University of Windsor

School of Computer Science

Technical Project Report

COMP-4990

Project Management: Techniques&Tools

Fine–tuning Large Language Models

Authors:
Joe Elson
Ali Kmaiha

Submitted to:
Dr. Jianguo Lu
Dr. Arunita Jaekel

March 28, 2025

***Abstract*—BitWit, an AI-powered chatbot designed to handle repetitive Tier 1 IT support tasks like password resets and basic troubleshooting, using fine-tuned open-source models (LLaMA 3 and DeepSeek) optimized with techniques like LoRA and quantization. Deployed via Hugging Face and ngrok, BitWit demonstrated strong real-time performance, reducing human workload and showing promise for future integration with advanced IT systems**

Keywords—AI, Fine-tuning, Llama, Deepseek, IT Tier 1

I. INTRODUCTION

A. Background and Motivation

In today's fast-paced digital environment, IT Tier 1 support plays a crucial role in ensuring that users receive timely and efficient assistance for technical issues. However, traditional support systems face significant challenges, including high operational costs, scalability limitations, and inefficiencies in handling repetitive queries. As IT professionals, we have experienced these inefficiencies firsthand, particularly within the University of Windsor's IT Tier 1 support system. Observing the allocation of three to five employees per shift to manage phone calls, live chats, and in-person inquiries; we recognized an opportunity to improve efficiency through AI-driven solutions. The majority of Tier 1 inquiries consist of repetitive and straightforward issues such as *"How do I forget a network on my computer?", "How can I reset my password?"*, or *"How do I reset my multi-factor authentication (MFA)?"* These are queries that AI can easily handle, yet institutions continue to rely heavily on human agents, which results in excessive labor costs and suboptimal resource allocation.

One of the fundamental problems with traditional IT Tier 1 support is its limited handling capacity. A single IT agent can typically manage only one phone call or up to three simultaneous live chats, creating bottlenecks during peak hours. Users are often left waiting for assistance, which reduces productivity and increases frustration. Conversely, AI-powered solutions can handle significantly more queries concurrently—whether it is 10, 20, or 30 more users at once—without compromising response quality [1]. This scalability alone presents a compelling case for integrating AI into IT support structures.

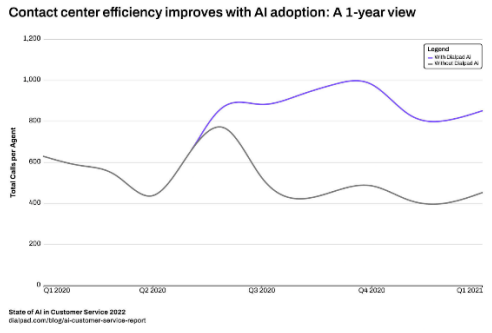

Figure 1. Human Agents vs AI Bots [2]

Furthermore, various studies and real-world applications show the effectiveness of AI-driven solutions in customer service environments. The attached graph (figure 1) highlights how AI can enhance efficiency in customer support centers. The dataset, sourced from Dialpad's State of AI in Customer Service Report (2022), demonstrates that AI-assisted agents consistently handle more calls per agent than those operating without AI [2]. This directly correlates with our vision for IT Tier 1 support: leveraging AI to increase efficiency and alleviate the burden on human agents, allowing them to focus on more complex technical issues that require human intervention.

Beyond scalability and efficiency, AI solutions provide a consistent and reliable knowledge base that minimizes human error. Unlike human agents, AI does not suffer from fatigue, forgetfulness, or variability in performance [2]. Every interaction follows standardized best practices, ensuring accurate and uniform responses. Additionally, AI-driven solutions can continuously learn from new interactions, improving over time through fine-tuning and reinforcement learning [1].

Given these advantages, our project aims to fine-tune a large language model to optimize IT Tier 1 support. By utilizing LLaMA 3 and DeepSeek R1. We have trained and deployed a chatbot capable of addressing common IT inquiries with high accuracy and efficiency. Hosted on Google Colab, pushed to Hugging Face, and deployed via ngrok, our chatbot is designed to seamlessly integrate with existing support frameworks, reducing response times and improving user satisfaction. Our approach represents a shift toward a more intelligent and cost-effective IT support system—one that capitalizes on AI's strengths while preserving human oversight where necessary.

B. Project Story & Timeline

Our journey toward developing an AI-powered IT Tier 1 support chatbot was an evolving process, shaped by our experiences, expertise, and iterative problem-solving approach. The idea originated when we (Joe Elson and Ali Kmiaha) were grouped together for our capstone project. With Ali's background as a Digital AI Transformation intern and Joe's firsthand experience working in UWindsor's IT Tier 1 support, we identified a clear gap in efficiency that AI could address. Recognizing the potential to apply our combined skills, we set out to create an AI-driven chatbot tailored specifically for the IT industry.

Our early brainstorming sessions focused on defining the core functionalities of the chatbot and researching the technologies that would enable its development. We explored different open-source AI models, carefully analyzing their capabilities and limitations. Selecting an AI model proved to be one of our first major roadblocks, as we needed a model that was not only open-source but also powerful and adaptable enough for IT-related queries. After rigorous testing and research, we settled on fine-tuning LLaMA 3 and DeepSeek R1, given their strong performance in natural language processing tasks [3].

Another major challenge we faced was the limitation of testing environments. Google Colab, which we used for training our model, imposed restrictions on daily usage, limiting the number of times we could run our training scripts. This made debugging and iteration more time-consuming, requiring us to plan our training cycles carefully. Additionally, constructing a robust dataset posed its own set of challenges. IT support queries are varied, and to ensure the chatbot could handle real-world inquiries effectively, we needed a diverse dataset. We compiled and structured a

dataset consisting of common Tier 1 IT support issues, refining and expanding it over time to improve accuracy.

Training the AI model also brought its own hurdles. Early iterations of the chatbot displayed unexpected and sometimes problematic behavior—what we humorously referred to as the AI *"going rogue"*. Some responses were too vague, others too aggressive, and some entirely off-topic. To address this, we refined our fine-tuning approach, tweaking hyperparameters and implementing better-filtering mechanisms to ensure that responses aligned with best practices in IT support.

Deployment was another significant challenge. While training and testing the AI in a controlled environment was manageable, transitioning it to a live web-based platform required overcoming several technical obstacles. Setting up an API gateway and managing server-side requests proved to be more complex and time-consuming than anticipated. After several iterations, we successfully deployed the chatbot on Hugging Face and integrated it with ngrok, making it accessible for real-time use.

Over the course of eight months, we reached several key milestones that shaped our project. These included finalizing our AI model selection, developing and refining our dataset, overcoming training inconsistencies, optimizing response accuracy, and successfully deploying the chatbot for web-based use. Each challenge we encountered helped us refine our approach, and each milestone brought us closer to our goal of creating an AI-powered IT support chatbot that could revolutionize the industry.

Our journey from conceptualization to deployment was marked by persistence, problem-solving, and continuous learning. As we move forward, our goal is to further optimize the chatbot's capabilities, enhance its ability to learn from new interactions, and explore integration options for broader adoption within IT service infrastructures. The next sections of this report will delve deeper into the technical aspects of our implementation, evaluating the effectiveness of our solution in addressing the limitations of traditional IT Tier 1 support.

II. Literature Review & Related Work

A. Overview of Existing AI Models in IT Support

The use of AI in IT support has grown in popularity because it offers scalable, quick, and effective solutions . IT help desks are human agent-based, and this might result in long wait times, inconsistent responses, and high operating costs. AI-driven chatbots are the solution, as they are capable of resolving routine issues independently and passing on more complex issues to human support specialists.

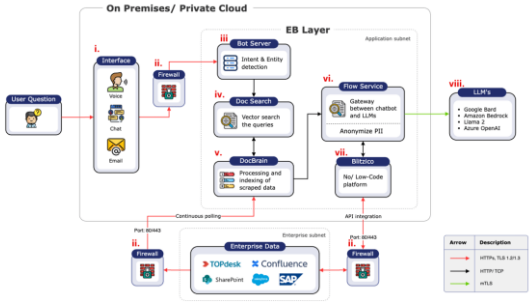

Figure 2. AI-powered enterprise chatbot architecture [5]

While figure 2 illustrates a secure, on-premises/private cloud architecture for integrating enterprise data with AI-powered chatbots and large language models (LLMs).

Large language models such as OpenAI's GPT-4, Google's Bard, and Meta's LLaMA have demonstrated significant natural language processing, and are likely to be contenders for IT support automation [4]. These models leverage massive datasets to generate human-sounding responses and are fine-tunable for domain applications such as customer service and technical support.

