Chapter 1

Introduction

Background and Motivation

The growing mental health awareness in recent times has not solved the problem of limited psychological support because of social stigma combined with financial obstacles and scarce qualified professionals. The digital transformation brings Artificial Intelligence (AI) at the forefront to resolve accessibility challenges through its provision of accessible non-bias mental support available at all times. The Artificial Intelligence system known as Virtual psychologists provides users access to a safe digital environment to share emotions while directing members to appropriate mental health services through their basic psychological assistance.

The creation of Large Language Models particularly GPT along with LLaMA has entirely transformed natural language understanding through human-like emotional interaction capabilities. The project uses modern technological breakthroughs to develop a virtual therapeutic platform which replicates appropriate empathic mental health interactions.

1.1 Problem Definition

The worldwide availability of affordable prompt service in mental health treatments continues to be a persistent issue. The availability of therapy combined with counseling remains effective but many people refrain from help because of public shaming concerns or payment difficulties or work restrictions. Most chatbots today do not grasp emotional context sufficiently well nor provide adequate empathetic responses which leads users to unhappy interactions. The project targets virtual psychologist development for emotional understanding combined with empathetic responses and highly responsive psychological support which maintains user safety alongside privacy protection.

1.2 Objective of Project

- Rephrase the task of building an empathetic conversational artificial intelligence system which conducts psychological support.
- The implementation of a process to refine a large language model by applying emotion-labeled and therapy-oriented datasets.
- The system needs capability to process emotions and sentiment through context-based response creation.
- Universal safety and non-offending user interactions will be achieved by using proper prompt engineering and content filtering solutions.
- The system requires storage space for keeping and monitoring dialogue logs which will be used for session preservation and user reflection while maintaining privacy standards.
- We will refine a large language model through the use of emotion-labeled along with therapy-oriented datasets.

- The solution includes detecting emotional markers inside the content to generate responses that match the context.
- The system enables safe communication by using ethical prompt engineering techniques and content filters.
- The system needs technology to manage conversation logs for maintaining session continuity while still respecting user privacy rules.

1.3 Scope of Project

- This system exists as an AI-run assistance tool that supplements therapy but does not function as a replacement. It provides:
- Basic emotional support through conversations.
- Mood tracking and journaling prompts.
- The program automatically leads users to emergency assistance when it identifies potential risks.
- A web-based together with application-based interactive interface connects users.
- The system functions strictly as a conversation tool that neither evaluates nor cures mental health disorders or delivers medical counsel to patients.

Chapter 2

Literature Survey

Artificial Intelligence (AI) technology specifically Large Language Models (LLMs) strengthens mental health support by providing virtual psychological agents to users. Modern AI technology represents an alternative approach to therapy which provides automated emotional understanding during unlimited availability for people experiencing psychological struggles. The review consolidates significant research findings from studies published between 2023 and 2025 about how empathy modeling meets AI safety requirements when used for context-sensitive mental health application communication techniques.

2.1 Review of Related Work

The development of virtual psychologists through AI chatbots aims to address the critical shortage of accessible mental health services while overcoming barriers such as stigma, cost, and privacy concerns. These chatbots uses advanced Natural Language Processing (NLP) and machine learning techniques to provide empathetic responses and emotional understanding, thereby enhancing user interactions and support.

2.2 Emotional Understanding and Empathy

Chatbots like “BlissBot” and the SAAC Mental Health Chatbot utilize NLP to interpret user emotions, allowing for tailored responses that resonate with users’ feelings [1, 2]. Techniques such as sentiment analysis and emotion recognition are employed to categorize user inputs, ensuring that the chatbot can respond appropriately to various emotional states [5, 3].

2.3 Accessibility and Privacy

These AI-driven solutions offer 24/7 support, making mental health resources available at any time, which is crucial for individuals hesitant to seek traditional therapy due to societal stigma [4, 2]. Privacy is a priority, with chatbots designed to maintain confidentiality and secure user data, fostering a safe environment for open discussions [1, 5].

