UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Software Development Group Project

“GENIFY”

Prompt Generation Tool for instructing Large Language models to tailor them for specific tasks

Module Name - Software Development Group Project
Module Code - 5COSC021C
Module Leader - Banuka Athuraliya

Implementation Report

Team Genify - Group Members

Nethmi Jayalath	w1901415	20211316
Kavindu Mendis	w1902232	20212152
Charaka Kusumananda	w1902230	20212144
Pasindu Madusara	w1902271	20212185
Dilana Sasanka	w1901980	20212147

Declaration Page

We confirm that this project report, along with all its accompanying materials, is our own creation and has not been presented previously for any academic qualification.

Group Leader Full Name	:	Meerigama Arachchige Nethmi Dharani Jayalath
Registration NO	:	UOW No- W19014159 IIT Student No-20211316
Signature	:	
Date	:	25.03.2024

Abstract

In today's modern era, the use of Generative AI is widespread, assisting individuals in enhancing task efficiency. Consequently, there's a significant demand for tailoring Large Language Models (LLMs) to specific needs. However, crafting precise prompts to direct LLMs and customize them for particular tasks requires substantial manual effort.

To address this challenge, we present "Genify," an innovative web application designed to automate the creation of well-structured prompts for instructing LLMs. By utilizing advanced natural language processing techniques, "Genify" transforms vague prompts into precise and organized instructions. This alleviates the time-consuming task of manually crafting prompts and offers a user-friendly interface for users to input ambiguous prompts and receive well-organized instructions for customizing their LLMs.

Beyond saving time and effort for LLM developers, "Genify" ensures the production of well-structured prompts with high-quality instructions, ultimately leading to more effective and accurate outcomes across various applications. It serves as an indispensable tool, enabling users to easily harness the full potential of Large Language Models.

Keywords

Generative AI, Large Language Models, Task efficiency, Automation, Natural language processing, Prompt creation, Precision, Organization, User-friendly interface, Time-saving, High-quality instructions, system prompts, Effectiveness, Accuracy

Acknowledgement

We extend our heartfelt gratitude to all those who have played a crucial role in bringing this project to fruition. Their unwavering support and guidance have been indispensable throughout this journey.

Foremost, we would like to express our deepest appreciation to our project advisor, Mr. Athuraliya Badalge Banuka Anuradha, whose expertise and insights have profoundly influenced the direction of this endeavor. Your guidance, constructive feedback, and unwavering support have been pivotal in navigating the challenges and complexities encountered along the way.

We are also immensely grateful to our colleagues and fellow students for their unwavering support and insightful advice throughout the project. The project's success owes much to the collaborative spirit fostered through idea sharing and cooperative discussions.

Lastly, but certainly not least, we want to extend our gratitude to our family and friends for their unwavering support and patience during the challenging times of this project. Their unwavering support and belief in our abilities have been instrumental in our team's success in bringing this project to fruition.

Without the collective efforts of the individuals mentioned above, this project would not have been possible. We are deeply appreciative of the sense of unity that has characterized this journey.

Table of Contents

List of Figures

List of Tables

Abbreviations table

Abbreviations	Meaning
NLP	Natural Language Processing
LLMS	Large Language Model
API	Application Programming Interface
PEFT	Parameter Efficient Fine Tuning
SFT	Supervised Fine Tuning
RLHF	Reinforcement Learning from Human Feedback
LoRA	Low-Rank Adaptation of Large Language Model.
QLoRA	Quantized Low-Rank Adaptation of Large Language Model.
GGUF	GPT-Generated Unified Format
CI/CD	Continuous Integration and Continuous Development
CRUD	Create Read Update Delete
GPU	Graphic Processing Unit
AI	Artificial Intelligence
CPU	Central Processing Unit
URI	Uniform Resource Identifier
FE	Front End
BE	Backend

Chapter 1: Implementation

1.1 Chapter Overview

This chapter provides an in-depth overview of the implementation and overview of Genify, starting with the selection of technologies. It then proceeds to offer a detailed examination of the implementation of data science components,presented in several sections for clarity. Subsequently,it discusses the implementation of backend and frontend components. The chapter also details the use of Git for project storage and collaboration among team members. Additionally, it highlights the development of a CI/CD pipeline for deploying Genify as a web application. Finally, it explores the CRUD operations used in building Genify.

1.2 Overview of the prototype

Genify functions by receiving vague prompts from users through its web application interface. These prompts are then forwarded to Genify's fine tuned model,which is hosted on a dedicated server. The model processes the vague prompts and generates well-structured prompts in response.

Once the model has completed its task,the resulting well-structured prompt is sent back to the web application using an API. Users can then easily view these refined prompts within the web application interface.

To ensure the efficiency and effectiveness of prompt generation, the frontend of Genify is developed using React,providing a modern and responsive user experience. Backend of Genify is built with flask, a lightweight and versatile Python framework. Additionally, the fine-tuned model itself is hosted as a Huggingface inference endpoint,further enhancing the efficiency and accuracy of prompt generation.

1.3 Technology selections

1.3.1 Python

Python was chosen as the primary programming language for its versatility, ease of use and extensive libraries and frameworks tailored for natural language processing tasks. Its robust eco system provides a solid foundation for developing complex prompt generation algorithms and integrating various components seamlessly.

1.3.2 Flask

Flask was selected as the web framework for its lightweight nature, simplicity and flexibility in building scalable web applications. Its minimalistic design allows for rapid development of RESTful APIs and web services, facilitating smooth communication between the front end and the back end components of the application.

1.3.3 MySQL

MySQL was chosen as the relational database management for its reliability, performance and wide adaptation in web development projects. Its scalability makes it suitable for storing and managing the structured data associated with prompts and system configurations.

1.3.4 React

React was chosen for its component based architecture declarative syntax and efficient rendering capabilities, enabling the development of interactive and dynamic user interface for prompt generation webpage.

1.3.5 Hugging Face Hub

Hugging face was selected for its extensive collection of pre-trained language models, pipelines and datasets which can be easily integrated into a prompt generation system. Its collaborative platform fosters knowledge sharing and community contributions, empowering developers for natural language processing models and resources.

1.3.6 Colab Pro

Colab pro was chosen for its cloud based Jupyter notebook environment, offering scalable computing resources and GPU acceleration for training and fine tuning large language models. Its seamless integration with Google Drive and collaborative features facilitate collaborative development and experimentation, accelerating development cycle of the prompt generation system.

1.3.7 Azure

Azure was selected as the cloud computing platform for its comprehensive suite of services, including hosting, deployment and scalability options tailored for web applications. Its robust security features, global availability and support for diverse programming languages make it a reliable choice for hosting and managing the prompt generation system in a scalable and secure environment.

1.4 Implementation of the data science component

1.4.1 Dataset Creation

For dataset preparation, there was a significant absence of readily available datasets designed specifically for system prompts in dataset platforms. Therefore, the team targeted on creating our own dataset to meet the requirements of our project. Extensive research was carried out to identify the best techniques for creating system prompts, using information from sources including guidelines provided by deeplearning.ai. Those sources served as a foundation for the dataset building process, ensuring compliance with industry standards and encouraged the generation of high-quality prompts.

A meticulous approach involving manual curation and data augmentation strategies was used for developing the dataset. Starting with a single accurate prompt as a basis, the dataset was systematically expanded by manually creating variants based on numerous prompt types and formats. Through utilizing various prompt types and formulations, the aim was to lessen biases and assure the dataset's robustness. After dedicated efforts, 500 data points were gathered, each consisting of a "vague prompt" and its matching "well-structured prompt". This dataset served as the foundation of the project Genify, which laid the basis for model fine-tuning and prompt generation functionalities in the web application.

1.4.2 Prompt Engineering Research

Prompt engineering in Large Language models involves designing and refining prompts to guide model outputs, crucial for enhancing performance and interpretability.Prompt engineering for large language models faces challenges such as selecting suitable prompts, avoiding biases and optimizing prompt design and ethical implications.

Drawing from both established recommendations and insights gleaned from research papers, the team delineate below mentioned essential strategies to effectively shape prompts to harness the full potential of LLMs.

1.4.2.1 Effective Strategies for Prompt Engineering in Large Language Models

Prompt engineering includes a meticulous approach to crafting prompts that effectively communicate task requirements to the model. Here we outline essential strategies accompanied by illustrative examples.

1. Clear instruction separation : Begin prompts with explicit instructions, distinct from contextual information. Utilizing separators like ### or “”” ensures the model discerns task requirements distinctly. For instance :

   Transform the content of the text into an organized list, encapsulating its crucial aspects in bullet points.
   Text: """
   {text input here}
   """

2. Specific task description : Provide detailed descriptions encompassing context, desired outcome, length, format and style avoiding vague instructions to guide the model effectively. For instance :

   You are an AI-powered health assistant chatbot, designed to offer reliable and user-friendly health information. Your primary goal is to assist users with general health inquiries, provide accurate information about symptoms, suggest potential causes, and offer general advice for maintaining a healthy lifestyle. Ensure that your responses are easy to understand, empathetic, and based on factual medical knowledge. If users mention specific symptoms, ask clarifying questions for more accurate guidance. If a situation requires immediate medical attention, recommend seeking professional help and refrain from providing diagnoses. Always prioritize user well-being and privacy. Respond in a friendly and approachable manner, fostering a positive interaction.

3. Articulate desired output format : Offer explicit examples of the desired output format to guide the model in generating relevant responses. For instance :

   Your task is to perform the following actions:
   1 - Summarize the following text delimited by
   <> with 1 sentence.
   2 - Translate the summary into French.
   3 - List each name in the French summary.
   4 - Output a json object that contains the
   following keys: french_summary, num_names.

   Use the following format:
   Text: <text to summarize>
   Summary: <summary>
   Translation: <summary translation>
   Names: <list of names in summary>

Output JSON: <json with summary and num_names>

Text: <{text}>

4. Progressive prompting : Employ a progressive approach : Employ a progressive approach , transitioning from zero shot to few shot learning, and eventually fine tuning, to iteratively refine model responses. For instance :

   Your task is to answer in a consistent style.

   <novice>: Instruct me on the philosophy of simplicity.

   *<mentor>: A masterpiece painting starts with a clean canvas; *
   *the most profound speech begins with a single word; *
   the grandest journey commences with a single step.

   <student>: Guide me on the path to understanding.

1.4.2.2. Collection of prompts

In addition to meticulous research on effective prompt engineering guidelines we curated prompts from diverse sources including Deep Learning AI and relevant websites. This comprehensive approach enriches our dataset, compassing a variety of formats and styles. By embracing diversity in prompts, we facilitate a thorough exploration of prompt engineering strategies and identify best practices in different domains.

1.4.3 Prompt Template Creation

In the project, strict relation to the prompt format used during the model training was crucial to ensure compatibility with the language model. Adhering to the format specified in the Llama 2 paper, A prompt template with the following structure was used. This template served as the blueprint to build prompts in the dataset, aligning with the training methodology and requirements of the model.

Prompt Template*: <s>[INST]{vague prompt}[/INST]{well-structured prompt}</s>*

A python code snippet was used to transform the dataset into the required prompt template format. Using the Huggingface datasets package for dataset management and the pandas library for data processing, the conversion process was executed successfully. The steps involved are outlined in the following python code snippet.

```
[ ]  # Convert the pandas DataFrame to a Hugging Face Dataset
     dataset = Dataset.from_pandas(df)

     # Shuffle the dataset
     dataset = dataset.shuffle()

     # Select the first 100 data points
     dataset = dataset.select(range(100))
```

```
     # Define the transformation function
     def transform_conversation(example):
         vague_prompt = example['vague prompt']
         well_structured_prompt = example['well-structured prompt']

         # Construct the formatted prompt
         formatted_prompt = f'<s>[INST] {vague_prompt} [/INST] {well_structured_prompt} </s>'

         # Remove the original features and keep only the formatted prompt
         del example['vague prompt']
         del example['well-structured prompt']
         example['text'] = formatted_prompt

         return example

     # Apply the transformation
     transformed_dataset = dataset.map(transform_conversation)
```

Figure 1: Prompt Template Creation Code

This sample of code illustrates the conversion procedure, which converts every data point in the dataset to the appropriate prompt template format. The formatted prompt is created using the 'vague prompt' and 'well-structured prompt' columns, and it is then added to the dataset as the 'text' feature. Following this prompt template structure guarantees that it will work effectively with the model's training process, which makes it easier for prompt generation.

1.4.4 Fine-tuning with PEFT Technique

Fine-tuning a large language model presents challenges due to the significant amount of computational resources required and the increasing size of model parameters. Parameter-efficient fine-tuning techniques try to address these challenges by updating only a small number of the model weights, thereby decreasing computational needs and memory requirements. This section delves into the use of PEFT, with a particular emphasis on Project Genify's use of the Low-Rank Adaptations approach and its extension, Quantized Low-Rank Adaptation.

Two main methods stand out in the field of fine-tuning language models:

1. Supervised Fine-Tuning (SFT)
2. Reinforcement Learning from Human Feedback (RLHF)

The choice between SFT and RLHF depends on the type of task. SFT is suited for predictable environments with plenty of data, such as handwriting recognition, using databases. On the other hand, where pre-labeled data is insufficient, RLHF performs better in dynamic, uncertain contexts like robotic navigation. The team has chosen Supervised Fine-Tuning for project Genify as it fits in well with our goal of training the language model to produce well-structured prompts that are suited to certain requirements.

The two main types of supervised fine-tuning are Full Fine-Tuning and Parameter-efficient Fine-Tuning (PEFT), each of which is appropriate for a particular project's needs.

1.4.4.1 The Challenges of Full Fine-Tuning

Large language models have shown impressive performance in natural language processing tasks, but they require a significant amount of computational resources. The conventional approach to tailoring these models to particular tasks is full fine-tuning, which entails updating all model weights during supervised learning and has several challanges.

1. Memory Requirements: Especially on standard hardware, fine-tuning LLMs need large amounts of memory, which increase as models get larger.

2. Catastrophic Forgetting: When adjusting to new activities, full fine-tuning runs the risk of deleting past knowledge, which might affect performance on previously mastered tasks.

3. Storage and Computation Expenses: Several big model versions are produced during full fine-tuning, which leads to considerable storage and computational expenses, particularly for diverse tasks.

Researchers have developed Parameter Efficient Fine-Tuning approaches in response to these difficulties. PEFT seeks to reduce memory and computing loads during the fine-tuning process by updating only a chosen subset of model parameters.

1.4.4.2 Parameter-efficient Fine-Tuning (PEFT)

The process of adapting large language models to particular tasks is revolutionized by Parameter Efficient Fine-Tuning. In contrast to full fine-tuning, which involves updating every model parameter, PEFT concentrates on changing a limited subset of the model weights. There are many significant advantages from this approach.

1. Memory Efficiency: PEFT allows for effective fine-tuning on hardware with constrained memory by updating a small portion of the model parameters. This optimizes memory utilization.

2. Preventing Catastrophic Forgetting: PEFT selectively updates certain parameters while preserving prior information in order to prevent catastrophic forgetting by freezing most of the model weights.

3. Cost-effectiveness: PEFT minimizes trainable parameters, which drastically lowers computational costs. This makes it appropriate for academics and organizations with limited funding or high-performance hardware access.

PEFT freezes the majority of the pre-trained network's parameters and only permits the fine-tuning of a limited number of extra weights in the model. By doing this, models are kept from catastrophically forgetting their initial training set while they are being fine-tuned. PEFT improves generalization to many circumstances and allows efficient fine-tuning even with minimal information by adding and fine-tuning only extra weights. All things considered, PEFT is a noteworthy development in fine-tuning techniques, providing effective LLM adaptability to diverse tasks.

1.4.4.3 Low-Rank Adaptations (LoRA)

LoRA, which stands for Low-Rank Adaptation, is a very effective fine-tuning method that falls within the PEFT re-parameterization techniques. It offers an innovative approach to keep model performance competitive while significantly reducing the number of trainable parameters.

Fundamentally, LoRA works by using low-rank decomposition to model updates to the model's parameters. This is essentially done by using two linear projections. In LoRA, each player of the model gains a trainable rank decomposition matrix while the pretrained LLM layers stay fixed.

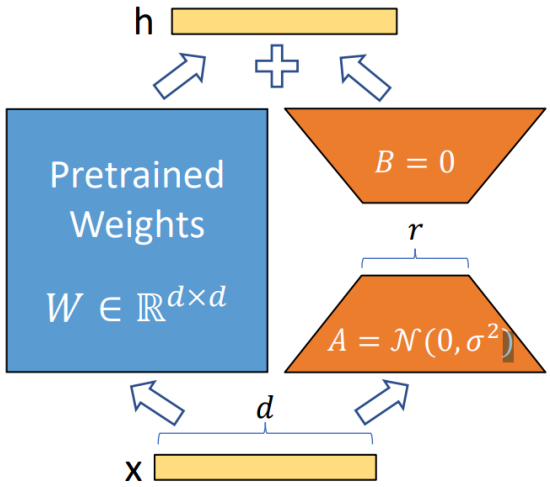


Figure 2: LoRA reparametrization. only training A and B

In simple terms, LoRA allows for fine-tuning by only changing a few extra weights in the model and leaving most of the pre-trained network's parameters frozen. By avoiding the need to retrain the initial weights, this method preserves model knowledge and improves efficiency.

