On-device System of Compositional Multi-tasking in Large Language Models

Ondrej Bohdal[1], Konstantinos Theodosiadis[2], Asterios Mpatziakas[2], Dimitris Filippidis[2], Iro Spyrou[2], Christos Zonios[2], Anastasios Drosou[2], Dimosthenis Ioannidis[2], Kyeng-Hun Lee[3], Jijoong Moon[3], Hyeonmok Ko[3], Mete Ozay[1], Umberto Michieli[1]

[1]Samsung R&D Institute UK, United Kingdom, [2]CERTH, Greece,
[3]Samsung Research, South Korea

Correspondence: o.bohdal.1@samsung.com

Abstract

Large language models (LLMs) are commonly adapted for diverse downstream tasks via parameter-efficient fine-tuning techniques such as Low-Rank Adapters (LoRA). While adapters can be combined to handle multiple tasks separately, standard approaches struggle when targeting the simultaneous execution of complex tasks, such as generating a translated summary from a long conversation. To address this challenge, we propose a novel approach tailored specifically for compositional multi-tasking scenarios involving summarization and translation. Our technique involves adding a learnable projection layer on top of the combined summarization and translation adapters. This design enables effective integration while maintaining efficiency through reduced computational overhead compared to alternative strategies requiring extensive retraining or sequential processing. We demonstrate the practical viability of our method within an on-device environment by developing an Android app capable of executing compositional tasks seamlessly. Experimental results indicate our solution performs well and is fast in both cloud-based and on-device implementations, highlighting the potential benefits of adopting our framework in real-world applications demanding high-speed operation alongside resource constraints.

1 Introduction

Generative AI has gained significant attention thanks to its ability to generate useful content (Gozalo-Brizuela and Garrido-Merchán, 2024) across modalities, including text (Zhao et al., 2023; Minaee et al., 2024), images (Yang et al., 2024c; Cao et al., 2024; Shenaj et al., 2025) and videos (Zhou et al., 2024). Most generative AI applications to date rely on remote servers with advanced hardware. Nevertheless, there has been growing interest in harnessing on-device generative AI capabilities (Xu et al., 2024). On-device AI offers several advantages, particularly enhanced privacy since sensitive data remain securely stored on the device without transmission over networks (Dhar et al., 2021). For text-based applications, compact yet proficient language models (LLMs)–typically ranging from 1B to 3B parameters–have emerged as viable options for deployment on mobile devices (Xu et al., 2024). Fine-tuning these pre-trained models using low-rank adaptation (LoRA) techniques (Hu et al., 2022) significantly boosts their effectiveness across various tasks such as translation and summarization (Mao et al., 2025).

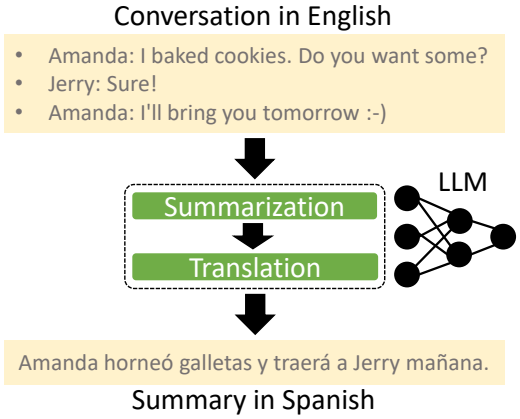

Figure 1: Compositional multi-tasking on the combination of summarization and translation. In our fully on-device system, we focus on the scenario where a conversation (Gliwa et al., 2019) in one language is summarized in another language.

A recent practical application of on-device LLMs concerns the so-called *compositional multi-tasking*, which entails performing multiple tasks simultaneously (Bohdal et al., 2025). Examples include generating translated summaries or adjusting tones in message replies. To tackle this challenge, researchers proposed a learnable calibration mechanism that combines the corresponding LoRA parameters and then corrects the combination via a small number of additional parameters. Remark-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 416–424
November 4-9, 2025

ably, these supplementary components require minimal storage space relative to standalone LoRAs (*e.g.*, 0.5%). We propose and implement an alternative efficient strategy that adds a projection layer on top of combined adapters. However, our primary goal is not to propose a new method that surpasses state-of-the-art compositional multi-tasking approaches in (Bohdal et al., 2025).