2.4 Limitations and Challenges

Despite the advancements, challenges remain in ensuring that chatbots can fully replicate the nuanced understanding of human therapists. Users may still prefer face-to-face interactions for complex emotional issues, highlighting the need for a hybrid approach that combines AI support with human oversight.

Chapter 3

System Design and Architecture

The Virtual Psychologist system is a conversational AI chatbot designed to provide emotionally aware, context-driven support to users seeking mental health-related dialogue. The core engine is built on a fine-tuned **LLaMA 3.2B** parameter model that enables natural, empathetic interactions. A defining feature of this system is its ability to **retain conversational context** across sessions, simulating the memory of a human counselor. While it does not perform sentiment or emotion classification, it prioritizes safe, structured, and context-aware responses.

3.1 User Flow and Conversation Pipeline

The following steps describe the chatbot interaction flow:

1. **User Message Input**: The user sends a message to the chatbot through the interface.
2. **Context Retrieval**: The system retrieves recent conversation history from stored session logs to provide conversational continuity.
3. **Prompt Construction**: A prompt is dynamically constructed using the current user message and relevant prior turns.

4. **LLM Response Generation**: The combined prompt is sent to the fine-tuned LLaMA model to generate a coherent, natural-sounding response.
5. **Safety Filtering**: The generated response is checked for unsafe or inappropriate content.
6. **Response Output**: The response is returned to the user in real-time.
7. **Session Logging**: The latest message pair (user + AI) is saved for future reference.

3.2 Module Details

3.2.1 LLM-Based Response Generator (LLaMA 3.2B)

At the heart of the system is the **LLaMA 3.2B parameter model**, fine-tuned for mental health-style dialogue. The model is trained to:

- Understand conversational context.
- Generate human-like, supportive responses.
- Maintain a neutral and empathetic tone across multiple exchanges.

The prompt fed into the model includes both the user's latest message and selected prior interactions, ensuring the system responds contextually, not just reactively.

3.2.2 Context Memory / Session Tracker

To simulate therapist-like continuity, the system maintains a **session history**. This module:

- Stores each user-bot exchange in a chronological format.

- Selects relevant recent turns to append to the current prompt.
- Ensures the model can reference earlier topics or concerns in the conversation.

This context tracking is crucial for building trust, reducing repetition, and making the conversation feel more personal and persistent.

3.2.3 Safety Filter & Escalation Protocol

Mental health discussions often touch on sensitive issues. To handle this responsibly, the system includes a **safety filter** that:

- Flags critical phrases using keyword matching (e.g., "I want to hurt myself").
- Filters or rewrites risky outputs using post-processing.
- Triggers escalation prompts such as offering crisis helpline numbers or encouraging users to seek professional help.

This filter ensures the chatbot adheres to ethical boundaries and avoids generating harmful content.

3.3 Tech Stack and Tools Used

Component	**Technology Used**
Language Model	LLaMA 3.2B (fine-tuned using LoRA/QLoRA)-Hugging Face & Meta
Backend	Python
Prompt Management	Transformers, PEFT, HuggingFace
Logging and Storage	JSON/CSV file-based session logs
Deployment Environment	Google Colab
Safety & Filtering	Custom regex-based rule engine

Table 3.1: Technologies and Tools Used

Chapter 4

Dataset

Overview

This project uses a custom-designed dataset which exists to prepare models for psychiatric case note generation in doctor-patient dialogues. The dataset required a nonpublic access to real doctor-patient transcripts because of privacy concerns therefore enacted dialogues based on real and adapted scripts were used to create the dataset.

Eight trained individuals staged doctor-patient sessions through scripted dialogues which combined hand-written scripts and modified Alexander Street transcripts. Automated recording of doctor-patient discussions generated both audio and text documentation for training and evaluating psychological conversational AI systems.