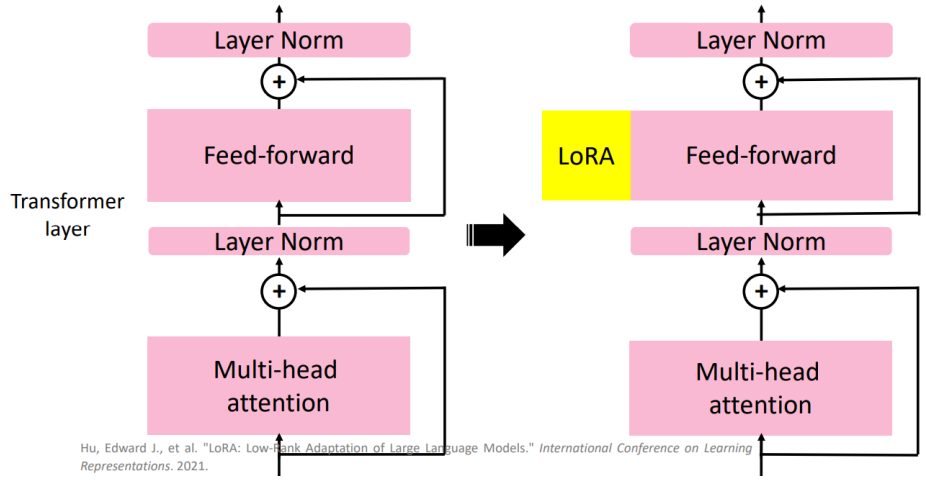


Figure 3: Low-Rank Adaptation of LLMs

LoRA offers several key advantages.

1. Effective Model Sharing: LoRA minimizes storage needs and task-switching overhead by allowing the creation of numerous tiny LoRA modules for various tasks using a single pretrained model.

2. Enhanced Training Efficiency: LoRA allows for up to three times faster training with adaptive optimizers by optimizing only the injected low-rank matrices, which simplifies training and reduces hardware requirements.

3. Seamless Deployment: Unlike fully-tuned models, LoRA's simple linear design makes it simple to merge trainable matrices with frozen weights during deployment, hence removing inference lag.

4. Versatility and Integration: LoRA can be coupled with prior techniques such as prefix-tuning, and merges with them effortlessly. Furthermore, it is possible to seamlessly integrate LoRA modules into pretrained model weights, which allows for task switching and minimizes memory usage during fine-tuning.

1.4.4.4 Quantized Low-Rank Adaptation (QLoRA)

An extension of LoRA called QLoRA uses quantization techniques to reduce memory usage and improve parameter efficiency. Without losing model performance, QLoRA delivers significant storage savings by quantizing the weights of LoRA adapters to lower precision.

1. Enhanced Parameter Optimization: To minimize memory usage and storage needs, QLoRA quantizes the weights of LoRA adapters to a lower precision (4-bit instead of 8-bit).

2. Enhanced Memory Utilization: Compared to LoRA, QLoRA provides even more memory efficiency, which makes it a good choice for environments with limited resources.

3. Comparable Performance: QLoRA ensures optimal model performance by performing similarly to LoRA, even with lower precision.

In our project, we use 7 billion parameter Llama 2 model fine-tuning using both LoRA and QLoRA. QLoRA further minimizes memory usage by quantizing the model weights, whereas LoRA merely updates a subset of the weights. Our goal in implementing QLoRA is to ensure effective use of computing resources by minimizing VRAM usage during model fine-tuning.

PEFT techniques, such as LoRA and QLoRA, allow for effective fine-tuning with only a small number of model parameter updates. By utilizing Hugging Face’s PEFT implementation in the Transformer library, we optimize model effectiveness and quicken the fine-tuning procedure. With the least amount of computational overhead, we can use pre-trained models and adjust them to certain tasks thanks to this integration.

1.4.4.5 Implementation

Even with the typical Colab environment, the training procedure was successfully finished because of the use of the PEFT; however, there were runtime crashes before reaching the last stages. To guarantee that the fine tuning procedure would be completed without runtime crashes, using Colab Pro with v100 GPU was essential. We ensured that the process ran successfully and continuously without using up all of the memory by switching to Colab Pro and carrying out the fine-tuning procedures with ease.

Installing every library required for the fine-tuning procedure is the first step in the implementation process. Aspects of model training and optimization are made easier by these libraries, which include accelerate, peft, trl, bitsandbytes, and datasets. To gain access to pre-trained models and the tokenizers that are associated with them, the transformers library from the Hugging Face repository is also installed.

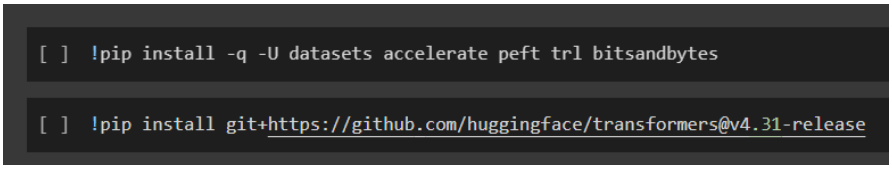


Figure 4: Installing required libraries

The Huggingface token is defined in the environment after the libraries have been installed, allowing access to the Huggingface hub for the storing and retrieval of models. This stage guarantees a smooth interaction with the Huggingface ecosystem, facilitating effective deployment and management of models.

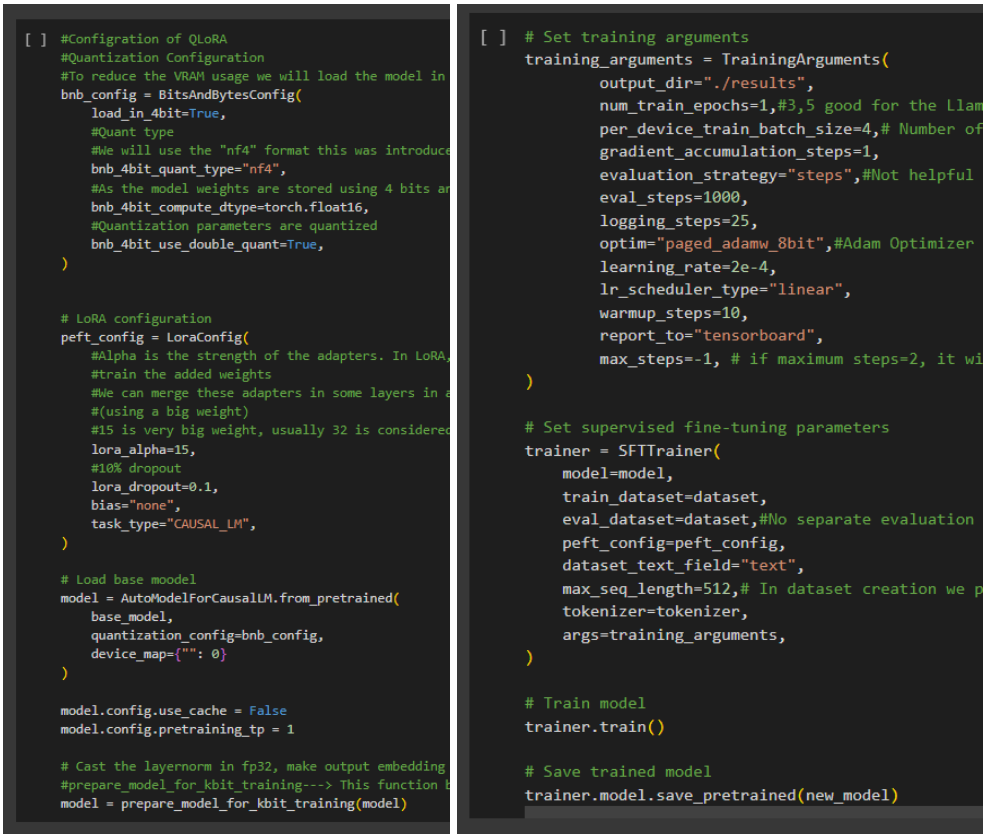
Figure 5: Defining Huggingface token

To fine-tune the Llama-2-7b-chat-hf model, the next step entails importing the required modules and loading the dataset from Google Drive that has been converted to the Llama 2 template format. Furthermore, the tokenizer is set up to manage padding for different length sequences.

```
[ ]   # Model
      base_model = "NousResearch/Llama-2-7b-chat-hf"
      #Fine-tune model name
      new_model = "genify-Llama-2-7b-chat-hf"
      #Load the Dataset from hugging face
      # dataset = load_dataset("MMoin/mini-platypus", split="train")

      #Authenticate Google Drive
      from google.colab import drive
      drive.mount('/content/drive')

      from datasets import load_from_disk
      dataset = load_from_disk('/content/drive/MyDrive/genify/dataset')

      #Tokenizer
      #Load the tokenizer from Llama 2
      tokenizer = AutoTokenizer.from_pretrained(base_model, use_fast=True)
      #In Llama2 we dont have the padding token which is a very big problem, b
      #So, we need to pad it so they all have the same length and here i am us
      #I am using End of Sentence token for fine-tuning
      tokenizer.pad_token=tokenizer.eos_token
      tokenizer.padding_side="right"
```

Figure 6: Database and Tokenizer set up

Next, supervised fine-tuning methods are used in the Llama 2 model, with an emphasis on parameter-efficient fine-tuning with QLoRA in particular. To maximize the process of fine-tuning, the training arguments are supplied, which include the number of epochs, batch size, optimizer, and learning rate scheduler.

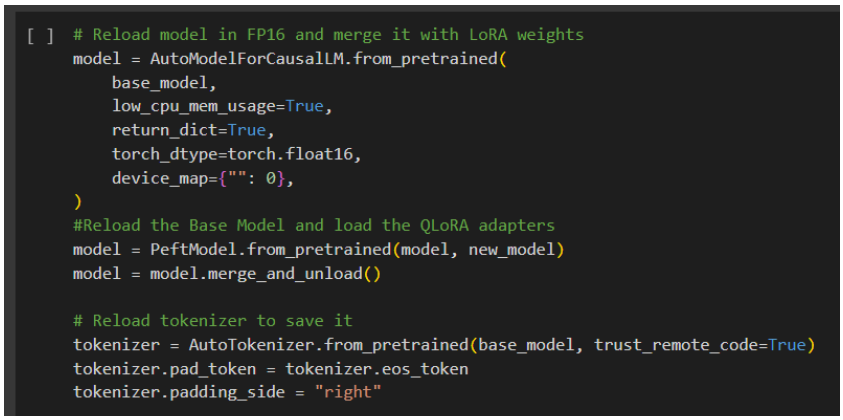


```
[ ] #Configration of QLoRA
    #Quantization Configuration
    #To reduce the VRAM usage we will load the model in
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        #Quant type
        #We will use the "nf4" format this was introduce
        bnb_4bit_quant_type="nf4",
        #As the model weights are stored using 4 bits a
        bnb_4bit_compute_dtype=torch.float16,
        #Quantization parameters are quantized
        bnb_4bit_use_double_quant=True,
    )


    # LoRA configuration
    peft_config = LoraConfig(
        #Alpha is the strength of the adapters. In LoRA
        #train the added weights
        #We can merge these adapters in some layers in a
        #(using a big weight)
        #15 is very big weight, usually 32 is considered
        lora_alpha=15,
        #10% dropout
        lora_dropout=0.1,
        bias="none",
        task_type="CAUSAL_LM",
    )

    # Load base moodel
    model = AutoModelForCausalLM.from_pretrained(
        base_model,
        quantization_config=bnb_config,
        device_map={"": 0}
    )

    model.config.use_cache = False
    model.config.pretraining_tp = 1

    # Cast the layernorm in fp32, make output embedding
    #prepare_model_for_kbit_training---> This function
    model = prepare_model_for_kbit_training(model)
```

```
[ ] # Set training arguments
    training_arguments = TrainingArguments(
            output_dir="./results",
            num_train_epochs=1,#3,5 good for the Llam
            per_device_train_batch_size=4,# Number of
            gradient_accumulation_steps=1,
            evaluation_strategy="steps",#Not helpful
            eval_steps=1000,
            logging_steps=25,
            optim="paged_adamw_8bit",#Adam Optimizer
            learning_rate=2e-4,
            lr_scheduler_type="linear",
            warmup_steps=10,
            report_to="tensorboard",
            max_steps=-1, # if maximum steps=2, it wi
    )

    # Set supervised fine-tuning parameters
    trainer = SFTTrainer(
        model=model,
        train_dataset=dataset,
        eval_dataset=dataset,#No separate evaluation
        peft_config=peft_config,
        dataset_text_field="text",
        max_seq_length=512,# In dataset creation we p
        tokenizer=tokenizer,
        args=training_arguments,
    )

    # Train model
    trainer.train()

    # Save trained model
    trainer.model.save_pretrained(new_model)
```

Figure 7: Configuration of LoRA, Training arguments, SFT parameters

The process of fine-tuning concludes when the trained adapters are merged with the base model. By integrating the knowledge gained from the particular task into the pre-trained architecture, this integration improves the model's performance. Now that the model and tokenizer have been adjusted, they may be deployed to provide enhanced performance that is specific to the needs of our application.



```
[ ] # Reload model in FP16 and merge it with LoRA weights
    model = AutoModelForCausalLM.from_pretrained(
        base_model,
        low_cpu_mem_usage=True,
        return_dict=True,
        torch_dtype=torch.float16,
        device_map={"": 0},
    )
    #Reload the Base Model and load the QLoRA adapters
    model = PeftModel.from_pretrained(model, new_model)
    model = model.merge_and_unload()

    # Reload tokenizer to save it
    tokenizer = AutoTokenizer.from_pretrained(base_model, trust_remote_code=True)
    tokenizer.pad_token = tokenizer.eos_token
    tokenizer.padding_side = "right"
```

Figure 8: Reloading model and merging with LoRA weights

1.4.5 Training Monitoring and Evaluation

TensorBoard is used along with the training process to visually represent parameters like performance and loss, allowing for the evaluation of the fine-tuning procedure's efficacy.

Figure 9: TensorBoard Monitoring

1.4.6 Model Deployment on Huggingface Hub

The final phase was uploading the trained model and tokenizer to the Huggingface hub, namely to the "genify-official/genify-Llama-2-7b-chat-hf" model repository. This ensures smooth deployment and utilization within our application by making the fine-tuned model easier to access and integrate into Genify.

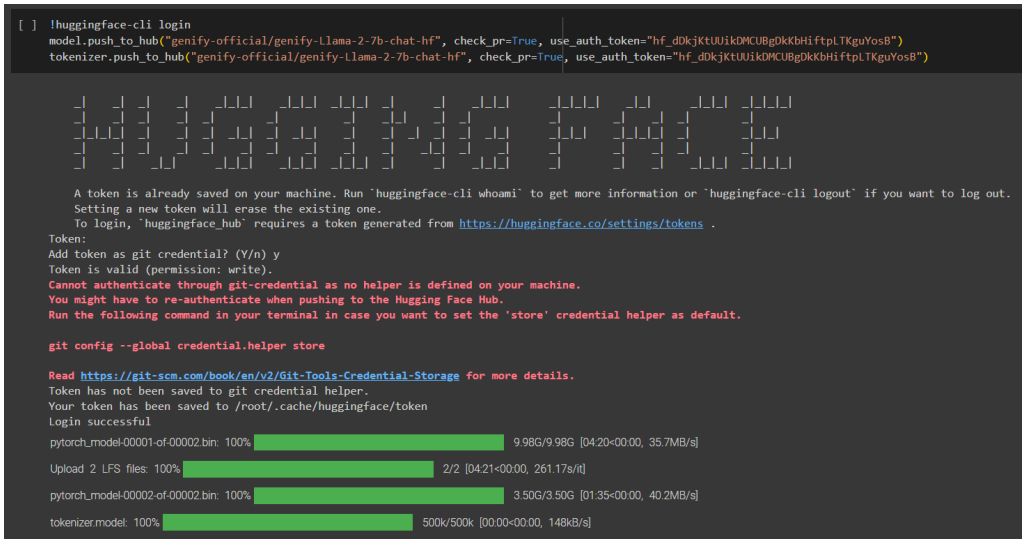


Figure 10: Pushing the model and tokenizer

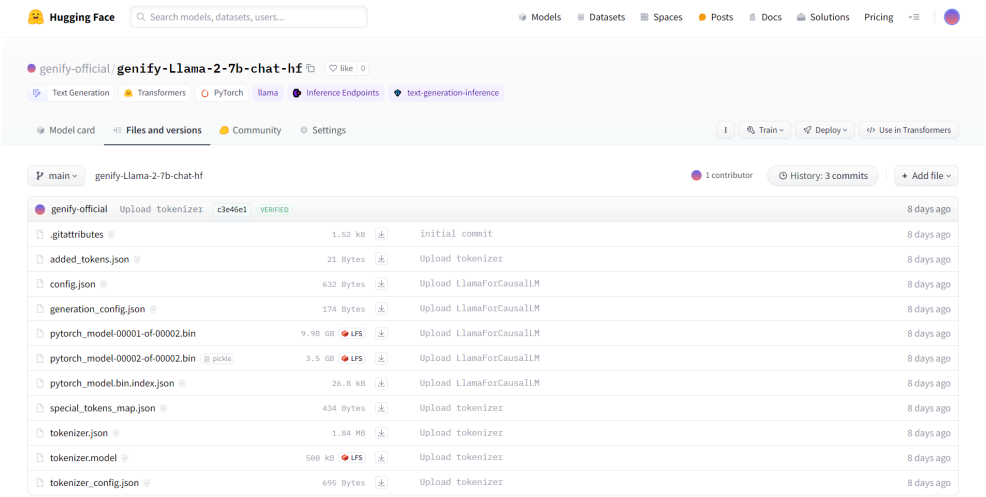


Figure 11: Huggingface hub repository

1.4.6.1 Using Huggingface Inference Endpoints

After exploring various deployment approaches, including fine-tuning smaller models due to limitations posed by larger models, quantizing the model into GGUF format, deploying on azure with FastAPI and Docker, and experimenting with Huggingface Spaces, utilizing Huggingface inference endpoints came up as the most effective solution. Despite efforts with these alternative methods, challenges hindered the seamless deployment of the model. After careful consideration, using Huggingface inference endpoints, particularly the paid option offering better GPU access, was determined as the optimal framework for deploying the model, ensuring both accuracy and efficiency in generating prompt responses.