In this paper, our main goal is to develop a fully on-device system implementing the compositional multi-tasking setup and to provide an associated on-device evaluation. We discuss details of how we developed an application running fully on-device for this conceptual setting, how the application works, and an experimental comparison between server and smartphone settings. More specifically, we focus on the combination of summarization and translation tasks, in an applied setting where we summarize messages that users receive on their phones. The considered scenario is illustrated in Figure 1. We note that this is a highly practical use-case as it can be beneficial, for example, when people move abroad and join local chat groups that use the local language. The tool enables users to easily see the summary of the conversation in their own language.

2 Related Work

2.1 On-device LLMs

Large Language Models (LLMs) typically include billions of parameters, with the largest models currently containing over 400B+ parameters (Dubey et al., 2024). Running these models, even only for inference, is costly and requires multiple high-end GPUs (Borzunov et al., 2024). In practice, this necessitates sending data to remote servers for generating suitable outputs. However, LLMs would also be highly beneficial in cases where private data (*e.g.*, messages) are used and when users prefer not to send data to the cloud (Dhar et al., 2021). As a result, smaller LLMs of sizes such as 1B or 3B have been developed, making it possible to deploy LLMs directly on mobile devices. In these cases, all computations run locally, avoiding data transmission to remote servers and also reducing operational costs for service providers. Various models suitable for on-device deployment have been developed, for example, LLaMA 3.2 1B (Dubey et al., 2024), Qwen2.5 1.5B (Yang et al., 2024a; Qwen Team, 2024), StableLM2 1.6B (Bellagente et al., 2024) and Gemma 2B (Team et al., 2024).

2.2 Multi-tasking in LLMs

LLMs can perform diverse tasks after their large-scale pre-training (Zhao et al., 2023; Minaee et al., 2024), but for strong performance on individual tasks, they are typically fine-tuned via parameter-efficient fine-tuning (PEFT) (Han et al., 2024; Ding et al., 2023). Fine-tuning is especially beneficial for on-device LLMs that have more limited resources. A common strategy for PEFT is the use of low-rank adapters (LoRA) that are injected into selected layers (Hu et al., 2022), introducing only a comparatively small number of parameters (*e.g.*, 10M for a model of size 1B). These LoRA parameters are then loaded into a shared LLM to perform individual tasks (Mao et al., 2025). It is common to store the single-task LoRAs on the device alongside the base LLM model (Gunter et al., 2024). In order to perform multi-tasking, the fine-tuned models can be merged into each other. Various strategies exist, including simple linear merging that computes a weighted average of the weights (Wortsman et al., 2022; Ilharco et al., 2023), TIES merging (Yadav et al., 2024), and more advanced learnable strategies such as LoraHub (Huang et al., 2024). Such model merging strategies have been shown to work well when doing multiple tasks separately (Yang et al., 2024b). However, they have been shown not to work well in compositional multi-tasking where specialized approaches need to be developed. Well-performing naïve baselines for compositional multi-tasking are inefficient as they either require training a new specialized LoRA or performing two inference passes with the LLM in sequence. In this work, we develop an on-device system that integrates compositional multi-tasking.

3 Problem Statement and Method

We focus on compositional multi-tasking, where two tasks are combined to be performed jointly. More specifically, we consider the summarization of messages in English as the primary task T_1 with the secondary task T_2 of translation from English to Spanish. The compositional task performs $T^C_{1,2}(x) = T_2(T_1(x))$, where $x \mapsto y_1 \mapsto y_2$ and a generic task T takes input text x and outputs text y; *i.e.*, $T(\cdot) : x \mapsto y$.