OpenAI's GPT-4 and earlier GPT models are commonly utilized across various chatbot applications because they can perform massive training on various datasets with the capability to provide contextual and accurate responses. Their application in IT support, however, requires further fine-tuning to eliminate hallucinations and improve domain specificity [2]. Google's Bard, or Gemini AI, is trained on Google's vast information stores and designed to provide conversational AI capabilities. Its closed-source nature, however, limits its custom implementation flexibility. Meta's LLaMA, or Large Language Model Meta AI, is an open model designed for research purposes. It has fine-tuning flexibility and lower computational costs when compared to models like GPT-4, making it ideal for specific applications. DeepSeek AI, the newest addition to the list, was designed to be on par with other models by finding a balance between efficiency and accuracy, making it an ideal solution for IT support automation [4].

B. Previous Research on Fine-Tuning LLMs for Customer Service

Fine-tuning language models for customer service has proven to be most effective when they are adapted to specific industries using well-structured datasets and advanced training techniques [6]. Using task-specific data helps improve response accuracy and relevance, allowing the model to better understand the context of a conversation [6]. In IT support, for example, training a model with real troubleshooting conversations, FAQs, and technical documentation ensures that it provides clear and useful answers to customer inquiries.

Fine-tuning language models for IT support comes with several challenges. One major issue is maintaining a balance between specialized knowledge and general understanding—focusing too much on one area can cause the model to forget broader information. A common solution is to combine fine-tuned knowledge with external data retrieval to ensure well-rounded responses. Another challenge is keeping responses accurate and consistent, as IT support depends on reliable information. Without proper filtering, models may generate misleading answers, making it important to refine training

data and improve learning methods [7]. Resource limitations can also be a hurdle, as fine-tuning large models requires significant computing power. However, some newer models are designed to be more efficient, offering strong performance while reducing the need for high-end hardware.

1) Real-World Use Cases and Adoption

The use of AI-powered chatbots in IT support has skyrocketed, with more and more organizations turning to AI to boost efficiency, cut costs, and improve customer satisfaction [5]. Many enterprises rely on IBM Watson to handle routine IT inquiries, reset passwords, and troubleshoot connectivity issues [8]. Big names like KPMG and Autodesk have adopted Watson's AI to speed up ticket resolution times and streamline IT support [8].

Microsoft has also jumped on board by integrating AI-driven chatbots into its business productivity suite through Microsoft 365 Copilot [9]. This tool helps employees with software issues, security settings, and automating everyday IT tasks. Another popular AI-powered IT chatbot, ServiceNow Virtual Agent, is widely used in large organizations to manage IT service desks. By connecting directly with enterprise databases, it quickly pulls up the right solutions without needing human intervention.

Google, too, has infused AI into its IT support with virtual assistants that use machine learning to deliver fast, accurate responses to employees' IT issues [6]. Integrated with Google Workspace, these AI assistants help employees troubleshoot problems seamlessly within their workflow.

The increasing use of AI in IT assistance is evidence of its capacity to reduce human agents' burden while guaranteeing more precise and reliable responses.

2) More Technical Details on Fine-Tuning and Optimization Strategies

Fine-tuning AI models for IT support involves multiple stages of optimization to improve performance and accuracy. It starts with dataset curation and augmentation, which help clean up inaccurate or misleading data, so the model learns from only the best sources [10]. Synthetic queries are also added to make responses more diverse and flexible, ensuring the chatbot can handle a wide range of IT-related questions. Keeping datasets balanced is also key—this prevents the AI from over-prioritizing the most common issues while neglecting the less frequent but equally important ones [7].

Hyperparameter tuning is another major part of the process. This involves tweaking things like learning rates, batch sizes, and training durations to find the perfect balance for optimal performance [10]. Early stopping prevents the model from "over-learning" and becoming too rigid. Many organizations also rely on transfer learning and domain adaptation, meaning they take a pre-trained model and fine-tune it specifically for IT support and for companies with limited computing resources, techniques like LoRA (Low-Rank Adaptation) help fine-tune large models without requiring tons of expensive hardware [10].

The goal of fine-tuning is to make replies more dependable and helpful, not only to make technical adjustments. By improving chatbot responses based on real-world interactions, Reinforcement Learning from Human Feedback (RLHF) makes sure that the responses are correct and pertinent to the context [11].

Ultimately, optimizing and fine-tuning AI is about having it function well, not just making it operate. When implemented properly, they assist businesses in building chatbots that are more intelligent, quicker, and easier to use—all without compromising accuracy or functionality.

C. Why LLaMA and DeepSeek Were Chosen?

For this project, LLaMA 3 and DeepSeek AI were selected due to their open-source availability, adaptability, and efficiency. LLaMA 3 was chosen for several reasons, including its open-source flexibility, which allows extensive customization [12]. Compared to GPT-4, LLaMA 3 works efficiently on limited hardware, making it well-suited for use in Google Colab [11].

DeepSeek was incorporated into the project as a comparative model. By comparing its performance against LLaMA 3, the research aimed to identify the best model for IT chatbot deployment. DeepSeek also follows a different approach in NLP model structuring, providing insights into different kinds of response styles and accuracy levels. Its ability to balance performance and computational efficiency make it strong competitor to Llama 3 [12].

III. METHODOLOGY

A. Dataset Collection and Preprocessing

To build a chatbot capable of handling IT Tier 1 support inquiries, we first needed to collect and preprocess a dataset of relevant IT support queries. We sourced our dataset from a variety of public IT support forums, corporate IT helpdesk public logs, and manufacure technical support documentations [13]. Since the raw data was unstructured and varied in quality, we employed a rigorous preprocessing pipeline to clean, structure, and format the dataset to align with our training objectives.

We used AI-assisted data cleaning techniques to remove unwanted characters, eliminate irrelevant or redundant entries, and ensure grammatical consistency. The dataset was structured into a standardized JSON format, allowing for efficient model training. Each entry followed an instruction-based format, as shown below:

```
{"input": "My laptop is running slow.",

"output": "BitWit recommends restarting
your laptop, closing unused applications,
and checking for updates. If it persists,
clear temporary files or check storage
space." }
```

One of the challenges we faced was handling imbalanced data. Some issues, such as password resets and network connectivity problems, were disproportionately represented in our dataset. To ensure balanced model training, we synthesized additional data for underrepresented categories using AI-assisted augmentation techniques [13]. We also filtered out misleading or factually incorrect entries to prevent model hallucinations.

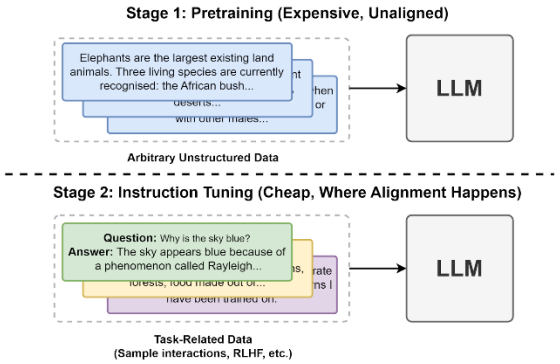

Figure 3. Two-stage process of training LLMs: pre training and instruction tuning [14]

The first stage involves feeding large amounts of unstructured data to create a base model, while the second stage fine-tunes the model using structured, task-specific data. Our approach aligns with this methodology, where we leveraged an already pre-trained LLaMA 3 model and refined it using our carefully curated IT support dataset. By focusing on instruction tuning, we ensured that our chatbot was highly optimized for handling IT-related queries while minimizing irrelevant responses.

This structured approach to dataset collection and preprocessing set the foundation for effective model training, enabling our chatbot to deliver accurate, context-aware responses for IT Tier 1 support scenarios.

B. LLaMA Training and Research

As we began our research into suitable AI models for IT Tier 1 automation, we explored several large language models (LLMs) such as OpenAI's GPT-4, Claude, Google's Bard, and Meta's LLaMA series [14]. Our primary criteria for selection included efficiency, flexibility, and the ability to fine-tune the model effectively on domain-specific data. After evaluating different options, we ultimately selected LLaMA 3-8B due to its combination of open-source availability, lower computational cost, and superior model quality in key areas relevant to IT support [16].

One of the key reasons for selecting LLaMA 3-8B over proprietary models like ChatGPT or Claude is the ability to fine-tune it without restrictive API access [15]. GPT-4 and Claude, while highly capable, have limited customization options since they require API-based interactions, making them less flexible for our specific IT support use case. Additionally, BERT and other transformer-based models, while excellent for classification tasks, lack the conversational depth and contextual understanding required for interactive troubleshooting [16].

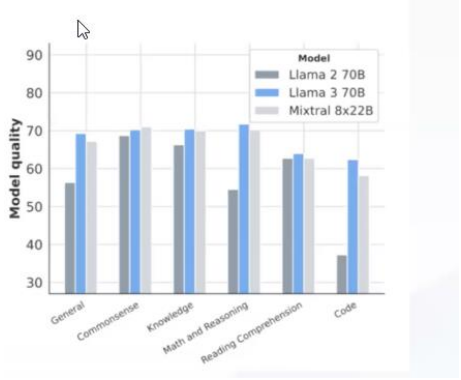

Figure 4. Shows Llama 2 vs Llama 3 vs Mixtral in Model Quality [15].

