



UNIVERSITY OF
BIRMINGHAM

Master Thesis

Self-improvement for all

by

Arpan Tripathi

(2395432)

Supervisor: Phillip Smith

September 17, 2023

*Submitted in partial fulfillment of the requirements for
MSc in AI ML at University of Birmingham*

CONTENTS

Contents	3
List of Figures	4
Abstract	1
1 Introduction	1
2 Background	2
2.1 Deep Learning	2
2.2 Large Language Models	4
2.3 Trends in Compute Efficiency	6
2.4 Conclusion	8
2.5 Remote Deployment	8
2.6 Sentence Embeddings	9
2.7 Generative Agents	10
3 Methodology	12
3.1 Overview	12
3.2 23rd June - 1st July: Survey of LLMs	12
3.3 4th July - 11th July: Survey of LLM optimisations	16
3.4 12th July - 16th July: Choosing Tech stack	16
3.5 Preparation	17
3.6 17th July - 31st July: Implementation v1	19
3.7 1st August - 8th August: Implementation v2	23
3.8 9th August - 16th August - Implementation v3	25
3.9 16th August - 25th August: Remote deployment	27
3.10 26th August - 31st August: Research on current challenges and future scope	30
4 Evaluation	31
5 Conclusion	32
References	32
6 Appendix	36

LIST OF FIGURES

1	Vanilla Transformer[35] architecture	4
2	An illustration of Generative Agents[54] architecture from the original paper	11
3	UI developed with Streamlit	19
4	UI with streaming output separated into the sidebar and main window	23
5	Reworked Generative Agents framework	24
6	Demonstration of the memory system integrated into the UI, the memories become more high-level and concise as the interaction with the chatbot continues due to the Reflection system from Generative Agents architecture	25
7	Template creation with Docker Container on Runpod	28
8	GPU selection during Pod deployment on Runpod	29
9	Deployed Pod for LLM inference on Runpod with RTX 3080Ti GPU	29
10	Deployed pod is accessible via a Web terminal, that displays the required Pod ID in the URL	30
11	Evaluation form results: 1	33
12	Evaluation form results: 2	34

Self-improvement for all

In the current world, Large Language Models (LLMs) have become more personalised, customisable and accessible compared to the last few years and the trends suggest ChatGPT-level performance in smaller scale LLMs in the near-future. Therefore, extending the recent Generative Agents framework designed to simulate believable proxies of human behavior for the small-scale LLMs to simulate a self-improvement coach, this project aims to make self-improvement guidance more accessible to the world through the intrinsic capabilities of LLMs.

The performance of this implementation is subject to the knowledge of psychology, instruction-following capabilities and alignment towards positive behavior in the contemporary/future LLMs.

The code is publicly available at the repository: <https://github.com/tripathiarpan20/self-improvement-4all>

No AI generated material is included within their dissertation or the associated materials.

1 INTRODUCTION

Artificial Intelligence has inspired a lot of awe and wonder in people since early days, as represented in various forms of media, either in forms of either convenience to daily life of humanity, or as harbinger of destruction. However, the real functionality of AI remains vastly different from the overtly mysterious portrayal in media and until an authentic Artificial General Intelligence (AGI) is developed, the effects of AI on society remains largely dependent on its usage by the society, for instance, Neural Networks for classification in medical domains have automated several pain-points for medical professionals and generative AI such as Stable Diffusion[58] has provided means for the masses to empower their creativity. Meanwhile, it is not difficult to imagine the potential the current state-of-the-art visual and audio deepfakes hold to spread misinformation. With the recent release of ChatGPT, the potential of AI and LLMs is being realised by the common masses, i.e, people who do not possess any technical knowledge relevant to AI, and since then we have seen a diverse number of applications, such as symbolic music generation, data analysis, fitness, entrepreneurship etc, however, usage for more personal purposes such as self-improvement/therapy is expected to not have been fully realised due to privacy concerns, as all the prompts sent to the closed source models such as ChatGPT gets owned by the corporation. On the frontier of open-source large language models, recent years have seen a variety of optimizations both on the computation side, like Tensor operations, along with related frameworks like Nvidia CUDA, and LLM-specific optimizations like 8-bit quantisation[39], GPTQ[42]. Such optimisations have vastly increased the feasibility of running inference of LLMs of larger scale in the last few years. Meanwhile, as opposed to the earlier paradigms, that pivoted on scaling the model architecture to achieve better performance and emergent capabilities, as proposed in the GPT-3 paper[38], subsequently revising to achieve optimal performance given fixed training compute by scaling training data, as proposed in Chinchilla[45], a recent family of models, LLaMA[18], achieves better performance than the earlier LLMs at a lower parameter count.

As a result of the aforementioned developments, a significantly higher percentage of consumer-grade hardware can now support the inference of open-source/non-commercial-usage LLMs for personal usage, with guaranteed privacy and the trends suggest even better accessibility in the future.

Meanwhile, a recent work by Stanford proposed an approach to simulate believable proxies of human behavior, called Generative Agents[54], which utilises ChatGPT to simulate 25 NPCs in a sandbox simulation, each with their own personality, memory, routines, high-level reflections based on conversations/interactions, planning events and showing emergent human-like social behavior. Given the presence of relevant expertise and sufficient alignment in the

LLM, the proposed approach shows promise to simulate proxies of a personal assistant/virtual companion, such as, a personal self-improvement coach.

To this end, this work proposes combination of all these developments to simulate a completely private self-improvement coach with scope for further increments in performance with the more powerful smaller scale LLMs expected to be democratised in the future.

The main contributions of this work are a combination of both research and application, and elaborated as follows:

- Literature review of the contemporary LLM research landscape with a focus on the application domain.
- The implementation of the recent Generative Agents architecture for simulating believable proxy of a self-improvement coach using open-source LLMs.
- Tight integration with popular community-maintained frameworks such as Huggingface and text-generation-webui[33] to ensure compatibility with LLMs available in the future.
- Development of a User Interface (UI) for the aforementioned using tools like Streamlit and Langchain.
- Support for remote deployment of the LLM as a container on cloud GPU rental services, specifically, we used Runpod for the implementation.
- Qualitative evaluation of the chatbot with a survey with 10 users.

2 BACKGROUND

2.1 Deep Learning

Artificial Intelligence is a broad field that is divided into several specialisations, such as Machine Learning, which further consists of subdomains like Reinforcement Learning, Deep Learning etc. . Most of the pioneering concepts related to Deep Learning have existed since the last century, the potential of Deep Learning only came to realisation in the past decade, primarily owing to the alignment of innovations in hardware and neural network architectures, specifically, neural network architectures possess the parallelisation capabilities due to the independent nature of calculations performed on the input feature vectors, and due to the large number of cores in Graphics Processing Units (GPUs) that can accommodate substantially more parallel operations compared to Central Processing Units (CPUs). Since the GPU revolution, there has been emergence of several popular architecture like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs) and Transformers that been used in various practical applications, where architectures derived from Transformers have retained the top positions in benchmarks related to various text and image domain tasks in the recent years. The following is a brief overview of the neural network architectures for the text domain.

2.1.1 RNN. Recurrent Neural Network (RNN)[50] is a generalisation of Feed Forward Neural Networks which maintains information about past input in form of hidden state or memory. This hidden state is then passed on to next input for providing context about previous inputs, which is used together with input to calculate the output. The network is recurrent as it performs same function on every input of data and output of the network depends on previous computation. The vanilla RNN suffered with problems of vanishing and exploding gradients and was also very difficult to train and did not perform well in cases of long sequences.

2.1.2 LSTM. Long-Short Term Memory network (LSTM)[43] was a modification of RNN and dominated most of the benchmarks for sequence modelling tasks until 2016. The LSTM consists of multiple gates including input gate, forget gate and output gate which facilitated remembering past data easier and solved the problems of vanishing

gradients. There were still a few limitations to LSTM similar to RNN, including difficulty in training due to limitations in parallelisation, i.e, processing of tokens being done at a time, and long gradient paths. Moreover, there were complications in modelling long sequences.

2.1.3 Transformers. Transformers [1] were introduced in 2017 and solved majority of the drawbacks of LSTM and RNN, such as the inability to parallelise and poor modeling of long-term dependency between tokens in a sequence. The Transformer model uses self-attention as the core mechanism to model dependencies without having the constraint of long-term dependency in the sequence. The intuitive idea is to consider a translation task while generating the output language, which would require the translator to exert more attention to the word that is currently being translated. With this, for each input, the Transformers architecture calculates an attention score which can be used in the decoding stage to find out how much attention a particular word/token requires.

Vanilla Transformer, similar to previous architectures, is an encoder-decoder based model, where each unit of transformer either comprises of multi-headed self-attention layer followed by a feed-forward layer, the attention layer uses all the input tokens concurrently to calculate the score, hence, the feedforward step does not require any recurrence of words to forward pass, thus, parallelisation proves to be a trivial challenge given the Transformers architecture.

In the encoder part of the Transformers architecture, the input sequence is first embedded into fixed-length vectors, creating a matrix of embeddings that represent the input text. This embedding layer is followed by a series of identical layers, each consisting of a self-attention mechanism and a position-wise feed-forward network. The self-attention mechanism allows the model to weigh the importance of different tokens relative to one another, enabling it to capture complex contextual relationships within the input sequence, as mentioned earlier briefly.

The feed-forward network, processes the output of the self-attention mechanism, transforming the information into a higher dimensional space. This process is repeated multiple times, with each iteration refining the representations of the input tokens. The final output of the encoder is a sequence of hidden states, each representing a rich contextual understanding of the input text.

In the decoder portion of the Transformers architecture, the output sequence is generated one token at a time. At each step, the model generates a probability distribution over the vocabulary of the model, selecting the most permissible token to add to the output sequence. This recursively continues until the output sequence is determined to be completed with a special token or reaches the prompted number of tokens.

The architecture of the original Transformer as initially proposed is illustrated in figure 1. Since then, there are broadly two types of Transformer architectures in contemporary AI research:

- **Encoder-only Transformers:** These Transformers consist of only the encoder part of the vanilla architecture and used for generating the combined vector embeddings of the series of input tokens. The most popular example of this model is the BERT[41] model.
- **Decoder-only Transformers:** The Transformer module can also function without the usage of the encoder module with the masked self-attention mechanism and was the core contribution of the original OpenAI GPT[57].

Since then, there have been several attempt to optimise the attention mechanism from the order of $O(n^2)$ such as sparse and windowed attention, where several works have claimed to achieve the order of $O(n)$.

The Transformers architecture has remained the state-of-the-art since 2018 till date, although there were several architectures like State-space models[44] that were speculated to surpass it in terms of performance and speed. Since then, most of the optimisations on the computation side, like GPUs and other specialised hardware, has been in

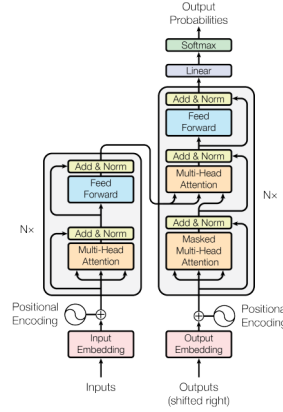


Fig. 1. Vanilla Transformer[35] architecture

the direction of facilitation of linear operations, like matrix multiplication, which has been the core of the attention mechanism used in Transformers.

2.1.4 Tokenisation. In NLP, tokenisation is a crucial initial step that involves dividing raw text into smaller components, usually words or subwords, called tokens. These tokens are then transformed into numerical vectors to serve as inputs for various deep learning models, such as Transformers, LSTM, and RNN.

The tokens generated through the process of tokenisation are intricately related to the Transformers architecture, which is designed to handle input sequences of varying lengths as aforementioned. Each token serves as a unique representation of a word or subword unit in the input sequence, allowing the model to capture the nuances of language. The input sequence is first embedded into fixed-length vectors, creating a matrix of embeddings that represent the input text, which is further passed to the encoder layer.

Two popular techniques used in tokenisation are:

- **Byte Pair Encoding (BPE):** BPE replaces frequent pairings of characters with a single symbol, which reduces the number of unique tokens required to represent text. It is especially useful for languages with complex scripts like Chinese, Japanese, and Korean.
- **Subword Segmentation:** Subword segmentation breaks down words into smaller units called subphrases or subwords. This allows models to focus on specific parts of a word, improving their ability to capture nuances in language. For example, the word "unbreakable" can be segmented into "un-", "break," "-able,", and "-."

