K-Merge: Online Continual Merging of Adapters for On-device Large Language Models

Donald Shenaj[1,2,3*] **Ondrej Bohdal**[1] **Taha Ceritli**[1] **Mete Ozay**[1]
Pietro Zanuttigh[3] **Umberto Michieli**[1]

[1]Samsung R&D Institute UK [2]University of Pisa [3]University of Padova

Abstract

On-device deployment of Large Language Models (LLMs) frequently leverages Low-Rank Adapters (LoRAs) to support diverse downstream tasks under tight resource constraints. To address the limited storage capacity of mobile devices, recent works have explored model merging techniques to fuse multiple LoRAs into a single one. In practice, however, LoRAs are often delivered incrementally, as users request support for new tasks (e.g., novel problem types or languages). This scenario introduces a new challenge: on-device online continual merging, where the objective is to incorporate new LoRAs while preserving the performance on previously supported tasks. In this paper, we propose a data-free and computationally efficient strategy for selecting and merging LoRAs when a new one becomes available, assuming the device can store only a limited number of adapters. Extensive experiments across real-world tasks demonstrate the superiority of our approach compared to alternative strategies while adhering to the storage budget and compute limitations of on-device settings. The project page is available at: https://donaldssh.github.io/K-Merge.

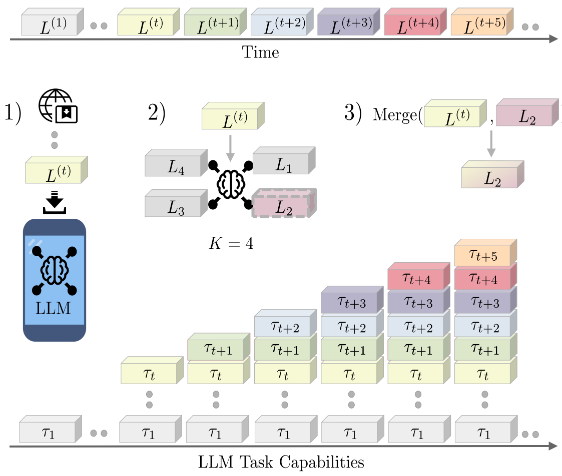


Figure 1: Online continual merging of adapters for on-device LLMs. Each adapter corresponds to a specific task τ_t (e.g., a specific problem type in a selected language). The objective is to increase the LLM capabilities over time without storing all adapters, but rather using a budget of K adapters (e.g., $K = 4$ here). The key steps are: 1) The device downloads new adapters over time. 2) The system selects the most similar stored adapter to the new one. 3) The system updates the selected stored adapter by merging it with the new adapter.

1 Introduction

Large Language Models (LLMs) are powerful general-purpose models that can be adapted to a wide range of problem types in many languages, including question answering (Sticha et al., 2024), translation (Zhu et al., 2024a), summarization (Liu et al., 2024b), text rewriting (Shu et al., 2024), and grammar correction (Rothe et al., 2021). Due to their substantial parameter count (Zhao et al., 2023; Minaee et al., 2024), fine-tuning these models for task-specific applications (e.g., a problem type in a given language) is commonly approached via parameter-efficient tuning methods (Ding et al., 2022; Han et al., 2024). Among these, Low-Rank Adapters (LoRAs) (Hu et al., 2022) are especially popular: they insert lightweight trainable modules into the base model and update only these modules during fine-tuning, freezing the LLM weights. This enables deploying a single base LLM and dynamically loading small LoRAs to support various tasks. A growing use case for LLMs is their deployment on mobile and edge devices (Dhar et al., 2021), driven by benefits such as improved privacy, lower latency, reduced server load, and offline access. However, the size of LLMs suitable for on-device deployment is typically limited to 1–2B parameters, much smaller than their server-scale counterparts, e.g., 70–400B models (Dubey et al., 2024). In such settings, prompting alone often results in sub-

[*]Research completed during internship at Samsung R&D Institute UK, while affiliated with the University of Padova.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3013–3029
July 2-7, 2026

par performance. Instead, task-specific LoRAs are commonly employed to recover acceptable accuracy (Burke, 2023; Gunter et al., 2024; Ceritli et al., 2025; Bohdal et al., 2026), offering a practical way to extend functionality without retraining the base model.

While a single LoRA may suffice for one task, a key challenge arises in real-world usage where LoRAs arrive incrementally, as users request new functionality over time. However, on-device storage is limited, making it infeasible to retain a separate LoRA for every task. One promising solution is to construct multi-task adapters by merging existing LoRAs (Yang et al., 2024b). Several merging strategies have been proposed, from linear combinations with fixed or learned coefficients (Wortsman et al., 2022; Ilharco et al., 2023) to more advanced approaches such as TIES (Yadav et al., 2023) and DARE (Yu et al., 2024).

A remaining question then is how to integrate each new LoRA given only the existing adapters and a limited number of storage slots. Crucially, the original single-task LoRAs and any training data are unavailable when a new LoRA arrives. Moreover, future task requests are *a-priori* unknown. Therefore, the device must continually update its set of LoRAs in a lightweight, data-free, and storage-aware manner, preserving existing capabilities while integrating new ones.