We assume that an LLM model and LoRAs for tasks T_1, T_2 parameterized by B_1, A_1 and B_2, A_2 are already stored on the device. More specifically, LoRA (Hu et al., 2022) introduces low-rank factorized matrices $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$ where the

rank $r \ll \min(d, k)$. Here, parameters k, d specify the input and output dimensions for the given layer respectively. Then, A and B are combined with model's weights $W_0 \in \mathbb{R}^{d \times k}$, resulting in an adjusted forward pass:

$$h = (W_0 + \Delta W)x = (W_0 + BA)x. \quad (1)$$

Various baselines can be considered: (i) a *zero-shot* approach where the model is prompted about the task, (ii) either the *primary-task LoRA* or the *secondary-task LoRA* paired with prompting, (iii) various merging strategies applied on the LoRA parameters (*e.g.*, *linear* (Wortsman et al., 2022), *concatenation* (Mangrulkar et al., 2022), *TIES* (Yadav et al., 2024) and *LoraHub* (Huang et al., 2024), (iv) inefficient baselines that perform well: *two-step LoRA usage* and *joint-expert LoRA* for the specific compositional task.

The goal is to obtain performance comparable to the inefficient baselines while being more efficient, in particular only requiring one inference pass and introducing only a limited number of additional parameters instead of a new LoRA that would use *e.g.* 50MB in storage.

We propose a new strategy that learns very few additional specialized parameters, using data $\mathbb{D}^C$ from the compositional task $T^C_{1,2}$. The inputs x are conversations in English, while the targets y are ground-truth summaries that have been translated from English to Spanish via a specialized translation model. These parameters are pre-trained on a server and then deployed to the device. Our technique adjusts the forward pass $h = (W_0 + \Delta W)x$ with update matrix ΔW computed as:

$$\Delta W = P_2 P_1 (0.5 B_1 A_1 + 0.5 B_2 A_2). \quad (2)$$

We refer to it as *projection merge* because it projects the average of the individual LoRA parameters via additional low-rank parameters $P_2 \in \mathbb{R}^{d \times s}, P_1 \in \mathbb{R}^{s \times k}$ for $s \ll \min(d, k)$. These projection parameters are shared across layers as well as components that have the same input and output dimensions (*e.g.* key, value attention projections). An overview of the method is provided in Figure 2.

Our solution has negligible storage overhead, unlike the *joint-expert LoRA* approach that requires storing a new LoRA. Further, the compute requirements are similar for both as our approach involves multiplying one LoRA matrix by the projection matrix. This introduces only negligible additional cost, as the overall runtime is dominated by inference with the base model.

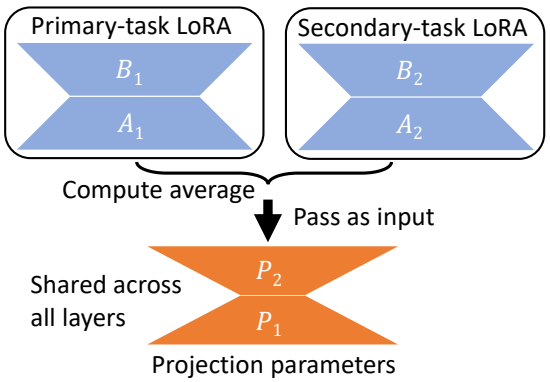

Figure 2: Overview of our projection merge method. We first merge the single-task LoRAs and then pass them through projection parameters.

4 System Framework

4.1 Proof-of-Concept Application on Server

We developed our application in two phases. First, we created a proof-of-concept (PoC) application running on a server using Python and React Native. The system consists of a React Native user client and a FastAPI server that manages communications between the user and the LLM. We used the PEFT Python library (Mangrulkar et al., 2022) to load a Llama-3.2-1B-Instruct (Dubey et al., 2024) model and the LoRA adapters, while the Transformers library (Wolf et al., 2019) handled the tokenizer and end-to-end LLM pipeline.

The application supports three methods for the joint task of summarization and translation: (i) our proposed projection merging method (*Merged LoRA*); (ii) base LLM model with prompting (*zero-shot*); and (iii) sequential LoRA adapter use (*two-step*) where one adapter is loaded to handle summarization and a second one later to handle translation of the summary. The system can also run automated experiments comparing all three methods, calculating ROUGE scores (Lin, 2004) and inference times. Figure 3 demonstrates the two-step method (left) and a comparison of all three methods (right). For easier English-reader assessment, the summary in the left panel has been translated back to English. All example conversations are from the SAMSum dataset (Gliwa et al., 2019).