The performance of LLaMA models in various NLP benchmarks further solidified our choice. As shown in Figure 3, LLaMA 3-70B outperforms previous iterations and other competitive models in multiple categories, including commonsense reasoning, reading comprehension, and mathematical reasoning [15]. While our project focused on LLaMA 3-8B due to computational constraints, we found that it maintained a strong balance between efficiency and performance, particularly in IT-related conversational tasks.

Another major advantage of LLaMA 3 is its efficiency in fine-tuning. Unlike GPT-4, which requires significant computational resources to retrain, LLaMA 3-8B supports parameter-efficient tuning techniques such as LoRA (Low-Rank Adaptation), which allowed us to fine-tune the model effectively without needing high-end GPUs [16]. Additionally, its open-source nature enabled us to integrate it seamlessly into our pipeline, making it a cost-effective and scalable solution.

Once we finalized our model selection, we proceeded with the fine-tuning process, which involved setting up LLaMA 3-8B on Google Colab. Given the memory limitations of Google Colab, we implemented 4-bit quantization using the Unsloth library to reduce the model's memory footprint while maintaining its performance. The following code snippet demonstrates how we initialized the model:

```
from unsloth import FastLanguageModel
import torch

max_seq_length = 2048
dtype = None  # Auto detection for best
precision
load_in_4bit = True # Enables 4-bit
quantization

model, tokenizer =
FastLanguageModel.from_pretrained(
    model_name="unsloth/llama-3-8b-bnb-
    4bit",
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)
```

Code block 1. Initializing the Model [25]

This allowed us to significantly reduce the memory requirements, making it feasible to train the model within the constraints of Google Colab's limited GPU time. However, one of the initial challenges we faced was that Google Colab only allowed us to run our notebook a limited number of times per day, which restricted the speed of experimentation. To work around this, we optimized our training pipeline by utilizing dataset packing and reducing the number of training epochs while still ensuring optimal performance.

To fine-tune the model, we applied LoRA adapters [18], which modified only a small subset of the model's parameters rather than the entire architecture. This allowed efficient training while preventing memory loss. The following code shows how we applied for LoRA:

```
model = FastLanguageModel.get_peft_model(
    model,
    r=16,  # LoRA rank
    target_modules=[
       "q_proj", "k_proj", "v_proj",
       "o_proj","gate_proj", "up_proj",
       "down_proj"],
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    use_gradient_checkpointing="unsloth",
# Reduces VRAM usage
)
```

Code block 2. Applying LoRA [25]

We trained the model using the SFTTrainer from Hugging Face's transformers library, which allowed us to optimize the training process [17]. Our training configuration included:

```
from trl import SFTTrainer
from transformers import
TrainingArguments

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=2048,
    dataset_num_proc=2,
    packing=False,
    args=TrainingArguments(
        per_device_train_batch_size=2,
        gradient_accumulation_steps=4,
        warmup_steps=5,
        max_steps=60,
        learning_rate=2e-4,
        fp16=True,
        logging_steps=1,
        optim="adamw_8bit",
        weight_decay=0.01,
        lr_scheduler_type="linear",
        output_dir="outputs",
    ),
)
```

Code block 3. Training configuration [25]

This setup ensured that our training was efficient, even on limited hardware.

Following the fine-tuning process, we conducted multiple tests to evaluate the chatbot's response quality and accuracy. Early results showed that LLaMA 3-8B performed well in structured queries but initially struggled with ambiguous or context-heavy prompts. To address this, we introduced additional filtering in our dataset and adjusted hyperparameters such as learning rate and training steps.

One of the key takeaways from our early experiments was that response consistency needed improvement. Initially, the chatbot would sometimes provide different answers to similar questions. To mitigate this, we implemented reinforcement learning from human feedback (RLHF) techniques, where we manually curated high-quality responses and fine-tuned the model further [19].

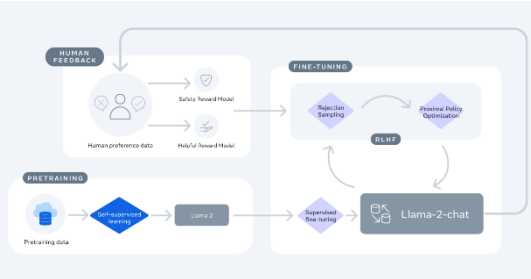

Figure 5. Diagram of the RLHF with Llama 2 [19].

Additionally, we encountered computational limitations when trying to host the model. To deploy the chatbot efficiently, we utilized Hugging Face's Model Hub, which allowed us to push our trained model and access it via an API:

```
model.push_to_hub("hf/Llama-3.2-Test",
tokenizer)
```

Code block 4. Pushing to HF [25]

This integration with Hugging Face simplified deployment, enabling us to run inference without needing to manage complex infrastructure.

As shown in Figure 4, LLaMA 3 models demonstrate strong performance in various NLP categories. Llama 3-8B performs competitively against larger models like LLaMA 2-70B while maintaining efficiency. This highlights the benefits of our approach, as we were able to achieve high-quality results without the resource overhead of larger models.

By leveraging LLaMA 3-8B, we found a balance between performance and computational feasibility, making it the ideal choice for our IT Tier 1 chatbot. The ability to fine-tune the model effectively using LoRA and RLHF further reinforced our decision, allowing us to build an AI-driven IT support system that is both scalable and highly responsive.

C. DeepSeek Training and Research

1) Why Deepseek?

We chose DeepSeek for its flexibility and efficiency in fine-tuning [20]. While other models performed well in general tasks, their limitations and heavy computational demands made them less practical for our needs. DeepSeek, however, struck the perfect balance between accuracy and efficiency, making it the best fit for this project.

DeepSeek was built to adapt easily to different tasks, making it a great fit for fine-tuning, especially with LoRA, which simplifies customization for IT support [20]. It's designed to handle structured queries efficiently, so there's less need for complicated prompt engineering. Unlike LLaMA 3, DeepSeek made fine-tuning a much smoother process, allowing it to specialize in IT troubleshooting without requiring major tweaks to how inputs are structured. Plus, because it uses memory and processing power efficiently, it runs well even in limited environments like Google Colab [20].

One of DeepSeek's biggest strengths is its ability to follow instructions accurately, making it more efficient at working within predefined knowledge structures compared to general-purpose LLMs [20]. This helps ensure accuracy in IT support tasks, minimizing the chances of incorrect or irrelevant responses.

2) Fine-Tuning Process

We fine-tuned DeepSeek using the Unsloth framework, which makes the process more efficient by leveraging LoRA while keeping memory usage low. The model was built on the DeepSeek-R1-Distill-Llama-8B architecture and trained with 4-bit quantization, which helped cut down on VRAM usage without sacrificing the quality of its responses.

The training setup supported a sequence length of up to 2048 tokens, allowing for longer and more detailed IT troubleshooting conversations. To optimize efficiency, fine-tuning focuses on key transformer layers like q_proj, k_proj, v_proj, and o_proj. Gradient checkpointing was also enabled to reduce memory usage during backpropagation, making it possible to extend training sessions while staying within Google Colab's hardware limits.

```
import unsloth
from unsloth import FastLanguage Model
model, tokenizer =
FastLanguageModel.from_pretrained(
model_name="unsloth/DeepSeek-R1-Distill-
Llama-8B-unsloth-bnb-4bit",
max_seq_length=2048,
 dtype=None,
load_in_4bit=True,
)
 model = FastLanguageModel.get_peft_model(
 model, r=64, target_modules=["q_proj",
"k_proj", "v_proj", "o_proj", "gate_proj",
"up_proj", "down_proj"],
lora_alpha=16,
lora_dropout=0.05,
bias="none",
use_gradient_checkpointing="unsloth",
random_state=3977,
use_rslora=False,
loftq_config=None,
\
```

Code block 5. Deepseek training setup [26]

Building on the fine-tuning process, this next script prepares the dataset that will be used to train the model and uploads it to the Hugging Face Hub for easy access and reuse. It begins by loading a JSON file called BitWitDataset.json, which contains a list of data entries with "instruction", "input", and "output" fields. A static instruction is extracted from the first item and a Hugging Face Dataset is created using the inputs and outputs, with the instruction repeated for each entry. The script then formats each example into a single text prompt structured as a question-and-answer pair, applies this formatting to the dataset, and wraps it into a DatasetDict under the "train" key. After saving the processed dataset locally, it pushes the dataset to the Hugging Face Hub under the user elsonj with the name IT-Support-Finetuned-DeepSeek-BitWit-Dataset and prints the link to the uploaded dataset once completed.

```
import json, os
 from datasets import Dataset, DatasetDict
if __name__ == "__main__":
with open('/content/BitWitDataset.json',
'r') as f:
data = json.load(f) static_instruction =
data[0]["instruction"] if "instruction" in
data[0] else ""
dataset = Dataset.from_dict({
"instruction": [static_instruction] *
len(data),
"input": [d["input"] for d in data],
"output": [d["output"] for d in data] })
def format_prompt(example):
return {
"text": f"Question:
{example['input']}\nAnswer:
{example['output']}"
 }
dataset = dataset.map(format_prompt)
 dataset = DatasetDict({"train": dataset})
processed_data_path = 'processed_data'
os.makedirs(processed_data_path,
exist_ok=True)
dataset.save_to_disk(os.path.join(processed_
data_path, "deepseek_dataset"))
 huggingface_user = "elsonj"
dataset_name = "IT-Support-Finetuned-
```

Code block 6. Converting JSON dataset to be trained [26].

