**MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

DeepSeek-based Chatbot system supports work management

by

Dinh Cong Bang

A thesis submitted in conformity with the requirements  
for the degree of Master of Software Engineering

© Copyright by Dinh Cong Bang 2025

**MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

DeepSeek-based Chatbot system supports work management

by

Dinh Cong Bang

A thesis submitted in conformity with the requirements  
for the degree of Master of Software Engineering

Supervisor:

Assoc. Prof. Phan Duy Hung

© Copyright by Dinh Cong Bang 2025

DeepSeek-based Chatbot system supports work management

Dinh Cong Bang

Degree Master of Software Engineering

FPT University

2025

Abstract

In the current era of digital transformation, integrating AI into enterprise systems has become a necessity. This thesis develops a work management system aimed at automating report generation, onboarding guidance, and point-of-contact lookup. The system enhances productivity by automatically generating reports, suggesting tasks, and guiding new employees. The AI model is trained on data from Jira and internal chat groups, enabling it to understand real-world contexts and business processes.

At the core of the system is the DeepSeek model, fine-tuned using the LoRA (Low-Rank Adaptation) technique combined with Multi-Stage Fine-Tuning. LoRA keeps the original weights frozen and only trains a small number of low-rank parameters, reducing the number of trainable parameters by thousands of times compared to full fine-tuning [1]. The model is trained in multiple rounds, with matrices orthogonalized in each round according to specific strategies and objectives. This approach improves the model’s ability to absorb and process information effectively.

Acknowledgments

I would like to express my deepest gratitude to Associate Professor Dr. Phan Duy Hung, who guided me from the very beginning in shaping the topic and setting the research direction. This work would not have been possible without his dedicated and persistent support.

I also sincerely thank my colleagues who supported me in providing data and infrastructure for AI training.

Finally, I would like to extend my heartfelt appreciation to my family – those who have always been by my side, loving and supporting me unconditionally throughout this research journey.

# List of Figures

[**Figure 1.** Overall architecture 16](#_Toc201868378)

[**Figure 2.** Data taken from Jira 18](#_Toc201868379)

[**Figure 3.** Data taken from Jira 21](#_Toc201868380)

[**Figure 4.** Data generate from Chat GPT 22](#_Toc201868381)

[**Figure 5.** Training result 28](#_Toc201868382)

# List of Table

[**Table 1.** Data in Hugging Face Chat Template format 20](#_Toc201868386)

[**Table 2.** Average token length and quantity statistics by data source 26](#_Toc201868387)

[**Table 3.** Parameters of each round 27](#_Toc201868388)

Table of Contents

[Acknowledgments 2](#_Toc201868393)

[List of Figures 3](#_Toc201868394)

[List of Table 4](#_Toc201868395)

[Table of Contents 5](#_Toc201868396)

[Chapter 1 Introduction 6](#_Toc201868397)

[1.1. Problem And Motivation 6](#_Toc201868398)

[1.2. Related Works 7](#_Toc201868399)

[1.3. Contribution 9](#_Toc201868400)

[Chapter 2 Background Study 10](#_Toc201868401)

[2.1. Business Workflow in Enterprises 10](#_Toc201868402)

[2.2. DeepSeek model 11](#_Toc201868403)

[2.3. Low-Rank Adaptation 12](#_Toc201868404)

[2.4. Orthogonal trong Low-Rank Adaptation 13](#_Toc201868405)

[2.5. Chain of LoRa 14](#_Toc201868406)

[Chapter 3 System Design 16](#_Toc201868407)

[3.1. System Architecture 16](#_Toc201868408)

[3.2. Data Preprocessing 17](#_Toc201868409)

[3.2.1. Data extraction 17](#_Toc201868410)

[3.2.2. Data Formatting for Instruction-Tuning 20](#_Toc201868411)

[3.3. Modeling Module 22](#_Toc201868412)

[Chapter 4 Experiments And Results 26](#_Toc201868413)

[4.1. Data collection 26](#_Toc201868414)

[4.2. Experiments 27](#_Toc201868415)

[4.3. Results 28](#_Toc201868416)

[4.4. Conclusion And Future Work 29](#_Toc201868417)

[References 31](#_Toc201868418)

# Introduction

## Problem And Motivation

In today's enterprise environment, the demand for effective task tracking, resource allocation, and progress evaluation is growing rapidly, especially with the increasing scale of projects and the complexity of operational workflows. Although many tools such as Jira or Trello have been widely adopted, they primarily serve as manual tracking and storage systems, requiring frequent human intervention. This leads to a time-consuming and inconsistent process when it comes to compiling reports, suggesting tasks, or onboarding new employees.

Simultaneously, the rapid advancement of large language models (LLMs) such as ChatGPT, Grok, and DeepSeek presents new opportunities to automate work management tasks. DeepSeek is an open-source LLM family trained on datasets consisting of trillions of tokens, with enhanced reasoning capabilities achieved through multi-stage fine-tuning strategies [4]. However, to effectively leverage these models in specific enterprise environments, fine-tuning on internal data becomes essential. This introduces challenges in terms of resource efficiency, scalability, and the risk of losing the model's foundational knowledge if not trained properly [2].

In response to these practical needs, this thesis focuses on developing an artificial intelligence system that supports enterprises in tracking, analyzing, and optimizing internal operations. The system is based on DeepSeek, one of the most prominent open-source LLMs today, trained on large-scale datasets, capable of contextual reasoning, and with Vietnamese language support. However, to effectively apply the model within a specific business environment — where internal language, domain-specific terminology, and unique workflows exist — fine-tuning is imperative. Fine-tuning not only allows the model to adapt to organization-specific data but also unlocks the ability to automate a range of processes such as: generating work progress reports, suggesting context-aware tasks, and onboarding new staff through personalized guidance.

To address these challenges while maintaining model stability and efficient use of resources, this paper proposes the application of LoRA (Low-Rank Adaptation) during the fine-tuning of the DeepSeek model. LoRA introduced by [1], allows for a significant reduction in trainable parameters by inserting low-rank matrices into the pre-trained model architecture. As a result, fine-tuning can be conducted with memory and compute costs tens of times lower than full model tuning, while still maintaining high performance. Since the original weights remain untouched, LoRA-fine-tuned models retain foundational knowledge, thereby mitigating the effect of catastrophic forgetting [2].

LoRA also provides high flexibility for enterprise deployment: only the added weight components (known as adapters) need to be stored, rather than the full fine-tuned model. This reduces storage costs and simplifies multi-version deployment across departments. In this project, training data is sourced from platforms like Jira, along with internal log files documenting work progress, employee feedback, and task histories. This data is processed and structured into a standard input format for the model, enabling the training of LoRA adapters for specific tasks: generating consolidated work reports, suggesting role-specific tasks, and guiding new employees on how to handle assigned tasks.

Integrating the fine-tuned DeepSeek model into a work management system not only automates many critical processes, but also acts as an “internal assistant” capable of understanding context, recommending actions, and supporting real-time decision-making. This represents a crucial advancement in enhancing operational capacity, especially as businesses face the growing pressures of digital transformation and resource optimization.