4.2 On-device Application

4.2.1 Framework Components

For the on-device implementation, we built the back-end in Rust and the front-end in React Native.

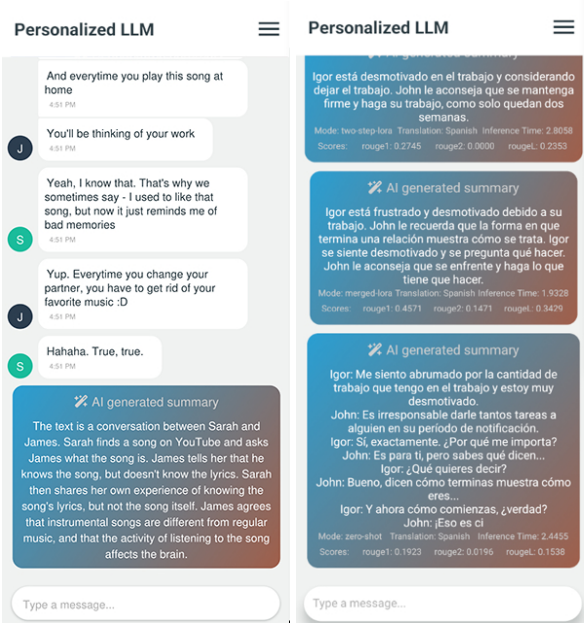

Figure 3: The user client for the PoC server-side application. The application operates either in a predefined mode such as two-step LoRA usage (left) or in an experimental mode where all methods are utilized and relevant metrics are reported (right). Tapping on the summary translates it back to English, which has been used in the example on the left side.

We used the *mistral.rs* package (Buehler, 2025) for LLM and adapter management, and the *axum crate* (tokio rs, 2024) for front-end/back-end communication. Figure 4 shows the system's high-level architecture. We describe each component of the architecture next. The *user interface* either receives an input dialogue from the user or can load example dialogues from the dataset so that the user does not need to enter long conversations. It communicates with the *LLM communication end-point*. The functionalities offered by the user interface are elaborated in Subsection 4.2.2. The LLM communication endpoint handles all communications between the front-end and back-end of the application, *e.g.* it is connected to the *dialogue emulator* that loads example dialogues and realistically replays them. It is also connected to the *inference API* that is responsible for handling the LLM prompt, calculation of metrics, and most importantly doing the actual inference that outputs the translated summary of the dialogue. Finally, *LLM setup and adapter handling* tackles the task of loading the LLM model and adapters while allowing for the dynamic configuration of these components.

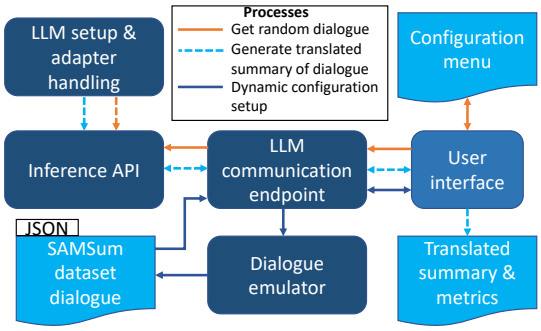

Figure 4: High-level architecture of our on-device system for compositional multi-tasking.

4.2.2 User Interface

The user interface provides a configuration menu and displays the translated summary and metrics below the conversation, as shown in Figure 5. Key features include: (i) method selection; (ii) task configuration, *i.e.*, whether to run summarization only or with translation; (iii) optional random conversation generation; (iv) single or comparative method evaluation mode, *i.e.*, whether to run one experiment with the selected method or with all methods; (v) target language selection.

