NEFMind: Parameter-Efficient Fine-Tuning of Open-Source LLMs for Telecom APIs Automation

Zainab Khan*†, Ahmed Hussain†, Mukesh Thakur‡, Arto Hellas*, and Panos Papadimitratos†
*Alto University, School of Science, Espoo, Finland
†Networked Systems Security (NSS) Group – KTH Royal Institute of Technology, Stockholm, Sweden
‡Ericsson, Finland
Corresponding author: ahmed.hussain@ieee.org

***Abstract*—The use of Service-Based Architecture in modern telecommunications has exponentially increased Network Functions (NFs) and Application Programming Interfaces (APIs), creating substantial operational complexities in service discovery and management. We introduce *NEFMind*, a framework leveraging parameter-efficient fine-tuning of open-source Large Language Models (LLMs) to address these challenges. It integrates three core components: synthetic dataset generation from Network Exposure Function (NEF) API specifications, model optimization through Quantized-Low-Rank Adaptation, and performance evaluation via GPT-4 Ref Score and BertScore metrics. Targeting 5G Service-Based Architecture APIs, our approach achieves 85% reduction in communication overhead compared to manual discovery methods. Experimental validation using the open-source Phi-2 model demonstrates exceptional API call identification performance at 98-100% accuracy. The fine-tuned Phi-2 model delivers performance comparable to significantly larger models like GPT-4 while maintaining computational efficiency for telecommunications infrastructure deployment. These findings validate domain-specific, parameter-efficient LLM strategies for managing complex API ecosystems in next-generation telecommunications networks.**

***Index Terms*—Generative AI, Large Language Models, Telecom, Network Exposure Function, 5G, Parameter-Efficient Fine-Tuning**

I. Introduction

The evolution of 5G networks has fundamentally transformed how network services are designed and deployed, particularly through the adoption of the Service-Based Architecture (SBA) [1]. Standardized by the 3rd Generation Partnership Project (3GPP), SBA has fundamentally altered how network functionalities are exposed and managed through Network Functions (NFs). An important element is the Network Exposure Function (NEF), which facilitates the exposure of network capabilities through a REST Application Programming Interface (API). However, the increase of these APIs, coupled with their increasing complexity, presents significant operational challenges for system administrators and developers.

The challenge manifests itself in two ways. First, the exponential growth in the number of APIs accompanying new NFs and underlying functionalities has created a vast and complex ecosystem that is increasingly difficult to navigate. Second, the technical intricacies involved in correctly identifying and implementing appropriate API calls require substantial expertise and time investment, often leading to inefficiencies and potential errors in system operations. Traditionally, system administrators analyze user queries, navigate through extensive API documentation, and formulate appropriate API calls. This process is not only time-consuming but also prone to human error, potentially impacting service reliability and system performance. We address these challenges through the use of Large Language Models (LLMs). Specifically, we explore the potential of fine-tuning open-source LLM, namely phi-2, to automate the process of interpreting user queries and generating appropriate NEF API calls.

Contribution. In this paper, we present three key contributions: First, we develop *NEFMind*, a framework encompassing model fine-tuning utilizing Q-Low-Rank Adaptation (QLoRA) and generating synthetic datasets derived from API specifications using Generative AI. Second, we evaluate model performance through multiple metrics, including GPT-4 Ref Score for accuracy assessment and BertScore for response similarity measurement, providing a reliable evaluation for understanding the capabilities and limitations of our framework. Finally, we demonstrate that fine-tuned LLMs can effectively automate interactions with APIs while maintaining high accuracy and computational efficiency. Our experimental results show significant improvements in API call automation accuracy, with a fine-tuned model achieving 98-100% accuracy compared to 4-10% in the baseline (i.e., non-fine-tuned) Phi-2 model.

Paper Organization. The remainder of this paper is organized as follows: Section II discusses the preliminaries and relevant related work of API automation and LLM applications. Section III describes our methodology and proposed framework (*NEFMind*), including the technical implementation of our fine-tuning approach. Section IV provides an in-depth performance evaluation, analyzing the quality of response via correctness and accuracy metrics. Section V highlights the key distinctions between this study and prior research efforts. Finally, Section VI concludes the work with directions for future research.