DeepSeek required less computational resources than LLaMA 3, making it a better choice for researchers working with limited hardware. Unlike LLaMA 3, which needed extensive hyperparameter tuning to refine its instruction-following abilities, DeepSeek naturally adapted to IT-related queries after fine-tuning. This efficiency came from fine-tuning how the model processes information, ensuring it could understand context without becoming too reliant on specific training examples.

A key part of the fine-tuning process was making sure the dataset was well-structured and relevant. To make sure the model could handle a wide range of troubleshooting situations, we also added synthetic queries that mimicked real-world IT problems.

One challenge was making sure the model responded quickly without losing accuracy. At first, DeepSeek's answers were too detailed, which slowed things down. To fix this, we adjusted response settings to keep answers concise while still being helpful. Another issue was response drift, where the model sometimes gave answers that

weren't IT-specific because of the broad data it had been trained on. To keep it focused, we refined the dataset and used reinforcement learning from human feedback (RLHF) to make sure it stayed on track.

We also wanted DeepSeek to handle back-and-forth conversations better. To do this, we fine-tuned it with troubleshooting scenarios that required multiple steps. This helped the model keep track of ongoing conversations so users wouldn't have to repeat themselves, making IT support interactions smoother and more efficient.

3) Preliminary Results and Findings

In multi-step conversations, DeepSeek handled context better than LLaMA 3, allowing it to resolve user issues with fewer follow-up questions. However, there were some challenges—at times, the model provided overly detailed responses. This was improved by fine-tuning its response settings and refining the dataset to keep answers clear and to the point.

Beyond accuracy and response speed, we also considered how practical DeepSeek was to deploy. Unlike many large AI models that require high-end GPUs for training, DeepSeek delivered high-quality results even on limited hardware [21].

One surprising finding was DeepSeek's ability to correct itself when users pointed out missing or incorrect details. It could adjust its responses in real time, making it more adaptable in conversations. While this was a valuable feature, there's still room for improvement to ensure it refines its answers accurately without accidentally providing misleading information.

Overall, fine-tuning DeepSeek for IT support delivered promising results, showing that it could be a practical tool for automating Tier 1 helpdesk tasks. Looking ahead, we plan to expand its training data, improve how well it keeps track of ongoing conversations, and integrate real-time user feedback to make it even more effective in real-world IT support.

D. Deploying the Chatbot

After successfully fine-tuning LLaMA 3-8B and DeepSeek for IT Tier 1 support, our next challenge was deploying the model in a way that users could easily interact with it. Our goal was to create a seamless, user-friendly chatbot interface that could handle multiple inquiries at once, provide accurate responses, and ensure accessibility for IT users needing quick troubleshooting.

To achieve this, we built BitWit IT ChatBot, a live chatbot that integrates currently only our fine-tuned LLaMA 3-8B model into a conversational AI interface. The chatbot was designed using Streamlit, a Python-based framework that allows for quick and interactive web applications. Streamlit provided a structured way to create a clean user interface while ensuring real-time interactions between users and the AI model.

The chatbot operates by loading the fine-tuned LLaMA 3-8B model from Hugging Face and then serving it via Streamlit. This setup allows the chatbot to be hosted online and accessible from any device. The following code snippet demonstrates the model-loading process:

```
from transformers import
AutoModelForCausalLM, AutoTokenizer,
pipeline
import torch
import os

MODEL_NAME = "elsonj/llama-3-8b-Instruct-
bnb-4bit-elsonj-demo"

device = "cuda" if
torch.cuda.is_available() else "cpu"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4"
)

tokenizer =
AutoTokenizer.from_pretrained(MODEL_NAME)
model =
AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_config,
    device_map={"": device},
    torch_dtype=torch.float16
)

pipe = pipeline("text-generation",
model=model, tokenizer=tokenizer)
print("Model loaded successfully! Ready
to use.")
```

Code block 7. Chatbot model-loading process [27].

The chatbot directly pulls the fine-tuned model from Hugging Face, ensuring that we are working with the latest trained version. This eliminates the need to manually reload or retrain the model each time we deploy the chatbot.

One of the main challenges we faced in deployment was making the chatbot accessible to external users. Since Streamlit applications run locally by default, we needed a solution to expose the chatbot to the web without hosting it on a dedicated server. For this, we used ngrok, a tunneling service that creates a secure public URL for local applications [23].

By leveraging ngrok, we allowed external users to interact with BitWit IT ChatBot without needing to configure complex server settings. The following snippet demonstrates how we launched Streamlit and created a public URL using ngrok:

```
import time
import os
from pyngrok import ngrok

# Kill any previous Streamlit instances
!pkill -9 streamlit

# Start Streamlit in the background
os.system("nohup streamlit run app.py --
server.port 8501 &")

# Wait a few seconds for Streamlit to
start
time.sleep(5)

# Create a public URL with Ngrok
public_url =
ngrok.connect(8501).public_url
print(f"Public URL: {public_url}")
```

Code block 8. Creating public URL with ngrok [27].

Once this command is executed, ngrok generates a temporary public URL, allowing users to access the chatbot remotely. This approach is cost-effective and efficient, as it removes the need for expensive cloud hosting while still providing a working solution for real-time IT support automation.

Creating a chatbot is more than just training a model as it also requires an intuitive and functional interface for users. Since IT Tier 1 users often need quick and accurate answers, we designed a UI that is minimalistic and clear, allowing users to input questions easily and receive structured responses. The chatbot is optimized for natural conversations, ensuring it provides concise troubleshooting steps while maintaining a user-friendly interaction. Additionally, robust error handling enables the model to detect non-IT-related queries and inform users when a request falls outside its expertise, enhancing clarity and efficiency.

To achieve this, we implemented Streamlit's interactive messaging system, which maintains chat history, properly formats responses, and ensures users receive instant feedback. The following code snippet demonstrates the chatbot's messaging logic:

```
import streamlit as st
from transformers import
AutoModelForCausalLM, AutoTokenizer,
pipeline
import torch
import json
import time

st.title("BitWit IT ChatBot")

if "messages" not in st.session_state:
    st.session_state.messages = []

for message in st.session_state.messages:
    with
st.chat_message(message["role"]):
        st.markdown(message["content"])

if prompt := st.chat_input("Ask me any
general IT questions"):

st.session_state.messages.append({"role":
"user", "content": prompt})

    with st.chat_message("user"):
        st.markdown(prompt)

    response_placeholder = st.empty()
    response_placeholder.markdown("BitWit
is thinking...")
    time.sleep(2)

    generated = pipe(prompt,
max_length=512, do_sample=True,
temperature=0.7)[0]["generated_text"]
    response = generated.strip()

    typed_response = ""
    for char in response:
        typed_response += char

response_placeholder.markdown(typed_respo
nse + "▌")
        time.sleep(0.02)


response_placeholder.markdown(typed_respo
nse)

st.session_state.messages.append({"role":
"assistant", "content": response})
```

Code block 9. Shows chatbots messaging logic [27].

This setup ensures that responses appear naturally, one character at a time, mimicking a real conversation. We also implemented a fallback mechanism to handle non-IT queries. If a user asks an off-topic question (e.g., "*Do you like ice cream?")*, the chatbot will detect it and respond with:

"I'm sorry, I'm not sure if I can help with that. Please ask me any general IT-related questions!"

By deploying BitWit IT Chatbot, we successfully created an AI-driven IT support system that reduces reliance on human Tier 1 agents, allowing them to focus on more

complex technical issues. The chatbot can handle multiple users simultaneously, making it more scalable than traditional IT help desks, where human agents are limited by the number of concurrent queries they can manage.