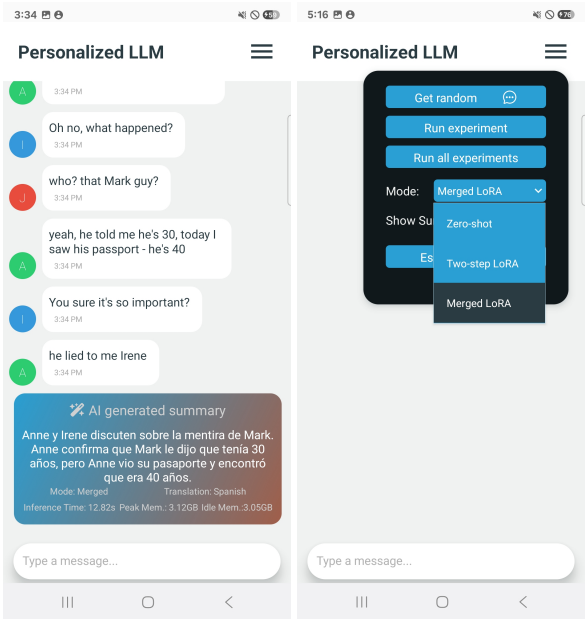

Figure 5: The application runs entirely on a Samsung S23 Ultra Android device. While similar to the server-side proof-of-concept application, additional information is displayed such as memory consumption.

4.2.3 Implementation Challenges and Solutions

Compared to a PoC solution on the server, developing an on-device LLM system presented several

technical challenges, including the following:

1) Adapter Integration: to handle this challenge, we modified the *mistral.rs* inference library. We exported the relevant classes from the library, and modified them to directly use local adapters. Consequently we were able to load the adapters within our Android app.

2) Model Loading: the *mistral.rs* library supports the *GGUF* quantized model format (Gerganov, 2024) with LoRA weights for Llama models. However, by default, it merges the LoRA weights into the base weights of the model upon loading. This behavior was undesirable for two reasons: if the weights are integrated into the model directly, we do not have the ability to dynamically switch between several adapters; further, in order to facilitate the suggested approach, the application should apply a custom merging of LoRA weights only. To address this, we modified the library's loading mechanism to bypass the default merging of LoRA weights, allowing greater control over adapter management.

3) Memory Management: another significant issue was related to using the main user interface (UI) thread to launch the inference back-end. Specifically, the application would not load the inference engine due to its high memory requirements. To overcome this hurdle, we moved all processing tasks to a separate thread dedicated to I/O operations using Kotlin, ensuring heavy tasks would not block the UI thread. This allowed the UI thread to remain free for loading the UI without delays and for handling user interactions. This approach improved the overall performance and responsiveness of the application and, more importantly, allowed the inference engine to function correctly.

5 Experiments

5.1 Implementation Details

We use Llama-3.2-1B-Instruct model (Dubey et al., 2024) with individual LoRA adapters trained on the SAMSum conversation summarization dataset (Gliwa et al., 2019) and TEDTalks English-to-Spanish translation dataset (Qi et al., 2018). For the on-device system, Q4_K_M 4-bit quantization of the model was used (AI, 2024). For the combined summarization-translation task, we created ground-truth data by translating SAMSum summaries using the Opus Machine Translation model (Tiedemann et al., 2023; Tiedemann and Thottingal, 2020). The statistics for each task are specified in Table 1. We chose the SAMSum dataset because it aligns well with our target use case of cross-lingual summarization on mobile devices. SAMSum features written-style conversations, which reflects well the input expected in our application. In contrast, the DialogSum dataset (Chen et al., 2021) used in (Bohdal et al., 2025) focuses on spoken-style dialogues.

Table 1: Dataset statistics across the different tasks.

	Training	**Validation**	**Test**
Summarization	14,732	818	819
Translation	196,026	4,231	5,571
Cross-lingual summarization	14,732	818	819

LoRAs are applied to attention components (query, key, value, output projections) and multilayer perceptron (MLP) components (up, down, gate projections) (Fomenko et al., 2024; Tunstall et al., 2024). We train them using the Adam optimizer with a learning rate of 5×10^{-5} and minibatch size of 3 for one epoch on the full training set. The LoRAs use rank $r = 32$, parameter $\alpha = 16$, dropout rate 0.05, resulting in 22.5M parameters and 45.1MB storage per adapter.