## Related Works

In recent years, the application of large language models (LLMs) in enterprise support systems has gained significant attention. Models such as GPT, LLaMA, and more recently, DeepSeek, have demonstrated strong potential in processing natural language both flexibly and accurately. These capabilities open new directions for developing intelligent systems such as virtual work assistants, report summarization tools, and smart task suggestion engines. However, to deploy such models effectively in specific enterprise environments, fine-tuning on internal data becomes a critical requirement..

Beyond computational efficiency, another major issue in fine-tuning is catastrophic forgetting, which occurs when a model is fine-tuned multiple times or continuously updated over time. Study [2] explores the relationship between the number of training steps and the degradation of foundational knowledge, showing that even with techniques like LoRA, if there is no proper strategy for knowledge management, the model may still suffer from severe loss of pre-trained knowledge. Reference [6] provides a comprehensive survey of continual learning methods for LLMs, including rehearsal, regularization, parameter isolation, and hybrid approaches such as adapters or LoRA, all aiming to maintain long-term model performance.

One of the most promising approaches to reduce computational cost and mitigate catastrophic forgetting is introduced in[1], which presents LoRA (Low-Rank Adaptation). This method retains all the original weights of the pre-trained model and only trains two low-rank matrices at the attention layers. This drastically reduces memory and compute costs during fine-tuning. LoRA has been shown to achieve performance comparable to full fine-tuning in many NLP tasks while significantly reducing the risk of forgetting pre-trained knowledge. Thanks to its lightweight and flexible nature, LoRA has also become the foundation for various extensions such as QLoRA [5], which applies LoRA on 4-bit quantized models to fine-tune effectively on low-memory GPUs.

Among open-source LLMs, DeepSeek is a high-potential model family designed to support the research community in deploying customizable models efficiently. In their latest technical report, the DeepSeek team[4] trained models ranging from 1.3B to 67B parameters using high-quality multilingual datasets, optimized for logical reasoning tasks. DeepSeek-R1, their fine-tuned reasoning model, has achieved results on par with commercial models like OpenAI GPT-3.5 in many multi-step reasoning tasks [7]. Importantly, DeepSeek is released under a fully open-source license, making it a practical choice for enterprises that cannot afford access to proprietary commercial models.

Experimental documentation has also demonstrated the applicability of DeepSeek to domain-specific tasks. For example, in a tutorial published by DataCamp [8], the authors fine-tuned DeepSeek-R1 Distill (8B) using LoRA to build a medical chatbot capable of chain-of-thought reasoning. Although not directly related to work management, the data preprocessing and fine-tuning strategies used in this study are highly transferable to similar tasks such as report generation, progress analysis, or task recommendation in enterprise settings.

Existing works have laid an essential foundation for integrating LLMs into enterprise assistant systems. However, there remains a lack of research that concretely addresses work management problems, which require integration with internal data (e.g., Jira, chat logs), maintaining knowledge stability, and operating efficiently in resource-constrained environments. This paper builds on the established approaches and extends them by combining DeepSeek, various LoRA-based techniques, and real-world enterprise task data to construct a work management support system tailored to the needs of modern organizations.

## Contribution

This study focuses on the design and development of a work management system based on the DeepSeek large language model, with components fine-tuned using the Low-Rank Adaptation (LoRA) technique to ensure efficient deployment in resource-constrained enterprise environments.

The paper proposes an AI-based work management system architecture tailored for small and medium-sized enterprises (SMEs), in which data is collected from work management platforms such as Trello, Jira, and internal documentation sources to construct a comprehensive dataset. From this data, the model is trained using the Chain of LoRA technique, where in each training round, the vectors in matrix A or B are softly orthogonalized, depending on the objectives of that specific round. The orthogonality constraints can be adjusted dynamically to optimize performance.

This approach enhances the model’s ability to learn from limited datasets, extract insights from hard-to-learn samples, and maintain the ability to share and transfer knowledge across tasks.

# Background Study

To successfully design and implement an AI-powered work management system tailored to the specific needs of an enterprise, it is essential to establish a solid knowledge foundation—both in terms of existing operational workflows and the latest technical advancements in large language models (LLMs).

First, the structure and characteristics of common work management methodologies will be analyzed to identify integration points where AI models can be effectively embedded. Following that, core technical components such as the DeepSeek model, LoRA (Low-Rank Adaptation) fine-tuning technique, and multi-stage training strategies will be explored. These components form the basis for selecting appropriate technologies to build a system that is efficient, lightweight, and easily adaptable to real-world enterprise environments.

## Business Workflow in Enterprises

In modern enterprise environments, effective work management plays a crucial role in ensuring productivity and collaboration across departments. Organizations often deploy Workflow Management Systems (WfMS) such as Jira or Trello to automate and monitor complex operational processes. This makes it feasible to collect communication and coordination data between different teams.

However, extracting data from workflow systems introduces a key challenge. Tasks are often related to the company’s products or services, but at the same time, the internal communication and contextual understanding of those tasks vary across departments or units. As a result, the data collected tends to have both similarities and differences in the details of each task.

For instance, multiple teams may be referring to the same software using shared domain-specific terminology, yet with different perspectives: the BA (Business Analyst) team focuses on user interaction and functional usage, while the Dev (Developer) team requires in-depth understanding of system functionalities at the code level. This highlights a key requirement for the model — it must be able to precisely capture the shared references when different departments refer to the same feature, while also distinguishing the specific functional needs unique to each department.

## DeepSeek model

The DeepSeek-R1-Distill-Qwen-1.5B is a large language model (LLM) with 1.5 billion parameters, developed based on the Qwen architecture and fine-tuned from the original DeepSeek-R1 version. The model is designed to deliver high performance in logical reasoning, programming, and mathematical tasks, while being optimized for deployment on resource-constrained systems.

It adopts a decoder-only Transformer architecture with the following technical specifications:

* Number of layers (n\_layers): 24
* Model dimension (d\_model): 2048
* Intermediate layer size (d\_intermediate): 5504
* Number of attention heads (n\_heads): 16
* Number of key-value heads (n\_kv\_heads): 16
* Vocabulary size: 102,400 (byte-level BPE)
* Maximum context length: 4096 tokens

The DeepSeek-R1-Distill-Qwen-1.5B is a large language model (LLM) with 1.5 billion parameters, developed based on the Qwen architecture and fine-tuned from the original DeepSeek-R1 version. The model is designed to deliver high performance in logical reasoning, programming, and mathematical tasks, while being optimized for deployment on resource-constrained systems [10]:

* Recommended GPU: NVIDIA RTX 3070 or equivalent, with at least 8GB of VRAM
* Actual VRAM usage: Approximately 3.3GB
* Quantization support: Can be reduced to ~4GB VRAM with 4-bit quantization

These optimizations make it feasible for businesses to deploy the model on standard workstations or even on properly configured personal computers.