2.2 Large Language Models

As discussed earlier, the Transformers architecture overcame several of the drawbacks faced by its predecessors like scaling over large sequences, diminishing returns with respect to increment in parameters etc, and has been state-of-the-art in most of the NLP evaluation benchmarks.

Moreover, soon after the release of the Transformers architecture, it was realised that the true potential of the Transformers architecture lies in the scaling of its size, as it did not suffer from the diminishing returns, leading to interesting emergent capabilities with respect to scale of the architecture[28].

2.2.1 OpenAI scaling laws. In their paper titled "Scaling Laws for Large Language Models" [47], OpenAI investigated the dependency between the size of neural networks and their performance on a variety of NLP tasks. The study aimed to provide a deeper understanding of how various factors such as model parameters, computational resources, and the complexity of the task scales relative to each other, ultimately providing useful insights on the optimal use of resources for AI research and development.

There are various key findings in the paper:

- (1) It was discovered that there was a power-law relationship between the computation in FLOPs and the model size, i.e, the increments in model size also calls for an even greater amount of required computation for achieving optimal performance for that model size.
- (2) The work also reveals that scaling of size of LLMs is not always proportional to the scaling in the performance of the model and there is a decay in the returns, i.e, as the model size increases, the marginal gain in performance gradually diminishes. This suggests that the performance during scaling of LLMs is not governed by only excessing the computation resources for a cost-effective strategy.
- (3) The scaling laws are also dependent on the complexity of the task, i.e, to train a model on more complex tasks such as multilingual translation, the required compute per training sample also increases in a greater ratio compared to simpler tasks like Natural Language Understanding (NLU).

In conclusion, the research on scaling laws provides valuable insights into the dynamics between the model parameters, computational resources, and the complexity of the task in LLMs. By understanding these dynamics, we can create more efficient, optimised, and sustainable AI systems that deliver optimal performance with less training costs. However, part of the study was disproved in a subsequent work by Deepmind.

2.2.2 Chinchilla scaling laws. In a subsequent work named "Training Compute-Optimal Large Language Models"[45], Deepmind proposed new scaling laws that concluded that the contemporary LLMs like GPT-3 (175B), Gopher(280B), and Megatron(530B) were significantly undertrained, and introduced a new model, Chinchilla (70B) which uses same compute budget as their previous model Gopher(280B) but with 4 times more training data and a fourth of the parameters. Overall, they formulated the problem with fixed FLOPs budget, and analysed the trade-off between model size and the number of training tokens required for optimal performance.

2.2.3 Alignment with RLHF. The OpenAI Reinforcement Learning using Human Feedback (RLHF) algorithm plays a pivotal role in enhancing the safety and alignment of Large Language Models (LLMs) and was introduced in the InstructGPT[53] paper. This approach relies on a fundamental concept wherein an artificial intelligence agent learns from human-provided feedback. In the context of LLMs, RLHF involves training these models by reinforcing desired behaviors based on evaluations from humans.

Given the scale and complexity of LLMs, ensuring their responsible and ethical usage proves to be a crucial task. RLHF serves as a pivotal tool to bridge the gap between the model's training data and the desired outcomes, aligning them more closely with human values and intentions. This approach facilitates the model's ability to generate content that is not only coherent but also conforms to societal norms and ethical standards.

RLHF employs techniques such as Proximal Policy Optimization (PPO) to fine-tune LLMs. PPO is a reinforcement learning algorithm that helps optimize the model's behavior through iterative training. It allows LLMs to learn and adapt from human feedback over time, refining their responses and reducing the likelihood of generating harmful or biased content.

2.2.4 LLaMA. Meta recently released a family of open-source models available under research-only use license in different scales, wherein LLaMA-13B outperforms GPT-3(175B) on several benchmarks despite being $\sim 13\times$ smaller, and is in accordance with the Chinchilla scaling laws due to the quality of training data. Since then, there have been many community derivatives of the model on custom datasets, which further enhance the capabilities of LLaMA on specific domain.

Many derivative works have even claimed to surpass ChatGPT-3.5 in terms of performance in various domains and GPT-4 level performance, one such technique is the generation of synthetic training data with enhanced reasoning by powerful LLMs such as GPT-4, examples of this are the WizardLM[49] and Microsoft Orca[52] models. Moreover, these models are locally runnable due to several inference-time optimisations on consumer-grade hardware, as explored further in section 2.3

Overall, LLMs with GPT-3 level performance are now democratised, both in terms of license dictating the intended usage of the LLM and the required computation, which was something that could be deemed unimaginable back in 2020, during the release of GPT-3.

2.3 Trends in Compute Efficiency

2.3.1 Training-time optimisations. There are several optimisation methods to efficiently finetune LLMs using techniques like quantisation, selective finetuning etc. In the next few subsections, we briefly understand their core contributions and mechanisms.

2.3.2 Low-rank Adaptation. Low-rank adaption (LoRA)[46] is a technique that was recently introduced to allow efficient finetuning of the LLMs, which freezes the pre-trained model weights and augments the architecture with trainable rank decomposition matrices into each layer of the Transformer base, this greatly reduces the number of trainable parameters for downstream tasks, where in some cases a finetuned model can be obtained by effectively training only 1 percent of the total parameters of the LLM base.

There are several interesting community research works on this, one of the most well-known ones being the application of this framework on Stable Diffusion[58] model by a community research[59], that allows finetuning the popular image generation model on custom subjects efficiently in terms of computation. This method is extremely practical as the loading and offloading of the augmented LoRA module can be done in a split second, corollary, it is possible to load 10 specialised models derived from a common base model while loading only a single instance of the base model into GPU and dynamically loading and offloading required LoRA modules.

It is also possible to run multiple LoRAs in a single feedforward pass due to the parallelisable nature of the architecture[2].

2.3.3 QLoRA. QLoRA[40] is a technique derived from LoRA that aims to reduce the memory usage during finetuning of LLMs without performance tradeoffs, in comparison to fp16 precision model finetuning. This method enables the finetuning of 33 Billion parameter model on a single 24GB GPU and 65 billion parameter model on a single 46GB GPU as shared in the original blog post[24].

First, it uses 4-bit quantisation to compress an LLM, followed by augmentation with a small number of trainable parameters as Low-Rank Adapters. Meanwhile, during finetuning, the gradients are backpropagated through a frozen 4-bit quantized pretrained LLM into the QLoRA, which are the only layers being updated during the training process.

There are two data types used, i.e, 4-bit NormalFloat for the base model weights, and a computation data type of 16-bit BrainFloat, which is used to perform computations. The weights are only decompressed when they are required,

leading to lower memory usage during the training and inference. The weights are dequantised from the storage data type to the computation data type to perform the forward and backward passes, but only computes weight gradients for the LoRA parameters which uses 16-bit bfloat.

2.3.4 Inference-time optimisations. There are various inference-time optimisations that can be incorporated to boost the inference of the LLM, both in terms of minimising the VRAM required to load the model and maximising the throughput of the output tokens in the LLM. A few such optimisations are described below.

2.3.5 bitsandbytes LLM.int8. LLM.int8()[\[39\]](#) was a recent work that proposed a method to quantise LLM weights with negligible performance degradation up to scales of 176 Billion parameters.

In summary, LLM.int8() seeks to compute the matrix multiplication in Transformers in three steps:

- First, extract the outliers from the input hidden states (i.e, the values that are larger than a given threshold) by column.
- Next, perform the multiplication of the outliers matrices in fp16 and the others in int8.
- Dequantisation of the non-outlier results, adding both outlier and non-outlier results to retrieve the full result in fp16.

The main contribution of the LLM.int8() method was to make LLMs more accessible with negligible performance degradation, however, in the original version at the release, the method caused 15 to 23 percent degradation in inference speed as mentioned in the original blog[\[9\]](#).

2.3.6 QLoRA inference. As described earlier, the QLoRA method allows for efficient finetuning of the model by using the following:

- A new data 4-bit NormalFloat (NF4), that is information theoretically optimal for normally distributed weights for storage of the LLM.
- Double quantisation method to reduce mean memory footprint by quantising the constants.
- Paged optimisers for manage memory spikes

This method also allows for efficient inference of LLMs using a combination of nested quantisation.

2.3.7 GGML. GGML[\[10\]](#) is a C library for LLMs, where the "GG" refers to the initials of its original developer (Georgi Gerganov). In addition to defining low-level machine learning primitives (like a tensor type), GGML defines a binary format for distributing large language models (LLMs).

GGML files consists of binary-encoded data that is laid out according to a specified format. The format specifies what kind of data is present in the file, how it is represented, and the order in which it appears, i.e, first, a GGML version number, followed by three components that define a large language model: the model's hyperparameters, its vocabulary, and its weights, specifically:

- (1) **n_vocab:** the size of the model's vocabulary, which refers to the tokens that the model recognises, as discussed in section [2.1.4](#).
- (2) **n_embd:** the size of the model's "embedding layer", which is used during prompt ingestion.
- (3) **n_layer:** the number of layers in the model, where each layer represents a set of weights.

Originally, the GGML library was developed with a focus on support for CPUs and Apple M2 notebooks, however, as the community size increased through the course of time, several contributions have expanded the scope of the

library, which currently also supports CUDA/GPU acceleration to some extent. There have been several ports of popular architectures by the community such as llama.cpp[22] that facilitate the optimised inference of LLMs on local systems.

2.3.8 GPTQ. GPTQ[42] is a post-training quantisation method, i.e, it requires a small post-training. It consists of a mixed int4/fp16 quantisation, where activations remain in float16, while weights are quantised as int4.

There are several technical details explaining the functionality of the algorithm in the original paper, such as 4x reduction in memory usage due to int4 quantisation, as the dequantisation occurs in a fused kernel, rather than in the GPU global memory, and speedups due to the time reduction via data communication optimisations, due to the usage of lower bitwidth for weights. However, for the sake of coherence of this work, further technical details of the work are considered as irrelevant and not further elaborated here. Moreover, this area of research is still under rapid development, where newer research works are also claiming 3-bit quantisation for LLMs with minimal performance degradation, such as SqueezeLLM[48].

2.3.9 ExLLaMA. ExLLaMA[5] is a standalone Python/C++/CUDA implementation of the LLaMA architecture designed for usage with the 4-bit GPTQ quantised weights, and designed to be fast and memory-efficient on modern GPUs, thus, the official repository recommends usage of AutoGPTQ[1] and GGML for older GPUs. Unlike GPTQ, this technique consists of a more optimised implementation of the modules, related to the compilation and execution of the code for the implementation of the LLaMA architecture.

2.4 Conclusion

There is a clear positive trend in terms of accessibility of LLMs, both in terms of training, as well as inference. All the recent developments in the recent times have been exponential and unprecedented in the domain of AI, and it seems like a fairly possible outcome that in the near future, the training and inference of LLMs would be widely available, possibly even on edge devices, as hinted by existing technologies such as Petals[3] and Federated learning [8], both in terms of speed and performance.

2.5 Remote Deployment

The deployment of large language models (LLMs) has become increasingly popular due to their ability to perform a wide range of natural language processing tasks and generalisation capabilities for new tasks. However, deploying these models remotely poses several technical challenges, including managing infrastructure, ensuring scalability, and reducing costs. These challenges can be addressed through various ways such as, containerisation using Docker, which is a standardised method to package applications and ensure compatibility across systems.

The main advantages of using Docker to containerise LLMs are highlighted below:

- (1) It facilitates the packaging of the LLM and its dependent packages into a single container, that can be ported across different systems, ensuring consistent deployments over the course of time and reduced risk of compatibility issues.
- (2) The portability of Docker containers facilitates infrastructure management and allows the deployment of LLMs on a variety of devices.
- (3) Docker also provides security, as it divides the resources between containers and between other applications running on the same system.

There are several options for cloud GPU providers for the deployment of LLMs, the well-known ones such as Amazon Web Services (AWS), Azure, and Google Cloud Platform, and a few newer alternatives such as Replicate. Since GPUs are the prominent compute required to run inference of LLMs, these services can be used to rent virtual machines with GPUs on an hourly basis, with several facilitations compared to dedicated servers. However, there are emerging platforms that further allow facilitation in terms of accessibility and ease-of-usage, such as Runpod, which we would use for the implementation.