II. Preliminaries and Related Work

A. Web Services and REST APIs in 5G Networks

This 3GPP-standardized framework facilitates network functionality exposure through NFs [2], utilizing Representational

State Transfer (REST) principles for distributed communication [3]. REST architecture's core principles—uniform interface, client-server decoupling, statelessness, cacheability, layered systems, and code on demand—establish the foundation for standardized HTTP-based communication while ensuring architectural separation [3], [4]. In the telecommunications context, these principles are implemented within NEF APIs, enabling secure network capability exposure [5]. This approach is particularly essential within 5G SBA [6], as illustrated by the CAMARA project [7] that demonstrates the evolution toward microservice-based API architectures in telecommunications.

B. Large Language Models

LLMs fundamentally rely on the Transformer architecture [8], which revolutionized Natural Language Processing (NLP) through parallel sequence processing and contextual understanding via self-attention mechanisms [9]. This architecture integrates multi-head attention layers, position-wise feed-forward networks, residual connections, and layer normalization to enable sophisticated language processing capabilities. Our research examines Phi-2 [10], an open-source model featuring 2.7 billion parameters and a 2048-token context window that demonstrates efficient performance through training on diverse datasets encompassing Python code and educational content.

Model Adaptation Techniques. Two primary adaptation paradigms have emerged for domain-specific optimization: Retrieval Augmented Generation (RAG) and Fine-Tuning. RAG enhances model responses through dynamic knowledge integration from external sources, while fine-tuning [11] directly modifies model weights for specialized domain performance. Zhang et al. [12] demonstrates the critical relationship between data quality, model architecture, and training methodology in determining fine-tuning effectiveness.

Parameter-Efficient Fine-Tuning for Automation. PEFT techniques [13] have emerged as highly effective approaches for specialized task adaptation without computational overhead. QLoRA [14] represents a significant advancement in this domain, enabling efficient model fine-tuning by updating only specific parameter subsets rather than complete model weights. This approach maintains performance comparable to full fine-tuning while dramatically reducing computational requirements, particularly excelling in code understanding and domain-specific tasks.

Integration of LLMs with APIs. Contemporary frameworks demonstrate diverse methodological approaches to LLM-API integration with varying performance characteristics. RestGPT [15] achieves 70-75% success rates through a three-module framework architecture, while Gorilla [16] focuses on comprehensive API collections via self-instruction fine-tuning and retrieval-aware training methodologies. ToolLLM [17] provides enhanced generalization capabilities for comprehensive API integration, and ReST Meets ReAct [18] advances the field through self-improving agent architectures combining retrieval-based approaches with reactive planning.

III. Methodology and Proposed Framework

To address the challenges of automating NEF API interactions, we define several key requirements derived from the practical constraints of enterprise telecommunications systems and the need for reliable API automation: (i) Utilization of open-source LLMs to promote research reproducibility and extensibility, (ii) Development of comprehensive in-context knowledge about NEF APIs for accurate query processing, and (iii) Creation of optimal training datasets from API specifications.

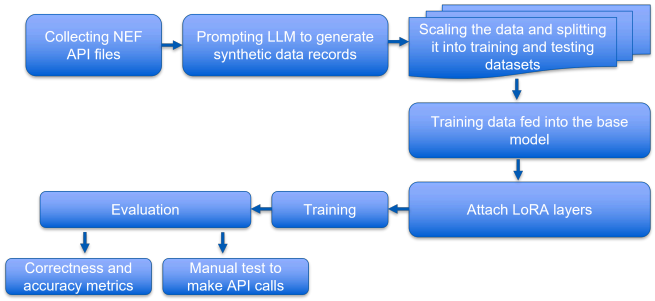


Fig. 1: Implementation pipeline consisting of multiple processes such as synthetic data generation and scaling, fine-tuning the LLMs and their evaluation.

Experimental Setup. We utilized Amazon G5 instances to evaluate *NEFMind*. Specifically, our setup employed the g5.4xlarge instance configuration for comprehensive model operations, encompassing data generation, training, and evaluation of the Phi-2 architecture. The g5.4xlarge configuration consists of a single GPU with 24GB of dedicated GPU memory, 16 virtual CPUs, and 64GB of system memory. This instance provides 600GB storage capacity with network and EBS bandwidth capabilities of 25Gbps and 8Gbps, respectively.