The parameters of our projection merge approach are trained using the Adam optimizer with learning rate 5×10^{-4} and minibatch size of 3, training for one epoch on 10,000 randomly selected examples from the training dataset. The examples used for training are conversations in English, while the targets are the ground-truth summaries translated from English to Spanish. We use rank $s = 4$, resulting in 0.1M additional parameters (0.2MB storage). For the alternative merging strategies, we selected weights of 0.5 for Linear, Concat, and TIES merges, and a density of 0.5 for the TIES merge. For all compared approaches, we include a system prompt specifying the task: *"Summarize the following text and translate it from English to Spanish"*.

5.2 Performance Analysis

As an initial step, we have performed experiments on a server with GPUs to compare our projection merge approach against the different baselines. The results are reported in Table 2 and we can extract the following key findings. (i) A simple zero-shot strategy obtains poor performance despite including a prompt that specifies the compositional multi-task objective. (ii) Primary or secondary-task LoRAs as well as various merging strategies perform

better on the compositional task examined, but their performance is significantly lower than that of the two-step LoRA usage or joint-expert LoRA. (iii) Our proposed projection merge achieves better performance than both two-step LoRA usage and joint-expert LoRA baselines. The efficiency of our projection merge is analyzed in Table 3, showing that our solution introduces only 0.4% of the parameters compared to a specialized joint-expert LoRA while requiring just one inference step.

Table 2: Evaluation on summarization of conversations in another language, test ROUGE-1/2/L (%, ↑). Our projection merge obtains comparable (slightly better) performance to the inefficient two-step LoRA usage or joint-expert LoRA that trains a full adapter for the given compositional task. Other baselines obtain significantly weaker performance.

	ROUGE-1	ROUGE-2	ROUGE-L
Zero-shot	18.55	4.60	13.72
Primary-task LoRA	23.14	6.37	17.62
Secondary-task LoRA	28.08	8.08	20.94
Linear merge	27.38	7.73	20.34
Concat merge	27.58	7.84	20.45
TIES merge	25.16	6.91	18.36
LoraHub merge	28.05	7.90	20.82
Two-step LoRA usage	37.26	13.64	29.25
Joint-expert LoRA	35.10	11.56	26.99
Projection merge (ours)	37.21	14.41	30.21

Table 3: Efficiency of our solution vs inefficient but well-performing baselines. Our method needs only one inference pass and just 0.4% of the parameters and storage compared to a new LoRA.

Method	Number of Inferences	Additional Parameters	Additional Storage
Two-step LoRA usage	2×	0.0M	0.0MB
Joint-expert LoRA	1×	22.5M	45.1MB
Projection merge (ours)	1×	0.1M	0.2MB

We note that the primary goal of our work is to detail how to develop an on-device system for compositional multi-tasking, without aiming to outperform state-of-the-art solutions for compositional multi-tasking. For a more complete context, we include a comparison with Learnable Calibration (Bohdal et al., 2025) on the task utilized in our paper. Results in Table 4 and 5 show that our projection merge achieves a balance between performance and efficiency, lying between the two Learnable Calibration variants.

Table 4: Evaluation on summarization of conversations in another language, test ROUGE-1/2/L (%, ↑). The performance of our projection merge lies between the two Learnable Calibration variants.

	ROUGE-1	ROUGE-2	ROUGE-L
Learnable Calibration	32.87	12.53	26.54
Learnable Calibration++	39.84	16.66	32.63
Projection merge (ours)	37.21	14.41	30.21

Table 5: Efficiency of our solution vs Learnable Calibration. The efficiency of our projection merge lies between the two Learnable Calibration variants.

Method	Number of Inferences	Additional Parameters	Additional Storage
Learnable Calibration	1×	24K	0.05MB
Learnable Calibration++	1×	176K	0.35MB
Projection merge (ours)	1×	102K	0.20MB

5.3 System Analysis

We have performed experiments on a subset of data (using about 20% of the test conversations to make the on-device evaluation faster) to compare inference times on the server and on the device. Our on-device experiments utilize a Samsung S23 Ultra Android device, while our server-side experiments in this section utilize an AMD Ryzen 7, 16 cores CPU. We compare *zero-shot*, *two-step LoRA*, and our *projection merge* approach in the evaluation. Figure 6 shows our method achieves the fastest inference times, taking around 6 seconds on the server and 24 seconds when running on the device. The standard deviations are relatively larger as the inference time depends on the length of the conversation and of the resulting summary.