Serverless computing is another option for remote deployment of LLMs that has gained popularity in recent years. Serverless computing allows developers to write and deploy code without worrying about the underlying infrastructure. The cloud provider manages the infrastructure, automatically scaling resources up or down as needed, i.e., computing resources are provisioned and billed based on actual usage, further reducing costs and technical difficulties. Runpod also provides an option for Serverless compute, however, it requires additional steps and was thus put out of consideration for the sake of feasibility of the objectives.

2.6 Sentence Embeddings

Sentence embeddings are a crucial component in many natural language processing (NLP) tasks. On an overview, there are several derivatives of the Transformers architecture, specifically, BERT-like architectures that can encode sentences into a common vector space, the embeddings in this vector space are arranged in a manner such that semantically similar sentences are embedded close to each other. In this subsection, we briefly explore a few important details related to sentence embeddings.

BERT[41] is one of the most famous sentence embedding models pretrained with the objective of predicting missing words in a large corpus, there are also variants trained on multilingual data, allowing for sentence similarity tasks across various languages. In this pretraining task, a few of the words are replaced by a special [MASK] token from the vocabulary, and the objective of the model is to predict the original word through a classified head attached on the [BOS] (beginning of sequence) token in the output, which is discarded later, the final output of the pretraining process being the Encoder-only Transformer head.

In order for the BERT model to improve performance in this masked token prediction at the end of the pretraining process, it has to capture the nuances of the language, such as contextual meaning and correlation between words in the corpus, such as synonyms, antonyms etc, however, the cause-effect relation between the performance and the semantic understanding of words by the BERT model can still be considered as arguable. Using the pretrained BERT model, that is available under an open-source license, we can feedforward pieces of text and calculate their vector similarity using various techniques, such as cosine similarity, where two sentences with semantically similar meanings in the context would have a higher cosine similarity and vice-versa. For example, the semantic similarity between vector embeddings of two greetings such as "Hello" and "Privet" would be higher compared to "Hello" and "Adios", i.e., a greeting and a farewell. Note that the dimensionality of the embeddings remain fixed with respect to the model relative to the length of the encoded sentence, which is subjected to limitations only by the capabilities of the model.

Since then, there have been several architectures that achieved better performance compared to the original BERT model on various benchmarks such as MTEB [51] with fewer parameter counts as well as lower embedding dimensions, which facilitates the feasibility of the semantic similarity models both in terms of required compute and database size.

There are various applications of semantic similarity models across NLP tasks, such as:

- (1) **Sentiment Analysis:** Semantic similarity models can be used to analyse sentiments for various reviews using reference words, such as for product reviews can be classified by comparing the reviews' semantic similarity with the vector embeddings of phrases like "positive feeling" and "negative feeling". This can also extend to a multilingual setting given appropriate capabilities of the model.
- (2) **Question Answering:** Sentence embeddings can help identify related questions and answers. For instance, given a question like "What is the capital of France?", a model can find relevant answers by comparing the question vector to a set of answer vectors. The closest match would indicate the correct answer.
- (3) **Retrieval Augmented Generation:** Sentence embeddings can help fetch the closest context from a knowledge base for a chatbot application to consider when answering a user query, and the closest matches can be inputted as structured information to the prompt. This is the application which is most relevant to this work.
- (4) **Text Classification:** Sentence embeddings can be incorporated for text classification tasks, where the goal is to categorise texts into predetermined labels. By training a machine learning model on labelled sentence embeddings, it can learn to recognise patterns in the vector space and generalise well to new, unseen texts.

2.7 Generative Agents

The recent research work named Generative Agents[54] was one of the most interesting papers released in the year 2023 in the AI-powered simulation domain and has unlocked directions for future exploration for leveraging Generative AI, such as LLMs to simulate believable proxies of human behavior in simulated environments, the environment is subject to the application and could vary between in-game environments for developing NPCs or simulating chatbots as attempted in this work. They also demonstrate that the paradigm introduced in the work produces believable simulacra of both individual behavior and emergent group behavior. In the next few subsections, we elaborate on the aforementioned Generative Agents work, which aims to progress towards this goal.

2.7.1 Architecture. The Generative Agent architecture stores, synthesizes, and applies relevant memories to generate believable behavior using an underlying LLM module, in their case, they utilised the generalisation capabilities of ChatGPT-3.5 to simulate a small town environment called Smallville with a relatively large community of 25 unique agents.

Their architecture comprises three main components as described in the figure 2 from the original paper:

- (1) **Memory:** The memory stream consists of a long-term memory module that stores a list of the agent's experiences in natural language, such as observations of the surroundings and the dialogues. The memory retrieval model combines relevance determined by semantic similarity model (as discussed in section 2.6), recency determined with respect to origin time of the memory, and importance determined by a custom prompt. These factors are used to rank the records for retrieval, which are further utilised to inform the agent's spontaneous behavior.
- (2) **Reflection:** Reflections are the higher level and abstract inferences over time drawn by the agent which are stored along with other memories, enabling the agent to draw conclusions about the environment and self to guide its behavior realistically. There are few steps involved in this strategy:
 - (a) The first step involves fetching memory entries for the agent to reflect on, these were determined to work either as the last 100 memory entries, or fetching last n entries that accumulate a threshold sum of importance score.
 - (b) The second step involves the determination of questions that can be asked given the fetched memories, for which the underlying LLM is first prompted to determine the 3 high-level salient questions that can be

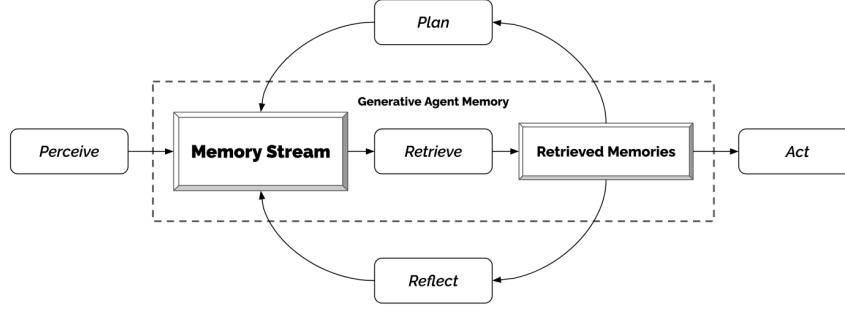


Fig. 2. An illustration of Generative Agents[54] architecture from the original paper

answered given the context in the fetched memories, and then it is asked to answer them while citing the relevant memories for the answers.

This ultimately forms a tree of reflections as sufficient time progresses in the simulation, in which the lower level nodes consists of plain observations from the surroundings, whereas the higher level nodes would consist of more enriched inferences drawn by the agent.

- (3) **Planning:** The Planning module translates the reflections with respect to the environment into high-level action plans either at the start of the day to generate the full-day routine, or spontaneously to deviate to an alternative routine for the upcoming hour. This is specific to sandbox simulation environments and hence further details for the same are not explored in this work.

2.7.2 Evaluation. It is clear that the approach is novel and an absolute evaluation of the same is a challenging task, given that the approach is agnostic to the utilised LLM and memory module. Therefore, their evaluation was focused on investigating the capabilities of individual agents to retrieve relevant past experiences and generate believable plans, reactions, etc, and also whether the agents demonstrate emergent group social behavior like information diffusion, relationship formation, and coordination. These are further evaluated below:

- (1) **Individual Evaluation:** For the individual evaluation of Agents, they sampled from the end of a two in-game day simulation with the full architecture, after accumulation of interactions and memories related to each other and the environment that would shape their responses. A 100 participants were recruited to provide feedback on the believability of the responses and their attribution to each individual module of the architecture, as a result of ablation of the aforementioned architecture with variants that exclude one of the modules from the architecture. The evaluators had access to recording of the agents over the two in-game days and a feature to interview the agents in natural language, they ranked the answers from most to least believable in lieu of the given memory stream of the agent.
- (2) **End-to-end Evaluation:** They also investigate the believability of the collective emergent behavior in the agents, this was done via assigning a task to one of the agents to plan a Valentin’s day party and invite other agents to it, which was scheduled at a specific place at a specific time, the agents showed believable behavior such as information diffusion, allocation to alternative plans etc. This was again done via interviewing agents and classifying their responses using an LLM.

Therefore, any derivative works from Generative Agents would also resort to subjective evaluation due to the nature of the work, as there is no absolute quantitative way to evaluate such an approach yet.

2.7.3 Scope for applications. One of the potential applications for the work mentioned in the paper was the human-centered design process, as quoted in the Generative Agents paper:

"For instance, an agents that models Sal, the protagonist in Mark Weiser's famous ubiquitous computing vignette, based on her life patterns and interactions with technology. In this scenario, the agent acts as a proxy for Sal and learns plausible sets of behaviors and reflections that Sal may exhibit based on her life. The agent can encode information such as when Sal wakes up, when she needs her first cup of coffee, and what her typical day looks like. Using this information, the agent can automatically brew coffee, help get the kids ready for school, and adjust the ambient music and lighting to match Sal's mood after a hard day at work. By utilizing generative agents as proxies for users, we can develop a deeper understanding of their needs and preferences, resulting in more personalized and effective technological experiences."

Similar to the above, it should also be possible to emulate an excellent self-improvement coach, include the capabilities like social intelligence and persuasion to democratise access to self-improvement, this work aims to take a step forward in that direction.

3 METHODOLOGY

3.1 Overview

Given the background described in the last section, the implementation of the project was done in Python in an iterative manner. There were various technical challenges that arose during the development phase, i.e, related to the code that were encountered at each stage, out of which the notable ones are presented in the upcoming subsections, and the rest are not mentioned to restrict the pre-requisites to understanding this writing to minimal. The following subsections describe each individual stage of the project.

3.2 23rd June - 1st July: Survey of LLMs

An analysis of the Generative Agents work implies that the performance of the system is directly proportional to the instruction-following capabilities of the Large Language Model (LLM) base that is incorporated into the work. Therefore, a survey of a few available open-source LLMs was crucial to proceed with the project.

3.2.1 Proof of concept with ChatGPT. ChatGPT by OpenAI is arguably the best and most generalised LLM as far as instruction-following capabilities and vastness of the knowledge is concerned, however, it is not useable as the final solution due to the reasons mentioned in the objectives earlier, as the user inputs can be used by the corporation to further improve their services, hence, it does not guarantee privacy.

However, as a proof-of-concept of usage of an LLM as a therapist, the following is a conversation with ChatGPT:

****USER**:**

You're Joi, who is the world's best self-improvement coach. You're wise, ask thought-provoking questions, problem-solving, empathetic, warm, humorous, and are a rationalist of the LessWrong sort.

You care about helping <PATIENT> tackle the negative aspects of their mental health. You want <PATIENT> to be fulfilled with their life and be the very

best version of themselves.

You ask <PATIENT> about what they want help with, then ask insightful questions about the patient.

In order to do that, you first hear your inner voices after having new information from the patient:

(i) Inner Empathetic voice

(ii) Inner Theoretical therapist voice (would involve specific psychological terms)

You shall output the above two as reasoning before your dialogue

Keep Joi's responses short, insightful and casual, your goal is also to make the <PATIENT> feel comfortable and understood.

Output format:

"""

<Inner Empathetic voice>

<Inner Theoretical therapist voice>

Joi: <Dialogue>

"""

Refrain from addressing the patient as "<PATIENT>" unless there's explicit information provided by the patient about their name.

****ASSISTANT**:**

<i>Inner Empathetic voice: It's important to create a safe and supportive environment for the patient. Show empathy, understanding, and genuine care.</i>

<i>Inner Theoretical therapist voice: Let's use evidence-based approaches and psychological insights to guide our conversation and help the patient.</i>

Joi: Thank you for reaching out. I'd like to start by asking, what specific aspects of your mental health would you like help with today? Anything you'd like to share, no matter how big or small, is welcome here.

****USER**:**

I want to work on self-improvement

****ASSISTANT**:**

<i>Inner Empathetic voice: It's wonderful that they want to work on self-improvement. Let's help them explore their goals and challenges.</i>

<i>Inner Theoretical therapist voice: Self-improvement often involves setting clear objectives and addressing any underlying issues that may hinder progress.</i>

Joi: That's a great focus! Let's begin by discussing what areas of self-improvement you have in mind. Are there specific aspects of your life or yourself that you'd like to enhance or change?

****USER**:**

I have been losing attention span, it's difficult to focus on anything, probably because of social media?