4.1 Data Collection Methodology

- **Script Sources:**
  - Hand-written scripts by Cheryl Bristow
  - Real patient-doctor transcript adaptations from Alexander Street
- **Voice Recordings:** Performed by pairs of students acting as doctors and

patients. Each session was recorded with separate audio channels for each speaker.

- **Transcription:** Audio recordings were transcribed using Google Cloud Speech-to-Text API.
- **Case Notes:** Annotators (students) used these transcripts to write formal psychiatric case notes, without any training or guided instructions.

4.2 Data Structure

4.2.1 Transcript Files

Transcripts are stored in JSON format, where each conversation is represented as an array of dialogue turns:

```
[
  {
    "speaker": 1,
    "dialogue": ["How are you feeling today?", "Are you overloaded with
   the work stress?"]
  },
  {
    "speaker": 2,
    "dialogue": ["Yes somewhat. These days are tough"]
  }
]
```

Field Definitions:

- `speaker`: Denotes the speaker (1 for doctor, 2 for patient).
- `dialogue`: A list of sentences spoken in a single turn.

4.2.2 Case Note Files

Each transcript is paired with case notes written by annotators. The notes reference sentences from the transcript and assign them to clinical categories.

```
[
  {
    "categoryId": "2",
    "sourceId": "7",
    "formalText": "The patient reports frequent anxiety and poor sleep."
  }
]
```

Field Definitions:

- `categoryId`: ID corresponding to one of the psychiatric note categories.
- `sourceId`: Refers to the index of the sentence in the transcript used to derive the case note.
- `formalText`: Paraphrased, clinically appropriate sentence written by the annotator.

4.2.3 Case Note Categories

Index	Category	Abbr.
0	Client Details	CD
1	Chief Complaint	CC
2	History of Present Illness	HPI
3	Past Psychiatric History	PPH
4	History of Substance Use	HSU
5	Social History	SH
6	Family History	FH
7	Review of Systems	RS

Table 4.1: Categories used for case note annotations [6]

4.3 Directory Structure

- `transcripts/source`: Original scripts used to create audio.
- `recordings`: Audio files of enacted conversations.
- `transcripts/transcribed`: Transcriptions generated via Google STT API.
- `casenotes`: Annotated formal case notes.
- `dataset.pickle`: Serialized file combining transcripts and notes for quick access in Python.

4.4 Licensing and Access

The dataset is publicly available under a **Creative Commons Attribution 4.0 International License**. It is intended for academic and research purposes only. Some transcripts were adapted from copyrighted materials with permission or under fair use.

4.5 Applications in Virtual Psychologist

This dataset serves as a training and evaluation backbone for the Virtual Psychologist system:

- It allows modeling of psychologically grounded dialogue.
- Case notes serve as structured outputs for clinical summarization tasks.
- Transcripts help in fine-tuning LLMs to understand therapy-style conversational flow.

Chapter 5

Implementation of Finetuning

This chapter elaborates on the implementation of finetuning the meta-llama/Llama-3.2-1B-Instruct model using parameter-efficient LoRA techniques. The goal was to customize the model for a virtual psychologist chatbot, trained on a conversational dataset. The implementation was carried out in Google Colab, leveraging GPU acceleration and efficient 4-bit quantization.

5.1 Environment Setup

The following packages were installed to support quantized finetuning:

```
!pip install torchtune peft bitsandbytes transformers auto-gptq optimum
   torch
!pip install huggingface_hub
```

These tools enabled:

- 4-bit quantized model loading via `bitsandbytes`
- LoRA finetuning with `peft`
- Dataset handling using `datasets`
- Model utilities from `transformers` and `optimum`

5.2 Model Loading

The base model used was Meta's LLaMA 3.2B Instruct. It was loaded with GPU support:

```
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-3.2-1B-
    Instruct").to(device)
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-3.2-1B-
    Instruct")
```