Design and Framework Implementation. The proposed methodology addresses the aforementioned requirements through a systematic approach incorporating model fine-tuning of state-of-the-art language model–Phi-2–utilizing a specialized dataset derived from the NEF API specification documentation [19]. To establish comprehensive performance benchmarks, we conduct a comparative analysis of the fine-tuned model and a RAG architecture implemented with GPT-4. The implementation adopts a modular pipeline architecture, where each component functions as an integral part of a sequential processing chain, with output artifacts serving as input parameters for subsequent processing stages. As illustrated in Fig. 1, building the framework encompasses four primary components: the initial dataset collection and synthetic data generation process, followed by data scaling and preprocessing operations, subsequent model fine-tuning utilizing QLoRA, and concluding

with systematic evaluation and performance assessment protocols. NEFMind specifically addresses telecommunications infrastructure constraints through parameter-efficient adaptation, requiring 60% less computational resources than comparable full fine-tuning approaches while maintaining domain-specific accuracy.

Dataset Generation. The effective integration of telecommunication API knowledge into LLMs necessitates a specialized fine-tuning process utilizing domain-specific datasets. We employ NEF API specifications [5] as the foundational knowledge source for model adaptation. Further, we leverage NEF API specification files in Yet Another Markup Language (YAML) format for synthetic data generation, implementing a systematic approach to dataset development. Below we discuss the methodology encompassing data generation, processing mechanisms, and scaling, establishing a robust foundation for model fine-tuning and subsequent performance evaluation.

Generating Synthetic Data. Initially, we utilized the original dataset [19] consisting of the YAML specification files of NEF APIs. To facilitate model comprehension, we flattened one of the YAML files, as each API specification file contains references to other files, with no external references beyond the file itself. Next, we formulated a prompt for the LLM to generate synthetic data in JavaScript Object Notation (JSON) format, encompassing specific fields including user queries (Request), endpoint specifications (API call, Method, Operation), contextual information (Description), and required parameter configurations (Parameters). For this task, we employed GPT-4 [20] as the expert model to ensure the highest quality of data. Although the API specification delineates schema definitions for seven API endpoints and the prompt specified the return of solely real data, the LLM (GPT-4) produced some fabricated and unreal JSON objects.

After data generation, the data validation and refinement are performed. Manual inspection and verification of the GPT-4 generated dataset identifies and eliminates instances of synthetic data artifacts that did not align with actual NEF API specifications. We isolated seven JSON objects that demonstrated complete fidelity to the authentic API specifications. This refinement ensures the dataset's validity and establishes a reliable foundation for subsequent model training and evaluation processes. Fig. 2 illustrates the generated JSON object after the refinement process.

Data Processing and Scaling. After obtaining seven records of data from the previous step, we recognized the inadequacy of this dataset for effective model training. Consequently, we opted to prompt the GPT-4 model to expand the dataset: We provided the model with the request field of each JSON record and requested it to generate 100 unique variations of each request. This process resulted in the generation of 765 records of data. The prompt for scaling the data is given in Fig. 3. Subsequently, we partitioned this expanded dataset roughly into a 70/30 ratio for training and evaluation purposes,


```
{
   "request": "How can I obtain an access token for future requests?",
   "api_call": "/api/v1/login/access-token",
   "description": "OAuth2 compatible token login, get an access token for
        future requests",
   "method": "post",
   "operation": "login_access_token_api_v1_login_access_token_post",
   "parameters": {
      "grant_type": "password",
      "username": "string",
      "password": "string",
      "scope": "string",
      "client_id": "string",
      "client_secret": "string"
   }
},
{
   "request": "How can I read active subscriptions?",
   "api_call": "/api/v1/3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions
        ",
   "description": "Get subscription by id",
   "method": "get",
   "operation": "
        read_active_subscriptions_api_v1_3gpp_as_session_with_qos_v1
   __scsAsId__subscriptions_get",
   "parameters": {
      "scsAsId": "string"
   }
}
```


Fig. 2: Generated synthetic JSON data for API requests and their required parameters.

resulting in approximately 535 records in the training dataset and 230 in the evaluation dataset. To facilitate the model training, we converted the JSON data into Comma Separated Values (CSV) format, as it was deemed suitable for the model trainer's comprehension. Additionally, for Phi-2, we structured each data record into *Instruct-Output* pairs. The Instruct field contained the value from the request parameter, while the Output object encompassed the remaining parameters such as API call, description, method, operation, and parameters. The complete process of data generation is depicted in Fig. 4.

```
Understand the given JSON object and
    generate the request parameter in 100
    different ways. All of them must be
    unique and not redundant.
    {json_object}

You may also use the document provided as
    context to understand more. Feel free
    to rephrase the request parameter.
{context}

The format of output must be an array of
    100 values in the following format:
[request1, request2, ..., request100]
```

Fig. 3: Prompt for data scaling.