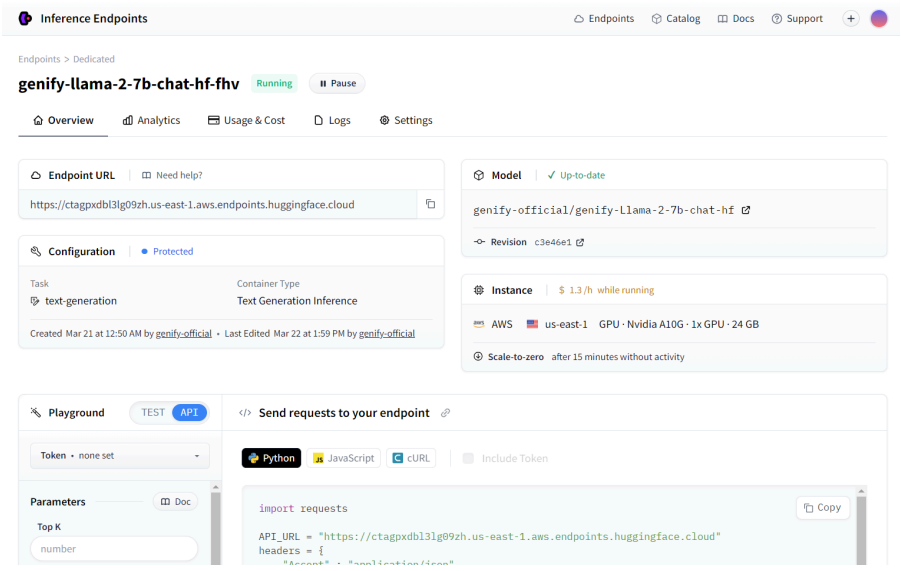


Figure 12: Model deployment Inference Endpoints

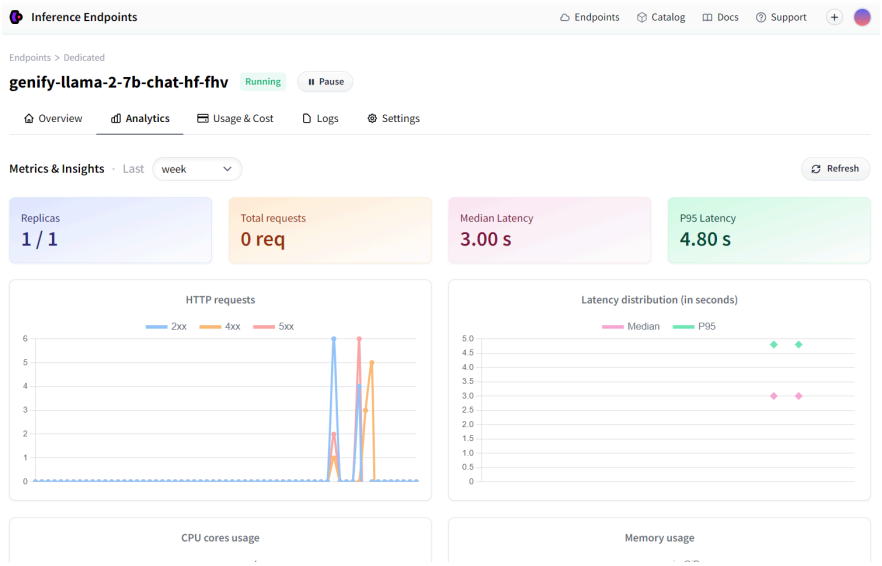


Figure 13: Model deployment Inference Endpoints Analytics

In order to receive the model-generated response, which included parameters like the maximum number of new tokens to generate, the Hugging Face inference endpoint was accessed by sending a POST request with the proper payload, which included the input text and a prompt structured using the llama 2 prompt format.

```
API_URL = "https://ctagpxdbl3lg09zh.us-east-1.aws.endpoints.huggingface.cloud"
HEADERS = {
    "Accept" : "application/json",
    "Authorization": "Bearer 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",
    "Content-Type": "application/json"
}

def query(payload):
    response = requests.post(API_URL, headers=HEADERS, json=payload)
    return response.json()
```

Figure 14: Getting model response

```
output = query({
                "inputs": f"<s>[INST]<<SYS>>{sys_prompt}<</SYS>{input_text}[/INST]",
                "parameters": {
                    "max_new_tokens": 150
                }
            })
```

Figure 15: Getting model response query

1.5 Implementation of the backend component

Genify backend serves as the foundation of the Genify platform, employing concepts such as ORM and OOP to ensure efficient data management and modular code organization with the use

of Flask and SQLAlchemy. The project structure outlined below describes its main elements and organizational structure.

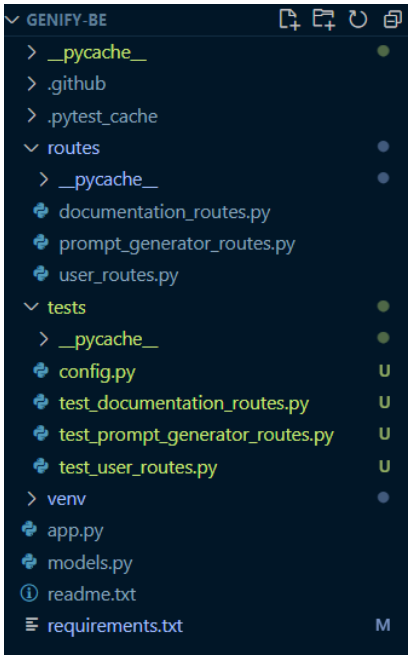


Figure 16: Backend Project Structure

The app.py file serves as the entry point for the Flask application. It registers the blueprints for various routes, initializes the Flask application, sets up the required parameters, including the SQLAlchemy database URI, and launches the Flask server.

```python
app = Flask(__name__)

CORS(app)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+mysqlconnector://genify:g3nify123$@genifydata

db.init_app(app)  # Register SQLAlchemy extension with the Flask app

with app.app_context():
    db.create_all()

app.register_blueprint(users_bp)
app.register_blueprint(documentation_bp)
app.register_blueprint(prompt_generator_bp)

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 16: App.py Code

The routes directory contains route handlers for different functionalities of the backend. The implementation of every route as a blueprint improves the readability and organization of the code. SQLAlchemy models for database entities, such as users and documentation, are defined in

the models.py file. These models communicate with the MySQL database that is deployed in Azure by using ORM techniques.

The prompt generation route's functionality is demonstrated by an example method from the prompt generator routes code. After receiving input text, this method uses a llama 2 prompt format to create a structured system prompt. It then sends a POST request to the Huggingface inference endpoint and returns the created response.

```python
@prompt_generator_bp.route('/generate', methods=['POST'])
def summarize():
    if request.method == 'POST':
        try:
            data = request.json
            input_text = data.get('input_text')
            if input_text:
                input = f"<s>[INST]<<SYS>>{sys_prompt}<</SYS>{input_text}[/INST]"
                output = query({
                                "inputs": input,
                                "parameters": {
                                    "max_new_tokens": 150
                                }
                            })
                return jsonify(output), 200
            else:
                return jsonify({'error': 'Input text is required.'}), 400
        except Exception as e:
            return jsonify({'error': str(e)}), 500
```

Figure 17: Prompt Generation Method

The Genify project's backend makes use of Flask, SQLAlchemy, and MySQL to deliver necessary features for the application. The backend guarantees modularity, scalability, and maintainability by utilizing Flask blueprints, ORM strategies, and hierarchical directory architecture.

1.6 Implementation of the front end component

Genify's front end is designed with modern technology and industry best practices for a user-friendly interface. It uses React.js, TypeScript, Ant Design, and Tailwind CSS for code reuse, maintainability, and aesthetics. The interface offers strong state management, seamless connectivity with backend APIs, and responsive design principles.

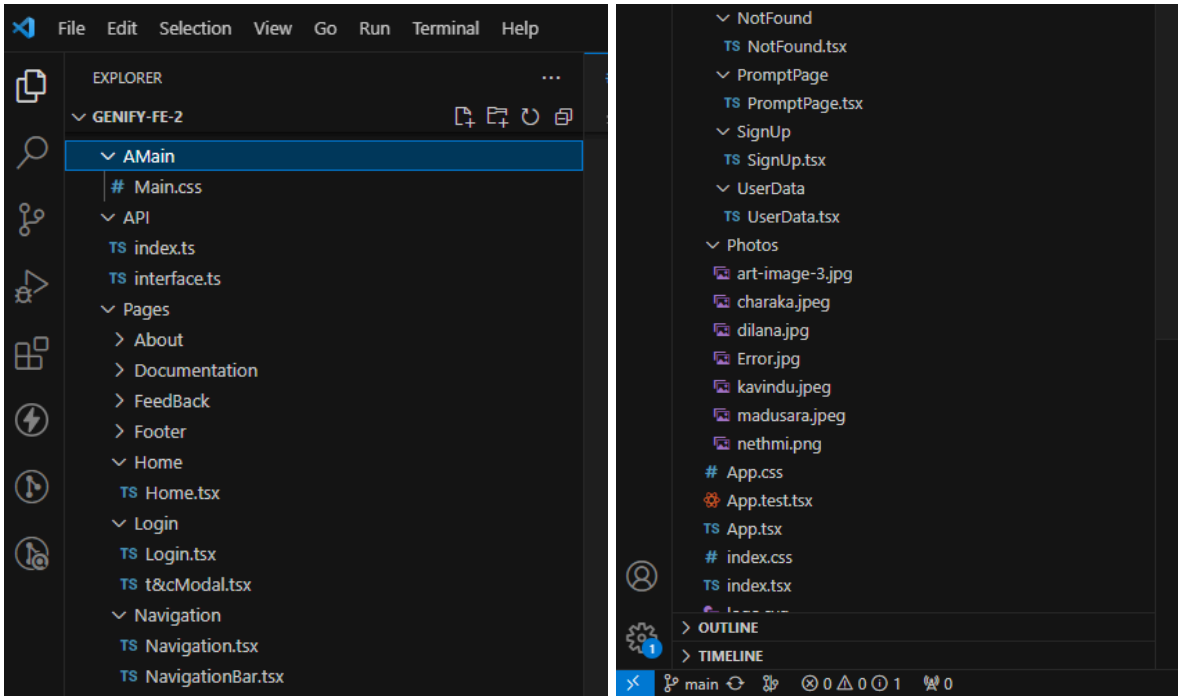


Figure 18: Frontend Project Structure

Welcome to our Login Page – the starting point for your Genify experience. With just a few clicks, access your account securely and dive into a world of productivity and collaboration. Your journey begins here, where simplicity and security intertwine to offer you a seamless login experience. Join us and unlock the full potential of Genify today!

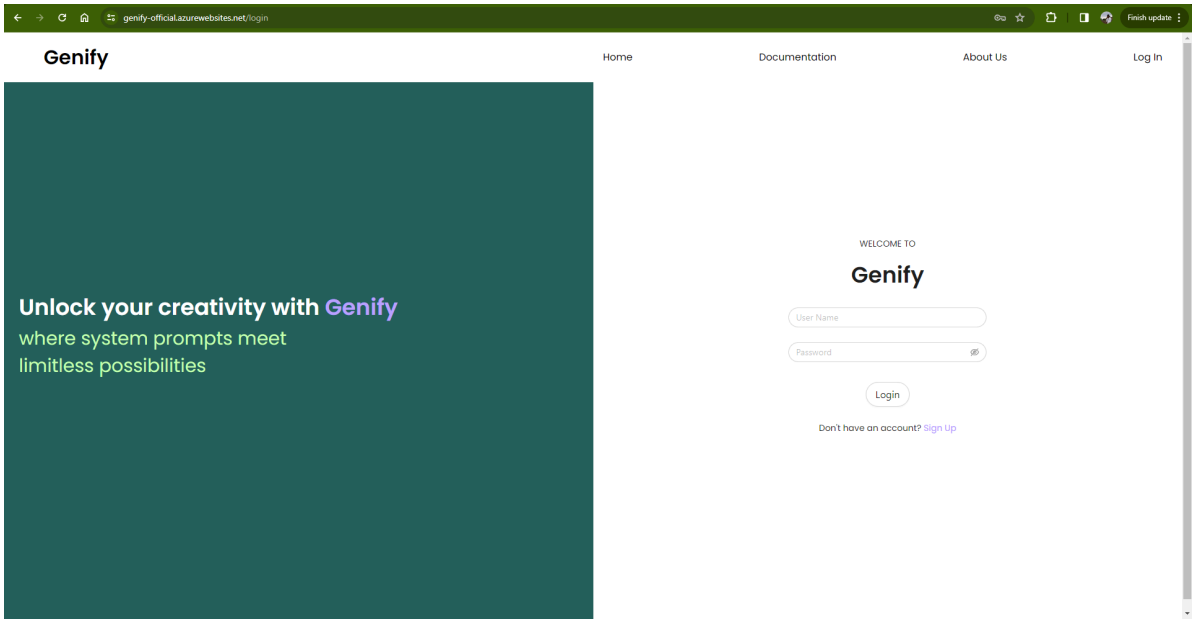


Figure 19: Frontend Login Page

Welcome to the heart of Genify – our front end interface designed to offer users a seamless and intuitive experience. Dive into a world where cutting-edge technologies converge with sleek design principles, enabling effortless navigation and interaction. Below, behold a glimpse of our meticulously crafted home page, where every element is thoughtfully curated to elevate your journey with Genify.

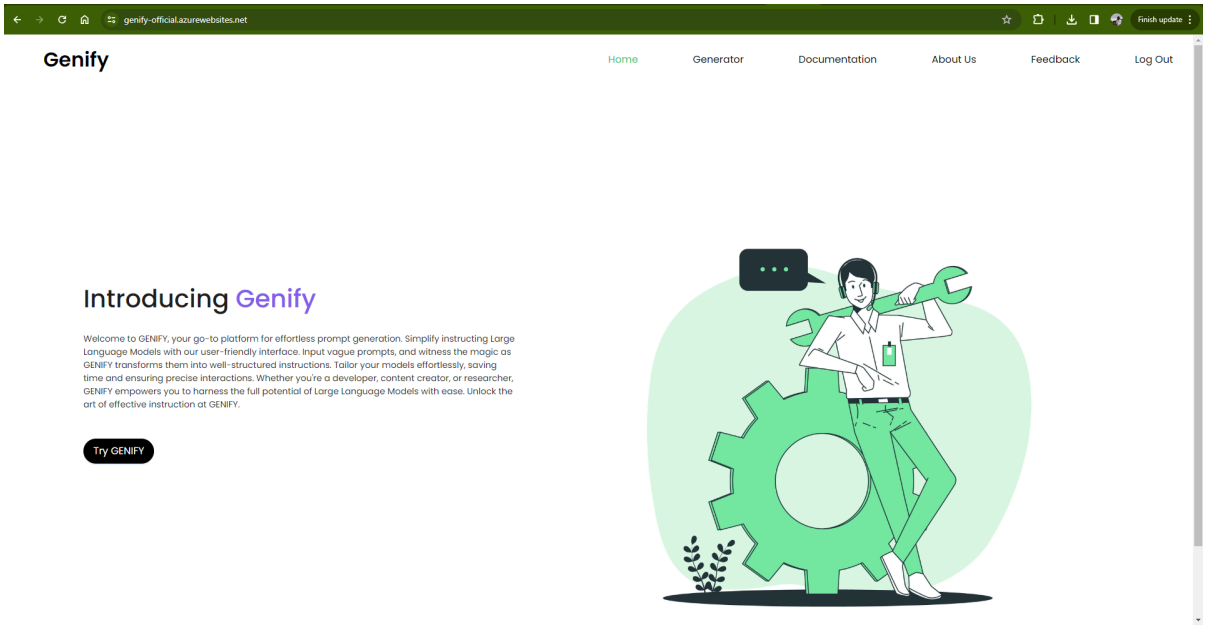


Figure 20: Frontend Home Page

Introducing our Prompt Generator Page – the hub where vague prompts are transformed into well-structured gems. Seamlessly navigating between ambiguity and clarity, this page harnesses advanced algorithms to refine your prompts with precision. Unlock the power of tailored communication as you witness your vague ideas metamorphose into coherent directives. With every click, embark on a journey towards enhanced productivity and streamlined communication within Genify.

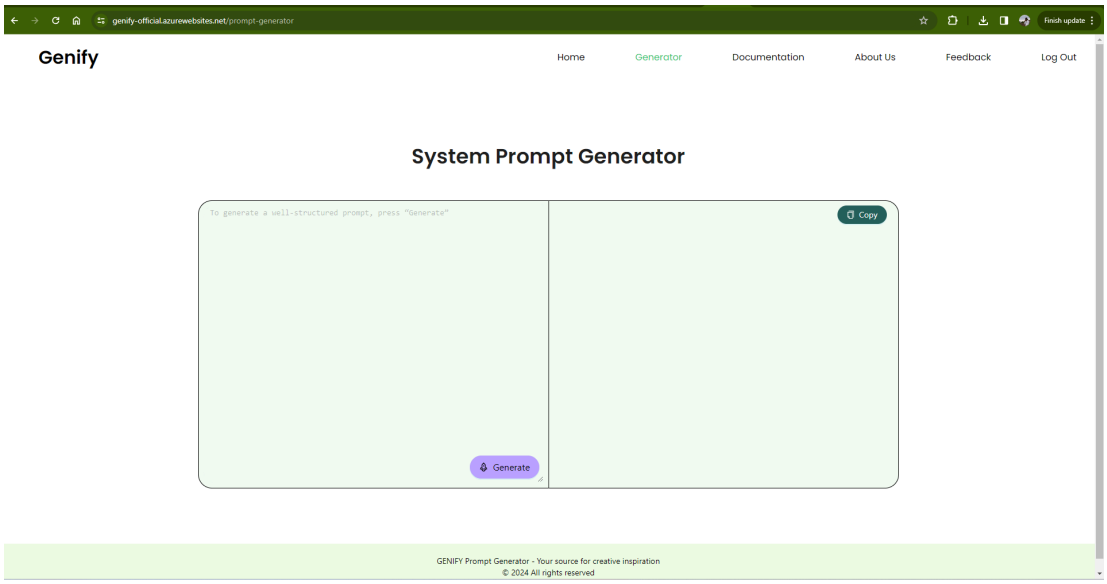


Figure 21: Frontend Generator Page

1.7 GIT Repository

Genify maintains three main repositories on GitHub: GENIFY-BE, GENIFY-FE, and Genify-Colab-Notebooks. The frontend, backend, and Colab notebooks are maintained in these repositories, respectively, and are essential to the development and teamwork of our project. Additional screenshots are provided in the Appendix A.1.

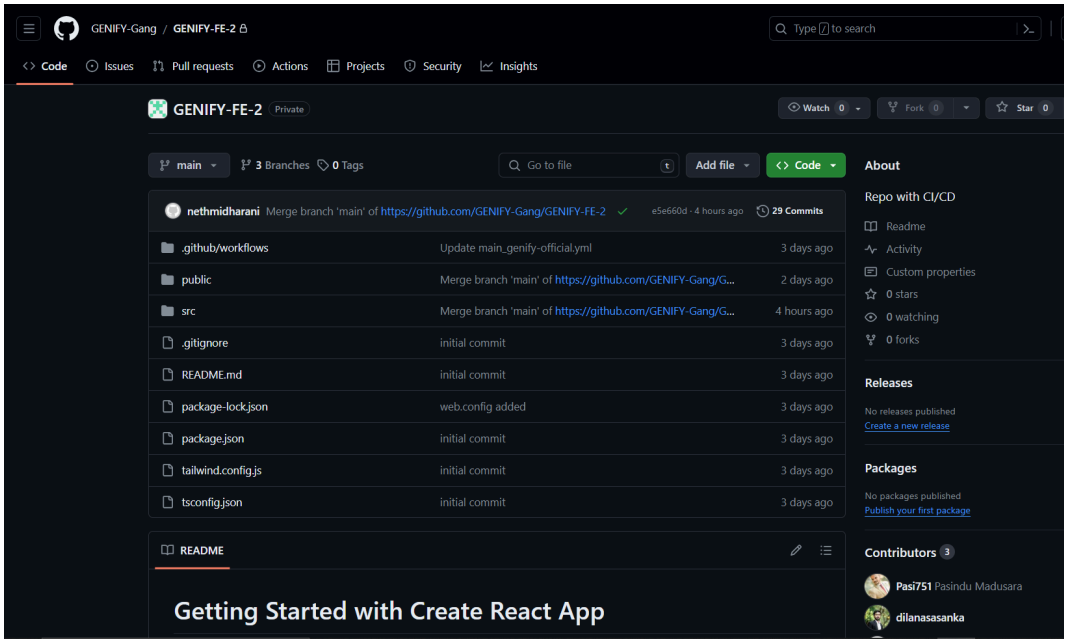


Figure 22: GIT Repository FE

1.8 Deployments/CI-CD Pipeline

The infrastructure of Microsoft Azure is used in the deployment of Genify; the database is hosted on an Azure Database for MySQL flexible server under the name genifydatabase.mysql.database.azure.com, and the backend is hosted on Web App with the domain genifybackend.azurewebsite.net. To guarantee a flawless user experience, the frontend is hosted on a different Web App with the URL genify-official.azurewebsite.net. Additional screenshots which demonstrate the deployment process can be found in the Appendix A.2.

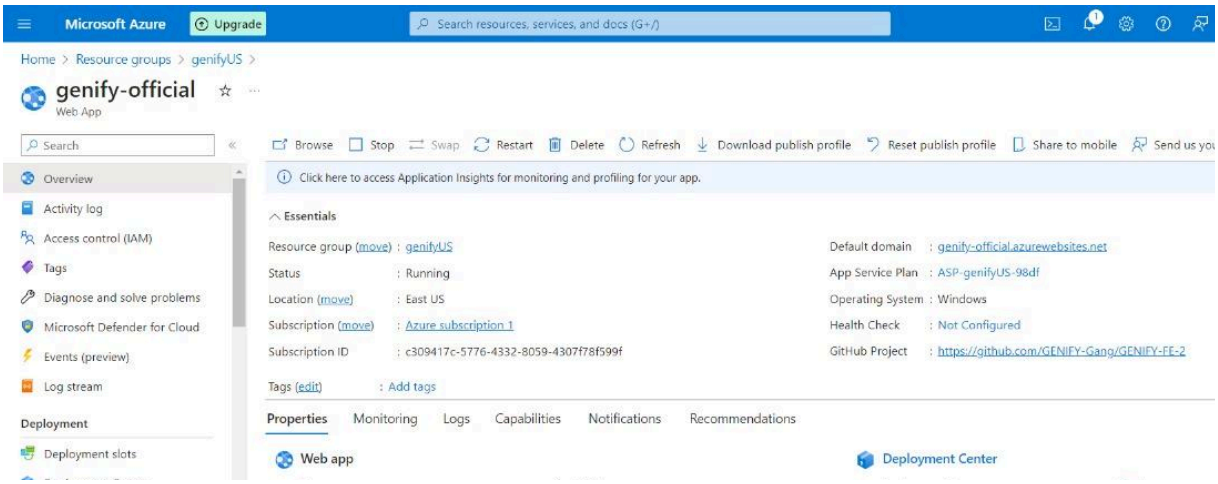


Figure 23: Azure deployment frontend

Changes made to the Github repository initiate automated build and deployment procedures through the use of the CI\CD pipeline, ensuring that Genify on Azure resources are updated immediately. The efficiency of development, responsiveness to user input, and flexibility in addressing changing requirements are all improved by this automation.

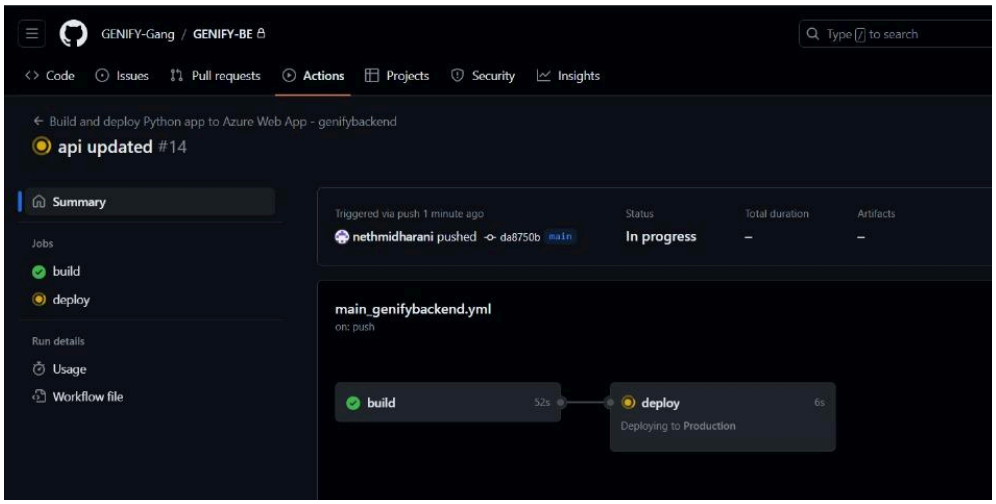


Figure 24: CI/CD Deployment

In summary, GitHub Actions for CI/CD combined with Genify's Azure deployment approach perfectly captures the essence of modern software delivery. Genify uses Azure's infrastructure and automation powers to provide reliable hosting, efficient deployments, and continuous innovation, ultimately enhancing the user experience.

1.9 CRUD operations

The Genify project's CRUD functions were implemented with excellent collaboration by all members. To guarantee reliable performance throughout a variety of user interactions, these procedures were carefully designed and put through testing. Remaining screenshots of the CRUD operations can be referenced in the Appendix A.4.