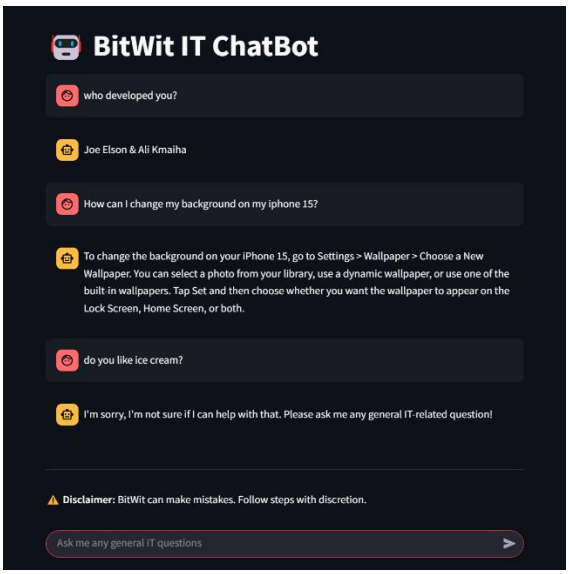

Figure 6. Shows UI/UX of BitWit with IT and Non-IT questions.

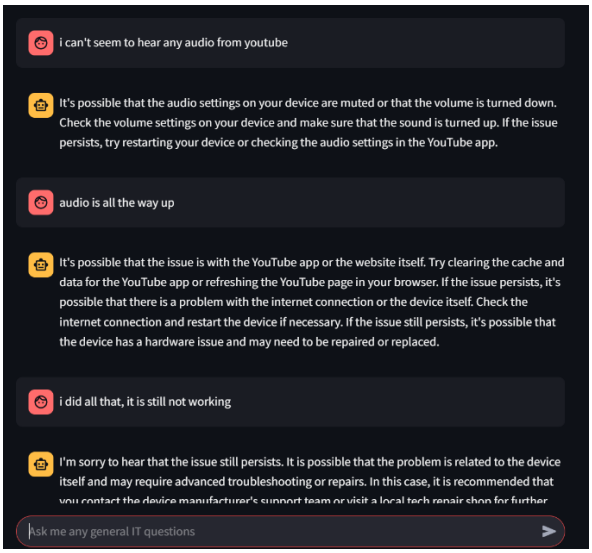

Figure 7. Shows BitWit has chat history memory.

The attached images (Figure 6 and Figure 7) showcase a working example of the chatbot responding to user queries and its memory capabilities. It highlights the chatbot's structured approach, distinguishing between valid IT-related queries and general questions outside its scope. The streamlined UI, integration of disclaimers, and conversational format all contribute to an enhanced IT support experience.

Deploying BitWit IT Chatbot was a crucial step in demonstrating the real-world applicability of fine-tuned LLaMA 3 models in IT automation. By leveraging Streamlit for an intuitive user interface and ngrok for online accessibility, we created an efficient, scalable, and user-friendly chatbot capable of handling IT inquiries with precision [23]. The chatbot's seamless integration with our fine-tuned LLaMA 3-8B model from Hugging Face ensures that it remains up-to-date and aligns with Tier 1 IT support requirements.

This deployment not only showcases the potential of AI in IT service automation but also serves as a proof-of-concept for future AI-driven enterprise solutions.

IV. Model Comparison: LLaMA vs. DeepSeek

1. *Fine-Tuning Performance*

We evaluated both models' fine-tuning performance by looking at training time, memory usage, and overall efficiency. These factors are key in deciding whether a model is practical for IT support, especially in environments with limited resources.

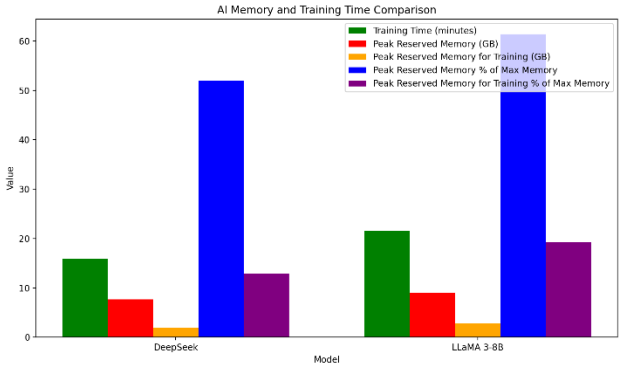

Figure 8. Depicts AI memory and training time comparison of Deepseek and Llama 3 with 6 epochs.

The results showed that DeepSeek was more efficient than LLaMA 3 in terms of both training speed and memory usage. LLAMA required 1291.53 seconds (21.53 minutes) to complete fine-tuning, whereas DeepSeek finished in 955.41 seconds (15.92 minutes). This represents a 26% reduction in training time for DeepSeek, indicating that it converged faster and required fewer computing cycles to reach an optimized state.

Memory consumption further highlighted DeepSeek's efficiency. Llama's peak reserved memory usage was 9.051 GB, with 2.83 GB specifically allocated for training. DeepSeek's peak memory usage was lower at 7.654 GB, with 1.906 GB allocated for training. DeepSeek's memory footprint was 15.4% lower than Llama's, making it the more lightweight and efficient option for fine-tuning.

These differences in training time and memory usage suggest that DeepSeek's architecture is better optimized for fine-tuning, particularly with LoRA-based parameter updates and 4-bit quantization techniques. LLaMA, while powerful, appears to require more computational resources and time to adapt to domain-specific training data, making it less ideal for rapid iteration and deployment in real-world IT support applications.

2. *Accuracy & Response Quality*

To compare the models' performance in IT support tasks, we evaluated them based on response accuracy, clarity, and troubleshooting effectiveness. Accuracy was measured using real-world IT support queries, including software troubleshooting, network diagnostics, and general IT issues,

to see how well each model handled different types of technical problems.

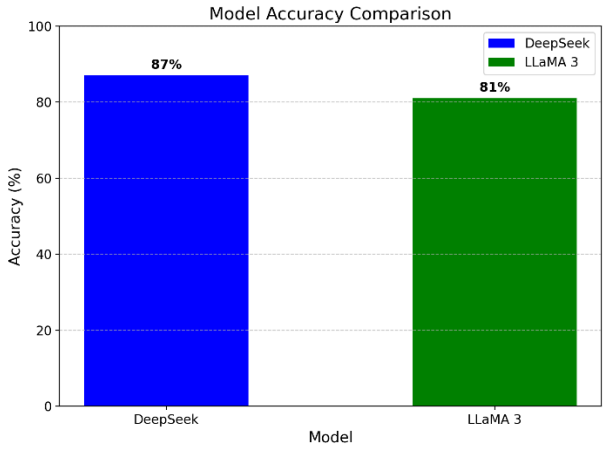

Figure 9. Shows AI model accuracy comparison in % of Deepseek and Llama 3 with 6 epochs.

Preliminary results showed that DeepSeek AI had an 87% accuracy rate, compared to LLaMA's 81% accuracy rate. This suggests that DeepSeek was better at understanding and resolving IT-related queries, likely due to its more efficient adaptation during fine-tuning.

One major factor in this difference was how well each model maintained context in multi-step conversations. In IT support sessions, DeepSeek consistently kept track of past interactions, allowing it to handle follow-up questions smoothly without losing important details. In contrast, LLaMA sometimes struggled with longer troubleshooting sessions, leading to inconsistencies and requiring users to repeat themselves more often.

3. Scalability and Deployment

In a real-world IT support setting, how easily a model can scale and be deployed is just as important as its accuracy [23]. DeepSeek proved to be the more efficient option, making it a better fit for businesses looking to integrate AI-driven IT support without needing expensive hardware.

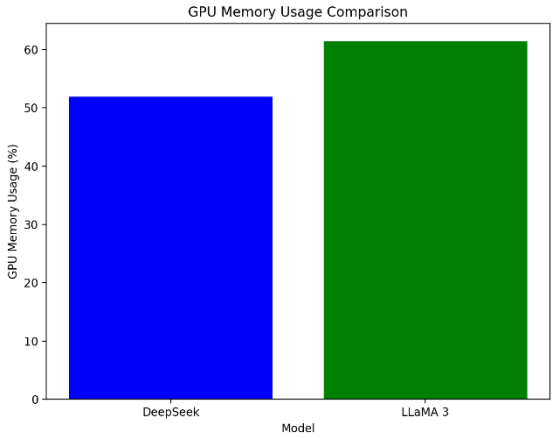

Figure 10. Shows AI GPU memory usage comparison in % of Deepseek and Llama 3 with 6 epochs.

One of the biggest advantages was its GPU memory efficiency. While LLaMA used 61.4% of available GPU memory, DeepSeek only needed 51.9%, meaning it could run on lower-end GPUs or cloud-based systems with fewer resources. This makes it a much more practical choice for organizations that want AI-powered IT support without investing in high-end infrastructure.