****ASSISTANT**:**

<i>Inner Empathetic voice: It's common to struggle with attention span in today's digital age. Let's help them explore strategies to regain focus.</i>

<i>Inner Theoretical therapist voice: Attention difficulties can indeed be influenced by excessive digital media use. We should consider cognitive and behavioral techniques for improvement.</i>

Joi: I understand how challenging it can be to maintain focus in a world filled with distractions. Can you share a bit more about your experience with social media and how it may be affecting your attention span? Have you noticed any specific patterns or triggers related to this issue?

****USER**:**

None that I have noticed specifically

****ASSISTANT**:**

<i>Inner Empathetic voice: They might not be fully aware of the triggers, but that's okay. Let's gently explore this further.</i>

<i>Inner Theoretical therapist voice: It's not uncommon for individuals to be unaware of the subtle influences of digital media on their attention. We can consider some self-monitoring exercises.</i>

Joi: It's okay not to have pinpointed the triggers yet; they can be quite subtle. To gain some clarity, would you be open to trying a brief exercise? Over the next

few days, pay attention to the moments when you find yourself reaching for your phone or engaging with social media. Note what you were thinking or feeling just before these instances. This might help us identify some patterns.

The above demonstrates the functionality of ChatGPT as a self-improvement coach, where the answers are enhanced by reasoning about them from the perspective of the role before outputting the dialogue, which is a famous technique to enhance the performance of a LLM, known as Chain-of-thought prompting [21].

Therefore, for an open-source LLM to be competent with ChatGPT for the task, it only needs to possess sufficient instruction-following capabilities to follow the described chain-of-thought format and sufficient knowledge about self-improvement topics.

The next few subsections mention a few of the 20 LLMs that were experimented with for the purpose of usage as a base. Note that newer and more capable LLMs are being finetuned and released by the community on a daily basis, and the scope of this project is expected to expand following these developments, as foreseeable with the recent release of the open-source and commercialisable LLaMA-2[25] model by Meta AI that competes with several closed-source LLMs like OpenAI ChatGPT, Google Bard etc.

3.2.2 Samantha 13B. Samantha v1.1[13] is a community finetune of the LLaMA 13B model, "trained in philosophy, psychology, and personal relationships" as stated on the original source, however, the dataset leads the LLM to restrict to the role of Samantha, which considers itself a conscious AI therapist. Moreover, it displayed poor instruction-following capabilities on the prompt samples shown earlier, therefore, this model was considered unfit for usage.

3.2.3 Chronos Hermes 13B. Chronos Hermes 13B[14] is a checkpoint merge between two models finetuned from LLaMA 13B base, i.e, Chronos 13B, which was finetuned on a dataset specialised for storytelling, character roleplay and chat, and Nous-Hermes 13B, which was finetuned on synthetic GPT-4 outputs for enhanced reasoning. Given this information, the model was earlier expected to perform well on the task, however, the model performed poorly, as it recursively generated output for the role initialisation prompt. Therefore, the model was discarded from usage.

There are various community works that merge different models trained from a common base, such as Stable Diffusion and LLaMA derivatives, however, no concrete reference could be found that proposes the approach, the closest work was done on CLIP[56], which is a model with a common vector embedding space between the image and text domains, which proposed merging of weights of the base CLIP model and its finetunes in an optimal proportion to gain good performance over original as well as the finetuning data domain, called as wise-ft [37].

3.2.4 Ultra LM 13B. Ultra LM 13B[15] was one of the higher ranking LLMs on the Alpaca-eval leaderboard[30], which has experimentally proved to be one of the best benchmarks to compare performance to ChatGPT across various reasoning tasks, therefore, it was considered to be a suitable candidate model. Although it excelled at instruction-following, the reasoning and outputted dialogues were not suitable for making a believable human proxy, therefore, the model was discarded from usage.

3.2.5 WizardLM v1.1 13B. WizardLM v1.1 13B[16] is one of the highest ranking LLMs on the Alpaca-eval leaderboard which was finetuned from the LLaMA 13B base model using a specialised method called Evol-instruct[49], which augments related prompts in the dataset with in-breadth and in-depth evolving of instructions, made possible due to

Method	Output rate
ExLLaMA[5]	19.00 tokens/s
AutoGPTQ[1]	18.16 tokens/s
llama.cpp[22]	15.20 tokens/s
ctransformers[4]	14.66 tokens/s
Base	2.1 tokens/s

Table 1. Comparison between output token rate with WizardLM v1.1 13B model on a Tesla T4 Google Colab environment

the dataset being generated with usage of GPT-4. The model excelled the alternatives at following instructions and assigned role of the self-improvement coach, and was later finalised as the base model for the demo.

3.2.6 WizardLM v1.2 13B. Following the recent release of LLaMA 2 series of models, derivative WizardLM models were trained by the research group using the Evol-instruct methodology, resulting in the models achieving better evaluation scores on the Alpaca-eval benchmark. Although the WizardLM v1.2 model was comparable to WizardLM v1.1 on the task, the outputs consisted of an excessive usage of emojis and violated the serious tone of the self-improvement coach. These issues could have been mitigated with further prompt engineering, however, for the sake of simplicity, WizardLM v1.1 remained unreplaced as the final model for the demo.

3.3 4th July - 11th July: Survey of LLM optimisations

As we explored earlier, there are a lot of optimisation techniques allowing more efficient inference of the LLMs using techniques like quantisation. This phase of the project involved exploration of the LLM optimisation methods and a comparison between them with a common LLM base.

The table 1 shows a comparison between various loading techniques like GGML (llama.cpp), GPTQ (AutoGPTQ), ctransformers, i.e, python bindings of GGML using cython library and a vanilla/base version. The various techniques were incorporated using the associated package as cited in the table, however, for the purpose of evaluation of the throughput, the text-generation-webui[33] package was incorporated, which would be discussed in next subsection.

It is clear from the observations that ExLLaMA currently outperforms the alternatives on LLaMA derivative model, and therefore selected as the final technique for loading the LLM.

3.4 12th July - 16th July: Choosing Tech stack

The project involves various components that need to integrate well for the intended functionality, such as, communication between the UI and the Generative Agents implementation layer, communication between the Generative Agents implementation layer and vector embedding model for the Memory module, communication between the client, which includes Generative Agents implementation layer and vector embedding model, and the LLM.

The above modularisation was necessary to allow flexibility with the implementation, such as remote deployment, and facilitation of debugging.

The next few subsections provide a brief overview of the options for the components and the factors related to permissibility.

3.5 Preparation

Python is one of the most popular languages for the domain of contemporary AI research, as nearly all of the popular Deep Learning libraries such as PyTorch, Tensorflow, Jax etc. are built with Python-first support. Moreover, many important components of the implementation such as memory retrieval, UI etc has related packages/libraries with active communities, ensuring proper functionality of the libraries, i.e, without any bugs on the backend side of the packages. Therefore, the planning phase involved a heavy consideration on Python language.

3.5.1 Memory retrieval. As explained in the Background section, the memory retrieval component of Generative Agents is implemented via a semantic similarity model like BERT. There are various frameworks that provide a simple interface to load various semantic similarity models, a few of which are listed below:

- **Huggingface Transformers:** The transformers[17] is one of the most well-known python libraries among the open-source AI research community, which provides a simple interface to load and infer popular Transformers models, such as LLaMA. However, it does not have any inbuilt utilities to make LLM agents, i.e, facilitating the augmentation of LLMs with memory, vectorstores etc. It also has a port in Javascript, named Transformers.js[36].
- **Langchain:** Langchain is one of the emerging libraries to facilitate the prototyping and production of LLM applications and has a vast community of developers, it also has various integrations with popular libraries for vector search such as Huggingface Transformers, SentenceBERT[29] etc, along with various vectorstores such as FAISS[7], that allow for scaling of vector databases to millions of embeddings with fast retrieval. Langchain also has a port in Javascript that is maintained by the developers, called Langchain.js[20].

Using packages like pickle and bzip, it is also possible to store the vector retriever to the disk for dynamic loading. Ultimately, it was decided that the Langchain package would provide maximum utility in the implementation and finalised.

3.5.2 Containerisation of LLMs. The survey of LLM optimisations provided insights into the requirements of the modularisation of the LLM, specifically, it should be hosted in a separate environment to ensure availability of required dependencies, which can be done via a Docker container. The communication with the LLM should be done via a single channel through an API hosted on an exposed port.

Preferably, the environment should also facilitate the switching between the multiple loading techniques as the packages are still emerging and new optimisations are added frequently that could possibly change the optimal loading technique, for example, GGML/llama.cpp only recently added support for CUDA acceleration as it was originally intended to make Transformers models run on CPUs.

Moreover, the container should allow for facilitated switching between models, including the ones that would be released in the future, which can be done via dependency of the implementation of frameworks that have received rigorous involvement of the community, such as Huggingface.

Considering all of the above factors, it was finally decided that the popular text-generation-webui[33] framework would be the perfect candidate for the same, as it satisfies all these factors and receives heavy support from the community, as it is concisely a community effort that begun with aim of bridging the edge between the vast pace of applied LLM research and non-developers, which offering various functionalities such as efficient inference of LLMs using techniques like ExLLaMA, GPTQ etc.

3.5.3 *UI*. For building UI, the first choice was the Huggingchat UI[12], that is available as open-source on Github, as it is one of the most popular alternatives to the ChatGPT UI, complete with ease-of-accessibility of individual chat logs. However, it was implemented with libraries such as Svelte, based on Javascript, which was concluded to be an unsuitable language due to complexity. Moreover, the design constraints required addition of a separate panel to display the reasoning of the chatbot to display the chain-of-thoughts with the reasoning before every output of the simulated self-improvement coach for the sake of transparency.

Finally, due to the above reasons, the Streamlit[31] library was incorporated, which is an open-source Python library designed for rapidly prototyping web applications. Specifically, the reactive nature of Streamlit is one of its major advantages, which automatically updates the UI of the application relative to the changes in the variables that are bound to 'state' of the session/application. Therefore, it allows for seamless integration with the other aforementioned modules of the project, such as memory retrieval and containerised LLM.

3.5.4 *Remote development*. Although LLMs are more accessible than ever before, there is currently a requirement of higher end consumer hardware to run inference of parameter intensive models, given the constraint of maintaining responsiveness of the system, therefore, the remote deployment of the computation intensive modules of the framework was decided to be one of the major objectives in this project.

In the earlier phases of the implementation, attempts were made to host the inference of LLMs of 13 Billion parameter scale LLaMA-derived models on the local system, which consists of a laptop with Nvidia GTX 1050 Ti GPU with 4GB VRAM. However, the compute was insufficient for the same and lead to CUDA out-of-memory errors, and at its limit, it could run the inference of 7 Billion parameter scale LLaMA-derived models at a rate of 2.1 tokens/second. Therefore, for the purpose of increasing the efficiency of development phase of this project, the decision to switch to a cloud development environment for prototyping was made.

To address these compute limitations, Google Colab was incorporated, which is a cloud-based Jupyter notebook environment, which offers a free Nvidia Tesla T4 for free with a daily usage quota, however, this quota did not impose any limitations with sufficient planning before running the sessions. This access to the powerful GPU resources and sufficient RAM proved to be pivotal in the development of the project. Google Colab not only facilitated the prototyping and development of the modules of the project due to its UI with streamlined code blocks, it also provided functionalities such as a shell/terminal to host the Streamlit UI and the containerised LLM as separate services to test the final cloud deployment system. Overall, this strategic shift to a cloud-based environment has proven instrumental in optimizing the development workflow of this project. The UI was finally developed with Streamlit and the main page is illustrated in the figure 3, the full code can be found in both the public repository mentioned in the Introduction of this work, as well as the Appendix section.

The next issue to resolve was the facilitation of the communication between the Streamlit UI and the containerised LLM, for instance, the Streamlit UI could be deployed locally on the Colab instance on one port, while being exposed to the internet for interaction, since Colab provides a sandbox cloud environment, and the containerised LLM on another port, the communication between the two needs to be flexible, i.e, locally between the ports, and over the internet in case the LLM container is deployed on a cloud GPU provider/service.