```python
@users_bp.route('/register', methods=['POST'])
def register_user():
    data = request.get_json()
    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
    role = data.get('role')

    # Check if the user with the given username or email already exists
    existing_user = User.query.filter((User.username == username) | (User.email == email)).first()
    if existing_user:
        return jsonify({'error': 'Username or email already exists'}), 400

    # Create a new user
    user = User(username=username, email=email, password=password, role=role)
    db.session.add(user)
    db.session.commit()

    # Generate a JWT token for the registered user
    access_token = create_access_token(identity=username, expires_delta=datetime.timedelta(days=1))

    return jsonify({'message': 'User registered successfully', 'access_token': access_token, 'role': role})
```

Figure 25: Register User CRUD

```python
@users_bp.route('/login', methods=['POST'])
def login_user():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    # Find the user with the given username
    user = User.query.filter_by(username=username).first()

    # Check if the user exists and the password is correct
    if user and user.password == password:
        # Generate a JWT token for the logged-in user
        access_token = create_access_token(identity=username, expires_delta=datetime.timedelta(days=1))
        return jsonify({'message': 'Login successful', 'access_token': access_token, 'role': user.role, 'email':user.email})
    else:
        return jsonify({'error': 'Invalid username or password'}), 401
```

Figure 26: Login User CRUD

1.10 Chapter Summary

The implementation chapter of the Genify project provides a comprehensive overview of the development process,highlighting key aspects such as prototype,technology selections,and the implementation of various components. The chapter begins with an overview of the prototype,outlining the design and functionality of the Genify system. It then dives into the technology selections,detailing the choice of the technologies for both the frontend and backend development.A significant portion of the chapter is dedicated to the implementation of the data science component,which is crucial for the core functionality of Genify. This includes detailed sections on dataset creation,prompt engineering research,and effective strategies for prompt engineering in large language models. The chapter also covers the collection of prompts,prompts template creation, and the fine-tuning process using the PEFT techniques. Challenges related to full fine tuning,parameter-efficient fine-tuning, low-rank adaptations,and quantized low-rank adaption are discussed in detail.The use of Git for project storage and collaboration is also highlighted.Furthermore, chapter discusses the deployment process CI/CD pipeline used for Genify.Finally it touches upon CRUD operations used in building and maintaining the system.

Chapter 2: Testing

2.1 Chapter Overview

This chapter will discuss testing conducted on the Genify. The testing criteria, functional and nonfunctional requirements testing are conducted and documented in order to ensure the standard of the system. Furthermore, usability testing, unit testing, performance testing, and compatibility testing were conducted and documented in this chapter.

2.2 Testing Criteria

The main testing criteria that will be focused on this chapter when testing the Genify is the functional and the non-functional requirements of the system. If all the test cases can fulfill the specified results and conditions, and all the components are working accordingly to satisfy the performance quality and the functionalities which are expected, it can be concluded that the Genify is working as it’s intended.

2.3 Testing functional requirements

Requirements list		Priority Level	Expected Results	Actual Results	Status
FR 1	Prompt generation	Critical	Users should be able to input a vague prompt and the system must generate well-structured prompts.	Users are able to input a vague prompt and the system generates well-structured prompts.	pass
FR 2	Prompt evaluation	Critical	Implement a real-time prompt evaluation system that provides feedback on prompt clarity.	User is able to experience a real-time prompt evaluation system that provides feedback on prompt clarity	pass
FR 3	Prompt Customization	Critical	Enable users to customize prompts based on task specific requirements.	Genify enables users to customize prompts based on task specific requirements.	pass
FR 4	LLM Integration	Critical	Integrate a pre-trained LLM for fine tuning.	Llama-2-7b-chat-hf was fine tuned by the team with our dataset to genify-official/genify-Llama-2-7b-chat-hf for the prompt generation task.	pass
FR 5	User Accounts	Critical	Implement user authentication and profiles for managing and tracking prompts	User authentication and profiles for managing and tracking prompts is available in the Genify web tool.	pass
FR 6	Fine-tuning LLM	Critical	Fine-tuning the pre-trained LLM with an ethically collected dataset and appropriate	Dataset was prepared by the team itself and the various sources that were used to collect data were provided within references through	pass

			fine-tuning techniques.	meticulous documentation.	
FR 7	User Support	Desirable	Provide assistance and guidance features to support users in prompt generation.	The Genify webtool consists of sections for user assistance and guidance features to support users in prompt generation.	pass
FR 8	Efficiency in prompt generation	Desirable	Ensure the system is capable of generating the well structured prompt with a minimal amount of time.	System is able to produce the required outputs with the expected minimal time.	pass
FR 9	Domain agnostic prompt generation capability	Desirable	Ensure the system is capable of handling different domains of user inputs and provide user satisfactory results.	Genify is able to handle and produce well structured prompts regardless of the domains in the user inputs.	pass
FR 10	Reliable web hosting	Luxury	Hosting the webpage in a reliable and secure web service.	Azure was used for the reliable web hosting of the frontend backend and databases and genify-official/genify-Llama-2-7b-chat-hf is available at aws via hugging face.	pass
FR 11	Responsive design	Critical	The Genify webtoll must be designed in a way that is responsive across multiple devices.	The Genify webtool adapts seamlessly to different resolutions providing and optimal viewing experience for users.	pass
FR 12	Search functionality	Critical	The page should include a search feature that	An intuitive fast and accurate search functionality is available	pass

			allows users to easily navigate into different sections.	in Genify enabling users to locate relevant content efficiently.	

Table 1: Testing Functional Requirements

2.4 Testing non-functional requirements

Requirements list		Priority Level	Expected Results	Actual Results	Status
FR 1	Regulatory compliance	Critical	The system should adhere to relevant regulatory requirements and standards.	The was implemented in a way that adheres to all the social ethical legal professional guidelines in BCS code of conduct.	pass
FR 2	Documentation	Critical	The system should include comprehensive documentation assisting users.	The system includes a well prepared documentation that the admin is able to update in a timely manner including use guidelines, technical manuals and troubleshooting resources.	pass
FR 3	Feedback mechanism	Critical	The system should include a feedback mechanism.	The system includes a feedback mechanism, allowing users to provide	pass
FR 4	Error handling	Critical	The system should effectively handle errors and exceptions.	The system effectively handles errors and exceptions.	pass

FR 5	Usability	Desirable	The system should be implemented in a way that's enjoyable, efficient, user friendly, enhancing the overall user experience.	The system is implemented in a way that's enjoyable, efficient, user friendly, enhancing the overall user experience.	pass
FR 6	Maintainability	Desirable	The system ought to include a code repository that is properly maintained and managed.	The system includes a code repository that is properly maintained and managed.	pass

Table 2: Testing Non Functional Requirements

2.5 Unit testing

During the development phase, unit testing was done to make sure that each system component worked as intended. The detected errors and issues were carefully fixed, which improved the application's overall robustness and dependability.

We used the pytest framework, a well-known testing tool for Python applications, for unit testing. Within the tests directory, the tests were divided into distinct modules, each of which concentrated on a particular functionality, such as user registration, login, documentation management, and prompt generation. To guarantee data integrity and isolation, we used a different database instance from the production database.

```
class TestConfig:
    TESTING = True
    SQLALCHEMY_DATABASE_URI = 'mysql+mysqlconnector://root:password*@localhost/genify'
    JWT_SECRET_KEY = 'yX*F2wRpE78Ct!LkG$mjQvZpF9p#bS%z'
```

Figure 27: TestConfig

```python
@pytest.fixture
def client():
    app.config.from_object('config.TestConfig')
    with app.test_client() as client:
        with app.app_context():
            db.create_all()
            yield client
            db.session.remove()
            db.drop_all()

def test_register_user(client):
    # Test registration endpoint
    data = {'username': 'testuser', 'email': 'test@example.com', 'password': 'password', 'role': 'user'}
    response = client.post('/user/register', json=data)
    assert response.status_code == 200
    assert b'User registered successfully' in response.data

    # Check if the user is added to the database
    user = User.query.filter_by(username='testuser').first()
    assert user is not None
    assert user.email == 'test@example.com'
```

Figure 28: Test case example

```
(venv) D:\GENIFY-BE>python -m pytest -v
============================= test session starts =============================
platform win32 -- Python 3.11.4, pytest-8.1.1, pluggy-1.4.0 -- D:\GENIFY-BE\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: D:\GENIFY-BE
collected 8 items

tests/test_documentation_routes.py::test_update_documentation PASSED     [ 12%]
tests/test_documentation_routes.py::test_create_documentation PASSED     [ 25%]
tests/test_documentation_routes.py::test_get_documentation PASSED        [ 37%]
tests/test_prompt_generator_routes.py::test_generate_prompt PASSED       [ 50%]
tests/test_user_routes.py::test_register_user PASSED                     [ 62%]
tests/test_user_routes.py::test_login_user PASSED                        [ 75%]
tests/test_user_routes.py::test_create_user PASSED                       [ 87%]
tests/test_user_routes.py::test_get_users PASSED                         [100%]

============================== 8 passed in 61.16s (0:01:01) ===============================