DeepSeek-R1-Distill-Qwen-1.5B has received high ratings in reasoning and programming evaluations, achieving performance comparable to much larger models such as Qwen3-235B-thinking. The model leverages chain-of-thought techniques from DeepSeek-R1 to enhance its reasoning ability, while maintaining strong efficiency and relatively low computational requirements.

With its balance between performance and deployability, DeepSeek-R1-Distill-Qwen-1.5B is a highly suitable choice for enterprise applications that require strong reasoning capabilities in a compact and efficient LLM.

## Low-Rank Adaptation

LoRA (Low-Rank Adaptation) is a parameter-efficient fine-tuning technique proposed by [1], aiming to minimize the number of parameters that need to be updated when fine-tuning large language models. In traditional fine-tuning, the entire weight matrix W of the model is updated during training, which requires substantial computational resources and memory—especially when the model has billions of parameters[1]. In contrast, LoRA keeps the original weights frozen and inserts additional low-rank matrices, significantly reducing both computation and memory costs during training.

Instead of directly updating the original weight matrix W, LoRA keeps W intact and adds a low-rank residual, denoted ΔW. For a weight matrix W∈R*d×k*, meaning d rows and k columns, LoRA introduces a new matrix computed as the product of two smaller matrices.

Where:

* A∈Rr×k
* B∈Rd×r
* r≪min(d,k): the rank of the update, representing the compression level.

Matrix A projects the input into a lower-dimensional subspace, capturing new learning directions of the original weight W. Matrix B expands this compressed representation back to the output space. As a result, all new information learned during fine-tuning is encoded into ΔW with very low parameter overhead.

Finally, the actual weight used in the model after applying LoRA becomes:

Where α is a scaling factor that ensures the overall influence of ΔW is balanced, regardless of the selected rank r. This design allows the rank r to be adjusted without disproportionately impacting the base model’s behavior [1].

In practice, there are several strategies to select an optimal α:

Determined AI recommends starting with α = r, then keeping this value fixed when changing the rank, avoiding the need to retune α [11]

Hugging Face AutoTrain documentation sets a default of α = 32 for rank = 16, and warns that setting α ≫ r can lead to overfitting [12].

Sebastian Raschka suggests a heuristic of α / r = 2 [13] to balance the strength of the update.

However, [14] found that using γr=α/r can slow down learning and reduce fine-tuning performance, especially with larger values of r. To address this, the author proposed a new approach called rsLoRA (rank-stabilized LoRA), which uses:

This helps stabilize the magnitude of ΔW as r increases, leading to improved convergence and training efficiency, without compromising the model’s reasoning ability. Kalajdzievski demonstrated that rsLoRA not only improves benchmark task performance, but also reduces computational resource requirements when fine-tuning large language models [14].

VUsing low-rank matrices allows LoRA-based models to achieve fine-tuning performance comparable to traditional methods, while requiring only 0.1% to 1% of the trainable parameters compared to full fine-tuning [1]. This makes LoRA especially useful in resource-constrained environments, such as training on CPUs or low-memory GPUs, enabling small and medium-sized businesses to train their own AI models affordably and effectively.

## Orthogonal in Low-Rank Adaptation

Orthogonalization in fine-tuning Large Language Models (LLMs) is a technique that forces the rows or columns of a matrix to be orthogonal to each other and normalized to unit length. This constraint compels the model to learn new directions instead of reusing existing ones, helping to prevent catastrophic forgetting[15] [16].

Studies such as Orthogonal Low-Rank Adaptation (O-LoRA) [15] propose splitting the data into separate tasks with distinct themes or objectives, and then training those tasks sequentially under the following principles:

* The matrix A for a new task must be softly orthogonal to the matrix A from previous tasks.
* The matrix A must also be orthogonal within itself, i.e., its row vectors should be mutually orthogonal.

By enforcing the learning directions (i.e., the rows of matrix A) of a new task to be orthogonal to those of previous tasks, O-LoRA ensures that the model does not overwrite previously learned knowledge, maintaining strong performance on older tasks without needing to revisit the original training data. Each task is thereby mapped into a distinct subspace, enabled by the orthogonal nature of the learned vectors. This allows the model to clearly distinguish between tasks, minimizing interference during multi-task training.

Moreover, because the learning space is constrained by orthogonality, the model is essentially forced to explore new representational directions, rather than overfitting by reusing previously adapted weights. This helps reduce the risk of overfitting, particularly for tasks with limited training data.

However, while orthogonalization is effective at preserving previous knowledge and reducing task interference through these constraints, it can also unintentionally limit the representational space the model can access. This becomes especially problematic in scenarios where tasks are closely related or share significant feature overlap—in such cases, forcing the model to learn in entirely new directions may prevent it from leveraging efficient, reusable representations.

As highlighted by [16] [17], models that are updated only within a restricted representational subspace, though resilient to interference, often exhibit reduced learning flexibility. They cannot fully adjust all weight components, which in turn limits their capacity to generalize or adapt optimally when tasks require overlapping or correlated features.

## Chain of LoRa

Chain of LoRA is a fine-tuning technique based on the principle of residual learning, proposed by [18] to overcome the limited generalization capacity of standard LoRA. While LoRA is resource-efficient, it can still underperform compared to full fine-tuning in certain tasks because it only updates a small subset of the model’s weights through low-rank matrices.

Chain of LoRA is inspired by the Frank-Wolfe algorithm in non-convex optimization, enabling a progressive increase in the rank of the weight updates without significantly increasing computational or memory costs. It addresses the learning capacity limitation by building a sequence of consecutive LoRA modules, where each module learns the residuals left unoptimized by the previous one—thus gradually approximating the optimal update.

The method unfolds the model across multiple stages:

* Tune LoRA: Train a new LoRA module on the current model weights.
* Tie a knot: Merge the newly learned weights into the backbone model.
* Extend the chain: Initialize a new LoRA adapter and continue learning the remaining residual.

This process is repeated until the full training objective is achieved. Chain of LoRA employs residual learning to iteratively enhance the model—each new adapter represents one step of fine-tuning in a multi-stage pipeline—without incurring extra memory or compute overhead. Additionally, Chain of LoRA introduces a rank decay strategy across stages, allowing the model to efficiently leverage previously learned knowledge while reducing training cost in later stages.

Compared to traditional LoRA, Chain of LoRA improves the model’s generalization ability by learning weight updates in residual increments, thereby approximating optimal weights more closely. Thanks to its chained structure, each iteration in Chain of LoRA focuses on the remaining information not yet captured in previous stages, enabling the model to fully utilize the representational space without overwriting prior learned patterns.

Merging the adapters into the backbone model after each round ensures constant memory usage, preventing the accumulation of redundant modules. Moreover, the rank decay strategy allows the model to operate with fewer parameters in later rounds while still maintaining output quality, effectively optimizing training cost.

This technique is particularly effective for complex tasks or small datasets, where dividing the learning objective into smaller, more optimizable steps increases training stability. Experiments on models such as OPT-1.3B and LLaMA-2-7B show that Chain of LoRA improves accuracy across all benchmark tasks (e.g., a 6.47% improvement on the WSC task) without increasing training cost compared to standard LoRA.