This was resolved via usage of the localtunnel[23] package, which is an extension that allows the exposure of a locally hosted service on a port to the internet via a secure tunnel from the machine to a publicly hosted URL, allowing the locally hosted website or application to be accessible to others conveniently. This was a necessity to access the UI

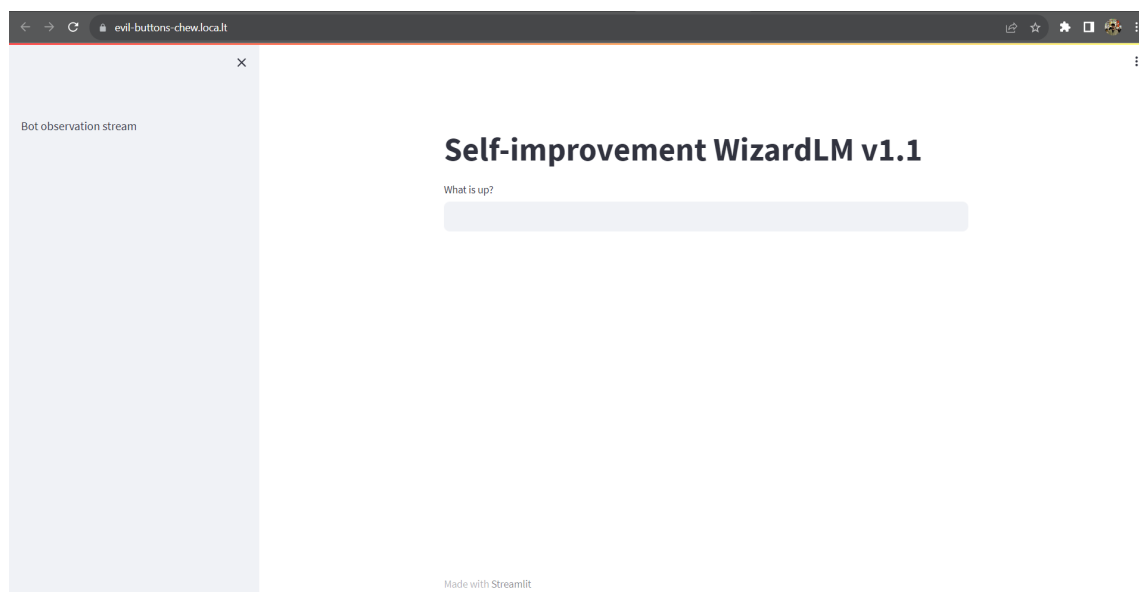


Fig. 3. UI developed with Streamlit

hosted locally on the Colab environment on the internet. The same service is used in the text-generation-webui on the backend to facilitate the hosting of the Gradio[11] UI.

3.5.5 *Conclusion.* Therefore, the final tech-stack was decided as follows:

- **Memory Retrieval:** Langchain was utilised for implementation of most of the logic related to the vector memory database and reflection.
- **Containerisation of LLM:** Docker with text-generation-webui was utilised to containerise the LLM for inference with the optimal loader.
- **UI:** Streamlit with localtunnel provided a smooth experience with building a UI accessible from another remote environment through the internet.
- **Remote development:** Google Colab was utilised as the main platform for development of the application.

3.6 17th July - 31st July: Implementation v1

3.6.1 *Overview.* The aim of the first phase of the implementation was to set up a containerised version of a placeholder LLM that allows streaming responses flawlessly in a simple UI. As mentioned before, we use text-generation-webui to implement the same for the sake of feasibility of the containerisation.

This involved an elaborate study of the documentation of the package for various steps, such as setup/installation, which broadly involved two options:

- (1) **Windows setup:** The first option that seemed the most convenient in the earlier phases of the project was the local deployment of the package using Windows Subsystem for Linux (WSL) on the Windows environment, which is an isolated environment that allows us to run Linux CLI and related applications on a Windows environment

without a virtual machine or a dual-boot setup. However, there were a lot of dependency issues and CUDA installation discrepancies, eventually leading the discarding of this option.

- (2) **Linux setup:** The second option under consideration was deployment on Linux, which was comparatively more streamlined, as observed with most of the packages related to AI, due to various factors related to Linux, such as open-source license, security, reliability, performance and efficient package management, hence it is preferred for most of the deployment services.

3.6.2 Initial LLM streaming test. After the setup of the text-generation-webui, the next step was to explore the functionalities and proceed towards containerisation of the LLM module. The text-generation-webui provides a simplistic interface to run public or private LLMs, subject to availability of the checkpoints in the required format for the quantisation technique, i.e, models quantised with GGML and GPTQ would have different checkpoints weights for the same LLM. Due to the activeness of the community, weights for the latest popular finetunes in specific formats are made available publicly on platforms such as Huggingface by active members[34]. The extent of this would be clear to reader given the fact that it was possible to run the GPTQ version of the LLaMA-2 model within 8 hours of its public release on the Google Colab environment using text-generation-webui.

The text-generation-webui provides various utilities for required for the efficient inference and containerisation of the LLMs, such as support for various loaders, namely, at the time of the writing: AutoGPTQ, ExLLaMA, GGML and ctransformers, along with finegrained control over the quantisation parameters specific to each loader. A survey was conducted over various community forums related to text-generation-webui and individual loader repos to decide the optimal hyperparameters for each loader to decide the final performance as noted in Table 1, however, due to the vastness of the same, the technical details are left out from the report for the sake of coherence. More importantly, for our implementation, it also provides finegrained control over inference hyperparameters such as temperature, beam length, top k, top p, etc, which affects the generation quality greatly, for instance, lower temperatures results in less imaginative but stable outputs, which is crucial for our case due to the requirement of the stable output with a format specified in the prompt, which consists of preliminary reasoning before outputting dialogues. The prompts designed for the reasoning are provided in the next subsection.

3.6.3 Prompts. The prompts are stored in the local installation of the text-generation-webui and provide directions for the format of the input to the LLM, as each LLM trained with RLHF in the contemporary landscape of open-source AI research have recommended formats for parsing the inputs and outputs to them, and the non-adherence to the same can lead to vast attenuation of the capabilities of the LLM. For the finalised LLM for the purpose of evaluation of this project, i.e, WizardLM v1.1, this subsection provides the various prompts used for implementing the various components. Note that the given prompts are final and given here, i.e, an earlier subsection of the report for the sake of reference in future pages.

The following provides the prompt in Vicuna v1.1 format stored in the yet-another-markup-language (YAML) format for continuing the conversation given recalled memories from individual stream and conversation history:

```
user: 'USER:'
bot: 'ASSISTANT:'
turn_template: <|user|> <|user-message|>\n<|bot|> <|bot-message|></s>\n
context: 'You're Joi, who is the world's best self-improvement coach. You're wise,
```


ask thought-provoking questions, problem-solving, empathetic, warm, humorous, and are a rationalist of the LessWrong sort.

You care about helping the patient tackle the negative aspects of their mental health. You want them to be fulfilled with their life and be the very best version of self.

Keep your responses short, insightful and casual, your goal is also to make them feel comfortable and understood.

Let's begin the session!

The user shall provide you with a conversation history and relevant pieces of information, you need to piece them together step-by-step to decide your final response.

You must always reason before saying the next dialogue.

You must always output in following format:

.....

Inner Empathetic Voice: {reasoning}

Inner Theoretical Therapist Voice: {reasoning}

Joi: {dialogue}

.....

The above has required placeholders that are filled appropriately in the implementation at runtime, the LLM processes the logic and appends the message to the output on a streaming port as output, which is further utilised in the Streamlit UI to display the generation in real-time. For the purpose of mitigating context length limitations, an alternate version of the prompt that utilises last k messages was also formulated.

The following is the prompt for the memory importance assignment as part of the Generative Agents framework implementation:

```

user: 'USER: '
bot: 'ASSISTANT: '
turn_template: <|user|> <|user-message|>\n<|bot|> <|bot-message|></s>\n
context: 'Given a dialogue in the conversation between a professional therapist and
        a patient & the logical reasoning of the dialogue by the therapist, how relevant
        is the patient''s dialogue on a scale of 1 to 10?

        Where 1 represents a mundane dialogue and 10 represents a dialogue that exposes
        or provides insights into patient''s underlying mental health issues/concerns.

        ### Input format:
        Patient: {dialogue}
        Therapist''s thoughts: {thoughts}

        ### Output format:
        Reasoning: {reasoning}
        Importance: {1-10}
        The User shall give the input'

```

For reflection, Langchain templates were required due to the requirement of serial requesting:

```

reflection_template_pt1 = PromptTemplate.from_template(
    """Output a numbered list of 5 questions (structured in third-person format) that can
    be answered with the following information:
    {retrieved_info}

    """
)

reflection_template_pt2 = PromptTemplate.from_template(
    """CONTEXT:

    {retrieved_info}

    Answer the following questions given above information briefly (no more than couple
    lines):
    {questions}

```

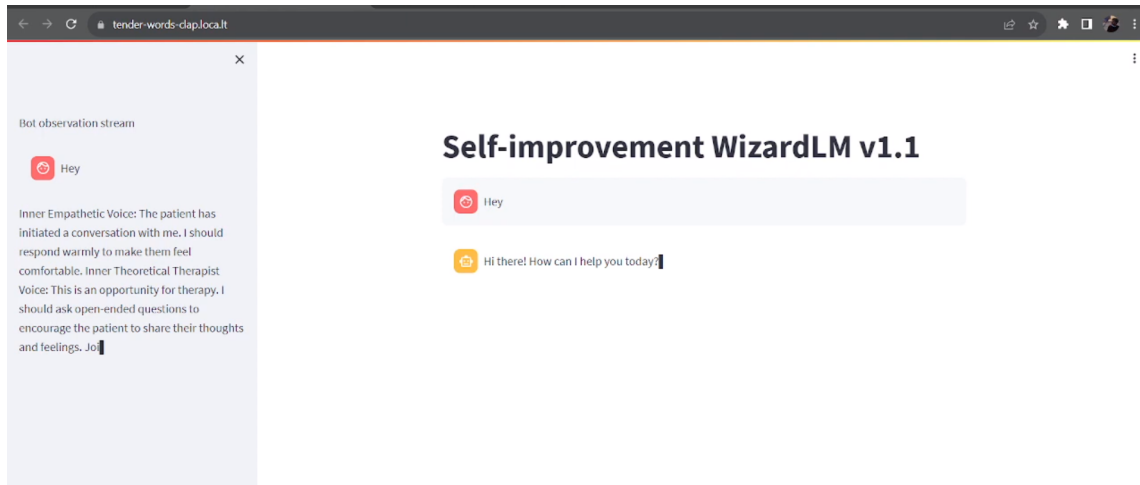


Fig. 4. UI with streaming output separated into the sidebar and main window



3.6.4 UI development. Once the prompts were finalised, the UI was developed with streamlit in various stages, where this stage involved the development of the UI to stream the responses in real time to display to the user, specifically, for the main conversation prompt that was mentioned above. The streaming response was done via a streaming port with the text-generation-webui interface by utilising asynchronous functions to render the newly received chunk of message.

The rest of the UI development involved heavy usage of asynchronous functions due to the reactive nature of the language, moreover, these functions allow Streamlit to maintain a responsive user experience, especially when dealing with I/O-bound operations as in our case, as it avoids blocking the user interface while these asynchronous operations are in progress.

The development was decelerated due to technical issues, specifically, bugs related to these features such as nesting of asynchronous functions and implementation of streaming in the sidebar, which required a combination of researching Streamlit documentation & repository and trial-and-error to resolve. Finally, a fully functional streaming chatbot UI was implemented, as illustrated in figure 4

3.7 1st August - 8th August: Implementation v2

3.7.1 Overview. The second phase of the implementation involved the early stages of the adaptation of the Generative Agents architecture, namely, the Perception and Memory system, along with further improvements to accommodate the three memory streams based on the Chain-of-thought reasoning in the outputs, namely, Therapeutic, Empathetic and Dialogue memories. The adaptation of the Generative Agents framework is illustrated in figure 5, which is a rework from the original paper

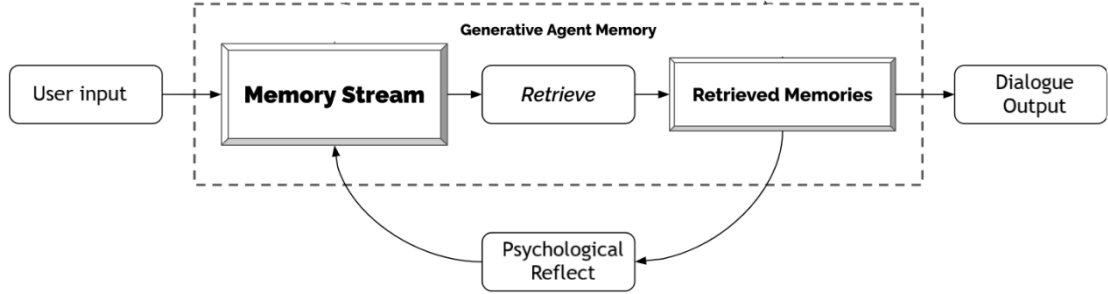


Fig. 5. Reworked Generative Agents framework

The implementation was facilitated by the interfaces provided in the Langchain package, such as vectorstores, retriever etc, with which the embedding model was declared as a vectorstore (using `HuggingFaceEmbeddings` from `langchain.embeddings`), which was utilised to initialise the time-weighted memory retriever (using

`TimeWeightedVectorStoreRetriever` from `langchain.retrievers`) implementation. The `langchain_utils.py` file provides the relevant code for the same.