The model was switched to training mode, and prepared for LoRA using gradient checkpointing:

```
model.train()
model.gradient_checkpointing_enable()
model = prepare_model_for_kbit_training(model).to(device)
```

5.3 LoRA Configuration

LoRA was configured using the following:

```
config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["q_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, config)
model.print_trainable_parameters()
```

Detailed Explanation of LoRA Parameters

- `r = 8` — **LoRA Rank**

  This refers to the **rank of the low-rank decomposition matrices** used in LoRA. Instead of updating the full weight matrices of a transformer layer (which is computationally expensive), LoRA introduces two smaller matrices $A \in R^{d \times r}$ and $B \in R^{r \times d}$. The update becomes:

  $$W' = W + \alpha AB$$

  A lower value of r keeps the model lightweight, while a higher r increases expressiveness but also compute cost. With $r = 8$, this configuration balances performance and efficiency.

- `lora_alpha = 32` — **Scaling Factor**

  `lora_alpha` is a scaling constant that adjusts the magnitude of the LoRA update. It rescales the low-rank product:

  $$W' = W + \frac{\alpha}{r} AB$$

  For $\alpha = 32$ and $r = 8$, the effective scale is 4. This ensures that the low-rank contribution is strong enough without overwhelming the pretrained weights.

- `target_modules = ["q_proj"]` — **Target Modules**

  This defines where LoRA adapters are inserted. Here, `q_proj` refers to the **query projection layer** of the attention mechanism. Limiting LoRA to this layer:

  - Reduces memory usage,
  - Focuses improvements on query representation,
  - Keeps fine-tuning lightweight.

- `lora_dropout = 0.05` — **Regularization Dropout**

  This applies a dropout probability of 5% to the LoRA outputs during training.

It serves as a regularization method to avoid overfitting, especially beneficial for smaller datasets.

- `task_type = "CAUSAL_LM"` — **Task Type**
  This parameter specifies that the fine-tuning task is **causal language modeling** (CLM), where the model predicts the next token in a sequence. It ensures that LoRA adapts only modules relevant to CLM, like the decoder's self-attention layers.

5.4 Dataset Preparation

The JSON dataset contained conversations between doctors and patients.

Splitting and Flattening

```
split_dataset = dataset['train'].train_test_split(test_size=0.1, seed
    =42)
```

Nested conversations were flattened:

```
text = ""
for pair in item["conversations"]:
    text += f"{pair['role'].capitalize()}: {pair['content']}\n"
```

5.5 Tokenization

The tokenizer was updated and used with truncation and padding:

```
tokenizer.pad_token = tokenizer.eos_token
```

Tokenization function:

```
def tokenize_function(examples):
    ...
    return tokenizer(
        texts,
        return_tensors="np",
        padding=True,
        truncation=True,
        max_length=512
    )
```

5.6 Data Collator

```
data_collator = transformers.DataCollatorForLanguageModeling(tokenizer,
   mlm=False)
```

5.7 Training Configuration

```
training_args = TrainingArguments(
    output_dir="/content/drive/MyDrive/Finetuning/shawgpt-ft",
    learning_rate=2e-4,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=10,
    weight_decay=0.01,
    logging_strategy="epoch",
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    gradient_accumulation_steps=4,
    warmup_steps=2,
```

```
    fp16=True,
    optim="paged_adamw_8bit",
)
```

5.8 Training Execution

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator
)
trainer.train()
```

Model was saved using:

```
trainer.save_model("/content/drive/MyDrive/FineTunedModels/final-shawgpt
    ")
tokenizer.save_pretrained("/content/drive/MyDrive/FineTunedModels/final-
    shawgpt")
```

5.9 Inference and Chatbot Integration

Model was loaded for inference:

```
pipe = pipeline("text-generation", model=model1, tokenizer=tokenizer,
    device=0)
```

Prompt example:

```
"""<|begin_of_text|>... Hello Doctor, I am Nancy ...<|eot_id|>"""
```

A loop allowed real-time chat interaction with the finetuned model.

Chapter 6

Summary and Conclusion

Summary

This implementation enables:

- Efficient LoRA-based training
- 4-bit quantization
- Instruction-style dialogue modeling
- Real-time chatbot integration

This optimized setup provides a highly capable, domain-specific virtual psychologist chatbot using minimal resources.

6.1 Results and Outputs

The Front-End of the project was built using Streamlit platform. The model was trained in Google Colab browser IDE. The resources used were runtim: T4- GPU on Google Colab.

To make the application hosted ngrok was used to make a tunnel from google colab to the streamlit public host.

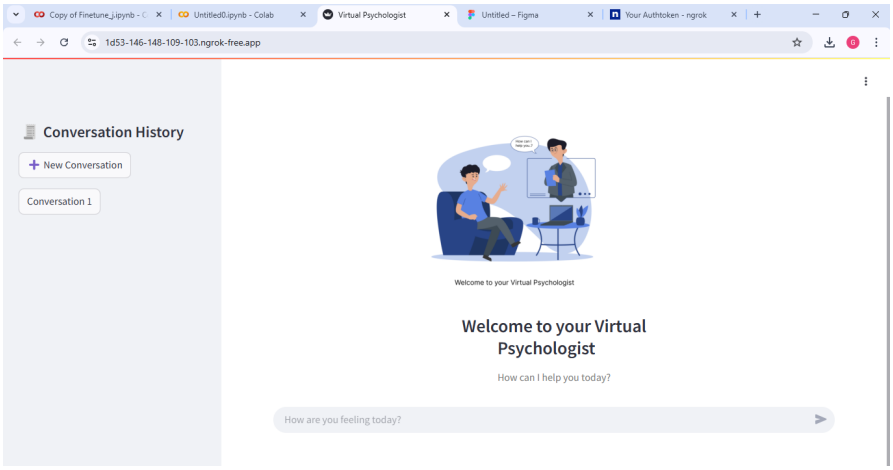

Figure 6.1: Empty State

The above image shows the empty state of the application. When no conversation is done, The empty state has the input text field and a tab having all the conversation history from when users can view the previous conversations.

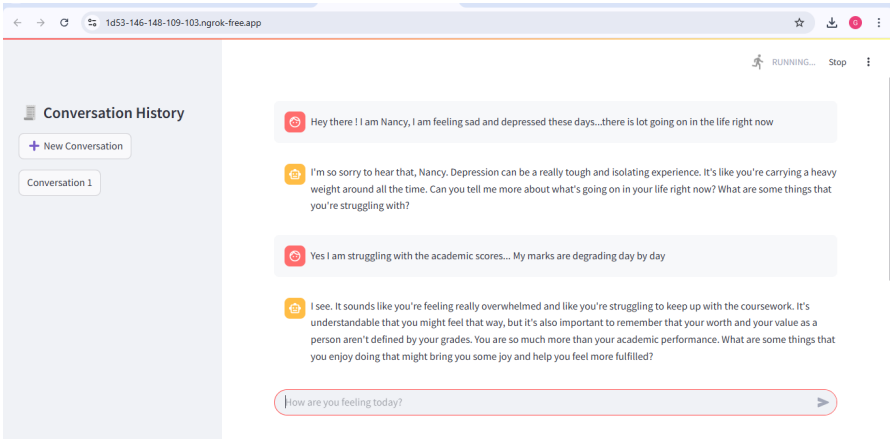

Figure 6.2: Default State

The above image shows the default and the normal state of the application.The Chatbot UI is very simple and it's inspiration is taking from what's app desktop UI. Which makes it more convenient for user to go through with easy navigation that enhances the user experience

6.2 Conclusions

Using LLMs in sensitive problem definition can be sensitive and very sensitive and need to be more precise and less recall. LLMs can fail and misguide the patient hence, high precision is required when dealing with the sensitive problem definition like mental health.