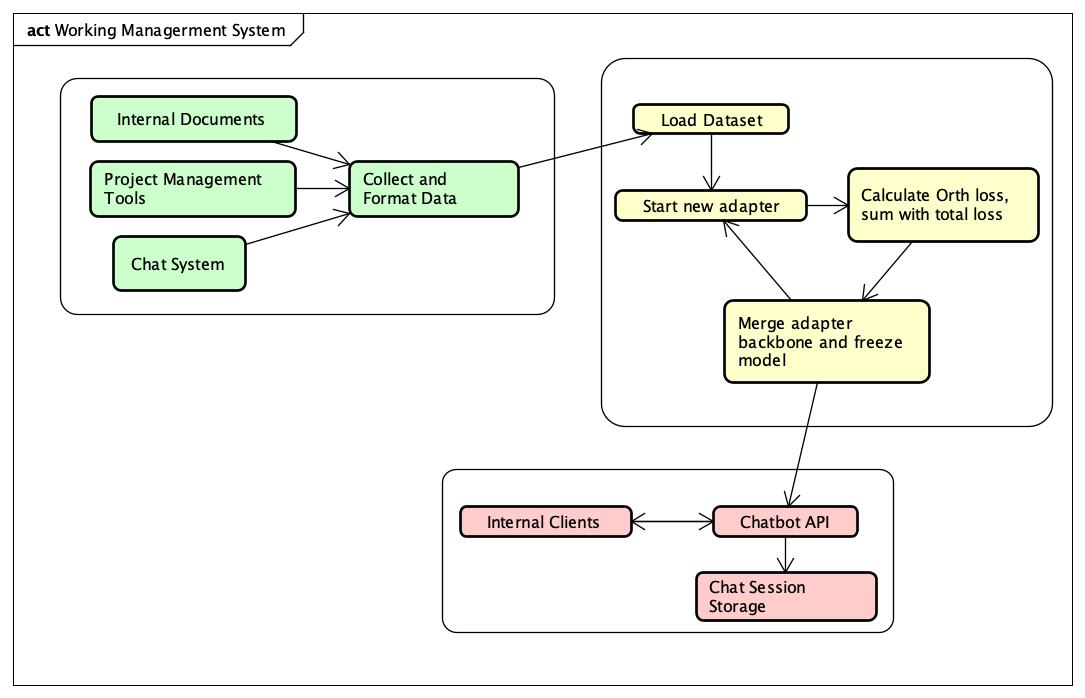
# System Design

## System Architecture

The internal chatbot system is designed using a three-tier architecture consisting of:

* Data Collection
* Model Processing & Training
* Inference Deployment via API

The overall architecture is illustrated in Figure 1.



**Figure 1.** Overall architecture

In the first tier, data is collected from three main sources:

* Internal documentation and training manuals
* Work management systems such as Jira or Trello
* Survey and interview data from internal personnel

Once collected, the data is standardized into question–answer pairs following an instruction-tuning format, which is suitable for training large language models (LLMs).

In the model training tier, the system applies the LoRA technique in combination with Chain of LoRA. The base model used is DeepSeek-R1-Distill-Qwen-1.5B, a lightweight and high-performance LLM with Vietnamese language support.

Training is conducted in multiple rounds, where each round adds a new LoRA adapter to learn the residual information not yet captured by previous adapters. During each round, the system computes an orthogonal loss and adds it to the total loss, ensuring that the newly learned directions (vectors) do not overlap with previous ones. This helps prevent catastrophic forgetting and enhances the model's ability to explore diverse learning directions.

In the inference tier, the fine-tuned model is stored and deployed via API using the Hugging Face Transformers platform. This API allows internal systems to query the chatbot and receive contextual responses. Additionally, it stores conversation history, which enhances the chatbot’s understanding of user context in future interactions.

To maintain high performance and context relevance, the system limits each inference prompt to the most recent 5–10 conversation turns, ensuring that prompt length remains efficient while retaining sufficient contextual information.

## Data Preprocessing

### Data extraction

The dataset is collected from three primary sources:

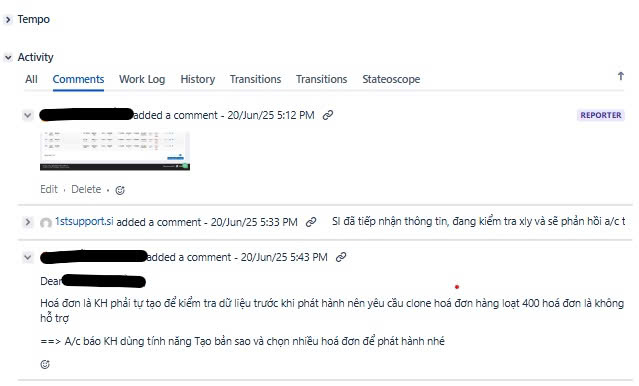
* Data extracted from work management systems such as Jira or Trello
* Data sourced from internal and publicly available documentation
* Data obtained from customer support groups

Data from work management systems typically has a high level of cleanliness, requiring only minimal preprocessing steps such as:

* Removing intermediary or redundant log entries
* Adjusting pronoun usage and role-specific references to fit the model's format

Once cleaned, this data can be directly used for AI training.

However, in organizations with multiple levels of management or communication, the data may include noisy comments from intermediate layers, which can disrupt the clarity of conversation flow — as illustrated in Figure 2.



**Figure 2.** Data taken from Jira

Comments from intermediate management levels often contain repetitive phrases or are system-generated with template-like structures. Common patterns include expressions such as "nhờ.\*hỗ trợ" ("please assist") or "xử lý.\*sớm" ("handle promptly"), which can account for more than half of the comment content. These patterns can be filtered effectively using query-based or regex-based filtering techniques.

For image-based data, although this paper does not cover specific implementation details, there are established techniques to convert images into text form. For example:

* Structured forms or entry forms can be extracted using tools like Tesseract + Layout parsing or LayoutLM, especially when the structure needs to be preserved.
* For images showing error messages or system displays, Tesseract OCR alone is sufficient to extract usable textual content.

A major limitation of data from work management systems is that many operational flows occur outside of Jira. For example, key information may be embedded in code files, manual system operations, or external platforms—while Jira only reflects status updates or summaries. As a result, only around 20% of the total collected data can be directly transformed into usable training samples.

Beyond data from systems like Jira and Trello, companies often maintain Excel-based FAQs used during product or service onboarding—either for customers or internal training. These are typically high-quality data sources, as the questions are already curated and redundant ones are filtered out, and the answers are often very detailed.

However, one limitation of such data is its time-sensitivity—for example, questions like "When will feature A or B be released?" can quickly become outdated. To improve dataset quality, the system should be able to detect and clean time-specific questions (e.g., phrases like "when will it be available", "planned release date", etc.) before training.