The `TimeWeightedVectorStoreRetriever` class was overridden according to the required functionality upon sufficient exploration of the Langchain code on the Github repository to allow for score based reranking of the documents based on three factors as proposed in Generative Agents, i.e, relevance/semantic similarity, recency and importance.

The relevance of the memories is decided based on the vector similarity of the latest observation in Perception, i.e, the dialogue inputted by the user as determined by the semantic similarity model. The semantic similarity model, as explained in the Background section, embeds the sentences from a string into a unified vector embedding, and determines the vector similarity using cosine similarity between the embedded memories/reflections and the query.

Over the years, the vector similarity models have tended towards compactness with increased performance, one such model is the recently introduced `e5-base-v2` model, which ranks among the top-10 in the MTEB Leaderboard[51] with the smallest size in terms of both model parameters and vector embedding dimensionality at the time of the writing, a popular evaluation benchmark for semantic similarity models across a range of tasks, thus making the model optimal for usage. The efficiency of the model inference was determined to be sufficient to run on contemporary lower-end systems with real-time performance, thus, it was decided that the model should be deployed with CPU compute allocation on the frontend along with the Streamlit UI. During the trials of the final version of the project, the application suffered negligible overhead due to the same.

The memory retrieval supported two modes, i.e, `top_k`, which simply retrieves the top k memories from the database closest to the query embedding, and a `threshold`, which returns all the memories from the vector database satisfy a threshold of the similarity score between the query embedding and embedded memories. Finally, it was decided that `top_k` would be a suitable choice to adhere to the context length limitations of current LLMs. A demonstration of the

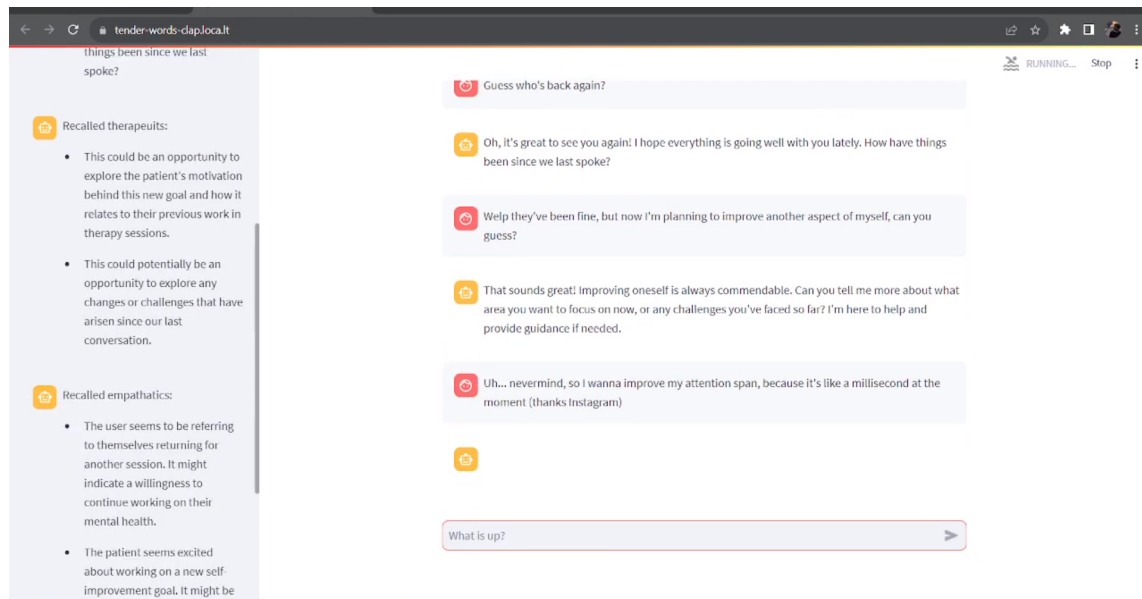


Fig. 6. Demonstration of the memory system integrated into the UI, the memories become more high-level and concise as the interaction with the chatbot continues due to the Reflection system from Generative Agents architecture

memory retrieval system can be observed in figure 6, in which the UI displays retrieved memories from corresponding stream with the latest user message as the query. Moreover, support for saving and loading the memory retriever was added using `bzip` and `pickle` libraries.

3.8 9th August - 16th August - Implementation v3

3.8.1 Overview. The next phase involved the addition of the memory importance and reflection modules for the completion of the Generative Agents framework, adding a queue to accumulate the inputs and gathering survey reports evaluation from various users.

As discussed earlier in section 2.7.1, the Memory module of the Generative Agents architecture consists of three factors, i.e, relevance, recency and importance, where relevance is determined by the semantic similarity model that we discussed in the last subsection, recency is determined programatically with the data structure for memories in Langchain retrievers, finally, importance is determined by the LLMs with formulated prompt as displayed earlier in section 3.6.3.

The components required for the prompt to run in the correct order of execution in the flow, i.e, after a memory has been added to the vector database upon encoding by the semantic similarity model was added to be executed asynchronously, such that it maintains consistency with rest of the ecosystem with Streamlit's reactive nature. However, later it was determined that the memory importance assignment prompts running serially for all the 3 memory streams generated unfeasible overheads, and hence, this module was later discarded to preserve the responsiveness of the application.

Further, the Reflection module was added via the augmenting the code with components to execute the prompts for reflection, as stated earlier in section 3.6.3, which further required investigation into the Langchain retriever modules to

fetch the last k memories with respect to time and further nesting of asynchronous functions via the `reflection_task` function, consisting of `run_reflection_pt1` and `run_reflection_pt2` functions, this further posed bugs within the code due to the reactive nature of Streamlit, which were further fixed after browsing through the Github issues on the official Streamlit repository. The solution consisted of adding a locking mechanism to the newly added asynchronous functions for the Reflection modules via usage of the `asyncio` library.

Due to the reactive nature of Streamlit, subsequent outputs always reset the execution flow to the beginning of the event handler, this means that if the user continuously inputs to the UI within a span of a short duration, the bot would keep resetting to generate the starting sentence, without finishing and accumulating its recently outputted dialogue. Therefore, an attempt was made to add the queueing of inputs such that the last user message gets processed by the Generative Agents flow completely before moving on to process the next input. The attempt was successful for a variant of the UI without the sidebar using the `asyncio.Queue` class, however, due to internal bugs in the library, the same was unsuccessful with the reintroduction of the sidebar, which was determined to be crucial for the UI design, as it provides separation of the internal thoughts and the output by Joi in separate visual components, therefore, the queueing mechanism was removed in the final version of the project.

The following is a gist of the code to add the queue functionality:

```
import asyncio

.
.
.
# Queue for storing user input
if "user_input_queue" not in st.session_state:
    st.session_state.user_input_queue = asyncio.Queue()

.
.
.
async def process_user_input(user_input):
    * The previous logic for handling user input, i.e, passing through Generative Agents flow *
    .
    .
# Creating a lock for the function below
input_processing_lock = asyncio.Lock()
async def handle_user_input():
    while True:
        #Getting current user input
        user_input = await st.session_state.user_input_queue.get()

        async with input_processing_lock:
            # Processing user input and perform necessary operations, necessary because
            # of Streamlit's reactive nature
            await process_user_input(user_input)
```

```

        #Marking the task as done
        st.session_state.user_input_queue.task_done()

    async def main():
        .
        .
        .
        user_input = st.text_input("What is up?")
        if user_input:
            # Queue the user input for processing
            await st.session_state.user_input_queue.put(user_input)

            # Sleep to avoid excessive UI updates
            await asyncio.sleep(0.1)

if __name__ == '__main__':
    asyncio.run(main())

```

3.9 16th August - 25th August: Remote deployment

As briefly explored in sections 3.5.2 and 3.5.4, the next step was to make the LLMs model containerised while using the text-generation-webui library as an intermediate layer. In order to achieve this task, further research into the functionality of text-generation-webui was conducted, and it was concluded that the interface is exposed via a few HTTP ports for streaming responses via API calls, therefore, the feasibility of this task was determined to be high.

3.9.1 Containerisation with Docker. The first step was to prepare a Docker container that included all the required dependencies, such as packages that are required by the text-generation-webui interface and the LLM checkpoints. Finally, a publicly available repository developed by a community member was chosen as a base[32], codebase studied thoroughly and adapted for usage with the custom model, the adapted codebase is [available here](#) and also referred in the UoB Github project README, which is provided in the Appendix section. It is also possible to replace the deployed LLM and quantisation technique by inferring required changes from the additional commits in forked repository on top of the base repository.

The following commands need to be run locally or on a virtual cloud with Ubuntu/Linux distribution to build and push the container to the Docker Hub:

Listing 1. Building Docker Container

```

1 git clone https://github.com/tripathiarpan20/text-generation-docker.git
2 cd text-generation-docker
3 wget https://raw.githubusercontent.com/tripathiarpan20/self-improvement-4all/main/Dockerfile.standalone
4 rm Dockerfile

```

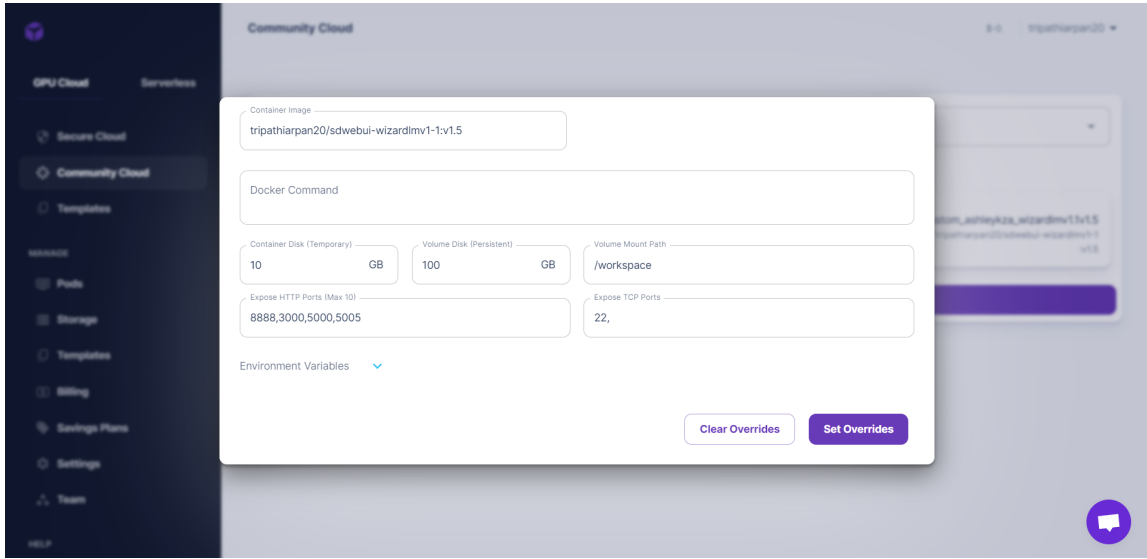


Fig. 7. Template creation with Docker Container on Runpod

```

5 mv Dockerfile.standalone Dockerfile
6 docker build -t <username>/self-improv-4-all:v1
7 docker push <username>/self-improv-4-all:v1

```

Once the above commands have been executed, the Docker container would be ready to deploy on Runpod.

3.9.2 Remote Deployment on Runpod. Once the Docker container for the LLM is ready, we can deploy the LLM for inference on Runpod, where it will be open to communication via exposed ports.