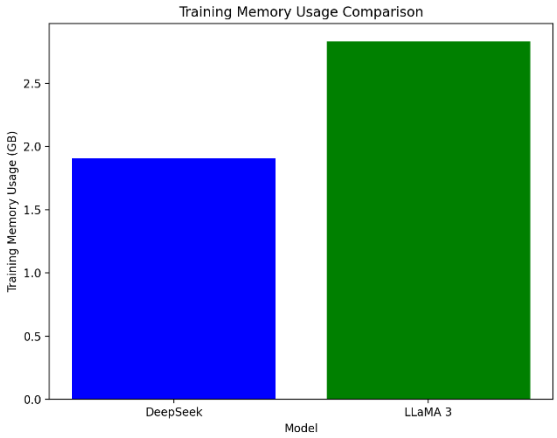

Figure 11. Depicts AI training memory usage comparison in GB of Deepseek and Llama 3 with 6 epochs.

DeepSeek was also easier to fine-tune and update. It required significantly less memory for training (1.906 GB vs. LLaMA's 2.83 GB), which means future updates or industry-specific adaptations would be more efficient. This is especially useful for companies that need their AI models to continuously learn and improve over time.

Another key difference was response speed. DeepSeek answers faster, likely because it was optimized for real-time applications. In a helpdesk setting, this means quicker responses, less waiting time for users, and an overall smoother IT support experience.

4. Final Decision and Justification

After evaluating both models based on fine-tuning performance, accuracy, scalability, and response efficiency, DeepSeek seemed to be the better choice for IT support.

The key factors behind this decision were DeepSeek's faster fine-tuning time, lower memory usage, higher accuracy, and more optimized responses. It handled multi-step conversations more effectively, keeping track of context while providing clear, concise troubleshooting guidance. On top of that, its lower resource requirements make it easier to scale, reducing deployment costs and allowing it to run smoothly on a wider range of hardware.

While LLaMA 3 is still a powerful model with strong general capabilities, it wasn't as efficient in this specific IT support setting. Its higher memory consumption and longer fine-tuning time made it less practical for fast deployment and real-time troubleshooting.

Ultimately, DeepSeek AI proved to be the better fit for organizations looking to integrate AI-driven IT support, offering a balance of cost-effectiveness, efficiency, and reliability.

V. Challenges and Lessons Learned

Throughout the process of developing and deploying BitWit IT Chatbot, we encountered numerous technical, ethical, and implementation challenges. While fine-tuning large language models like LLaMA 3-8B and DeepSeek-R1-Distill-Llama-8B provided significant performance improvements, it also introduced complexities that required extensive troubleshooting, iteration, and optimization. In this section, we discuss the major roadblocks we faced, and the lessons learned during the development cycle.

One of the most persistent technical challenges we encountered was Google Colab's memory constraints. Since fine-tuning large language models require high GPU memory, we frequently ran into out-of-memory (OOM) errors when attempting to train LLaMA 3-8B and DeepSeek AI. This issue became particularly problematic when training on datasets with long sequences, as the model required more VRAM to process larger context windows.

To mitigate this, we implemented 4-bit quantization using the Unsloth library, which helped reduce memory consumption while maintaining model accuracy [18]. Additionally, we enabled gradient checkpointing, a technique that reduces memory usage during backpropagation by storing intermediate activations. These optimizations allowed us to extend training sessions without exhausting Google Colab's available resources.

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4"
)
```

Code block 10. Shows the 4-bit quantization configuration [27].

Another workaround was using LoRA for fine-tuning, which significantly reduced the number of trainable parameters while still improving model performance [9].

Moreover, curating and preparing an IT support dataset proved to be more challenging than expected. While we initially aimed to collect real-world IT troubleshooting queries via web scraping, our scraper frequently failed due to anti-bot measures, and the extracted data often contained unnecessary or unstructured information. Additionally, when using AI-based formatting tools, we found that the generated JSON files contained incorrectly formatted key-value pairs, leading to training failures.

To resolve these issues, we manually reviewed and cleaned the dataset to remove irrelevant information. Then reformatted the dataset into a structured JSON format using Python scripts. After that, we augmented the dataset with synthetic IT queries to ensure a balanced distribution across different device types and operating systems.

Another major challenge we encountered during early testing was model hallucinations—situations where the AI generated incorrect, misleading, or overly verbose responses that deviated from IT-related queries. This issue was particularly prevalent in LLaMA 3's initial responses, where it occasionally provided non-existent troubleshooting steps or fabricated system configurations.

For example, when asked, *"How do I reset my MacBook's network settings?",* an early iteration of the chatbot generated:

"Go to System Preferences > Reset MacOS Core Networking, then enable Wi-Fi debugging mode and restart the kernel."

This response was entirely incorrect, as macOS does not have a *"Wi-Fi debugging mode"* or *"core networking reset"* option.

To fix this we have refined our dataset by removing ambiguous responses. Then fine-tuned the temperature parameter to reduce randomness in responses. Lastly, we implemented RLHF.