As for official documents, such as internal manuals, regulatory documents, and government decrees, these are the most factually accurate sources, especially legal texts. However, the main challenge lies in the generalization requirement, as such content is not naturally formatted as work-related interactions.

A simple and effective solution is to use large language models (e.g., ChatGPT, Claude, or Gemini) to generate structured training data in JSONL format. Based on real-world experimentation, the recommended approach is:

* Let the model read the full document first to build a knowledge foundation.
* Then instruct it with a fixed prompt to generate question–answer pairs in JSONL, where each line includes exactly one instruction and one response.
* Limit each generation session to 25–50 pairs to ensure data quality.
* Focus on a specific section or theme within the document to keep context consistent.

As for internal chat group data, due to privacy and security concerns, this paper does not include or analyze data from these sources.

### Data Formatting for Instruction-Tuning

Once the question–answer pairs are collected, the data needs to be formatted for training. This paper proposes using the Hugging Face Chat Template format [17]. This format supports multi-turn conversational data, commonly used in assistant-style LLMs. Data is structured as a list of {role, content} pairs, and the chat\_template function defined in the tokenizer is responsible for converting this list into a training prompt. The format helps the model learn natural conversation structures, better context retention, and supports tasks like question answering, data explanation, and function calling.

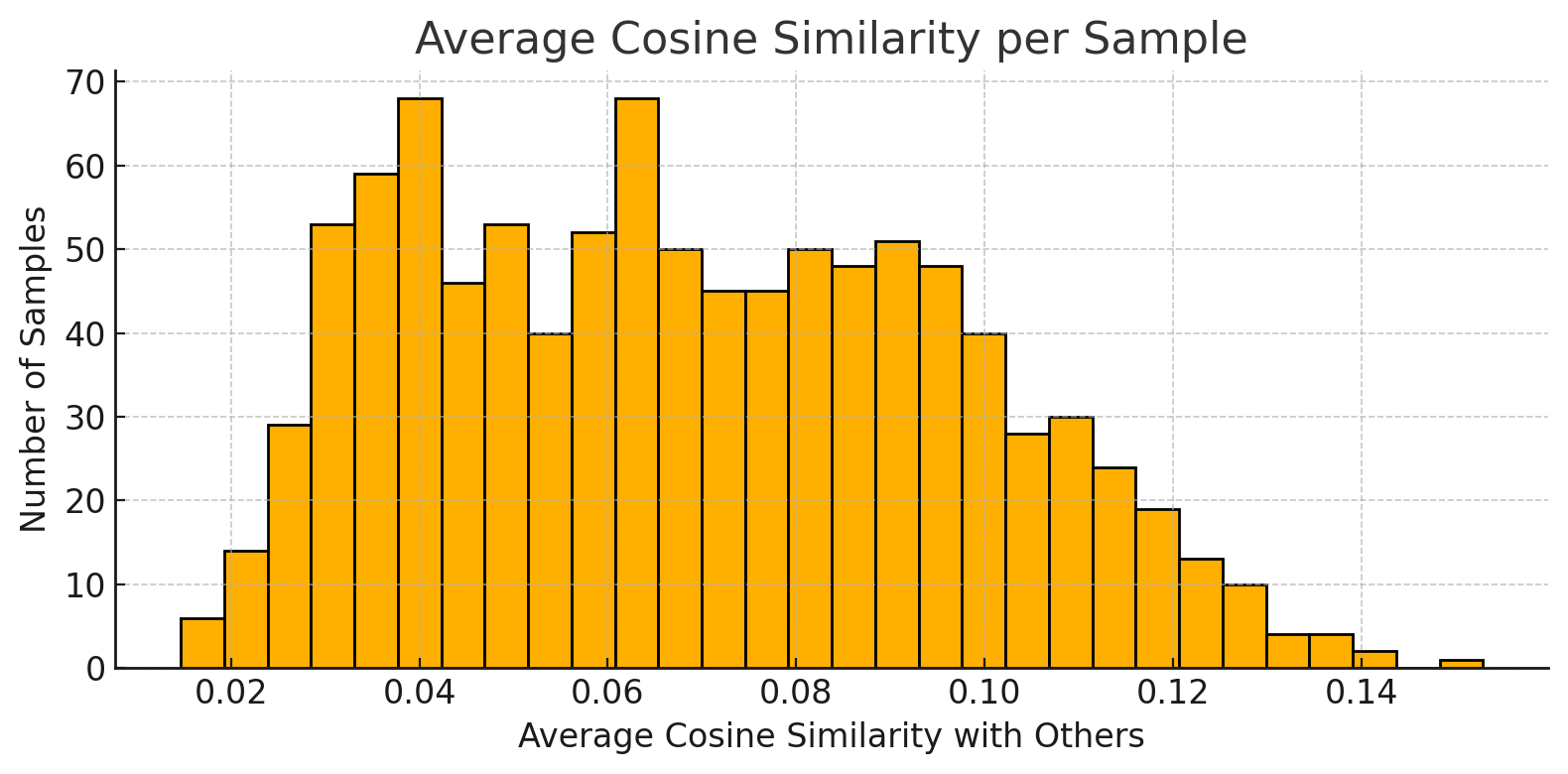
This format is currently adopted by models such as LLaMA2-chat, Mistral, and DeepSeek, and is natively supported by the Hugging Face Transformers library for training and inference.

**Table 1.** Data in Hugging Face Chat Template format

|  |
| --- |
| {"messages": [{"role": "user", "content": "How can I reset my password?"}, {"role": "assistant", "content": "To reset your password, click on 'Forgot password' on the login page and follow the instructions."}]}  {"messages": [{"role": "user", "content": "What is the return policy of your store?"}, {"role": "assistant", "content": "Our return policy allows returns within 30 days of purchase, with the original receipt and packaging."}]}  {"messages": [{"role": "user", "content": "Can you explain how invoices work in this system?"}, {"role": "assistant", "content": "Invoices record transactions and are automatically generated when a purchase is made. You can view them under the 'Billing' section."}]}  {"messages": [{"role": "user", "content": "Is there an API to check invoice status?"}, {"role": "assistant", "content": "Yes, you can use the `getInvoiceStatus` API by providing the invoice ID as a parameter."}]}  {"messages": [{"role": "user", "content": "Show me an example of a valid XML invoice payload."}, {"role": "assistant", "content": "<Invoice><ID>12345</ID><Date>2025-06-24</Date><Amount>100.00</Amount></Invoice>"}]} |