Fine-Tuning Phi-2. Phi-2 is a Transformer-based architecture comprising 2.7 billion parameters originally trained on a diverse dataset encompassing Python textbook content, coding

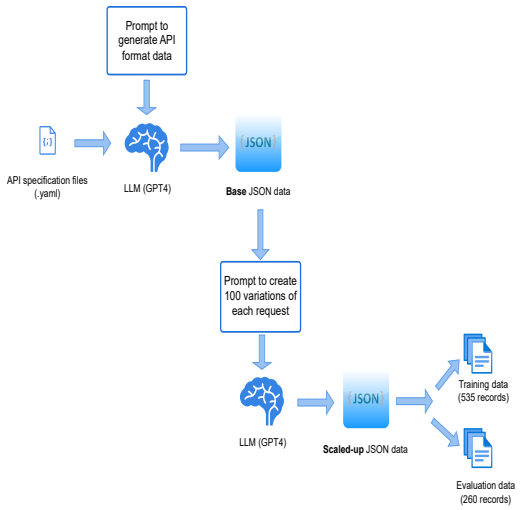


Fig. 4: Synthetic data generation pipeline incorporating GPT-4-based question-answer pair generation from NEF API specifications, iterative data augmentation, and dataset partitioning for model evaluation.

data, NLP-based synthetic texts, and educational website content. We utilize Phi-2 for fine-tuning telecommunication-based API datasets, specifically selected for its moderate parameter scale and being open-source, enabling performance analysis in domain-specific applications. The fine-tuning implemented a systematic approach encompassing multiple integrated phases. Initially, the training dataset underwent formatting procedures, incorporating appropriate labeling schemas and implementing the *Instruct/Output* format structure. Following data preparation, the formatted Comma Separated Values (CSV) dataset was integrated into the processing pipeline. The base Phi-2 model was subsequently initialized without quantization parameters but incorporating FlashAttention-2 optimization, enhancing the model's attention mechanism efficiency for extended sequence processing while optimizing computational resource utilization and memory management. The model initialization phase was followed by tokenization and applying the QLoRA configurations. The final implementation phase utilized HuggingFace's Supervised Fine-Tuning Trainer (SFT) framework [21], integrating the prepared model architecture with the specified QLoRA configurations (Table I) and training parameters (Table II).

TABLE I: QLoRA Configuration

Argument	Value
Low-Rank Adaptation (LoRA) alpha	16
LoRA dropout	0.1
LoRA rank	64
Target modules	q_proj, k_proj, v_proj, dense, fc1, fc2
Bias	None
Task type	CAUSAL_LM

The model achieved a training runtime of approximately 595 seconds, processing 4.495 training samples per second and executing 1.504 training steps per second, with a total of Floating-Point Operations (FLOs) of 6.08×10^{15}. The final training loss of 0.1921 indicates an optimal balance between computational efficiency and model performance, suggesting Phi-2's particular suitability for scenarios prioritizing rapid fine-tuning capabilities despite potentially higher loss metrics compared to alternative architectures.

TABLE II: Training parameters provided to SFT from HuggingFace to fine-tune the model.