In this work, we formalize this setting as online continual merging of LoRAs under a storage budget, and introduce a novel method tailored to this challenge. Our approach identifies which existing adapter is most similar to the incoming LoRA, decides whether to allocate a new slot (until the storage budget is hit) or to merge adapters using an efficient data-free strategy (see Fig. 1 for an overview of our pipeline).

Our contributions are summarized as follows:

1. We introduce a new and practical setting for online continual merging of adapters in resource-constrained on-device LLMs under a storage budget.

2. We propose a lightweight, data-free merging strategy that identifies suitable adapters for merging using information from the history of merges and employs dynamic weighting to balance old and new capabilities.

3. We conduct extensive evaluation on real-world tasks representative of mobile device usage: our approach achieves strong performance under realistic constraints.

2 Related Work

Parameter-Efficient Fine-Tuning (PEFT): PEFT techniques adapt pre-trained models by training only a small subset of parameters (Ding et al., 2022; Han et al., 2024). Among these, LoRAs (Hu et al., 2022) have emerged as a widely adopted approach. LoRA inserts two low-rank matrices into each target layer of the model and updates only these during fine-tuning, keeping the base model weights frozen. This yields strong performance while significantly reducing memory and compute overhead (Mao et al., 2025). Subsequent work has focused on improving LoRA's efficiency and expressiveness, e.g., AdaLoRA (Zhang et al., 2023), Delta-LoRA (Zi et al., 2023), DoRA (Liu et al., 2024a), VeRA (Kopiczko et al., 2024), and Tied-LoRA (Renduchintala et al., 2024). However, all these methods assume a static task setting and do not address continual merging of adapters under resource constraints.

Model Merging: Model merging aims to combine multiple models or adapters trained for different tasks into a single multi-task model. Early approaches relied on simple weight averaging (Wortsman et al., 2022; Ilharco et al., 2023), but more sophisticated methods have since emerged (Huang et al., 2024; Xiao et al., 2024; Gauthier-Caron et al., 2024; Bohdal et al., 2025; Hammoud et al., 2024). Two widely used techniques are TIES (Yadav et al., 2023), which resets negligible parameters and resolves sign conflicts before merging, and DARE/DARE-TIES (Yu et al., 2024), which selectively drops and rescales parameter deltas. While many model merging techniques target full-model weight merging, recent works have focused on merging LoRA adapters specifically (Stoica et al., 2025; Zhao et al., 2025; Shenaj et al., 2025; Ceritli et al., 2025), often enabling more modular and efficient adaptation. Other efforts address merging in continual learning scenarios (Marczak et al., 2024; Sokar et al., 2025), particularly for discriminative tasks such as image classification (Tang et al., 2025; Coleman et al., 2024) and retrieval (Dziadzio et al., 2025). TIME (Dziadzio et al., 2025) sequentially trains and merges multimodal experts using tailored initialization and merging strategies for continual integration. In contrast, OPCM (Tang et al., 2025) merges models without retraining by projecting updates orthogonally to prior ones, aligning better with our setting. However, its performance deteriorates as the number of merges increases. Our work instead focuses on on-device continual merging of LoRA adapters for generative text tasks, enabling dynamic integration of new functionality over time under strict compute and memory constraints.

On-device LLMs: Standard LLMs often consist of several billion parameters and require GPU-based training and inference (Borzunov et al., 2024). Since most users lack access to such infrastructure, LLM services are commonly deployed via remote servers. However, privacy concerns arise when user data must be transmitted externally–particularly in applications involving personal conversations or sensitive content. As a result, there is growing interest in on-device LLMs to perform inference locally on mobile or edge hardware (Dhar et al., 2021). Applications of interest include summarizing private chats (Gliwa et al., 2019) or generating personalized responses (Jandaghi et al., 2024), while ensuring data remains local. To meet the resource demands of mobile platforms, LLMs are typically downsized to 1–2B parameters and are often further optimized via quantization or pruning (Zhu et al., 2024b). Recent open-source models designed for on-device deployment include Llama-3.2-1B (Dubey et al., 2024) and Qwen-2.5-1.5B (Yang et al., 2024a; Qwen Team, 2024). These models deliver strong language capabilities within the computational and memory limitations of modern smartphones.

3 Problem Formulation

Training multi-task or multi-lingual LoRAs from scratch is impractical due to its lack of flexibility, as it requires access to all datasets and necessitates retraining whenever a new capability is added. Hence we assume only single-task LoRAs are sent to the device. While storage is limited, storing a small number of (possibly multi-tasking) adapters is typically feasible. Formally, we assume the availability of a storage budget for storing a collection of $\leq K$ LoRAs on the device, $\mathcal{L} = \{L_i\}_{i=1}^{K}$.

In the conventional static setting, where all single-task LoRAs $L^{(t)}, \forall t$ are available simultaneously, multi-task adapters can be constructed by grouping LoRAs into K clusters based on task similarity and merging each group into a shared multi-task LoRA $L_i, i = 1, \ldots, K$.