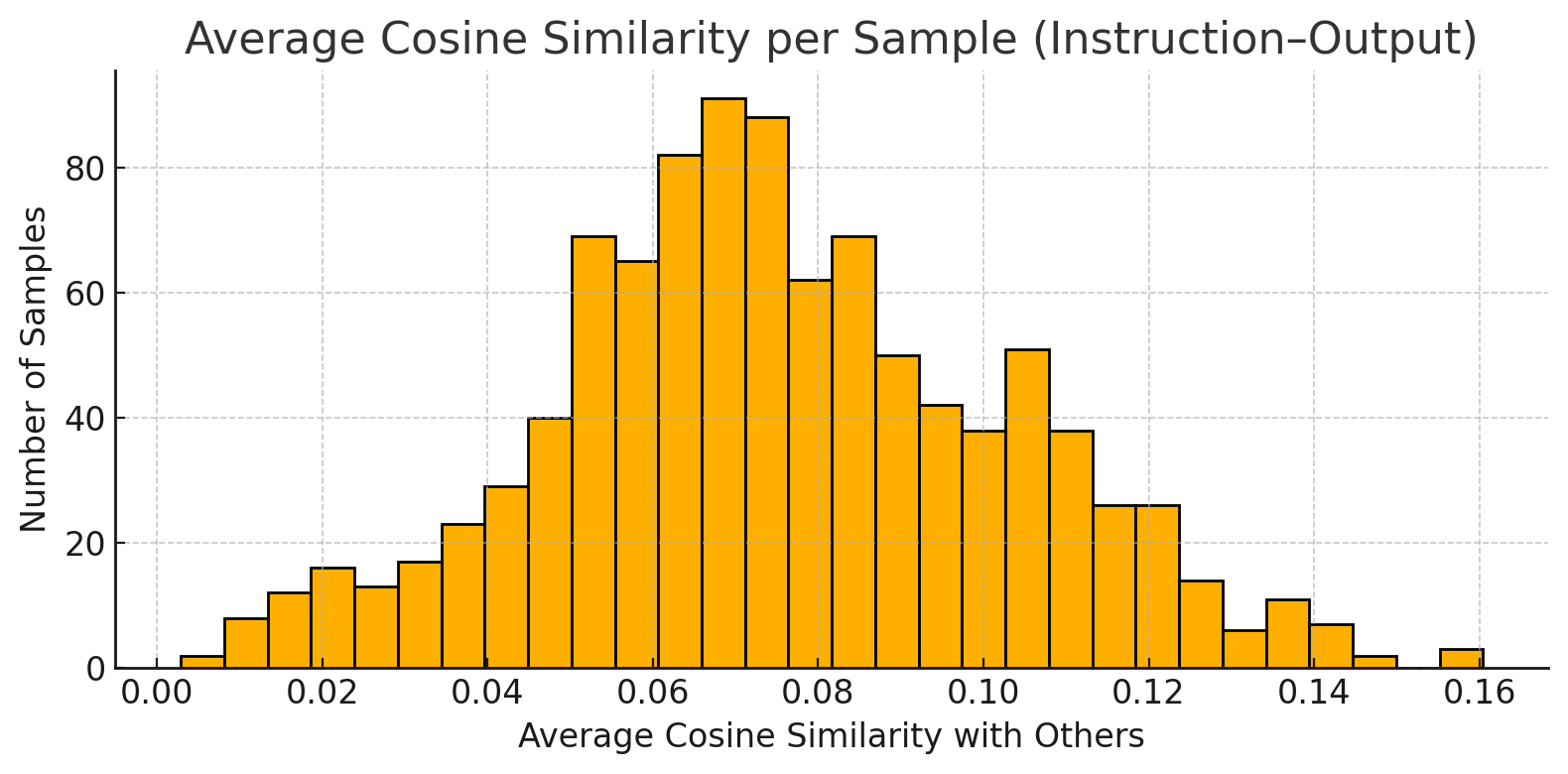
For data collected from work management systems, the task creator is assigned the "user" role, as they typically initiate the request or inquiry. Other participants are labeled as "assistant". Since these dialogues often involve multiple turns, consecutive messages by the same person are merged into a single message. This helps maintain coherent reasoning and allows the model to infer responses in a structured, user–assistant pattern, reducing confusion during training.

In one dataset of 1,000 samples, consisting of 300 from Jira and 700 from Excel-based FAQs gathered during software deployment, the cosine similarity between each sample and the rest was computed. The results showed a left-skewed distribution, with most similarity scores ranging from 0.04 to 0.10, indicating a high degree of diversity across samples and reducing the risk of overfitting. Only ~5–10% of samples had a similarity score above 0.12, the threshold where overlap or paraphrasing might begin to affect training.



**Figure 3.** Data taken from Jira

In contrast, data generated by ChatGPT, while structured according to a predefined prompt, exhibited more redundancy. Over 120 pairs had cosine similarity scores above 0.90, implying significant overlap or rewording of content. This is attributed to the model’s limited understanding of internal processes, domain-specific terminology, and operational context, which results in surface-level linguistic mimicry rather than genuine expertise. Despite this, ChatGPT-generated data remains valid for use in PEFT fine-tuning settings such as LoRA or CoLA, especially when used to enhance generalization.



**Figure 4.** Data generate from Chat GPT

To ensure the model retains both factual accuracy and expressive flexibility, this paper adopts the mixing strategy from[18], which recommends the golden ratio (61.8 : 38.2) for real-to-generated data. This ratio preserves the authenticity of real-world data while expanding the linguistic diversity through synthetic samples.

In this context, with 1,000 real samples, approximately 618 generated samples should be added, resulting in a dataset of around 1,618 total samples—optimizing model diversity without degrading fine-tuning quality.

## Modeling Module

The base model used is DeepSeek-R1-Distill-Qwen-1.5B, fine-tuned using a Chain of LoRA approach combined with orthogonality constraints, following the pipeline below:

* Tokenization & Data Formatting: Both real and generated data are preprocessed and formatted in a chat format compatible with Qwen models, then tokenized using DeepSeek's tokenizer.
* Orthogonality Loading: Before each round, the A matrices from previously trained adapters are reloaded to compute orthogonality constraints against the current adapter. In each round, a new LoRA module is initialized and trained while previous adapters remain frozen. The trained LoRA parameters are saved separately (as individual adapters) but are not merged into the backbone until the final stage. The rank is halved in each subsequent round (e.g., 16 → 8 → 4).
* Orthogonality Enforcement: During each round, the rows of the current A matrix must be orthogonal to each other (intra-module), and also orthogonal to A matrices from previous rounds (inter-module). The model computes an orthogonality penalty at each step and adds it to the training loss to gradually enforce this constraint.
* Final Merging: In the last round, all LoRA modules are merged into the base model to produce a standalone, fully integrated final model without dependency on external adapters.

In addition to typical hyperparameters like rank, learning rate, and epochs, two special parameters are introduced to ensure orthogonality constraints:

* Lambda internal: Controls the orthogonality strength among the row vectors of the same A matrix (within a LoRA module). A higher value enforces greater diversity in learned features.
* Lambda external: Controls orthogonality between the current A matrix and those from previous adapters. It prevents redundant learning and encourages the model to discover new directions in the representation space.

To optimize both efficiency and learning quality, orthogonality is only applied to certain modules, including:

* q\_proj: Influences query vectors in attention—orthogonalization ensures learning of diverse queries.
* k\_proj: Generates key vectors—orthogonality avoids overlapping key directions.
* v\_proj: Produces value vectors—orthogonality allows learning of distinct semantic values.
* gate\_proj: Controls information flow—orthogonal gates enable varied nonlinear transformations.