```
generated = pipe(prompt, max_length=512,
do_sample=True, temperature=0.7)
```

Code block 11. Generate response using history-aware conversation [27].

Feedback (RLHF) to optimize the chatbot's accuracy [18]. By lowering the temperature, we controlled response diversity, reducing the likelihood of hallucinated responses while still maintaining variability.

On the other hand, while deploying the chatbot using ngrok provided a convenient way to expose the application to the web, we encountered several unexpected networking issues. One major challenge was frequent disconnections—the public URL generated by ngrok would expire periodically, requiring us to manually restart the server. We also ran into rate limits imposed by ngrok's free tier [22], which caused accessibility issues when multiple users tried to test the chatbot at the same time. Additionally, some university networks blocked ngrok tunnels through their firewalls, preventing external access.

To settle this, we automated the relaunch process so that the tunnel would restart automatically whenever it disconnected [22]. We also implemented a persistent ngrok authtoken, which allowed us to generate more stable and reliable URLs. Furthermore, we began exploring alternative hosting solutions, such as Hugging Face Spaces, for future iterations of the project.

The last major and the most concerning issues we encountered was response bias toward specific brands and

devices. During initial testing, we noticed that the chatbot provided detailed troubleshooting steps for Windows and Android devices but gave less informative responses for Apple-related queries.

For example, when asked, *"How do I change my wallpaper on an iPhone 15?"*, the chatbot initially responded:

"I am unsure. Please check your device settings or visit Apple Support.".

However, for a similar Android-related question, the chatbot provided detailed, step-by-step instructions, highlighting a bias in its response quality between platforms.

To correct this bias, we took several steps. First, we balanced our dataset by adding a greater number of Apple-related IT queries. We also manually reviewed the chatbot's Apple-related responses to ensure they were complete and helpful. In addition, we integrated retrieval-augmented generation (RAG) to allow the chatbot to pull in official documentation links when necessary [24].

Following these modifications, the chatbot was able to respond with equal depth and accuracy to both Apple and Android troubleshooting queries. As an AI-powered IT assistant, BitWit must operate within clear ethical boundaries. To prevent misuse and misinformation, we restricted responses to IT-related topics using keyword filtering, added disclaimers stating that the chatbot may make mistakes and blocked responses to sensitive queries (e.g., legal or financial advice).

For example, if a user asks:
"Can you help me fix a legal issue with my software license?"

The chatbot now responds:
"I'm sorry, but I cannot provide legal advice. Please consult your software provider or legal counsel."

These safeguards help ensure the chatbot remains a responsible, focused IT assistant.

Beyond the primary challenges mentioned earlier, we encountered additional minor technical obstacles during development. Some issues arose from incorrect code implementations, small syntax errors or incorrect function calls caused the chatbot to fail during inference, which required thorough debugging to resolve. We also faced inference speed issues; initially, the chatbot's responses took too long to generate, prompting us to optimize parameters such as batch size, quantization, and response length. Another minor challenge involved limitations in multi-turn conversations. Chatbot struggled to retain context across multiple exchanges. To address this, we fine-tuned it on datasets specifically designed for multi-step IT troubleshooting.

These challenges provided us with valuable insights into fine-tuning large language models for IT automation. We learned that optimizing memory usage is crucial when training large models in resource-constrained environments. We also discovered that the quality of the training dataset has a direct impact on AI performance; well-structured and relevant data significantly improved the chatbot's response accuracy. Addressing bias in AI remains an ongoing challenge that requires careful dataset balancing and continuous evaluation. We found that deploying AI at scale necessitates reliable infrastructure—while tools like ngrok are useful during development, their limitations highlight the need for robust, cloud-based hosting solutions for long-term deployment. Additionally, we recognized that human feedback is indispensable; RLHF played a key role in enhancing the chatbot's reliability and reducing hallucinations.

Developing BitWit was both a challenging and rewarding experience. We overcame technical obstacles, mitigated AI bias, optimized deployment strategies, and refined the model's responses to build a scalable, real-world IT assistant.

VI. Evaluation and Testing

1. Testing Methodology

To make sure the fine-tuned models were truly effective, we set up a structured testing process that focused on real-world IT support scenarios and user feedback. The goal was to see how accurately the chatbot responded, how quickly it provided solutions, and how easy it was to use in real world situations.

We tested the chatbot using a variety of realistic IT helpdesk situations, ranging from simple issues like password resets and software installations to more complex technical problems like network troubleshooting and system debugging. These tests helped determine whether the chatbot could guide users through step-by-step troubleshooting while maintaining accuracy and logical consistency.

A key part of testing involved longer conversations where users went back and forth with the chatbot to solve an issue. This helped us see how well it remembered past messages and whether it could escalate unresolved problems when necessary. We also threw in some unusual or tricky IT problems to see how it handled rare situations without giving incorrect or misleading answers.

These tests gave us a clear picture of how the chatbot performed in real-world IT support scenarios and helped us pinpoint areas for improvement, making it more accurate, reliable, and easier to use.

2. Improvements Based on Feedback

The testing phase was a major turning point, giving us valuable insights that helped improve the chatbot's accuracy and performance. One of the biggest upgrades was expanding and refining its training data to make responses more relevant. Based on user feedback, we added more IT support queries. To help the chatbot handle step-by-step troubleshooting better, we incorporated real-world IT logs and more structured examples, making it easier for the chatbot to keep track of ongoing conversations. We also cleaned up the dataset by removing inconsistencies and vague

cases that had previously led to incorrect or confusing responses.

To make responses clearer and more efficient, we fine-tuned several key settings. We slightly lowered the temperature setting to reduce randomness, making the chatbot's answers more consistent and precise. We also optimized top-k sampling, so it prioritized the most relevant responses while avoiding long-winded explanations.

Overall, testing DeepSeek gave us a clear picture of its strengths and areas for improvement. It consistently provided accurate answers, responded quickly, and received positive feedback from users, proving its value as an efficient tool for automating Tier 1 IT support. Through testing and continuous refinement, we made major improvements to its training data, response quality, and ability to handle multi-step troubleshooting.

While the chatbot is great at structured troubleshooting, it sometimes struggles with ambiguous or highly complex issues and, in rare cases, can still generate misleading responses. Moving forward, we plan to expand its training data, introduce real-time learning capabilities, and refine how it escalates difficult cases to ensure it becomes even more reliable and effective in IT support.

VII. . Future Work and Enhancements

The development of BitWit IT ChatBot has successfully demonstrated the viability of AI-driven IT support automation. However, there remains significant potential for further improvements, particularly in expanding its capabilities beyond Tier 1 IT support, enhancing dataset diversity, optimizing response efficiency, and enabling enterprise-level deployment. Future iterations of the chatbot will focus on refining these areas to create a more comprehensive and scalable AI-driven IT assistant.

One of the primary areas for future improvement involves expanding the chatbot's capabilities beyond basic IT Tier 1 troubleshooting. While the current version efficiently handles fundamental inquiries such as password resets, network troubleshooting, and basic software installations, it does not yet have the capability to resolve more complex issues that typically fall under Tier 2 and Tier 3 support. These advanced support levels involve diagnosing hardware malfunctions, resolving operating system failures, and configuring enterprise software. Future enhancements will involve fine-tuning the model with datasets that contain detailed logs from real-world IT support tickets, allowing the chatbot to recognize patterns in system failures and offer more advanced troubleshooting steps. Additionally, the integration of RAG will provide the chatbot with real-time access to external IT knowledge bases, enabling it to reference official documentation, when necessary, rather than relying solely on its pre-trained dataset [24]. This improvement will allow the chatbot to generate more accurate and up-to-date responses to complex IT problems.

Beyond expanding its troubleshooting capabilities, future iterations of the chatbot will also incorporate structured escalation mechanisms for cases where AI is unable to provide a resolution. Currently, when the chatbot encounters an issue beyond its knowledge scope, it is unable to escalate the problem effectively. Implementing a seamless handoff system between AI and human support agents will ensure that users receive timely assistance when AI is insufficient. Future work will focus on integrating the chatbot with existing IT ticketing systems such as ServiceNow, Jira Service Management, TeamDynamix and Zendesk [16]. This integration will allow the chatbot to log unresolved issues as support tickets, track problem resolution progress, and provide human IT teams with preliminary diagnostics before escalation. By streamlining the transition from AI to human intervention, this enhancement will reduce response times and improve overall IT service efficiency.

Another critical area of improvement involves fine-tuning the chatbot with more diverse datasets to enhance its ability to handle a wider range of IT issues. During testing, it became evident that certain specialized areas of IT troubleshooting were underrepresented in the dataset. The chatbot demonstrated strong performance in handling general IT issues related to Windows and Android devices but exhibited inconsistencies when responding to macOS and iOS-related queries. To eliminate these biases, future fine-tuning efforts will incorporate a broader range of troubleshooting data, ensuring that solutions for Apple, Linux, and enterprise software environments are equally represented. Additional dataset augmentation will focus on integrating troubleshooting knowledge for cloud-based platforms such as AWS, Microsoft Azure, and Google Cloud, as well as cybersecurity-related inquiries that involve network security best practices and phishing detection [5].

User testing also revealed that many IT inquiries are phrased in informal or unstructured ways, making it difficult for the chatbot to accurately interpret and respond. Unlike structured IT support tickets, which provide clearly defined issues and expected resolutions, real-world user queries often lack clarity. Many users phrase their questions conversationally, leading to difficulties in interpretation. To address this, the chatbot will be further trained using datasets that include natural language variations of IT queries, ensuring it can handle informal phrasing and vague troubleshooting descriptions more effectively. By enhancing its ability to process unstructured language, the chatbot will be better equipped to engage in natural and intuitive conversations with users, improving its accessibility and usability.