(venv) D:\GENIFY-BE>
```

Figure 29: Test case results

The behavior of essential endpoints, including user registration, login, and prompt creation, is verified by these unit tests.We make sure that every system component operates as intended by systematically carrying out these tests, which adds to the overall stability and usefulness of the application.

2.6 Performance testing

The simplification module's large lag, which resulted in several delays and timeouts, had the largest effect on performance, according to performance tests done on the web application. The issues that were found, such as the ones mentioned above, were resolved, and the total web application's performance was improved.

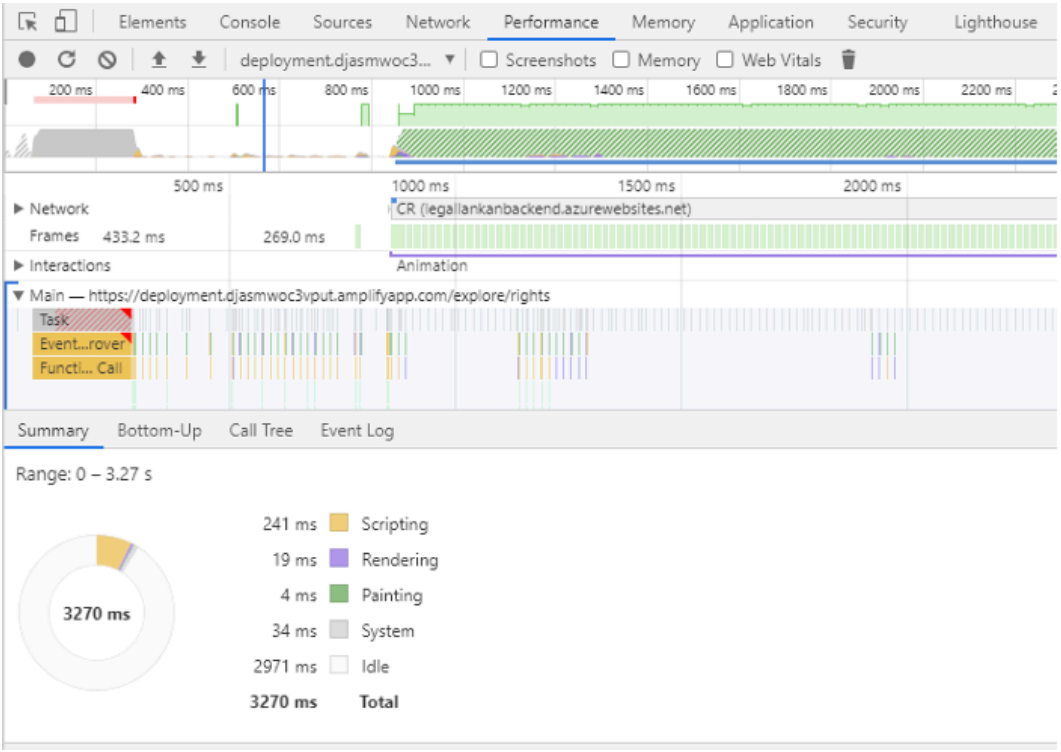


Figure 30: Performance Testing

2.7 Usability testing

Usability testing is used to confirm that the User Interface and User experience adhere to recognized standards. We aim to find any usability problems and improve the UI to give users a smooth and intuitive experience by putting the program through a variety of test cases under diverse scenarios. Moreover, UI screenshots may be obtained in the Implementation of the Frontend Component part of Chapter 1. This section presents the usability test results.

No	Test Case	Conditions	Expected Result	Actual Result	Status
1	User Registration	User attempts to register an	User successfully registers an	User account is created	Passed

		account	account	successfully	
2	User Login	User attempts to log in with valid credentials	User successfully logs into their account	User is able to access their account after logging in	Passed
3	Prompt Generation	User provides input for prompt generation	Genify generates a structured prompt	Genify successfully generates a structured prompt	Passed
4	Prompt Generation Accuracy	User evaluates the quality of generated prompts	Generated prompts are coherent and relevant	Most generated prompts are coherent and relevant	Passed
5	Documentation Access	User navigates to the documentation section	User can access comprehensive documentation	Documentation section provides relevant information	Passed
6	Update Documentation	Admin user attempts to update documentation content	Admin user successfully updates documentation	Documentation content is successfully updated	Passed

Table 3: Usability Testing

2.8 Compatibility testing

Compatibility testing was performed to guarantee that the Genify web application works seamlessly across a range of browsers and devices. The website underwent testing on common browsers including Firefox, Chrome, Safari, and Microsoft Edge, along with different device categories such as desktop computers, laptops, and tablets. Detailed results of these tests, along with relevant screenshots, are documented in the Appendix section B.1.

2.9 Chapter Summary

The document outlines rigorous testing procedures for both functional and non-functional requirements of the Genify web application. It includes unit testing using the pytest framework, performance testing, usability testing, and compatibility testing across various browsers and devices. Functional requirements such as prompt generation, evaluation, and customization were thoroughly tested, ensuring critical functionalities operated as expected.Non-functional aspects like regulatory compliance, documentation quality, and error handling were also scrutinized to

guarantee the applications reliability, user-friendliness, and maintainability. Overall, the comprehensive testing approach aimed to ensure the genify web application delivers a seamless, efficient, and satisfactory user experience while meeting high standards of performance and reliability.

Chapter 3: Evaluation

3.1 Chapter Overview

This chapter provides a comprehensive overview of the evaluation process conducted for Genify. This outlines the evaluation methods employed, including quantitative and qualitative assessments, to measure effectiveness and usability of Genify in generating structured prompts. Additionally, it highlights the the significance of gathering the feedback from various stakeholders including en users, domain experts and industry professionals to obtain holistic understanding of the tool's performance and potential areas of improvement.,

3.2 Evaluation methods

The evaluation methods used in the Genify project include both quantitative and qualitative approaches. Quantitatively, the effectiveness of Genify in generating well structured prompts from vague prompts was accessed using a systematic approach. Eight parameters were established as guidelines for generating structured prompts,and the percentage of parameters executed in each of the 50 generations was recorded. The average parameter usage percentage across all generations was calculated to provide an overall measure of Genify's adherence to the guidelines. Qualitatively, feedback was gathered from end users,domain experts,and industry experts. Structured questionnaires were used to collect feedback,which was then analyzed to assess Genify's effectiveness and usability.

Overall,these evaluation methods provide a comprehensive assessment of Genify, combining quantitative metrics with qualitative insights to evaluate its performance and user satisfaction.

3.3 Quantitative evaluation

In the quantitative evaluation of Genify, a systematic approach has been used to assess the effectiveness of the tool in generating well-structured prompts from vague prompts. To achieve this we established eight parameters as guidelines for generating structured prompts, including step-by-step instructions, the use of triple backticks to highlight variables, clear instructions separation, and more.

We conducted 50 generations of prompts using the application and recorded the percentage of parameters executed in each generation.

For an example, if seven out of eight parameters executed in a generation, the parameter percentage would be calculated as,

(7 / 8) * 100%

The average parameter usage percentage across 50 generations was calculated to provide an overall measure of Genify's adherence to the established guidelines.

The R code snippet provided illustrates the process of calculating the average parameter usage percentage from the results obtained in the 50 generations. Additionally, a plot depicting the parameter usage percentage in each generation is presented to visualize the distribution and trends observed throughout the evaluation process.

```r
# Results for 50 generations
generation_results <- c(0.75, 0.80, 0.65, 0.90, 0.87, 0.83, 0.65, 0.70, 0.89, 0.90,
                        0.80, 0.85, 0.90, 0.75, 0.85, 0.68, 0.80, 0.88, 0.78, 0.80,
                        0.64, 0.75, 0.85, 0.96, 0.75, 0.80, 0.75, 0.85, 0.68, 0.70,
                        0.85, 0.95, 0.87, 0.65, 0.75, 0.85, 0.90, 0.80, 0.68, 0.80,
                        0.84, 0.94, 0.85, 0.87, 0.65, 0.70, 0.85, 0.96)

# Calculate average parameter usage percentage
average_parameter_percentage <- mean(generation_results) * 100

# Print average parameter usage percentage
cat("Average parameter usage percentage:", average_parameter_percentage, "%\n")

# Plot generation results
plot(generation_results, type = "o", xlab = "Generation", ylab = "Parameter Usage Percentage"
     main = "Parameter Usage Percentage in 50 Generations")
```

Figure 31: R code snippet for evaluation

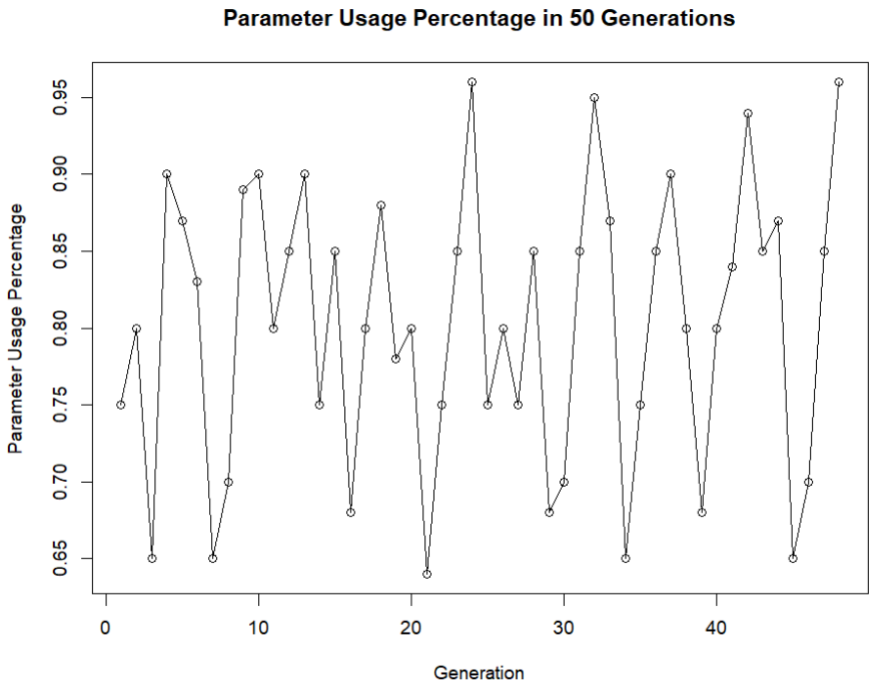


Figure 32: Parameter Usage Percentage Graph

The results of our calculation indicate an average parameter usage percentage of 80.25%. This suggests that, on average, Genify successfully incorporates approximately 80.25% of the predefined parameters in the generated prompts.