Orthogonality is not applied to modules like o\_proj (output projection), since these do not produce new representations and forcing orthogonality may distort the model’s learned output structure.

In early rounds, the model needs to build foundational representations. Hence, configurations are chosen to allow wide expressive capacity:

* High Rank: Ensures the ability to learn diverse and complex patterns.
* Higher Learning Rate: Accelerates parameter updates for newly initialized modules.
* Moderate Epochs: Prevents overfitting while ensuring enough time to learn structure.
* High Lambda internal: Encourages diversity among vectors in A, maximizing initial representational capacity.

In later rounds, the focus shifts to residual learning—capturing errors and features not fully learned. Settings aim for refinement and stability:

* Decreasing Rank: e.g., 16 → 8 → 4, under the assumption that residuals are simpler.
* Lower Learning Rate: Ensures smoother convergence and prevents overwriting prior knowledge.
* Increased Epochs: Later-stage learning is harder due to narrower representation space; more steps help stabilize convergence.
* Reduced Lambda internal: Diversity has already been established; moderate values avoid redundant vector learning.
* Increased Lambda external: Stronger orthogonality to ensure novel learning directions. However, in the final round, this may be reduced to 0 to allow overlap where necessary to capture any remaining information.

Since the model is fine-tuned on a relatively small dataset (<10,000 samples), the strategy is adjusted accordingly:

* Fewer LoRA Rounds (2–3): Ranks become too low in later rounds to meaningfully contribute.
* Higher Lambda internal & Lambda external: Stronger orthogonality enforcement helps avoid overfitting on limited data.
* Epochs Increase in Later Rounds: Smaller representation space and more difficult residuals require longer optimization to avoid premature convergence.
* Lower Alpha Scaling: To prevent over-amplification, the scaling factor α is computed based on the square root of the rank, as proposed in [14].

This training strategy allows the model to refine knowledge incrementally using residual learning, while maintaining orthogonal directions across rounds to avoid redundancy. For low-resource scenarios, it offers a balanced trade-off between generalization and expressive diversity, while conserving compute resources through rank control and targeted learning rates.

# Experiments And Results

## Data collection

Our dataset was collected from an internal Jira-based work management system and documentation related to the electronic invoice processing workflow. Among these, Excel-based FAQ files were gathered during the implementation of the e-invoicing system in accordance with Circular 70. Additionally, a portion of the data was automatically generated from official instructional documents released after 25/05/2025, using the ChatGPT model to ensure broad coverage and linguistic diversity within the dataset.

The data primarily focuses on two departments:

* Developer team: This subset includes materials related to the usage of the system’s base code for extending functionality, as well as documentation intended for third-party integration with the electronic invoice system.
* IT Help Desk: This subset focuses on customer support and procedural guidance regarding invoice management in accordance with Circular 70.

The dataset was constructed using the “golden ratio” split between real and generated data, consisting of 66.6% real-world data (1,000 samples) and 33.3% synthetic data (618 samples), totaling 1,618 samples.

After collection, the data underwent preprocessing, including deduplication and normalization, and was formatted into a standardized chat-based structure compatible with language model training. A summary of the dataset distribution and token-level statistics is provided in Table 2:

**Table 2.** Average token length and quantity statistics by data source

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Source | Number of Samples | Median | Longest | Average Length |
| Jira(Real data) | 550 | 96 | 367 | 111.12 |
| Excel | 450 | 101 | 185 | 102.84 |
| GPT Generate | 618 | 88 | 161 | 88.35 |
| Total | 1618 | 101 | 237.67 | 100.1 |

## Experiments

In the experimental setup, the model utilized was DeepSeek-R1-Distill-Qwen-1.5B, a lightweight yet high-performance variant within the DeepSeek family. This model supports Vietnamese language and exhibits strong logical reasoning capabilities. With 1.5 billion parameters and open-source availability, it is suitable for deployment and fine-tuning in small to medium-sized enterprise environments without requiring high-end computational infrastructure.

The training process followed the Chain of LoRA strategy, in which each round trains a new LoRA module on top of the updated model from the previous round. These LoRA modules progressively learn the residual components of the weight space that have not yet been optimized. Additionally, orthogonality constraints were applied to ensure representational diversity and reduce redundancy across rounds. The hyperparameters for each training round are detailed in Table 3.

**Table 3.** Parameters of each round

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Round | Rank | Epochs | Learning Rate | λ internal | λ external | Reason |
| 1 | 16 | 5 | 2e-5 | 0.1 | 0.0 | Initial round for learning foundational representations |
| 2 | 8 | 7 | 1.5e-5 | 0.05 | 0.05 | Second round for capturing residual patterns |
| 3 | 4 | 9 | 1e-5 | 0.0 | 0.1 | Final round for fine-tuning remaining minor errors |

Due to practical constraints on cost and feasibility, the training was conducted on an Apple MacBook platform using an M-series chip with an integrated GPU supporting Metal Performance Shader (MPS). Although not as powerful as high-end NVIDIA GPUs, the MPS framework proved sufficient for training medium-scale language models like DeepSeek-R1-Distill-Qwen-1.5B, provided that batch size and token lengths are kept within reasonable limits.

Environment specifications:

* Device: MacBook Pro M2 with 16GB RAM
* Software stack: Python 3.10, PyTorch 2.x, Transformers 4.41+, PEFT 0.9, Datasets
* MPS Configuration: Fallback enabled and memory watermark adjusted to avoid out-of-memory errors during training.

## Results

The model was trained across three rounds, with the loss consistently decreasing to a range of approximately 3.0–4.0, stabilizing within that interval. In Round 1, the loss dropped sharply from 86.98 to 2.96, indicating that the adapter quickly acquired foundational knowledge. Round 2 started at 27.97, gradually decreased, and converged around 3.0, showing that the second adapter reinforced the learning direction established in the first round and contributed to smoother representations. The final adapter only slightly reduced the loss from 8.03 to around 4.0. A slight increase in final loss across later rounds can be attributed to the use of lower rank values, which limit representational capacity, and the enforcement of orthogonality constraints, which further restrict learning flexibility. Nonetheless, given the constraints of the dataset and token limits, this level of loss is considered acceptable. Moreover, the loss not declining too steeply suggests that the model avoids overfitting.

A graph of a graph

AI-generated content may be incorrect.

**Figure 5.** Training result

Testing on a held-out test set—consisting of paraphrased or duplicated questions derived from the training data—revealed distinct differences in how well the model absorbed specific types of knowledge:

* For definition-based knowledge (e.g., "What is an invoice?" or "What is a cash register invoice?"), the model provided accurate or near-accurate definitions.
* For scenario-based questions (e.g., customer name mismatch, incorrect invoice totals), the model generally offered appropriate handling guidance. However, for more complex scenarios, such as issuing a replacement invoice that is still incorrect, the responses remained vague and did not meet expected specificity.
* For error messages with lengthy and complex content (e.g., "-1: Tax ID, form code, invoice number are not unique – check the data format"), the model failed to comprehend the structure and provided irrelevant answers—likely due to limitations in contextual understanding and sentence parsing.
* For technical coding questions related to .NET or React, the model demonstrated minimal learning. Nonetheless, it began to recognize certain terminology, such as BaseController. For questions involving error codes like ERR, comprehension remained limited under standard settings. However, increasing the number of epochs significantly (e.g., 10–15) allowed the model to understand specific codes such as ERR:5. For closely related error codes (e.g., ERR:1, ERR:10–19), the model either produced redundant responses or generalized its answers.