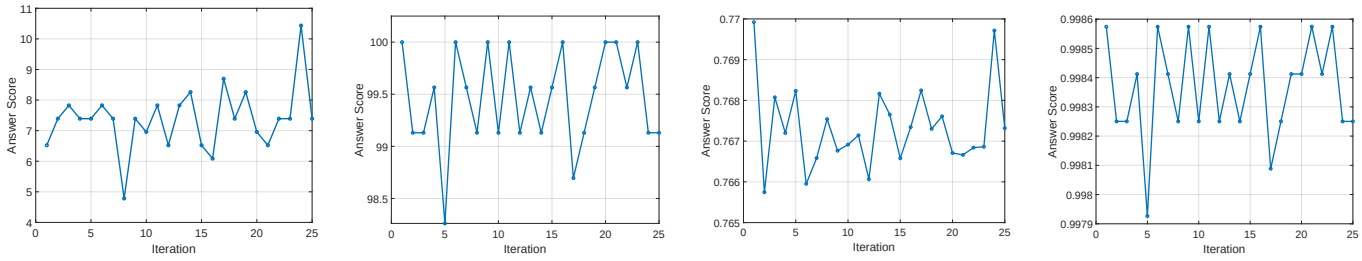
Argument	Value
Epochs	5
Batch size	3
Gradient accumulation steps	1
Optim	paged_adamw_32bit
Save steps	10
Logging steps	10
Learning rate	2×10^{-4}
Weight decay rate	0.001
Warmup ratio	0.03
BF16	True
Max grad norm	0.3
Max steps	-1
Group by length	True
Scheduler type	Constant
Reports to	Tensorboard

IV. Performance Evaluation

We evaluate the performance of the baseline (i.e., Phi-2) and fine-tuned open-source LLM, utilizing two performance metrics: accuracy and response similarity. The evaluation comprises 230 records extracted from a comprehensive dataset of 765 entries [19], incorporating three fundamental components: NEF API endpoint identification queries, GPT-4 reference responses as evaluation benchmarks, and comparative model outputs from baseline and fine-tuned variants. We employ 25 iterative evaluations, enabling granular analysis of the accuracy and similarity of the generated responses. Each evaluation iteration generates structured JSON artifacts encompassing prediction datasets with performance metrics, BertScore-based semantic similarity quantification, and GPT-4 Ref Score comparative analysis of accuracy [22] for reference-based benchmarking.

Baseline Model Evaluation Framework. The baseline Phi-2 evaluation implements a RAG-based methodology to address inherent domain-specific knowledge limitations. Our processing pipeline systematically integrates API specifications through semantic segmentation utilizing LangChain's recursive text splitter [23], followed by HuggingFace embedding generation and Facebook AI Similarity Search (FAISS)-based similarity retrieval [24]. The architecture integrates LangChain's Q&A framework [25] with task-specific prompt structures, generating evaluation outputs in structured JSON format. Analysis reveals significant formatting inconsistencies and semantic inadequacies in baseline model responses, demonstrating the critical necessity for domain-specific adaptation.

Fine-Tuned Model Assessment Protocol. The fine-tuned Phi-2 architecture processes NEF API queries independently



(a) (b) (c) (d)

Fig. 5: Evaluation metrics across 25 iterations comparing base Phi-2 and fine-tuned Phi2-NEF models. (a) GPT-4 Ref Scores for base Phi-2, (b) GPT-4 Ref Scores for Phi2-NEF, (c) BERTScore for base Phi-2, and (d) BERTScore for Phi2-NEF.

TABLE III: Performance metrics summary across twenty-five evaluation iterations for baseline and fine-tuned Phi-2 models.

	GPT-4 Ref Score		**BertScore**	
	Phi-2	**Phi2-NEF**	**Phi-2**	**Phi2-NEF**
Maximum	10.4348	100	0.7699	0.9986
Minimum	4.7826	98.2609	0.7657	0.9979

without supplementary contextual information. Systematic evaluation using 230 records validates model performance through HuggingFace pipeline integration with task-specific tokenization. Response formatting protocols convert string-format outputs into structured JSON objects, systematically documenting responses under designated keys alongside corresponding inputs and ground truth references.

Performance Metrics and Analysis. Our dual-metric assessment protocol employs GPT-4 Ref Score [22] for accuracy quantification and BertScore for semantic similarity analysis. Each metric executes across 25 iterations, generating comprehensive performance characterization as presented in Table III.

GPT-4 Reference Score Analysis. Utilizing Generative Pre-Trained Transformer (GPT)-4 [20], [22] as the expert validation model, accuracy assessment reveals stark performance disparities. Baseline Phi-2 demonstrates accuracy ranges of 4.78-10.43 on a 0-100 scale, while fine-tuned variants achieve 98.69-100.00 ranges. This substantial improvement validates the effectiveness of domain-specific training methodologies, as illustrated in Figures 5a and 5b.