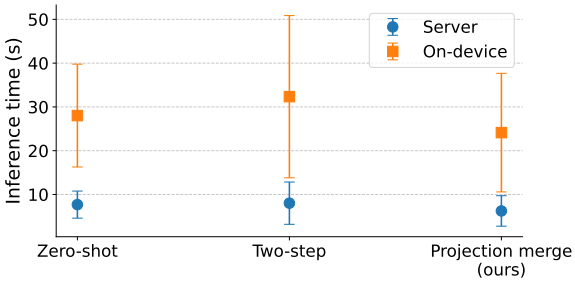

Figure 6: Comparison of inference times for selected approaches on the server and on the device, with mean and standard deviation across the trials. Our projection merge is faster than the baselines, taking around 6 seconds on the server and 24 seconds on the device.

While GPU inference reduces these times to under 1 second, our experimental evaluation suggests that the time required when performing all compu-

tations on the device is manageable, even though it could be optimized further by exploring more advanced quantization methods such as BitNet (Wang et al., 2023). The observed speedup of our method is comparatively larger when running on-device in contrast to when it runs on the server.

We remark the time employed by the two-step solution is not twice as long as of the other approaches because the inference time depends on the number of processed and generated tokens. The first inference pass performs summarization and so the input for the second inference pass is significantly shorter. When an approach generates long outputs, it takes a proportionally longer time, so the overall inference time depends strongly on how much the solution leads to generating long outputs.

Analysis of memory has shown all methods require around 3.12GB of peak memory, with 3.01GB being the mean idle memory requirement after loading the model and adapters. Hence most memory is used for loading the model and adapters, while using the model for inference consumes only a small amount of additional memory.

5.4 Discussion

Our work establishes the feasibility of running compositional multi-tasking LLMs entirely on-device, ensuring user privacy by eliminating the need for remote server communication. The modular design of our application supports straightforward extension to additional languages and compositional tasks. For example, additional compositional tasks could include reply suggestions in addition to summarization and their combinations with translation and tone adjustment.

The current implementation faces some practical limitations, namely: inference times of over 20 seconds may be too long for some use cases; memory requirements of 3GB may make deployment challenging for tiny remote devices. These limitations could be addressed through more aggressive quantization techniques, model architectures optimized for mobile devices, shared parameters across adapters, and dynamic adapter loading based on user needs. However, the most promising avenue for significant speedups would be running an LLM integrated into the mobile operating system, rather than within our application.

6 Conclusion

This paper presented an on-device system for compositional multi-tasking in LLMs, focusing on the practical use case of summarizing conversations in another language. This capability is particularly valuable for users engaging with foreign language content, such as travelers participating in local chat groups. Our solution introduces a lightweight projection layer on top of single-task LoRAs, achieving superior performance with minimal parameter overhead compared to efficient but poorly-performing baselines or well-performing but inefficient approaches. Experimental evaluation on a smartphone has confirmed the practical benefits of our solution, highlighting its speed and viability in fully on-device settings.

Limitations

While we show it is possible to use our method in fully on-device settings, inference times of over 20 seconds may be considered too long. As a result, further optimization of the system, *e.g.* via more aggressive quantization, would be needed before wider deployment. The system requires a manageable amount of memory (3GB), but even this could still make it suitable only for mid and higher-end devices. The solution requires additional parameters that are specific to the given compositional task, so for each new compositional task we would store these on the device. However, the amount of additional storage is small and negligible compared to storing a full adapter, making it possible to scale also to larger numbers of compositional tasks.

Ethical Considerations

Ability to perform compositional multi-tasking fully on-device has significant practical benefits for users. User's data remain fully private and no connection to the internet is needed. While this has broad benefits for users, it also has implications for cases when the users would want to use the compositional multi-tasking abilities to achieve undesirable goals. While the original LLM model may have been aligned for safety, fine-tuning it via LoRA and performing subsequent LoRA merging can diminish the robustness of the safeguard mechanisms.