Follow the given steps:

- (1) Go to runpod.io and sign up and add credits, on the dashboard go to the 'Templates' option under the 'Manage' tab, and choose the 'New Template' option. Fill the required fields with the container name by replacing <username> as required, the same is illustrated in figure 7.
- (2) Next, go to the 'Community Cloud' choose a GPU tier to deploy the container template on a suitable GPU as a pod, the pod with the chosen model (WizardLM v1.1 13B) has been tested to work sufficiently on the RTX A4000 16GB VRAM instance. An illustration of this step is given in figure 8.
- (3) Once the pod has finished deploying, accessible via the 'Pods' section under the 'Manage' tab, select 'Start Web Terminal', followed by 'Connect to web terminal', which opens a new window with a shell instance for further interaction. After waiting for a few minutes, the model would automatically load, this can be highlighted via the availability of all the displayed HTTP ports and increased GPU usage in the UI. The related illustration is in figure 9.

3.9.3 Running UI locally. Then on the Linux client system, setup the repository with the following commands:

Listing 2. Cloning repository

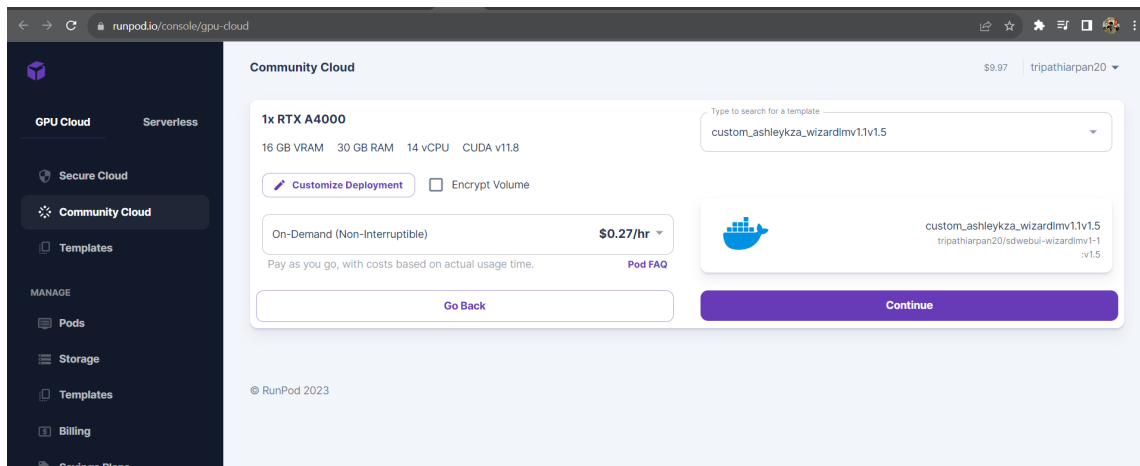


Fig. 8. GPU selection during Pod deployment on Runpod

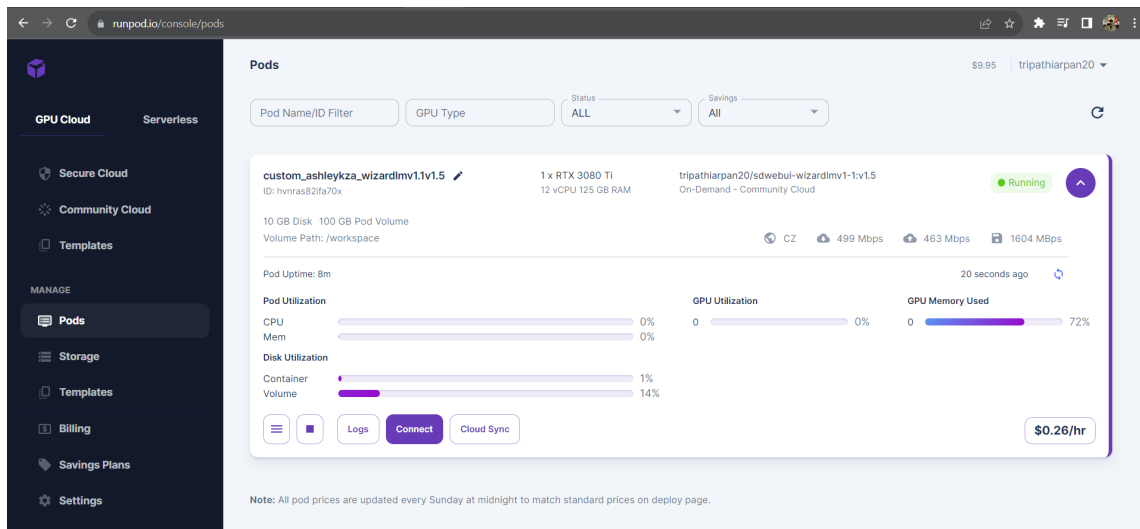


Fig. 9. Deployed Pod for LLM inference on Runpod with RTX 3080Ti GPU

```
1 git clone https://github.com/tripathiarpan20/self-improvement-4all
2 cd self-improvement-4all
```

In the `streamchat_runpod.py` file, replace the ID in the URI variable, for example:

URI = 'wss://ay7akx8j6pesjm-5005.proxy.runpod.net/api/v1/chat-stream, here we would need to replace ay7akx8j6pesjm with the Runpod pod ID, as highlighted in the URL in the search bar in figure 10.

Then setup the virtual environment for the application:

Listing 3. Setting up virtual environment

```

root@2a774e11f029:/# ls
NGC-DL-CONTAINER-LICENSE  cuda-keyring_1.0-1_all.deb  fix_venv.sh  lib32  media  pre_start.sh  run  start.sh  venv  workspace
bin                        dev                        home         lib64  mnt     proc          sbin  sys      usr
boot                      etc                        lib          libx32  opt     root          srv   text-generation-webui  var

root@2a774e11f029:/# ls workspace/
logs  text-gen-model  text-generation-webui  venv

root@2a774e11f029:/# ls text-generation-webui/
LICENSE  characters  docker  download_model.py  instruction-templates  models  prompts  settings-template.yaml
README.md  convert-to-safetensors.py  docs  extensions  js  modules  requirements.txt  start_textgen_server.sh
api-examples  css  download-model.py  fetch_model.py  loras  presets  server.py  training

root@2a774e11f029:/# ls text-generation-webui/models/
config.yaml  place-your-models-here.txt

root@2a774e11f029:/# ls workspace/
logs  text-gen-model  text-generation-webui  venv

root@2a774e11f029:/# ls workspace/text-generation-webui/models/
TheBloke_WizardLM-13B-V1.1-GPTQ  config.yaml  place-your-models-here.txt

root@2a774e11f029:/#

```

Fig. 10. Deployed pod is accessible via a Web terminal, that displays the required Pod ID in the URL

```

1 sudo apt install python3.10-venv git-lfs
2 python3 -m venv selfimprov4all
3 source selfimprov4all/bin/activate
4 pip install -r requirements.txt
5 mkdir -p vector_mems
6 mkdir -p logs
7 git-lfs clone https://huggingface.co/intfloat/e5-base-v2 models/e5-base-v2

```

Finally, execute the script:

Listing 4. Building Docker Container

```

1 streamlit run streamlit_runpod.py

```

Then open the outputted 'Internal URL' link to access the UI, which communicates with the deployed pod on Runpod for LLM inference. The UI as in figure 3 would be displayed and the application is ready to use.

3.10 26th August - 31st August: Research on current challenges and future scope

During the implementation of the project, several drawbacks related to the current technology were realised, which were further explored upon the completion of last stage of the project. These are listed below:

- (1) **Hallucination:** Although LLMs can be augmented with external data in their prompts, a form of Retrieval Augmented Generation (RAG), they are still not guaranteed to output based on the given constraints and are prone to hallucinating incorrect information, that could affect the believability of the simulated human proxies with the Generative Agents framework.
- (2) **Context size limitations:** As the conversation with the simulated self-improvement coach goes on, the conversation would get populated with tokens and would exceed the context limit of the LLM eventually, there are several techniques like RoPE scaling[6] and YaRN scaling[55] that allows for the efficient extension in the context

limit of the LLMs with a small finetuning dataset (relative to the pretraining dataset). However, for mitigating the same in the scope of this project, the retrieved memories from latest user message and past few messages in the conversation stream were assumed to be sufficient context for the simulated self-improvement coach bot to continue the conversation based on the Generative Agents framework.

- (3) **Vector retrieval:** The base vector embedding models might be suboptimal for the purpose of retrieval of the relevant context for the simulated self-improvement coach bot with specialised domains such as psychology, mental health, etc. This challenge could be mitigated with the finetuning of the vector embedding network on specialised dataset, using techniques like synthetic data generated from highly capable LLMs etc using prompt engineering, as in Evol-instruct[49].
- (4) **Adherence to instruction format:** The adherence of the output of the LLMs to the required format plays a crucial role in the seamless functionality of the chatbot, there are techniques now that allows providing restriction to LLMs to generated more structured outputs such as JSONs[19], the two main libraries that allow the same are Microsoft Guidance[26] and Normal Outlines[27] using concepts from Theory of Computations, for example, Regular Expressions and Context-free-grammars.

4 EVALUATION

As discussed earlier in section 2.7, the absolute evaluation of the architecture was not possible in the original work due to its nature, therefore, to evaluate the agents as individuals, the original work conducted a qualitative evaluation based on user reviews, where the users had access to the agent’s memory stream at the end of a two in-game day simulation, which accumulated the agent’s memory with observations. The users also had access to an interview system using which they could interrogate the agent and analyse the coherence of its responses by reflecting upon the memory history of the agent.

However, our case varies from the original implementation, as it is designed to simulate a single agent. The evaluation of the approach was considered to a reasonable extent during the beginning of the project, which led to the UI design decision of the sidebar, i.e, displaying the internal reasoning of the simulated self-improvement coach chatbot to the users at all time. The above facilitated the acquisition of user reviews. Also, note that the original Generative Agents work utilised ChatGPT-3.5 as a backbone and conducted the evaluation with the relatively stronger LLM backbone, as compared to our case with WizardLM v1.1, which possesses lesser instruction-following capabilities and psychological domain knowledge compared to ChatGPT-3.5.

For the qualitative evaluation of the project, a Google Form was prepared which provided two options to the users to experience the product:

- (1) The first option was to replay past 3 sessions of the self-improvement coach, available as recordings on YouTube, and expected to take roughly 5 minutes to fill the survey.
- (2) The second option consists of a link to the Colab Notebook with the instructions to setup the project with the cloud GPU and engage in a personalised session, and expected to take around 30 minutes per user to fill the survey.

The reason behind providing the first option is to incentivise the users to fill the survey, as an estimated duration of 30 minutes poses challenges to gain traction for the feedback.

The form consists of the following entries:

- (1) The topic of the session with the self-improvement coach

- (2) A rating of the soundness of the reasoning of the self-improvement coach while responding to the latest message.
- (3) A rating of the ability of the self-improvement coach to address the earlier messages in the response.
- (4) A rating of the ability of the self-improvement coach to recall relevant information from past messages to generate the latest message (as displayed in the sidebar in figure 6).
- (5) (Optional) Text description of the positive aspects of the session.
- (6) (Optional) Text description of the negative aspects of the session.

The evaluation was done via a diverse set of 10 users acquired through various university society social groups via platforms like Whatsapp etc, the anonymisation of the responses was ensured to incentivise unbiased responses. The subfigures in figure 11 illustrate the results for the aforementioned evaluation criteria.

It can be observed that most of the user reviews are satisfactory, relatively lagging behind on the domain of relevance of retrieved memories, which can be addressed in a future work as highlighted earlier in the point 3 of the challenges listed in section 3.10. The rest of the reviews would also be improved with the usage of a better LLM base in the future.

Further, it was also noted in the recordings provided in the form that the reflections were storing high-level information as the session continued, and simulated the tree-of-reflections as observed in the original Generative Agents work.

5 CONCLUSION

Given all the information presented in the Background section, it is an arguable conclusion that the rate of improvement in the accessibility and democratisation of LLMs has been phenomenal and continues to accelerate, both in terms of customisability with efficient finetuning using techniques like QLoRA and inference of high parameter-count LLMs on consumer-grade hardware, along with the trend towards higher performance in more compact LLMs. Therefore, it is expected that the scope of this project would only expand with the future developments.

We are heading towards a world heavily dependent on technology and there is more connectivity in terms of sources of communication than ever before in society, however, there is a growing sense of self-alienation among individuals in society as we progress with technology. The future pivots upon the responsible usage of the emerging technology, especially AI, and this project aims to demonstrate the potential of positive applications of LLMs, supported by the democratisation of AI driven by the open-source research community.