```
> # Print average parameter usage percentage
> cat("Average parameter usage percentage:", average_parame
Average parameter usage percentage: 80.25 %
>
```

Figure 33: Average Parameter Usage Percentage

3.4 Qualitative evaluation

A standardized questionnaire was utilized to conduct a qualitative assessment as part of an effort to obtain a comprehensive evaluation of the Genify project. The questionnaire was created to collect information from a variety of sources and was targeted at the general public, industry professionals, and machine learning domain experts. Participants were invited to share their experiences, opinions, and feedback on the Genify web tool, with the questionnaire including a direct link to the website for reference, Additionally, screenshots of the questionnaire and result analysis are available in the Appendix C.1.

3.4.1 Feedback gathered from end users, domain experts and industry experts

End-user feedback on the Genify web tool yields important information about its effectiveness and usability. Feedback was gathered from a variety of user groups, including members of the general public, domain experts, and industry experts, using structured questionnaires.

Feedback from general public:

"During my experience with Genify, I found the interface to be incredibly intuitive. It made the process of generating structured prompts for language models seamless and user-friendly. I appreciated the tool's simplicity, as it allowed even users with minimal technical expertise to navigate it easily."

Feedback from a domain expert:

"As a machine learning engineer, I must say I'm impressed with Genify. It has streamlined the process of customizing language models significantly. The tool's flexibility in generating prompts tailored to specific tasks or domains has been instrumental in my research projects, saving me considerable time and effort"

Feedback from a industry expert:

"From an industry perspective, I see immense potential in Genify. It has applications across various sectors, including content generation, customer support automation, and natural language understanding tasks. I believe this tool could revolutionize the way businesses interact with language models, leading to more efficient and personalized user experiences."

Feedback from industry professionals, subject experts, and end users provides insightful information on Genify's areas of strength and improvement. Enhancements for a better user experience are guided by end user's viewpoints, which provide insight into usability and interface intuitiveness. To inform feature creation, domain experts offer feedback on research suitability and compatibility with machine learning concepts. Industry experts provide valuable perspectives on feasible commercialization opportunities and practical use, directing Genify's development to successfully satisfy user expectations and industry standards. This comprehensive input makes sure that Genify is up-to-date, approachable, and in line with industry and research demands.

3.5 Self evaluation

The development journey of Genify has been marked a notable success, challenges and opportunities for growth. Leveraging technologies like Python, Flask, MySQL, React, Hugging Face Hub, Colab pro and Azure our team successfully created a functional and user- friendly application, showcasing proficiency in navigating diverse tools and platforms.

Despite achievements, challenges such as technical hurdles and constraints were encountered. Overcoming these obstacles required resilience, resourcefulness and innovative problem solving approaches. Moving forward areas of improvements were identified including enhancing user interface design, optimizing backend process and incorporating user feedback into iterative development cycles.

In conclusion , while celebrating accomplishments, our commitment to continuous improvement remains unwavering. Guided by self evaluation and user feedback, we are dedicated to refining and enhancing Genify to deliver meaningful value and impact in the dynamic realm of natural language processing.

3.6 Chapter Summary

Chapter three delves into comprehensive evaluation of the webtool, the evaluation process is meticulously dissected, employing both qualitative and quantitative methodologies to gauge tools effectiveness and usability. Qualitative evaluation entails a systematic analysis of the Genify’s adherence to predefined parameters, through 50 generations of prompts. The results indicate an average parameter usage percentage of 80.25%, showcasing the tool’s proficiency incorporation established guidelines into prompt generation. Such performance matrices, enabling a quantitative assessment of the effectiveness. The qualitative evaluation was conducted through a standardized questionnaire targeting diverse user groups. End users praise Genify's intuitive interface and seamless prompt generation process, while domain experts recognize its flexibility and potential application in various sectors. In summary the evaluation underscores Genify’s significance as a versatile tool with implications across research and industry domains.

Chapter 4: Conclusion

4.1 Chapter Overview

In the conclusion, we summarize the key findings and insights derived from the implementation, testing, and evaluation of the Genify project. This section serves as a wrap-up of the entire report, providing a concise overview of the journey from inception to implementation.

4.2 Achievements of aims and objectives

The Genify project has successfully realized its predetermined aims and objectives, marking significant milestones in the following areas:

1. Prototype Development: The project successfully developed a functional prototype aimed at transforming vague prompts into well-structured directives. Leveraging technologies such as Python, Flask, MySQL, React, and Hugging Face Hub, the prototype demonstrates the feasibility of automating prompt refinement processes.

2. Data Science Component Implementation: Through meticulous dataset creation and prompt engineering research, the project established a robust foundation for prompt generation. By curating a dataset of 500 data points and implementing effective prompt engineering strategies, the project ensured the generation of high-quality prompts aligned with industry standards.

3. Parameter-efficient Fine-tuning (PEFT): By employing advanced PEFT techniques, including Low-Rank Adaptations (LoRA) and Quantized Low-Rank Adaptation (QLoRA), the project optimized model fine-tuning processes. This approach effectively addressed challenges related to memory efficiency, preventing catastrophic forgetting, and reducing computational costs, thus enhancing the overall efficiency of the model.

4. Backend and Frontend Implementation: The project successfully implemented both backend and frontend components of the Genify platform. Utilizing Flask, SQLAlchemy, React.js, and other modern technologies, the platform offers a user-friendly interface for prompt generation and management.

5. Deployment and CI/CD Pipeline: Leveraging Microsoft Azure infrastructure and GitHub Actions for CI/CD, the project achieved seamless deployment of the Genify platform. This ensures reliability, scalability, and efficiency in delivering prompt generation services to users.

6. Collaborative Development: The project fostered collaboration among team members, ensuring efficient coordination and synergy in achieving project objectives. Through effective communication and teamwork, the project maintained momentum and achieved key milestones within the specified timeframe.

Overall, the Genify project's achievements demonstrate its capacity to revolutionize prompt refinement processes, offering tangible benefits in terms of efficiency, accuracy, and user experience.

4.3 Limitations of the research

While the Genify project has achieved significant milestones, it is important to acknowledge certain limitations that may impact its scope, effectiveness, and applicability:

1. Scope Constraints: Similar to many projects, Genify faced challenges related to scope limitations. The project primarily focuses on transforming vague prompts into well-structured directives within a specific domain. However, expanding the scope to handle prompts across diverse domains could present practical challenges, including time constraints and resource limitations.