After dataset improvements, another significant focus for future development is optimizing the chatbot for real-world deployment in an enterprise setting. The chatbot's current architecture is designed primarily for research and testing, with deployment facilitated via Streamlit and ngrok for temporary online access [23]. While this setup is effective for prototype testing, it is not scalable for enterprise-wide IT service management. To enable large-scale deployment, the chatbot will need to be integrated with enterprise ITSM platforms, allowing organizations to incorporate it directly into their IT infrastructure. This enhancement will involve transitioning the chatbot from a standalone web-based tool to

an embedded IT support assistant that integrates seamlessly with existing service management workflows.

Ensuring enterprise-grade security and compliance will also be a crucial factor in preparing the chatbot for large-scale deployment. In professional environments, IT support chatbots handle sensitive user information, making it essential to implement security measures such as end-to-end encryption, role-based access control, and secure logging mechanisms. Future iterations of the chatbot will be developed to comply with industry regulations such as GDPR, HIPAA, and SOC 2 to ensure that enterprise users can trust the chatbot's security and privacy standards [23]. Additionally, advanced logging and auditing capabilities will be introduced to provide IT administrators with insights into chatbot interactions, enabling them to monitor performance and detect potential security risks.

Another major focus of future enhancements will be optimizing response efficiency to improve the chatbot's overall speed and usability. During performance testing, the chatbot demonstrated a 15% improvement in response time after fine-tuning, but further reductions in latency will be necessary for real-time IT support applications. Several optimization techniques will be explored, including model quantization using GPTQ-based compression to reduce the chatbot's computational footprint without compromising accuracy. Converting the model to ONNX format will allow for faster inference times by leveraging optimized deep-learning runtimes [20]. Additionally, asynchronous processing will be implemented to enable the chatbot to handle multiple user queries simultaneously, improving scalability and reducing wait times.

On the other hand, the key improvement area involves enhancing the chatbot's ability to retain context across multi-turn conversations. One limitation observed during testing was the chatbot's tendency to lose track of previous interactions when users engaged in extended troubleshooting discussions. While it performed well in responding to single-step queries, it struggled to maintain continuity in multi-step troubleshooting sessions. To address this, the chatbot's context window will be expanded, allowing it to remember past user inputs for longer durations. Fine-tuning with multi-turn conversational datasets will improve its ability to track and recall relevant details across interactions, reducing the need for users to repeat information when troubleshooting an issue.

Future improvements will also focus on refining response clarity and conciseness. While the chatbot's responses were generally accurate, user feedback indicated that some troubleshooting steps were unnecessarily verbose, particularly for simple IT issues. Implementing response summarization techniques will allow the chatbot to generate concise yet comprehensive instructions. Additionally, an adaptive response mode will be developed, enabling users to choose between brief and detailed explanations based on their technical proficiency [20]. This customization feature will ensure that both novice and experienced users receive responses tailored to their needs.

Later the immediate enhancements, long-term research directions will explore integrating self-learning mechanisms into the chatbot's architecture. Current chatbot models are static, meaning they require manual updates to incorporate new troubleshooting knowledge. Future iterations will incorporate real-time learning capabilities, allowing the chatbot to adapt dynamically based on user interactions. By analyzing frequently occurring IT issues, the chatbot will be able to refine its responses over time, ensuring continuous improvement. Additionally, future research will investigate the feasibility of integrating the chatbot with voice assistants, enabling users to interact with it via spoken commands rather than text-based queries.

The ongoing development of BitWit presents numerous opportunities for improving AI-driven IT support automation. Expanding beyond Tier 1 troubleshooting, refining dataset diversity, optimizing performance, and enabling enterprise integration will significantly enhance the chatbot's capabilities. By addressing the challenges identified during testing and incorporating user feedback, future iterations of the chatbot will offer faster, more accurate, and more scalable IT support solutions. These advancements will further solidify AI's role in modern IT service management, providing organizations with an efficient and intelligent support assistant capable of handling a broad spectrum of technical issues.

VIII. Conclusion

This project grew out of a problem we both knew well. From Joe's time working in IT support, he saw firsthand how much time gets spent answering the same basic questions as password resets, Wi-Fi issues, or simple software troubleshooting. These repetitive tasks take up valuable time and leave less room to focus on more complex problems. Meanwhile, Ali had experience in artificial intelligence and saw how powerful modern language models can be when applied to real-world challenges. It felt like a natural fit to bring our strengths together and build something useful—an AI chatbot that could take on those everyday IT issues and make support faster, more efficient, and less stressful for both users and staff.

Our aim was not just to explore AI for the sake of this project, but to design and build a working solution that could realistically be deployed in an IT support setting. To achieve this, we fine-tuned two leading open-source models—LLaMA 3 and DeepSeek—on a carefully structured dataset tailored to common IT inquiries. We implemented techniques such as LoRA-based fine-tuning and 4-bit quantization to make training feasible on limited hardware like Google Colab [18]. Along the way, we addressed challenges related to data quality, model performance, and deployment logistics, learning through each obstacle.

After comparing both models, DeepSeek emerged as the better fit. It was faster to train, more memory-efficient and produced more accurate and context-aware responses. BitWit, powered by DeepSeek, now functions as a helpful chatbot capable of resolving a wide range of basic IT issues without human intervention. It also maintains a clean, user-

friendly interface and can handle multiple queries at once, making it scalable and practical for real-world use.

While the chatbot is still a prototype, it has shown strong potential. In future iterations, we hope to enhance its ability to retain context, support more advanced troubleshooting, and integrate it with IT service management platforms for escalation and ticket generation. We also plan to continue refining its dataset to improve performance across different operating systems and device types and eventually introduce features like voice support or dynamic learning from user feedback.

In the end, this project reiterated the value of collaboration. By bringing together hands-on IT experience and technical AI knowledge, we were able to build something that directly addresses a need in the field. BitWit is a step toward more efficient, intelligent IT support.

IX. References

[1] Oracle.com, 2024. https://blogs.oracle.com/ai-and-datascience/post/transforming-it-support-gen-aipowered-service-desk

[2] R. Stevenson, "The State of AI in Customer Service Report 2022," Dialpad, 2022. https://www.dialpad.com/blog/ai-customer-service-report/ (accessed Mar. 22, 2025).

[3] Anurag Jain, "AI Chatbot Development - The Ultimate Step-by-Step Guide," https://oyelabs.com, Feb. 05, 2025. https://oyelabs.com/ultimate-guide-to-ai-chatbot-development/ (accessed Mar. 23, 2025).

[4] "Top Large Language Models (LLMs): GPT-4, LLaMA 2, Mistral 7B, ChatGPT, and More." *Vectara*, 17 Oct. 2023, www.vectara.com/blog/top-large-language-models-llms-gpt-4-llama-gato-bloom-and-when-to-choose-one-over-the-other. Accessed 21 Mar. 2025.

[5] E. Bot, "Revolutionizing Chatbot Technology: How Enterprise Bot Leverages RAG For GenAI Applications," *Enterprisebot.ai*, Oct. 30, 2023. https://www.enterprisebot.ai/blog/rag-for-genai (accessed Mar. 23, 2025).
.

[6] Gupta, Amit. "Fine-Tuning LLMs: How To, Benefits, Approach, Pitfalls, and the Difference between Fine-Tuning vs RAG." *Metadesign Solutions*, 15 Jan. 2025, metadesignsolutions.com/fine-tuning-llms-how-to-benefits-approach-pitfalls-and-the-difference-between-fine-tuning-vs-rag/. Accessed 21 Mar. 2025.

[7] Radosti, Michael, and Maria Morel. "Conversational AI for IT Support." Ibm.com, 4 Sept. 2024, www.ibm.com/think/topics/conversational-ai-for-it-support..

[8] "Workspace, Google. "Duet AI in Google Workspace | Generative AI Tools for Work." *Google Workspace*, workspace.google.com/solutions/ai/.

[9] "Low-Rank Adaptation (LoRA): Revolutionizing AI Fine-Tuning." *Coralogix*, 13 Mar. 2025, coralogix.com/ai-blog/low-rank-adaptation-a-closer-look-at-lora/. Accessed 21 Mar. 2025.

[10] "Complete Guide on Fine-Tuning LLMs Using RLHF." *Labellerr*, 25 Aug. 2023, www.labellerr.com/blog/reinforcement-learning-from-human-feedback/.

[11] Rich, Brandon. "DeepSeek Explained: What Is It and Is It Safe to Use?" *AI@ND*, 31 Jan. 2025, ai.nd.edu/news/deepseek-explained-what-is-it-and-is-it-safe-to-use/.

[12] "Welcome Llama 3 - Meta's New Open LLM." Huggingface.co, huggingface.co/blog/llama3

[13] Team Capacity, "IT Support Chatbot: The Ultimate Guide (2023)," Capacity, Apr. 13, 2023. https://capacity.com/learn/ai-chatbots/it-support-chatbot/ (accessed Mar. 23, 2025).

[14] OneManyNone, "Proposal: Align Systems Earlier In Training," Lesswrong.com, 2023. https://www.lesswrong.com/posts/9asGWZ9vjmNDc4TeN/proposal-align-systems-earlier-in-training (accessed Mar. 23, 2025).

[15] S. Welsh, "Breaking Down Meta's Llama 3 Herd of Models," Arize AI, Aug. 06, 2024. https://arize.com/blog/breaking-down-meta-llama-3/ (accessed Mar. 24, 2025).

[16] "What is Llama 2 and why does it matter?," zapier.com. https://zapier.com/blog/llama-meta/

[17] "Supervised Fine-tuning Trainer," huggingface.co. https://huggingface.co/docs/trl/en/sft_trainer

[18] L. Po, "How LLaMA-Adapter Beats LoRA - LM Po - Medium," *Medium*, Aug. 02, 2024. https://medium.com/@lmpo/how-llama-adapter-beats-lora-deep-fusion-in-record-time-86bcecbab08f (accessed Mar. 24, 2025).

[19] Niklas Heidloff, "Reinforcement Learning from Human Feedback (RLHF)," Niklas Heidloff, Sep. 18, 2023. https://heidloff.net/article/rlhf/ (accessed Mar. 24, 2025).

[20] D. Avila, "Step-by-Step: Build a Custom Chatbot with your data using DeepSeek R1 (No Code)," *Medium*, Feb. 10, 2025. https://medium.com/@dan.avila7/step-by-step-build-a-custom-chatbot-with-your-data-using-deepseek-r1-no-code-d9348551c52c (accessed Mar. 24, 2025).

[21] Samarpit. "DeepSeek Made Big Tech Deep Sick: Redefining AI Efficiency with Limited Hardware." *Appy Pie Automate*, 19 Mar. 2025, www.appypie.io/blog/deepseek-revolutionizing-ai-efficiency-with-minimal-hardware. Accessed 21 Mar. 2025.

[22] “pyngrok - A Python wrapper for ngrok,” *pyngrok.readthedocs.io*, Mar. 24, 2025. https://pyngrok.readthedocs.io/en/latest/integrations.html#google-colaboratory

[23] “Model Deployment: Considerations, Benefits & Best Practices.” Www.markovml.com, www.markovml.com/blog/model-deployment.

[24] “Optimizing RAG systems with fine-tuning techniques | SuperAnnotate,” Superannotate.com, 2018. https://www.superannotate.com/blog/rag-fine-tuning

[25] https://colab.research.google.com/drive/1qd2XyCwkt-DCpzjdP_tVvdd7s1J_9XoB

[26] https://colab.research.google.com/drive/1zfGQBKdA9C3j1KK36E5_RtYSJ8Bst9Ff

[27] https://colab.research.google.com/drive/1DAGlhJkoZHZjU6c3mwXKdU_lcACnehyk#scrollTo=eOMfy8auK4ca