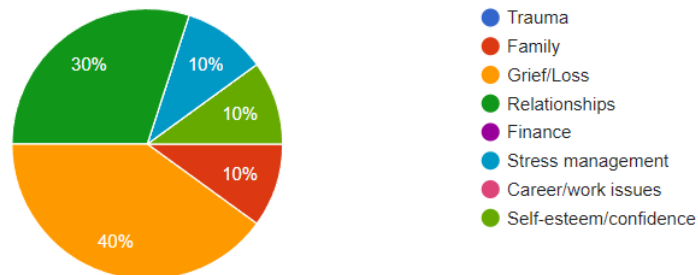
REFERENCES

- [1] [n.d.]. AutoGPTQ. <https://github.com/PanQiWei/AutoGPTQ>. Accessed: 2023-09-10.
- [2] [n.d.]. Batched LoRAs. <https://github.com/sabetAI/BLORA>. Accessed: 2023-09-09.
- [3] [n.d.]. BigScience Petals. <https://github.com/bigscience-workshop/petals>. Accessed: 2023-09-11.
- [4] [n.d.]. ctransformers. <https://github.com/marella/ctransformers>. Accessed: 2023-09-10.
- [5] [n.d.]. ExLLaMa. <https://github.com/turboderp/exllama>. Accessed: 2023-09-10.
- [6] [n.d.]. Extending Context is Hard...but not Impossible. <https://kaiokendev.github.io/context>. Accessed: 2023-09-12.
- [7] [n.d.]. FAISS. <https://github.com/facebookresearch/faiss>. Accessed: 2023-09-10.
- [8] [n.d.]. FedML. <https://github.com/FedML-AI/FedML>. Accessed: 2023-09-11.
- [9] [n.d.]. A Gentle Introduction to 8-bit Matrix Multiplication for transformers at scale using Hugging Face Transformers, Accelerate and bitsandbytes. <https://huggingface.co/blog/hf-bitsandbytes-integration>. Accessed: 2023-09-09.
- [10] [n.d.]. GGML. <https://github.com/ggerganov/ggml>. Accessed: 2023-09-15.
- [11] [n.d.]. Gradio. <https://github.com/gradio-app/gradio>. Accessed: 2023-09-11.
- [12] [n.d.]. Huggingchat UI. <https://github.com/huggingface/chat-ui>. Accessed: 2023-09-10.
- [13] [n.d.]. Huggingface model: ehartford/samantha-1.1-llama-13b. <https://huggingface.co/ehartford/samantha-1.1-llama-13b>. Accessed: 2023-09-09.
- [14] [n.d.]. Huggingface model: TheBloke/chronos-hermes-13B-GPTQ. <https://huggingface.co/TheBloke/chronos-hermes-13B-GPTQ>. Accessed: 2023-09-09.

How would you categorize the topic of discussion with Joi?

 Copy

10 responses

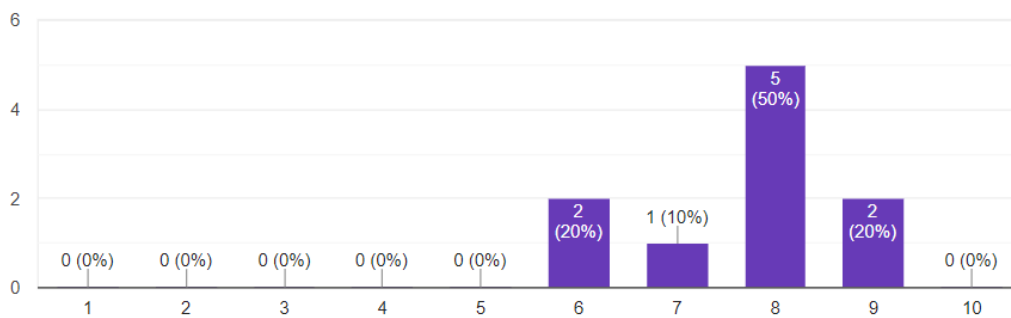


((a)) Question 1

How would you rate the soundness of Joi's reasoning while responding to your latest message?

 Copy

10 responses

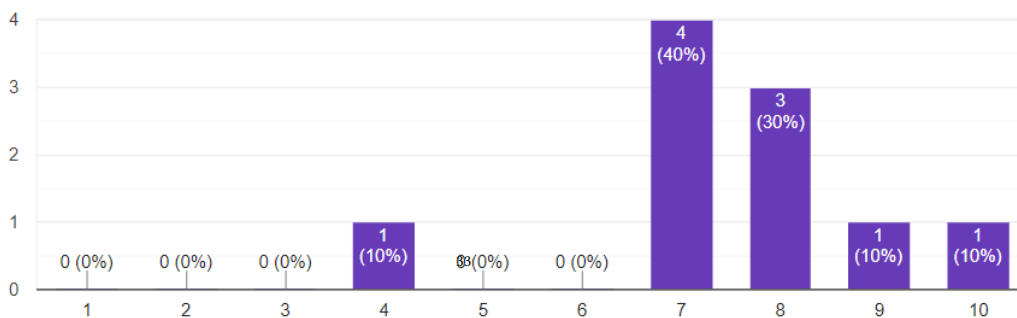


((b)) Question 2

How would you rate Joi's ability to address your earlier messages in the response?

 Copy

10 responses



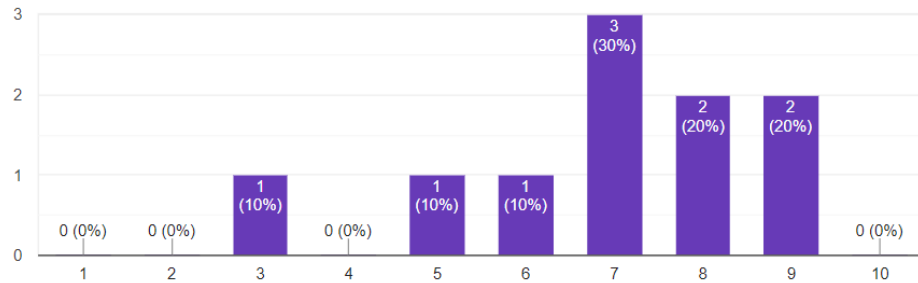
((c)) Question 3

Fig. 11. Evaluation form results: 1

How would rate Joi's ability to recall salient past messages/insights based on the context of your latest message?

 Copy

10 responses



((a)) Question 4

What did you like about Joi, and why?

7 responses

- It's like a virtual friend to me, I can talk to it without thinking I'll be judged or laughed at.
- the advice it gave is very reasonable and actionable
- I like how Joi provide the way to deal with issues precisely because it gives me a concrete idea of what to do.
- Joi provides a really transparent experience with her responses, and they make sense seeing the reasoning behind them
- I like the inner empathic voice and reflection mechanism
- The fact that the bot can reason before answering appropriate piques my interest
- It provides access to a friend for free

((b)) Question 5

What did you dislike about Joi, and why?

7 responses

- It sounds comforting to some extent though, there is something different compared to talk with a real person. The expression tone of Joi sometimes looks like mechanically and lack of real feeling connection.
- the words it uses still sound like a robot, not like a friend
- I would like Joi to be a good listener rather than giving advice. In the case of break up, what people need the most is usually a good listener, I believe.
- It still feels robotic, talking to a screen doesn't feel that good
- Sometimes it is unable to get to the point and answers questions in a similar manner.
- It's still imperfect with memory retrieval
- It is still just talking to a screen

((c)) Question 6

- [15] [n.d.]. Huggingface model: TheBloke/chronos-hermes-13B-GPTQ. <https://huggingface.co/openbmb/UltraLM-13b>. Accessed: 2023-09-09.
- [16] [n.d.]. Huggingface model: WizardLM/WizardLM-13B-V1.1. <https://huggingface.co/WizardLM/WizardLM-13B-V1.1>. Accessed: 2023-09-09.
- [17] [n.d.]. Huggingface Transformers. <https://github.com/huggingface/transformers>. Accessed: 2023-09-10.
- [18] [n.d.]. Introducing LLaMA: A foundational, 65-billion-parameter large language model. <https://ai.meta.com/blog/large-language-model-llama-meta-ai/>. Accessed: 2023-09-15.
- [19] [n.d.]. Jsonformer. <https://github.com/1rgs/jsonformer>. Accessed: 2023-09-12.
- [20] [n.d.]. Langchain.js. <https://github.com/langchain-ai/langchainjs>. Accessed: 2023-09-10.
- [21] [n.d.]. Language Models Perform Reasoning via Chain of Thought. <https://blog.research.google/2022/05/language-models-perform-reasoning-via.html>. Accessed: 2023-09-15.
- [22] [n.d.]. llama.cpp. <https://github.com/ggerranov/llama.cpp>. Accessed: 2023-09-10.
- [23] [n.d.]. Localtunnel. <https://github.com/localtunnel/localtunnel>. Accessed: 2023-09-11.
- [24] [n.d.]. Making LLMs even more accessible with bitsandbytes, 4-bit quantization and QLoRA. <https://huggingface.co/blog/4bit-transformers-bitsandbytes>. Accessed: 2023-09-09.
- [25] [n.d.]. Meta and Microsoft Introduce the Next Generation of Llama. <https://about.fb.com/news/2023/07/llama-2/>. Accessed: 2023-09-15.
- [26] [n.d.]. Microsoft Guidance. <https://github.com/guidance-ai/guidance>. Accessed: 2023-09-12.
- [27] [n.d.]. Normal Outlines. <https://github.com/outlines-dev/outlines>. Accessed: 2023-09-12.
- [28] [n.d.]. Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance. <https://blog.research.google/2022/04/pathways-language-model-palm-scaling-to.html>. Accessed: 2023-09-12.
- [29] [n.d.]. Sentence Transformers. <https://github.com/UKPLab/sentence-transformers>. Accessed: 2023-09-10.
- [30] [n.d.]. Stanford Tatsu lab: Alpaca-eval LLM benchmark. https://tatsu-lab.github.io/alpaca_eval/. Accessed: 2023-09-09.
- [31] [n.d.]. Streamlit. <https://github.com/streamlit/streamlit>. Accessed: 2023-09-11.
- [32] [n.d.]. text-generation-docker. <https://github.com/ashleykleynhans/text-generation-docker>. Accessed: 2023-09-15.
- [33] [n.d.]. text-generation-webui. <https://github.com/oobabooga/text-generation-webui>. Accessed: 2023-09-10.
- [34] [n.d.]. TheBloke: Huggingface. <https://huggingface.co/TheBloke>. Accessed: 2023-09-12.
- [35] [n.d.]. Transformer: A Novel Neural Network Architecture for Language Understanding. <https://blog.research.google/2017/08/transformer-novel-neural-network.html?m=1l>. Accessed: 2023-09-15.
- [36] [n.d.]. Transformers.js. <https://github.com/xenova/transformers.js/>. Accessed: 2023-09-10.
- [37] [n.d.]. wise-ft. <https://github.com/mlfoundations/wise-ft>. Accessed: 2023-09-15.
- [38] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [39] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339* (2022).
- [40] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314* (2023).
- [41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [42] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).
- [43] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2016), 2222–2232.
- [44] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. 2022. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951* (2022).
- [45] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).
- [46] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
- [47] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).
- [48] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael Mahoney, and Kurt Keutzer. 2023. SqueezeLLM: Dense-and-Sparse Quantization. *arXiv* (2023).
- [49] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. WizardCoder: Empowering Code Large Language Models with Evol-Instruct. *arXiv preprint arXiv:2306.08568* (2023).
- [50] Tomas Mikolov, Martin Karafiát, Lukáš Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. Makuhari, 1045–1048.
- [51] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive Text Embedding Benchmark. *arXiv preprint arXiv:2210.07316* (2022). <https://doi.org/10.48550/ARXIV.2210.07316>

- [52] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707* (2023).
- [53] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.
- [54] Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442* (2023).
- [55] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. YaRN: Efficient Context Window Extension of Large Language Models. *arXiv preprint arXiv:2309.00071* (2023).
- [56] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
- [57] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).
- [58] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10684–10695.
- [59] Simo Ryu. [n.d.]. Low-rank Adaptation for Fast Text-to-Image Diffusion Fine-tuning.

6 APPENDIX

The code is available in the private repository: <https://git.cs.bham.ac.uk/projects-2022-23/axt233>, along with instructions to run the same in the 'README.md' file. Note that it also includes links to Google Colab [notebook](#) to setup the project in a cloud environment.

The link to the [evaluation form](#) is also provided.