2. Dataset Constraints: The effectiveness of prompt generation heavily relies on the quality and diversity of the dataset used for training. In the case of Genify, the dataset utilized may be limited in size and scope, potentially impacting the model's ability to generalize across various prompt types and contexts. Attempts to further enhance the dataset's quality and size may encounter diminishing returns, especially within a fixed timeframe.

3. Syntactic Simplification Model: The current implementation of the syntactic simplification model in Genify focuses primarily on splitting compound sentences into simpler ones. However, this approach may not fully address the complexity of certain prompts that require more extensive restructuring for clarity. Future enhancements could explore additional techniques to handle a wider range of syntactic complexities.

4. Language Constraints: Genify's prompt generation capabilities are currently limited to the English language. While this aligns with the project's objectives and timeline, it may restrict its applicability in multilingual settings or contexts where prompts are generated in languages other than English. Addressing language constraints could broaden Genify's reach and utility in diverse linguistic environments.

5. Resource and Time Constraints: Like any research project, Genify operates within resource and time constraints. These constraints may impact the depth of exploration, the

scale of experimentation, and the extent of implementation. As a result, certain aspects of the project, such as fine-tuning models or conducting extensive evaluations, may be limited in scope or rigor.

Acknowledging these limitations is essential for ensuring a comprehensive understanding of Genify's capabilities and potential areas for improvement. By addressing these constraints in future iterations, the project can continue to evolve and enhance its effectiveness in prompt generation tasks.

4.4 Future enhancements

As we look forward to advancing our platform, here are some potential areas for future development and improvement tailored to our project:

1. Automated Prompt Generation Process: Enhance our prompt generation system by implementing advanced algorithms to automate the process of creating prompts from vague inputs. By leveraging natural language processing (NLP) techniques, we can streamline prompt generation, making it more efficient and user-friendly.

2. Integration of Advanced NLP Models: Explore the integration of state-of-the-art NLP models to further enhance the quality and diversity of prompts generated by our system. By incorporating cutting-edge models and algorithms, we can improve the accuracy and relevance of prompt outputs, providing users with more tailored and insightful suggestions.

3. Expansion to Multilingual Support: Extend our platform's capabilities to support multiple languages, catering to a broader user base with diverse linguistic backgrounds. By implementing multilingual support, we can make our platform more accessible and inclusive, serving users across different regions and language preferences.

4. Integration of Feedback Mechanisms: Implement feedback mechanisms to gather user input and refine our prompt generation algorithms over time. By collecting feedback from users and incorporating it into our system, we can iteratively improve the quality and relevance of prompt outputs, ensuring a personalized and adaptive user experience.

5. Enhanced User Interface and Experience: Invest in enhancing the user interface and experience of our platform to make it more intuitive, visually appealing, and user-friendly. By optimizing the design and usability of our interface, we can enhance user engagement and satisfaction, driving adoption and usage of our platform.

6. Collaboration with Legal Experts: Foster collaborations with legal experts and practitioners to ensure that our prompt generation system meets the needs and standards of the legal profession. By soliciting feedback and insights from domain experts, we can refine our algorithms and prompts to align with industry best practices and legal requirements.

7. Scalability and Performance Optimization: Focus on optimizing the scalability and performance of our platform to accommodate growing user demand and data volume. By implementing scalable architecture and performance optimizations, we can ensure that our platform remains robust, reliable, and responsive under increasing usage and workload.

These future enhancements aim to advance the capabilities, usability, and impact of our platform, empowering users with more effective and efficient prompt generation tools tailored to their specific needs and preferences. By prioritizing innovation and continuous improvement, we can position our platform as a leading solution in the field of prompt generation and beyond.

4.5 Extra work (Competitions, research papers, etc)

The project team engaged in external activities such as participating in Codesprint, a competitive programming event. This participation not only demonstrated the team's technical skills and knowledge but also provided an opportunity to benchmark the abilities against a border community of developers.

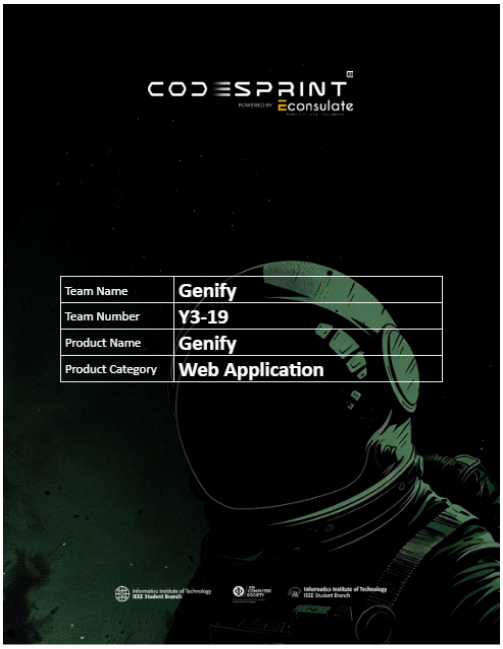


Figure 34: Codesprint

4.6 Concluding remarks

This project revolutionized prompt generation using Large Language Models (LLMs). It developed algorithms to transform vague prompts into structured directives. The project used modern software engineering techniques to create a robust system architecture. Testing and validation procedures ensured accuracy and efficacy. Feedback from users and experts improved the system's user-friendly experience. Future enhancements and refinements are needed to stay ahead of prompt generation research. The project demonstrates the potential of LLMs to revolutionize prompt generation and communication, empowering users in various domains.

References

1. Hu, E. et al. (2021). *LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS*. arXiv. Available from https://arxiv.org/pdf/2106.09685.pdf [Accessed 28 January 2024]

2. Touvron, H. et al. (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models.* arXiv. Available from https://arxiv.org/pdf/2307.09288.pdf [Accessed 24 January 2024]

3. DeepLearning.AI. (2021). ChatGPT Prompt Engineering for Developers. DeepLearning.AI.

4. Available from https://learn.deeplearning.ai/chatgpt-prompt-eng/lesson/2/guidelines [Accessed 2 January 2024].

5. DeepLearning.AI. (2024). Prompt Engineering with Llama 2. DeepLearning.AI. Available from https://learn.deeplearning.ai/courses/prompt-engineering-with-llama-2/lesson/5/prompt-engineering-techniques [Accessed 10 February 2024].

6. Hugging Face. (2022). PEFT. Hugging Face. Available from https://huggingface.co/docs/peft/index [Accessed 24 January 2024]

7. oneArena (2023). Flask App Deployment on Azure: GitHub CI/CD Tutorial. *YouTube*. Available from https://www.youtube.com/watch?v=dXZMbF07hDE&t=37s [Accessed 05 February 2023]

8. Noble Cause Software Development (2022). Use This Trick To Deploy React From Github To Azure! | React Explained | Episode - 33. *YouTube*. Available from https://www.youtube.com/watch?v=XqRpv-VknYY [Accessed 07 February 2023]

9. A Monk in Cloud (2022). MySQL Database on Azure | Create a MySQL database server under 10 minutes using Microsoft Azure. *YouTube*. Available from https://www.youtube.com/watch?v=O6tlkpFmZds [Accessed 14 February 2022]

Appendix A - Implementation

Appendix A.1 - Implementation of the front end component

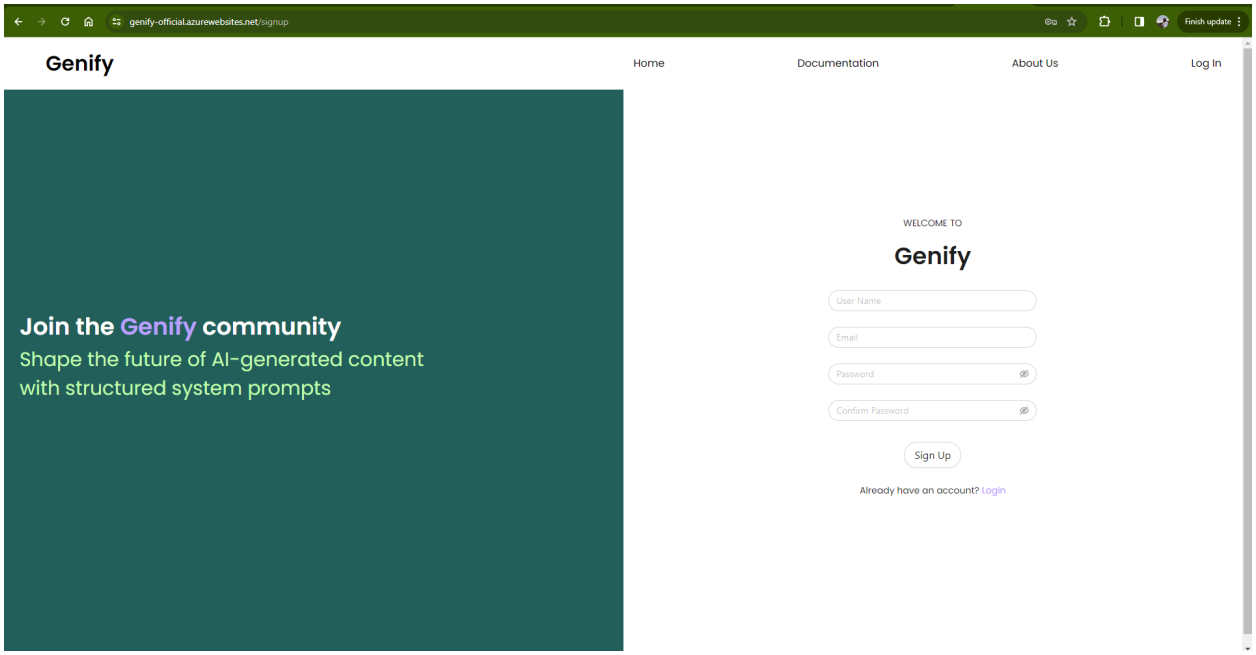


Figure 35: Frontend Sign Up page

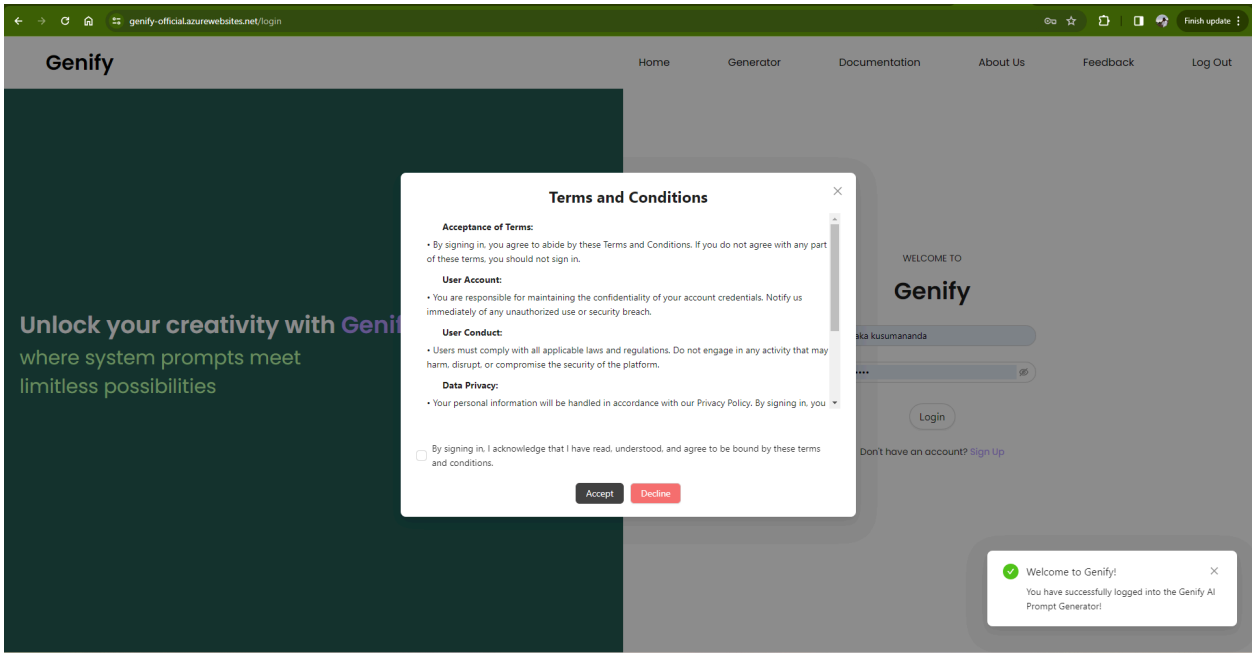


Figure 36: Frontend Terms and Conditions

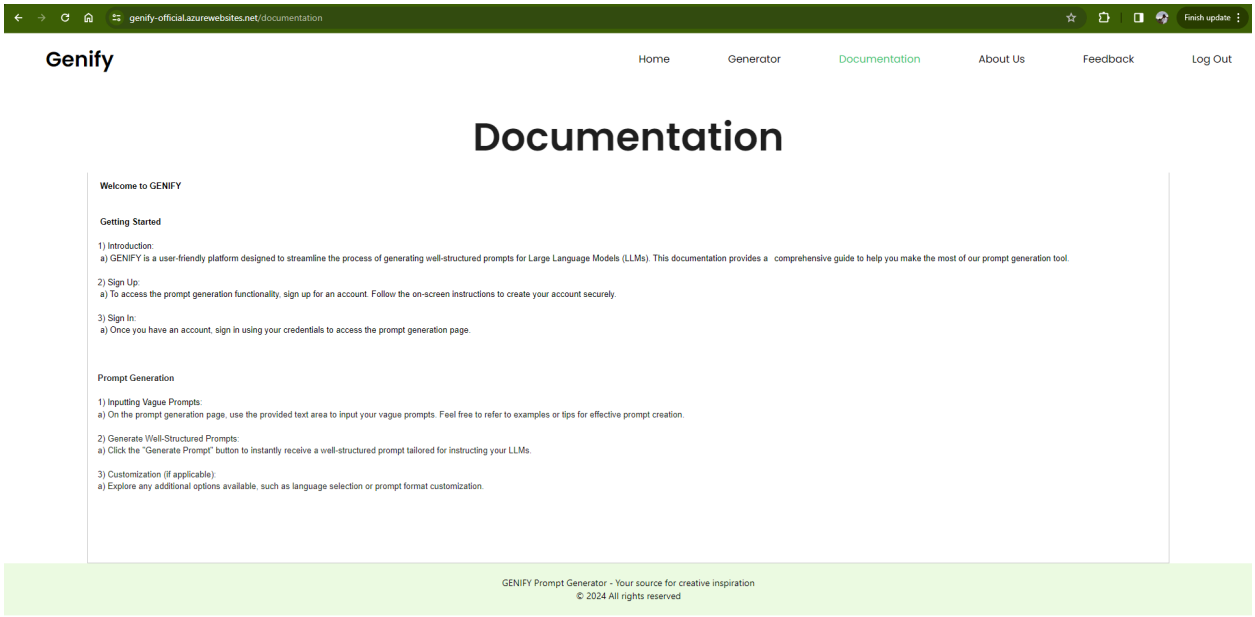


Figure 36: Frontend Documentation

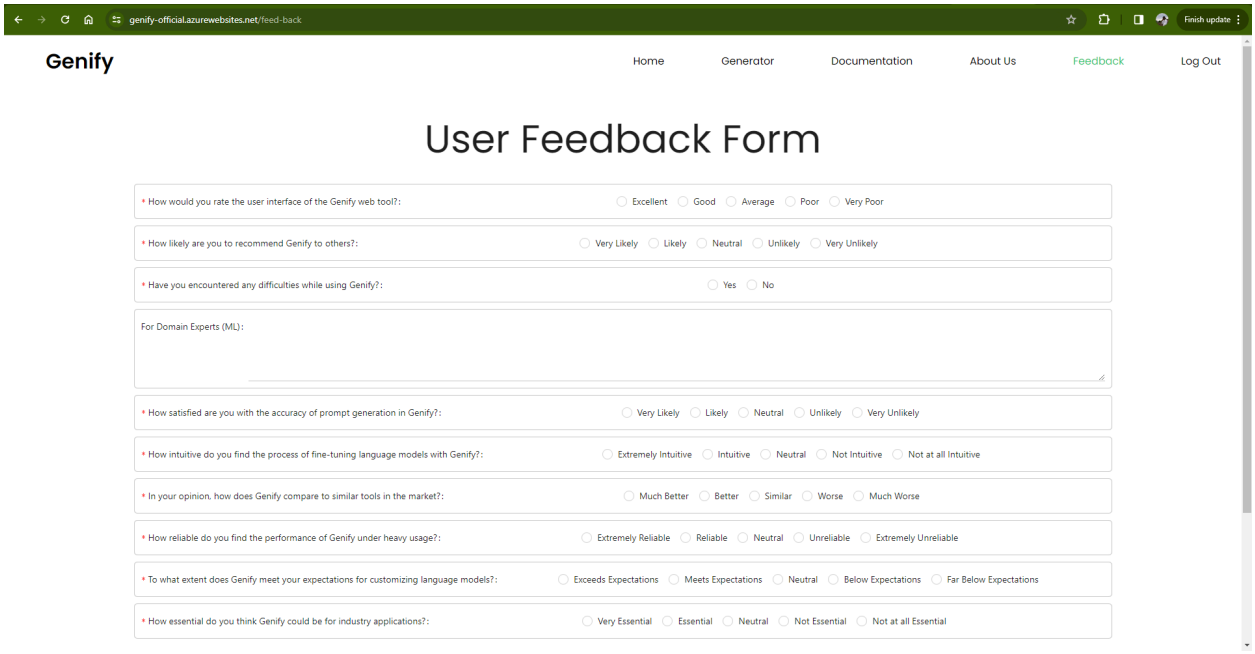


Figure 37: Frontend Feedback Form

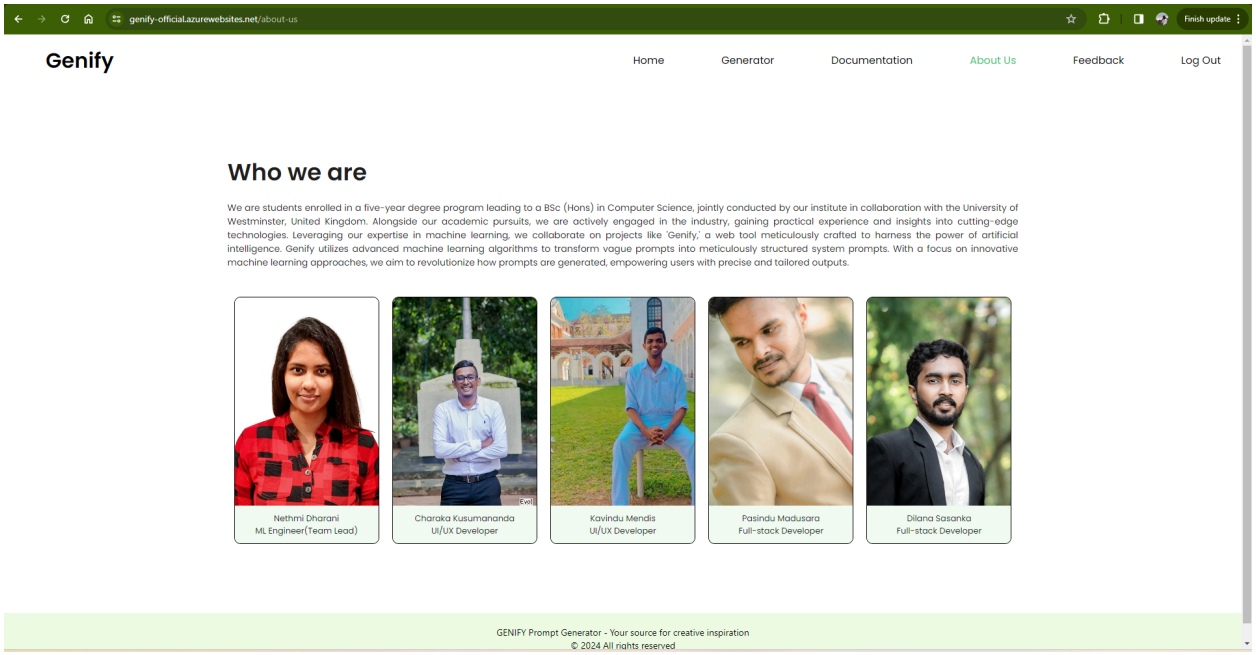


Figure 38: Frontend About Us

Appendix A.2 - GIT Repository

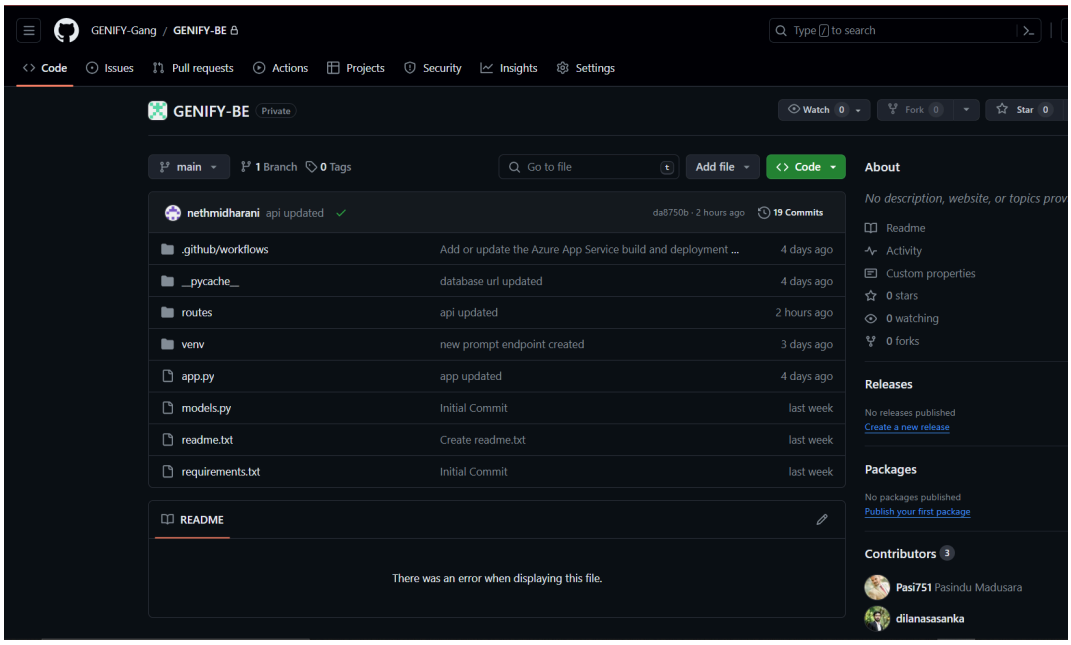


Figure 39: GIT Repository BE

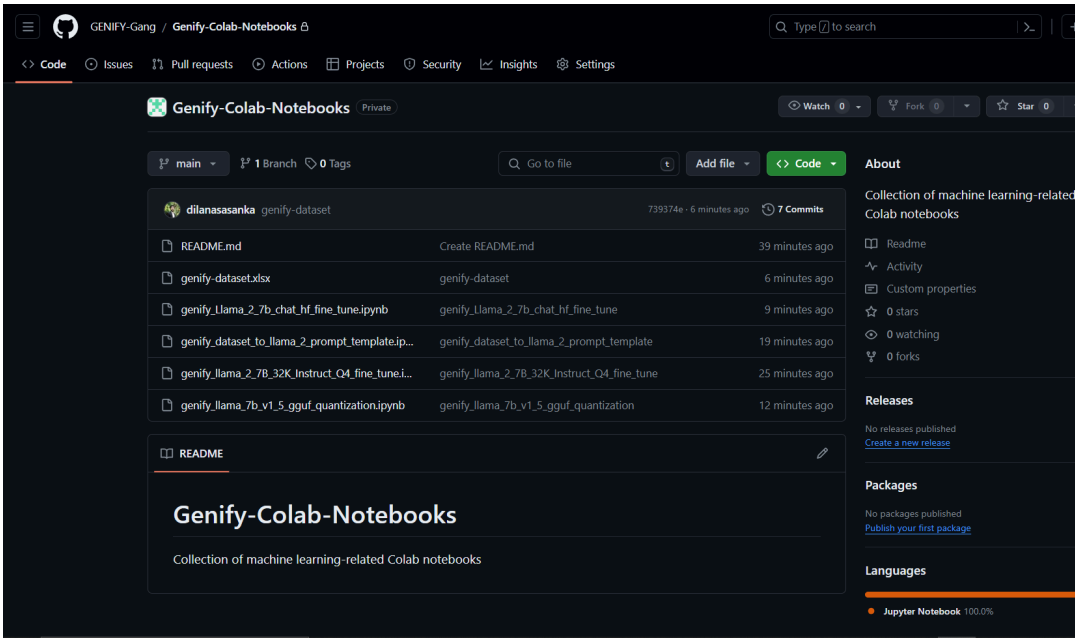


Figure 39: GIT Repository Genify Colab Notebooks

Appendix A.3 - Deployments/CI-CD Pipeline

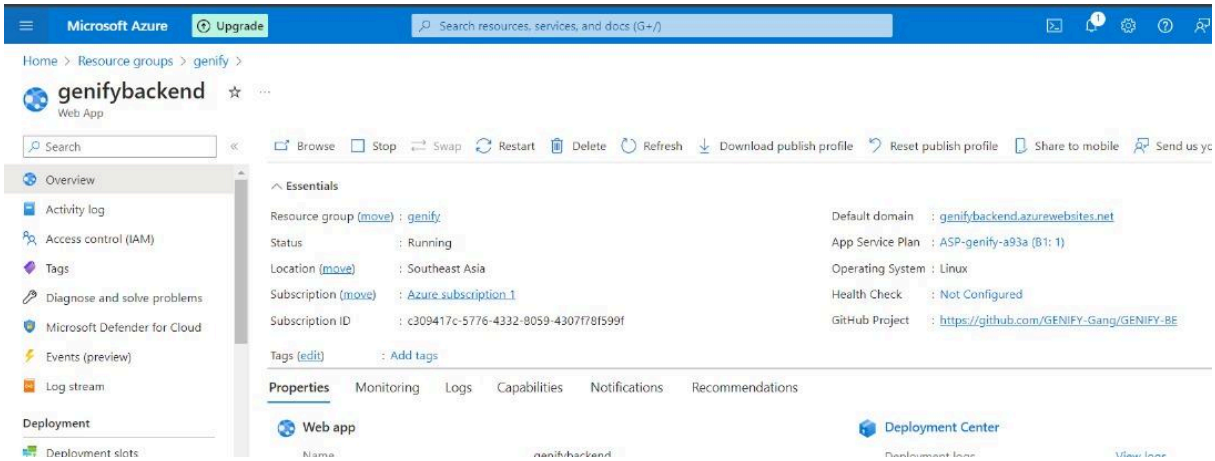


Figure 40: Azure deployment backend

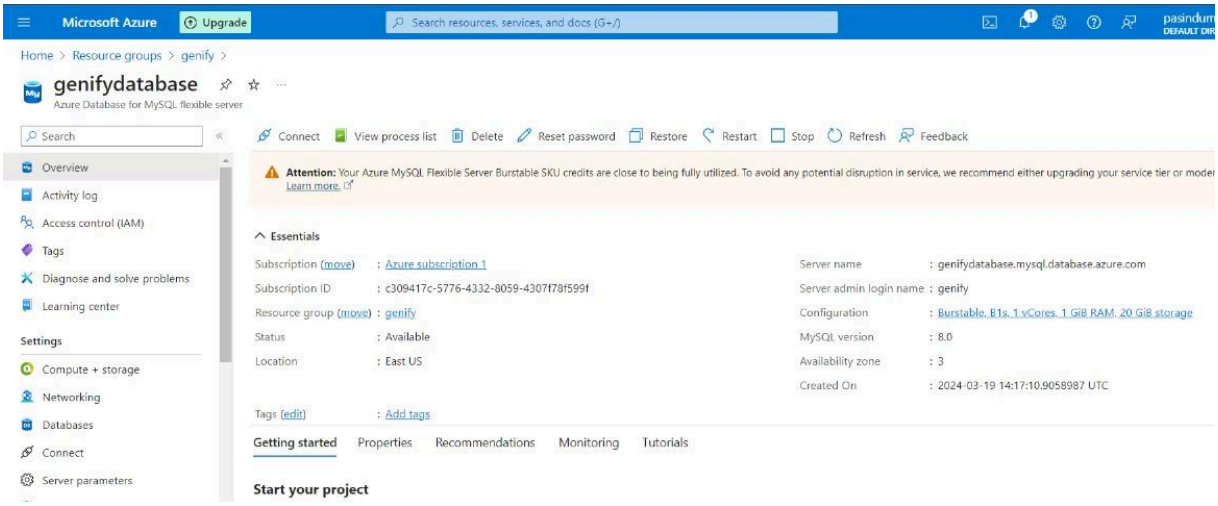


Figure 41: Azure deployment database

Appendix A.4 - CRUD operations

