 1–7. DOI: 10.1109/ICCUBEA47591.2019.9129347.

[20] Brian Lester, Rami Al-Rfou, and Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. arXiv: 2104.08691 [cs.CL].

[21] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[22] Ron Li, Andre Kumar, and Jonathan H. Chen. "How Chatbots and Large Language Model Artificial Intelligence Systems Will Reshape Modern Medicine: Fountain of Creativity or Pandora's Box?" In: *JAMA Internal Medicine* 183.6 (June 2023), pp. 596–597. ISSN: 2168-6106. DOI: 10.1001/jamainternmed.2023.1835. eprint: https://jamanetwork.com/journals/jamainternalmedicine/articlepdf/2804310/jamainternal_li_2023_ic_230009_1685974606.93544.pdf. URL: https://doi.org/10.1001/jamainternmed.2023.1835.

[23] Xiang Lisa Li and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation". eng. In: *ACL-IJCNLP 2021 - 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, Proceedings of the Conference.* 2021, pp. 4582–4597. ISBN: 1954085524.

[24] Yunxiang Li et al. "ChatDoctor: A Medical Chat Model Fine-Tuned on a Large Language Model Meta-AI (LLaMA) Using Medical Domain Knowledge". English. In: *Cureus* 15.6 (2023). Place: Palo Alto, United States Publisher: Cureus Inc. DOI: 10.7759/cureus.40895. URL: https://www.proquest.com/docview/2844019753/abstract/D0819359FCD04B27PQ/1 (visited on 09/12/2023).

[25] INELDA International End of Life Doula Association. *What Is A Doula?* URL: https://inelda.org/about-doulas/what-is-a-doula/ (visited on 11/13/2023).

[26] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing.* 2021. arXiv: 2107.13586 [cs.CL].

[27] Xiao Liu et al. *GPT Understands, Too.* 2023. arXiv: 2103.10385 [cs.CL].

[28] Xiao Liu et al. *P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks.* 2022. arXiv: 2110.07602 [cs.CL].

[29] *Ludwig.* 2023. URL: https://ludwig.ai/latest/ (visited on 01/20/2024).

[30] Aniket Maurya. *How to build a chatbot using open-source LLMs like Llama 2 and Falcon.* 2023. URL: https://lightning.ai/pages/community/tutorial/how-to-build-a-chatbot-using-open-source-llms-like-llama-2-and-falcon/ (visited on 01/20/2024).

[31] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: 1301.3781 [cs.CL].

[32] Chanin Nantasenamat. *How to build an LLM-powered ChatBot with Streamlit.* 2023. URL: https://blog.streamlit.io/how-to-build-an-llm-powered-chatbot-with-streamlit/ (visited on 01/20/2024).

[33] Sharan Narang and Aakanksha Chowdhery. *Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance.* 2022. URL: https://blog.research.google/2022/04/pathways-language-model-palm-scaling-to.html (visited on 01/08/2024).

[34] *OpenAI API: Overview.* 2023. URL: https://platform.openai.com/docs/overview (visited on 01/20/2024).

[35] Oded Ovadia et al. *Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs*. 2023. arXiv: 2312.05934 [cs.AI].

[36] "PaLM: Scaling Language Modeling with Pathways". eng. In: *arXiv.org* (2022). ISSN: 2331-8422.

[37] Andrew Peng et al. *GPT-3.5 Turbo fine-tuning and API updates*. 2023. URL: https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates (visited on 12/01/2023).

[38] Sundar Pichai and Demis Hassabis. *Introducing Gemini: our largest and most capable AI model*. 2023. URL: https://blog.google/technology/ai/google-gemini-ai/ (visited on 01/20/2024).

[39] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[40] Google Research. *Google Colaboratory*. URL: https://colab.google/ (visited on 11/14/2023).

[41] Frank Rosenblatt. "The Design of an Intelligent Automaton". In: *Research Trends 6, no. 2* (1958).

[42] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal* (1959).

[43] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. July 2023. DOI: 10.48550/arXiv.2307.09288. URL: http://arxiv.org/abs/2307.09288 (visited on 09/17/2023).

[44] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2302.13971. URL: http://arxiv.org/abs/2302.13971 (visited on 09/17/2023).

[45] A. M. Turing. "Computing Machinery and Intelligence". In: *Oxford University Press on behalf of the Mind Association* (1950).

[46] Ashish Vaswani et al. "Attention is all you need". eng. In: *Advances in Neural Information Processing Systems*. Vol. 2017-. 2017, pp. 5999–6009.

[47] Jingfeng Yang et al. "Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond". eng. In: *arXiv.org* (2023). ISSN: 2331-8422.

[48] Tianyi Zhang* et al. "BERTScore: Evaluating Text Generation with BERT". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SkeHuCVFDr.