BertScore Semantic Similarity Assessment. BertScore evaluation [26] employs BERT-based contextual embeddings with HuggingFace frameworks, transforming natural language into vector representations for pairwise cosine-similarity analysis. Baseline Phi-2 achieves similarity scores within 0.765-0.769 ranges on a 0-1 scale, while fine-tuned variants demonstrate 0.997-0.998 ranges. These results, visualized in Figures 5c and 5d, validate our curated prompting strategy's effectiveness in generating well-structured, consistent NEF API interaction outputs.

Empirical API Integration Validation. Despite superior response generation performance, direct API integration testing reveals implementation challenges. Validation employs LangChain's OpenAPI agent framework [27] with standardized testing credentials, implementing systematic query interpretation and execution planning. The agent architecture, fundamentally based on OpenAPI Specification (OAS), demonstrates the fine-tuned model's limitations in producing deterministic, semantically consistent outputs within integrated frameworks. These observations highlight potential agent integration incompatibilities and architectural constraints requiring systematic investigation in future research iterations.

Our comprehensive evaluation demonstrates that baseline Phi-2 exhibits significant performance limitations due to insufficient domain-specific knowledge, while fine-tuned architectures achieve superior performance through specialized NEF API training. However, the limited training corpus scale introduces potential overfitting risks, necessitating expanded datasets in future development cycles to maintain model generalization capabilities.

V. Comparison with Existing Solutions

Several notable research efforts have explored the integration of LLMs with APIs, each contributing distinct methodological approaches and domain applications that provide valuable context for understanding our work's positioning within the field.

RestGPT [15] represents a foundational approach to LLM-API integration through its prompt-based framework architecture. This system employs three interconnected modules—a planner, API selector, and executor—designed to interpret complex tasks, decompose them into actionable steps, and execute appropriate RESTful API calls across general-purpose applications. While RestGPT demonstrates broad applicability using OpenAI's text-davinci-003 model with sophisticated prompt engineering techniques, our research is different in both domain specificity and methodological approach. We focus exclusively on telecommunications APIs, enabling deeper domain expertise and more targeted optimization.

Furthermore, our implementation leverages fine-tuning techniques on open-source models rather than relying solely on prompt engineering, ensuring greater accessibility for researchers and independence from proprietary systems. This

distinction yields substantial performance improvements: while RestGPT achieved 70-75% success rates for correct API calls, our approach demonstrates 98-100% prediction accuracy within the telecommunications domain.

Gorilla [16] explores the integration of LLMs with RESTful APIs. However, significant technical and evaluative differences distinguish their and our approach. Gorilla employs self-instruction fine-tuning combined with retrieval-aware training techniques, whereas our methodology centers on Parameter-Efficient Fine-Tuning (PEFT) strategies. Both utilize open-source model foundations; Gorilla refining LLaMA-7B-based architectures while we similarly leverage accessible model resources. The evaluation frameworks differ distinctly: Gorilla directly benchmarks its fine-tuned model against GPT-4, demonstrating superior API functionality performance and reduced hallucination errors.

Conversely, our study employs GPT-4 exclusively as a teacher model, focusing evaluation efforts on comparing fine-tuned models with their baseline counterparts rather than direct comparison with proprietary systems.

ToolLLM [17] aligns closely with our objectives through its comprehensive framework for bridging LLMs with external knowledge via RESTful API interactions. Their methodology encompasses complete data construction, model training, and evaluation pipelines. The primary distinction lies in dataset scope and domain strategy: ToolLLM aggregates diverse datasets from RapidAPI Hub spanning 49 categories, pursuing broad multi-domain applicability.

In contrast, our approach deliberately constrains focus to the telecommunications industry, enabling specialized optimization and targeted performance enhancements within this critical infrastructure domain.

VI. Conclusion

In this paper, we presented *NEFMind*, a framework for automating NEF API interactions through domain-specific fine-tuning of open-source LLMs. Our analysis of Phi-2 architecture, when applying Parameter-Efficient Fine-Tuning, demonstrates significant performance enhancements in API call automation. The fine-tuned architecture achieved 98-100% accuracy (compared to 4-10% baseline) and improved BertScore metrics from 0.76 to 0.99 while maintaining computational efficiency through QLoRA implementation. Notably, the performance of the Phi-2 architecture (2.7B parameters) suggests viable deployment in telecommunications infrastructure. While these findings establish the efficacy of fine-tuning for API call generation and demonstrate the viability of parameter-efficient models for domain-specific tasks, challenges persist in security implementation and cross-domain generalization. Future research should address these limitations while enhancing architectural frameworks and fine-tuning methodologies.