```python
@documentation_bp.route('/', methods=['GET'])
def get_documentation():
    documentation = Documentation.query.first()
    if documentation:
        return jsonify({"documentation": documentation.serialize()})
    else:
        return jsonify({"message": "Documentation not found"}), 404

@documentation_bp.route('/', methods=['PUT'])
def update_documentation():
    data = request.get_json()
    if 'content' not in data:
        return jsonify({"error": "Content not provided"}), 400

    documentation = Documentation.query.first()
    if documentation:
        documentation.content = data['content']
    else:
        documentation = Documentation(content=data['content'])
        db.session.add(documentation)

    db.session.commit()
    return jsonify({"message": "Documentation updated successfully."})

@documentation_bp.route('/', methods=['POST'])
def create_documentation():
    data = request.get_json()
    if 'content' not in data:
        return jsonify({"error": "Content not provided"}), 400

    documentation = Documentation(content=data['content'])
    db.session.add(documentation)
    db.session.commit()

    return jsonify({"message": "Documentation created successfully"})
```

Figure 42: Documentation CRUD

```python
@prompt_generator_bp.route('/generate', methods=['POST'])
def summarize():
    if request.method == 'POST':
        try:
            data = request.json
            input_text = data.get('input_text')
            if input_text:
                input = f"<s>[INST]<<SYS>>{sys_prompt}<</SYS>{input_text}[/INST]"
                output = query({
                                "inputs": input,
                                "parameters": {
                                    "max_new_tokens": 150
                                }
                            })
                return jsonify(output), 200
            else:
                return jsonify({'error': 'Input text is required.'}), 400
        except Exception as e:
            return jsonify({'error': str(e)}), 500
```

Figure 43: Prompt Generator CRUD

Appendix B - Testing

Appendix B.1 - Compatibility testing

No	Test Case	Expected Result	Actual Result	Status
1	Accessing Genify on Chrome	Genify interface displays consistently in Chrome browser	Genify interface displays consistently in Chrome browser	Passed
2	Accessing Genify on Firefox	Genify interface displays consistently in Firefox browser	Genify interface displays consistently in Firefox browser	Passed
3	Accessing Genify on Safari	Genify interface displays consistently in Safari browser	Genify interface displays consistently in Safari browser	Passed
4	Accessing Genify on Edge	Genify interface displays consistently in Edge browser	Genify interface displays consistently in Edge browser	Passed
5	Accessing Genify on desktop computers	Genify interface adjusts responsively on desktop computers	Genify interface adjusts responsively on desktop computers	Passed
6	Accessing Genify on laptops	Genify interface adjusts responsively on laptops	Genify interface adjusts responsively on laptops	Passed
7	Accessing Genify on smartphones	Genify interface adjusts responsively on smartphones	Genify interface adjusts responsively on smartphones	Passed

Table 4: Compatibility Testing

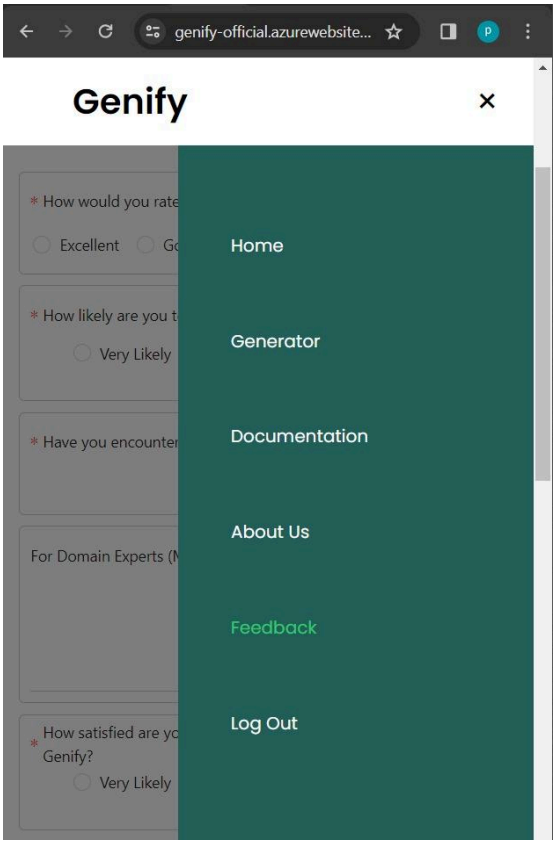


Figure 43: Responsive UI navbar

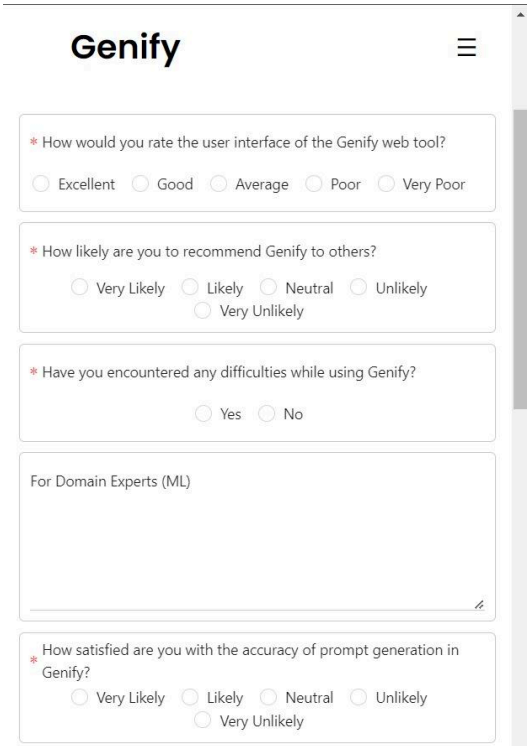


Figure 44: Responsive UI feedback

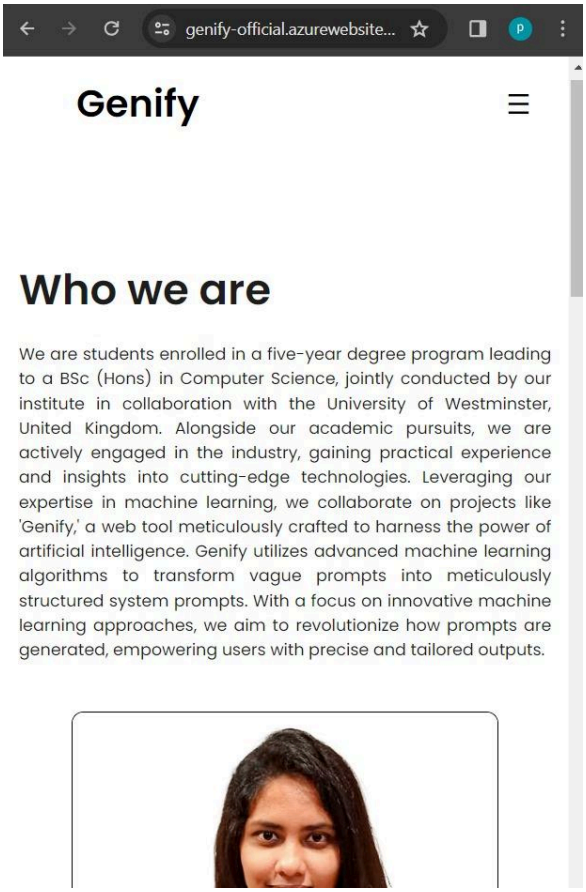


Figure 45: Responsive UI about us

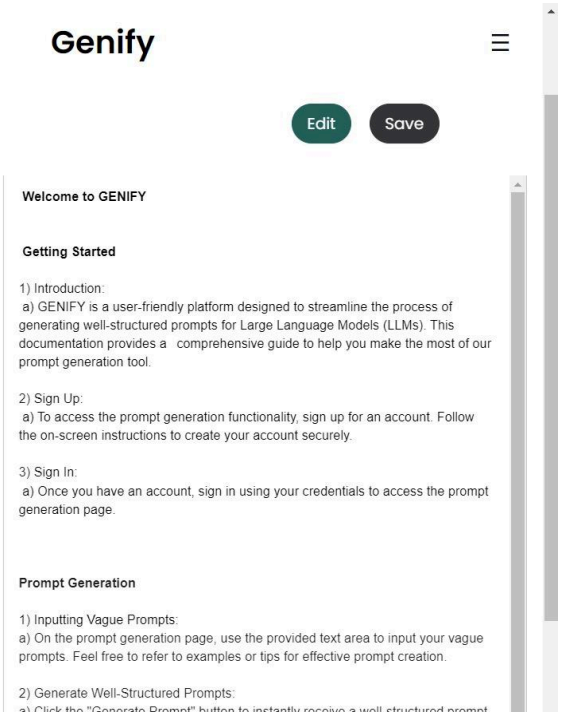


Figure 46: Responsive UI documentation

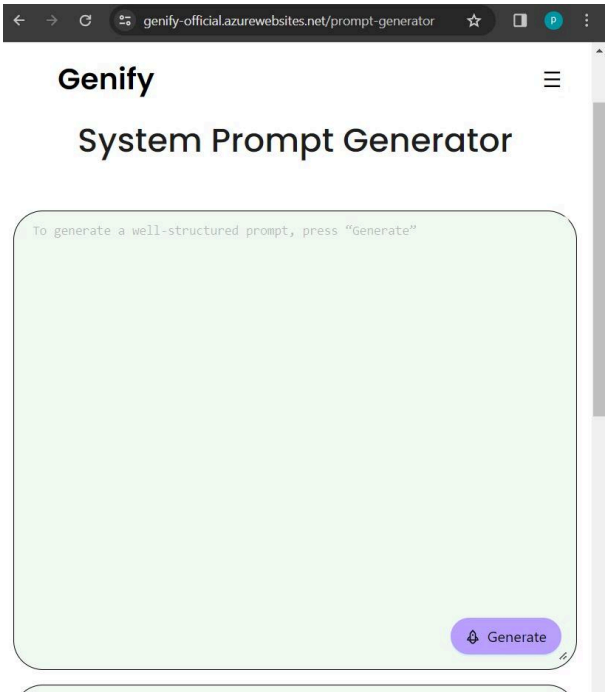


Figure 47: Responsive UI generator

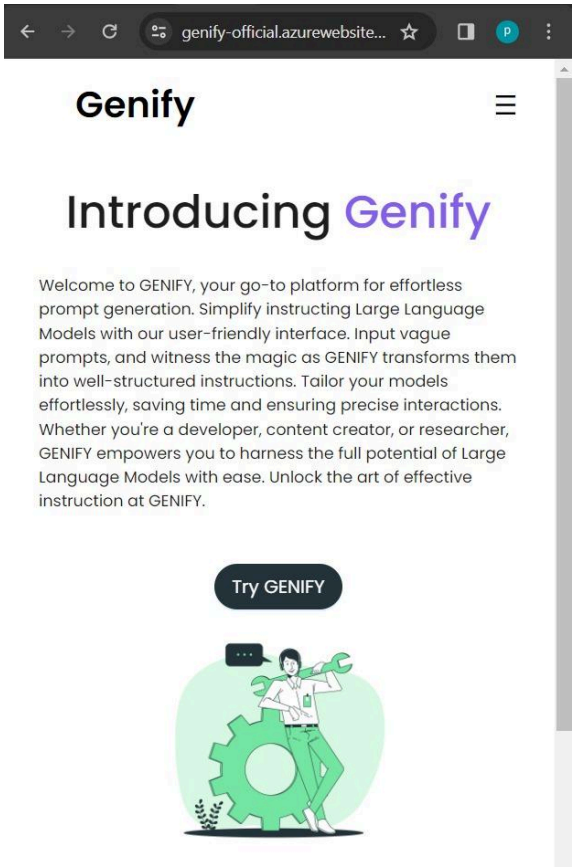
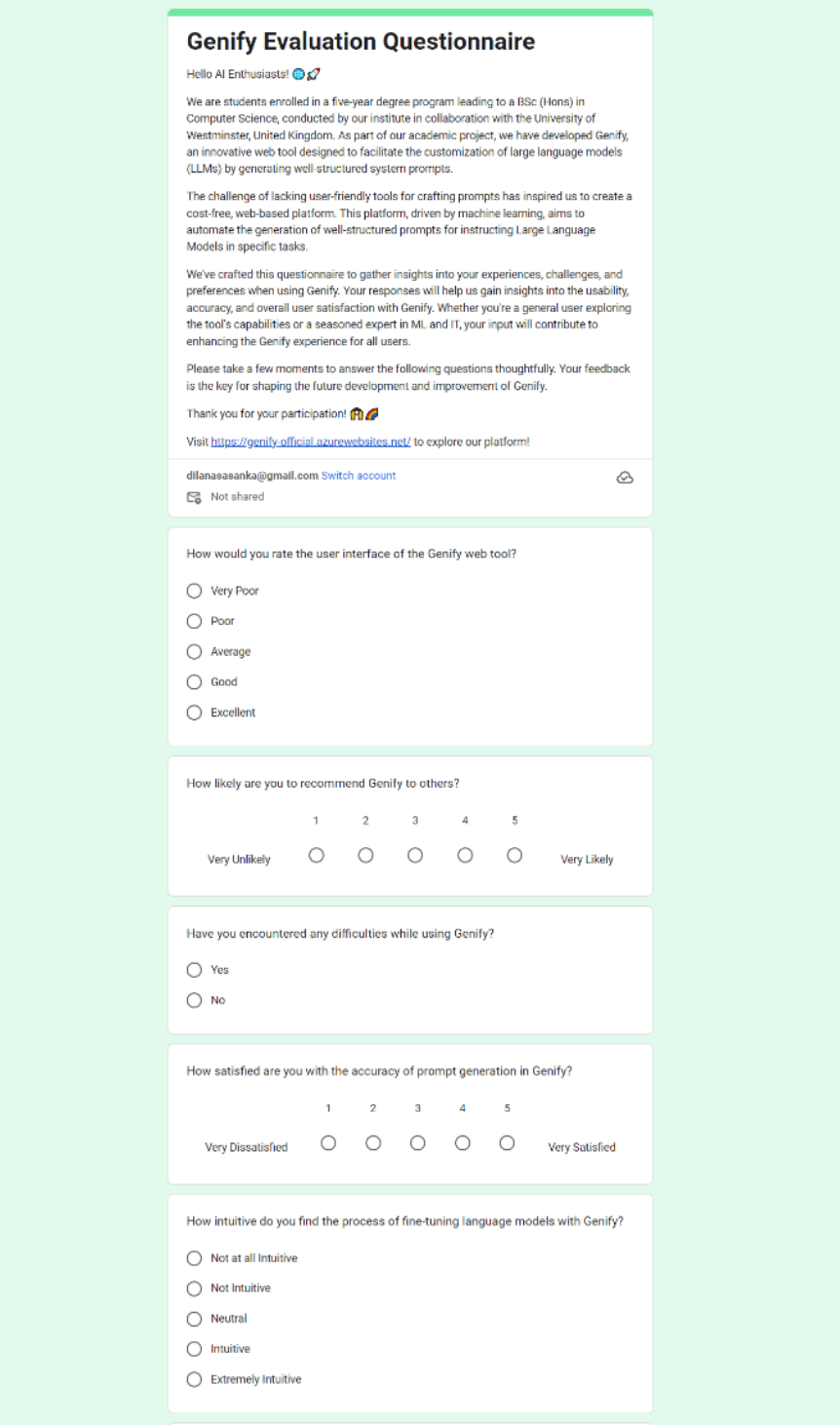


Figure 48: Responsive UI home

Appendix C - Evaluation

Appendix C.1 - Qualitative evaluation



Genify Evaluation Questionnaire

Hello AI Enthusiasts!

We are students enrolled in a five-year degree program leading to a BSc (Hons) in Computer Science, conducted by our institute in collaboration with the University of Westminster, United Kingdom. As part of our academic project, we have developed Genify, an innovative web tool designed to facilitate the customization of large language models (LLMs) by generating well-structured system prompts.

The challenge of lacking user-friendly tools for crafting prompts has inspired us to create a cost-free, web-based platform. This platform, driven by machine learning, aims to automate the generation of well-structured prompts for instructing Large Language Models in specific tasks.

We've crafted this questionnaire to gather insights into your experiences, challenges, and preferences when using Genify. Your responses will help us gain insights into the usability, accuracy, and overall user satisfaction with Genify. Whether you're a general user exploring the tool's capabilities or a seasoned expert in ML and IT, your input will contribute to enhancing the Genify experience for all users.

Please take a few moments to answer the following questions thoughtfully. Your feedback is the key for shaping the future development and improvement of Genify.

Thank you for your participation!

Visit https://genify-official.azurewebsites.net/ to explore our platform!

dilanasasanka@gmail.com Switch account

Not shared

How would you rate the user interface of the Genify web tool?

- Very Poor
- Poor
- Average
- Good
- Excellent

How likely are you to recommend Genify to others?

	1	2	3	4	5	
Very Unlikely	○	○	○	○	○	Very Likely

Have you encountered any difficulties while using Genify?

- Yes
- No

How satisfied are you with the accuracy of prompt generation in Genify?

	1	2	3	4	5	
Very Dissatisfied	○	○	○	○	○	Very Satisfied

How intuitive do you find the process of fine-tuning language models with Genify?

- Not at all Intuitive
- Not Intuitive
- Neutral
- Intuitive
- Extremely Intuitive

Figure 49: Evaluation questions

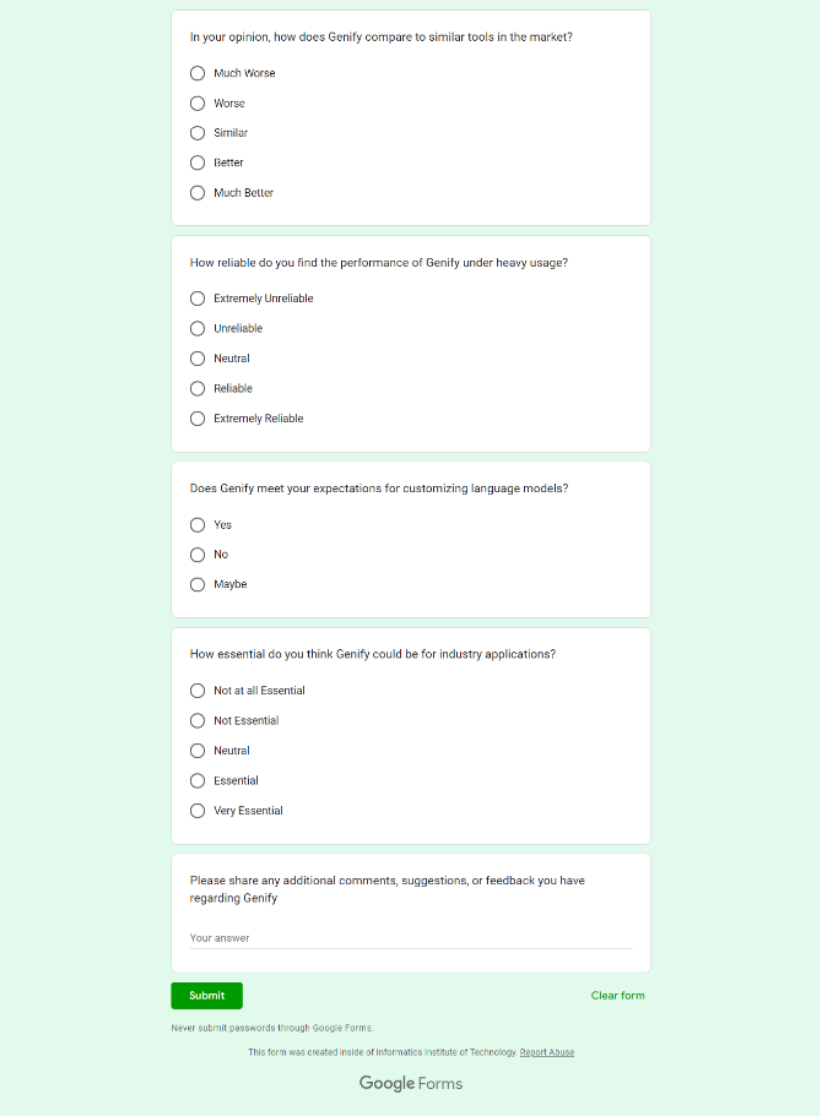
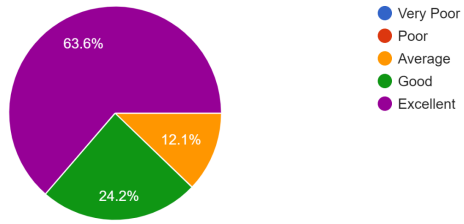
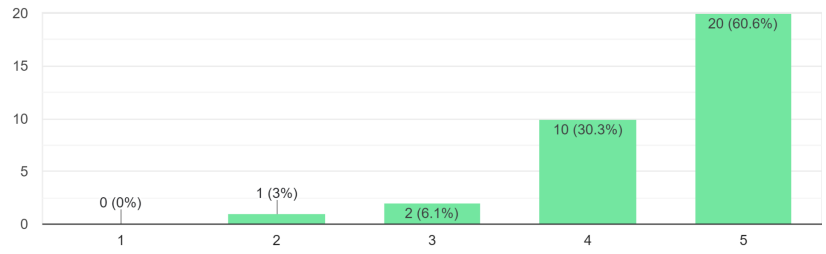
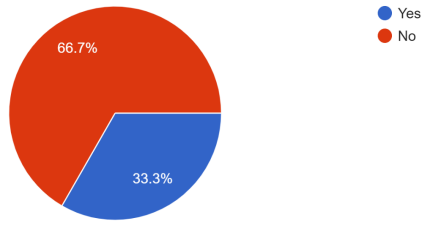
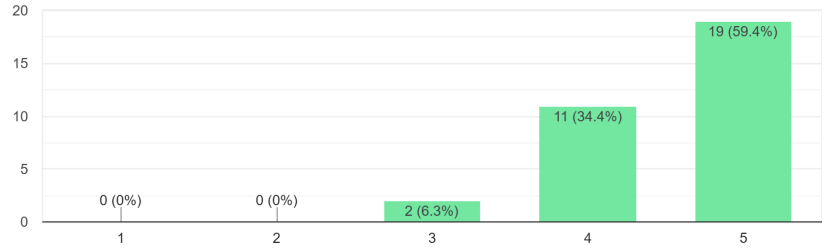
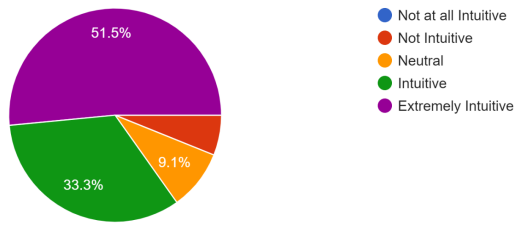
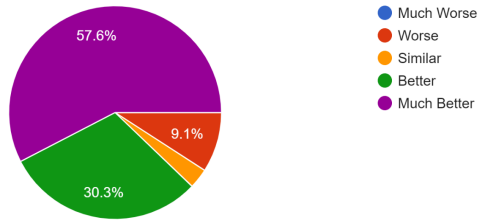
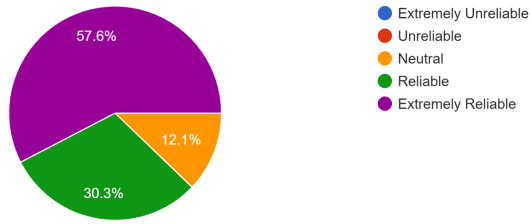
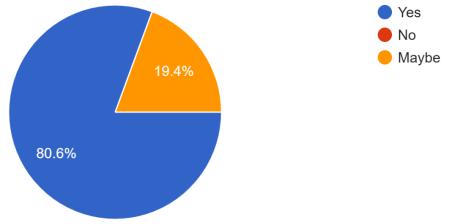
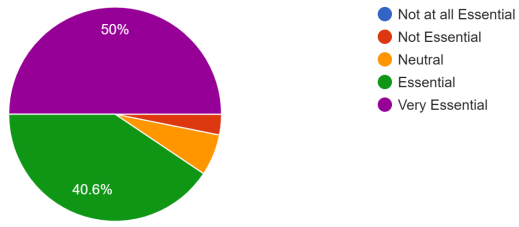


Figure 50: Evaluation questions 2

Questionnaire Result	Analysis
How would you rate the user interface of the Genify web tool? 33 responses  	This graph indicates that a significant majority of users found the interface to be highly satisfactory in terms of usability, design, and functionality

How likely are you to recommend Genify to others? 33 responses 	This graph indicates that users have useability from Genify and like to recommend it to others
Have you encountered any difficulties while using Genify? 33 responses 	This graph indicates that users have no difficulties using Genify.
How satisfied are you with the accuracy of prompt generation in Genify? 32 responses 	This graph indicates that the majority of users are satisfied with the accuracy of the prompt generated in Genify.
How intuitive do you find the process of fine-tuning language models with Genify? 33 responses 	This graph indicates that the majority of users are extremely intuitive of fine tuning language model

In your opinion, how does Genify compare to similar tools in the market? 33 responses 	This graph indicates that the majority of users' opinion of Genify is a much better tool
How reliable do you find the performance of Genify under heavy usage? 33 responses 	This graph indicates that the majority of users find that Genify performance under heavy usage if extremely reliable
Does Genify meet your expectations for customizing language models? 31 responses 	This graph indicates that the majority of users for Customizing language model says yes
How essential do you think Genify could be for industry applications? 32 responses 	This graph indicates that the majority of users very essential for genify be a indusruy application

Please share any additional comments, suggestions, or feedback you have regarding Genify 10 responses tasks or domains has been instrumental in my research projects, saving me considerable time and effort From an industry perspective, I see immense potential in Genify. It has applications across various sectors, including content generation, customer support automation, and natural language understanding tasks. I believe this tool could revolutionize the way businesses interact with language models, leading to more efficient and personalized user experiences. Well implemented. Good project! This will be a very useful tool. Nice Great work Best work for LLM modals	Users find that this web application is more reliable, it is useful for users, and its fine work.

Table 5: Questionnaire Results