These findings suggest that larger models with more parameters may be capable of learning and responding accurately to such complex queries.

## Conclusion And Future Work

The model successfully achieved its primary objective of acquiring foundational knowledge and responding appropriately to support- and development-type scenarios, particularly for structured and short-form inputs. Although it did not reach strong generalization capabilities across all types of technical errors or queries, the model demonstrated potential—especially when exposed to higher training epochs—by correctly learning non-redundant error codes such as ERR:5 or ERR:7. The noticeable performance improvement across training rounds validates the effectiveness of the chained training strategy and orthogonal LoRA in enhancing representational diversity while mitigating overfitting.

Several factors may have contributed to the model’s limited learning capabilities:

* Limited contextual understanding, particularly in multi-layered or logic-intensive queries. This is partly due to short token lengths and partly due to the small-scale architecture, which lacks sufficient capacity for deep semantic representation.
* Technical error codes or code-related queries (e.g., .NET, React) were not effectively learned. This may be attributed to insufficient task-specific training data or the need for programming domain knowledge, which this model is not explicitly designed to handle—unlike code-specialized versions such as DeepSeek Coder.
* Token pattern overlaps in certain error codes (e.g., ERR:1 and ERR:10–19) may introduce noise into the learned vector space, leading to misinterpretations or repetitive responses.

Trong With access to larger datasets and more computational resources, training with larger-scale models (e.g., DeepSeek-R1 7B or Qwen 7B) could significantly enhance context comprehension and representation capacity. This would enable better discrimination between syntactically similar but semantically distinct error patterns.

Furthermore, separating tasks such as development and support into distinct training objectives—while applying parallel orthogonal training strategies—could improve task specialization without sacrificing shared knowledge transfer. This multi-task orthogonal learning setup may result in more robust and domain-aligned conversational agents.

# References

|  |  |
| --- | --- |
| [1] | Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, "LoRA: Low-Rank Adaptation of Large Language Models," 17 Jun 2021. [Online]. Available: https://arxiv.org/abs/2106.09685. [Accessed 12 05 2025]. |
| [2] | D. Kalajdzievski, "Scaling Laws for Forgetting When Fine-Tuning Large Language Models," 11 Jan arxiv. [Online]. Available: https://arxiv.org/abs/2401.05605. [Accessed 13 05 2025]. |
| [3] | Hong Liu, Saisai Gong, Yixin Ji, Kaixin Wu, Jia Xu, Jinjie Gu, "Boosting LLM-based Relevance Modeling with Distribution-Aware Robust Learning," 17 Dec 2024. [Online]. Available: https://arxiv.org/abs/2412.12504. [Accessed 13 05 1015]. |
| [4] | DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang,..100 additional authors not shown), "DeepSeek-V3 Technical Report," 2025 . |
| [5] | Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," 23 May 2023. [Online]. Available: https://arxiv.org/abs/2305.14314. [Accessed 25 May 2025]. |
| [6] | Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, Hao Wang, "Continual Learning of Large Language Models: A Comprehensive Survey," 25 Apr 2024 . [Online]. Available: https://arxiv.org/abs/2404.16789. [Accessed 25 May 2025]. |
| [7] | DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi. (100 additional authors not shown), "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," 22 Jan 2025. [Online]. Available: https://arxiv.org/abs/2501.12948. [Accessed 25 May 2025]. |
| [8] | "Fine-Tuning DeepSeek R1 (Reasoning Model)," 27 Jan 2025. [Online]. Available: https://www.datacamp.com/tutorial/fine-tuning-deepseek-r1-reasoning-model. [Accessed 25 May 2025]. |
| [9] | E. Page, "Fine-Tuning LLMs using Intel Xeon CPUs," 18 Mar 2025. [Online]. Available: https://lenovopress.lenovo.com/lp2179-fine-tuning-llms-using-intel-xeon-cpus. [Accessed 25 May 2025]. |
| [10] | DeepSeek-AI, "DeepSeek-R1-Distill-Qwen-7B," [Online]. Available: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B. |
| [11] | Sze Wai Yuen, Liam Li, Kevin Musgrave, "Finding the best LoRA parameters," 11 Sep 2024. [Online]. Available: https://www.determined.ai/blog/lora-parameters. [Accessed 7 Jun 2025]. |
| [12] | H. Face, "Seq2Seq Parameters," [Online]. Available: https://huggingface.co/docs/autotrain/en/seq2seq\_params. [Accessed 07 Jun 2025]. |
| [13] | S. Raschka, "Practical Tips for Finetuning LLMs Using LoRA (Low-Rank Adaptation)," 19 Nov 2023. [Online]. Available: https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms. [Accessed 08 Jun 2025]. |
| [14] | D. Kalajdzievski, "A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA," 28 Nov 2023. [Online]. Available: https://arxiv.org/abs/2312.03732. [Accessed 08 Jun 2025]. |
| [15] | Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, Xuanjing Huang, "Orthogonal Subspace Learning for Language Model Continual Learning," 22 Oct 2023. [Online]. Available: https://arxiv.org/abs/2310.14152. [Accessed 08 Jun 2025]. |
| [16] | Quan Cheng, Yuanyu Wan, Lingyu Wu, Chenping Hou, Lijun Zhang, "Continuous Subspace Optimization for Continual Learning," 17 May 2025. [Online]. Available: https://arxiv.org/abs/2505.11816. [Accessed 17 Jun 2025]. |
| [17] | Ruiyu Wang, Sen Wang, Xinxin Zuo, Qiang Sun, "Lifelong Learning with Task-Specific Adaptation: Addressing the Stability-Plasticity Dilemma," 08 Mar 2025. [Online]. Available: https://arxiv.org/abs/2503.06213. [Accessed 18 Jun 2025]. |
| [18] | Wenhan Xia, Chengwei Qin, Elad Hazan, "Chain of LoRA: Efficient Fine-tuning of Language Models via Residual Learning," 08 Jan 2024. [Online]. Available: https://arxiv.org/abs/2401.04151. [Accessed 08 Jun 2025]. |
| [19] | K. Miltos, H. Robin and M. Arnab, "Metro railway safety: An analysis of accident precursors," *Safety Science,* vol. 50, no. 7, pp. 1535-1548, 2012. |
| [20] | Y. Cao, Y. An, S. Shuai, G. Xie and Y. Sun, "A statistical study of railway safety in China and Japan 1990–2020," *Accident Analysis & Prevention,* vol. 175, 2022. |