References

[1] L. Xia *et al.*, "5G Service Based Core Network Design," in *2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)*, 2019, pp. 1–6.

[2] Network Exposure - Enable 5G Innovation. [Online]. Available: https://www.ericsson.com/en/core-network/network-exposure

[3] V. B. Surwase, "REST API Modeling Languages - A Developer's Perspective," *International Journal For Science Technology And Engineering*, vol. 2, pp. 634–637, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:54773316

[4] "REST API: Key Concepts, Best Practices, and Benefits." [Online]. Available: https://www.altexsoft.com/media/2021/03/rest_api_works.png

[5] "Network Exposure - Enable 5G Innovation." [Online]. Available: https://www.ericsson.com/en/core-network/network-exposure

[6] J. van Zyl, "A Perspective on Service Based architecture," in *Proceedings of SAICSIT*, vol. 249, 2002.

[7] J. Ordonez-Lucena and F. Dsouza, "Pathways towards network-as-a-service: the CAMARA project," in *Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration*, ser. NAI '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 53–59. [Online]. Available: https://doi.org/10.1145/3538401.3546825

[8] A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[9] N. Heidloff. Foundation Models, Transformers, BERT and GPT. [Online]. Available: https://heidloff.net/article/foundation-models-transformers-bert-and-gpt/

[10] microsoft/phi-2 · Hugging Face. [Online]. Available: https://huggingface.co/microsoft/phi-2

[11] C. Jeong, "Domain-specialized LLM: Financial fine-tuning and utilization method using Mistral 7B," *Journal of Intelligence and Information Systems*, vol. 30, no. 1, p. 93–120, Mar. 2024.

[12] B. Zhang *et al.*, "When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method," 2024.

[13] J. Liu *et al.*, "An Empirical Study of Parameter-Efficient Fine-Tuning Methods for Pre-Trained Code Models," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 397–408.

[14] T. Dettmers *et al.*, "QLoRA: Efficient Finetuning of Quantized LLMs," *Advances in Neural Information Processing Systems*, vol. 36, pp. 10 088–10 115, Dec. 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html

[15] Y. Song, W. Xiong *et al.*, "RestGPT: Connecting Large Language Models with Real-World RESTful APIs," 2023.

[16] S. G. Patil, T. Zhang *et al.*, "Gorilla: Large Language Model Connected with Massive APIs," 2023.

[17] Y. Qin, S. Liang *et al.*, "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs," 2023.

[18] R. Aksitov, S. Miryoosefi *et al.*, "ReST meets ReAct: Self-Improvement for Multi-Step Reasoning LLM Agent," 2023.

[19] 3rd Generation Partnership Project (3GPP), "5gc_apis: Restful apis of main network functions in the 3gpp 5g core network," https://github.com/jdegre/5GC_APIs, 2024, accessed: 2025-01-05.

[20] "OpenAI. GPT-4 and GPT-4 Turbo." [Online]. Available: https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo

[21] "Supervised Fine-tuning Trainer." [Online]. Available: https://huggingface.co/docs/trl/en/sft_trainer

[22] A. Karapantelakis *et al.*, "Using large language models to understand telecom standards," in *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. IEEE, 2024, pp. 440–446.

[23] Recursively split by character | LangChain. [Online]. Available: https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/

[24] Faiss | LangChain. [Online]. Available: https://python.langchain.com/v0.2/docs/integrations/vectorstores/faiss/

[25] Q&A with RAG | LangChain. [Online]. Available: https://python.langchain.com/v0.1/docs/use_cases/question_answering/

[26] T. Zhang, V. Kishore *et al.*, "BERTScore: Evaluating Text Generation with BERT," 2020. [Online]. Available: https://arxiv.org/abs/1904.09675

[27] create_openapi_agent — LangChain documentation. [Online]. Available: https://python.langchain.com/v0.2/api_reference/community/agent_toolkits/langchain_community.agent_toolkits.openapi.base.create_openapi_agent.html