References

Meta AI. 2024. Llama-3.2-1B-Instruct-GGUF Q4_K_M 4-bit Model. https://huggingface.co/bartowski/Llama-3.2-1B-Instruct-GGUF.

Marco Bellagente, Jonathan Tow, Dakota Mahan, Duy Phung, Maksym Zhuravinskyi, Reshinth Adithyan, James Baicoianu, Ben Brooks, Nathan Cooper, Ashish Datta, et al. 2024. Stable lm 2 1.6 b technical report. *arXiv preprint arXiv:2402.17834*.

Ondrej Bohdal, Mete Ozay, Jijoong Moon, Kyeng-Hun Lee, Hyeonmok Ko, and Umberto Michieli. 2025. Efficient compositional multi-tasking for on-device large language models. In *EMNLP*.

Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. 2024. Distributed inference and fine-tuning of large language models over the internet. In *NeurIPS*.

Eric L. Buehler. 2025. mistral.rs: A rust implementation of mistral models. https://github.com/EricLBuehler/mistral.rs.

Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z. Li. 2024. A survey on generative diffusion models. *IEEE Transactions on Knowledge and Data Engineering*, 36(7).

Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. 2021. DialogSum: A real-life scenario dialogue summarization dataset. In *ACL Findings*.

Sauptik Dhar, Junyao Guo, Jiayi (Jason) Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2021. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Trans. Internet Things*, 2(3).

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *Nature Machine Intelligence*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Vlad Fomenko, Han Yu, Jongho Lee, Stanley Hsieh, and Weizhu Chen. 2024. A note on lora. *arXiv preprint arXiv:2404.05086*.

Radoslav Gerganov. 2024. ggml. https://github.com/ggerganov/ggml.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *ACL*.

Roberto Gozalo-Brizuela and Eduardo C Garrido-Merchán. 2024. A survey of generative ai applications. *Journal of Computer Science*, 20(8).

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, et al. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. In *TMLR*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *ICLR*.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. Lorahub: Efficient cross-task generalization via dynamic lora composition. In *COLM*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. In *ICLR*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out ACL Workshop*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Yuren Mao, Yuhang Ge, Yijiang Fan, Wenyi Xu, Yu Mi, Zhonghao Hu, and Yunjun Gao. 2025. A survey on lora of large language models. *Frontiers of Computer Science*, 19(7).

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Ye Qi, Devendra Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. 2018. When and why are pre-trained word embeddings useful for neural machine translation? In *NAACL*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Donald Shenaj, Ondrej Bohdal, Mete Ozay, Pietro Zanuttigh, and Umberto Michieli. 2025. Lora.rar: Learning to merge loras via hypernetworks for subject-style conditioned image generation. In *ICCV*.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale,

Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Jörg Tiedemann, Mikko Aulamo, Daria Bakshandaeva, Michele Boggia, Stig-Arne Gronroos, Tommi Nieminen, Alessandro Raganato, Yves Scherrer, Raul Vazquez, and Sami Virpioja. 2023. Democratizing neural machine translation with OPUS-MT. *Language Resources and Evaluation*, (58).

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *EAMT*.

David Pedersen tokio rs, Carl Lerche. 2024. axum: Web framework for building async apis in rust. https://crates.io/crates/axum.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Shengyi Huang, Kashif Rasul, Alvaro Bartolome, Alexander M. Rush, and Thomas Wolf. 2024. The Alignment Handbook.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. Bitnet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*.

Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. 2024. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2024. Ties-merging: Resolving interference when merging models. In *NeurIPS*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024b. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *arXiv preprint arXiv:2408.07666*.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. 2024c. Diffusion models: A comprehensive survey of methods and applications. *ACM Comput. Surv.*, 56(4).

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Pengyuan Zhou, Lin Wang, Zhi Liu, Yanbin Hao, Pan Hui, Sasu Tarkoma, and Jussi Kangasharju. 2024. A survey on generative ai and llm for video generation, understanding, and streaming. *arXiv preprint arXiv:2404.16038*.