In many real-world scenarios, however, LoRAs arrive incrementally over time, giving rise to the online continual merging setup considered here. As a result, single-task LoRAs are not accessible all at once, and merging must be performed iteratively, based solely on the incoming LoRA and the current set of stored adapters, as described next.

The merging happens on-device and not on the server, because it would be impractical and inefficient to store a copy of the deployed LoRAs on both the server and the device.

3.1 Online Continual Merging

At each discrete time step $t > 0$, the current on-device collection $\mathcal{L}^{(t-1)}$ contains $|\mathcal{L}^{(t-1)}| \leq K$ LoRAs, where $|\cdot|$ indicates the cardinality of the set. A new single-task LoRA $L^{(t)}$ arrives, trained to support a new user-selected task τ_t. The system must incorporate $L^{(t)}$ into $\mathcal{L}^{(t-1)}$ to obtain an updated set $\mathcal{L}^{(t)}$ whose LoRAs preserve performance across all tasks (Fig. 1). We initialize $\mathcal{L}^{(0)} = \emptyset$ and denote the set of all tasks seen at the end of step t by $\mathcal{T}^{(t)} = \{\tau_i\}_{i=1}^{t}$.

A system tackling the online continual merging setup must:

1. Update the on-device collection whenever a new adapter arrives, by performing one of the following two actions:
   (a) Selecting an adapter $L_c \in \mathcal{L}^{(t-1)}$ to be merged with $L^{(t)}$ and replace it in $\mathcal{L}^{(t-1)}$ to obtain:
   $$\mathcal{L}^{(t)} = \{\mathcal{L}^{(t-1)} \backslash L_c\} \cup \{\mathrm{merge}(L_c, L^{(t)})\}, \tag{1}$$
   where the $\mathrm{merge}(\cdot)$ operation returns the merged LoRA and should be data-free and computationally-efficient, suitable for on-device execution (see Sec. 4.2);
   (b) Allocating a new slot, i.e., adding the incoming LoRA to the current collection by:
   $$\mathcal{L}^{(t)} = \{L^{(t)}\} \cup \mathcal{L}^{(t-1)}. \tag{2}$$
   Note that this operation is only available if there is still storage available, i.e., if $|\mathcal{L}^{(t-1)}| < K$.
2. Define a map $\Theta^{(t)}$ such that, whenever an input sample x of task $\tau_i \in \mathcal{T}^{(t)}$ arrives, it identifies the corresponding adapter's identifier, $\hat{c} = \Theta^{(t)}(i)$, to use for inference,
   $$\Theta^{(t)} : i \in \{1, \ldots, |\mathcal{T}^{(t)}|\} \rightarrow \hat{c} \in \{1, \ldots, |\mathcal{L}^{(t-1)}|\}. \tag{3}$$

Then, the adapter $L_{\hat{c}}$ is loaded from storage and plugged into the model to perform inference.

3. Maintain strong performance on all previous tasks in $\mathcal{T}^{(t)}$.

3.2 Evaluation Protocol and Aggregate Score

In the considered experimental setting, each task τ_i is characterized by a problem type and a language. To evaluate performance, we construct a benchmark spanning α problem types and β languages, totaling $\gamma = \alpha \cdot \beta$ distinct tasks for which we obtain γ single-task LoRAs. Tasks arrive at the device in random order, one per time step. At each time step t, the system updates the collection of on-device LoRAs $\mathcal{L}^{(t)}$, and the performance is measured on all the tasks seen so far $\mathcal{T}^{(t)}$. In our setup, $t = 1, \ldots, \gamma$, however, γ is unknown to the device at any time t.

Each task τ_i is associated with a dataset $\mathcal{D}_{\tau_i}$ and an evaluation metric M_{τ_i} that takes as input a LoRA and provides as output the performance on the associated dataset. Since different problem types may be evaluated by different metrics that operate on different scales, we normalize performance on each task by comparing it to the performance of a single-task LoRA trained directly on that task. Let L denote a generic adapter. For each task $\tau_i \in \mathcal{T}^{(t)}$, where $t > 0$, let $M_{\tau_i}(L; \mathcal{D}_{\tau_i})$ be the score obtained by the LLM using L, and $M_{\tau_i}(L^{(i)}; \mathcal{D}_{\tau_i})$ the single-task performance, i.e., the score obtained by the LLM using $L^{(i)}$. We define the normalized aggregate score $S^{(t)}$ as follows:

$$S^{(t)} = \frac{1}{|\mathcal{T}^{(t)}|} \sum_{\tau_i \in \mathcal{T}^{(t)}} \frac{M_{\tau_i}(L; \mathcal{D}_{\tau_i})}{M_{\tau_i}(L^{(i)}; \mathcal{D}_{\tau_i})}. \tag{4}$$

This score quantifies how closely the performance of a generic LoRA L aligns with that of single-task LoRAs $L^{(i)}$. In our experiments, we evaluate LoRAs $L \in \mathcal{L}^{(t)}$; however, the proposed score formulation is generic and allows the evaluation of any given LoRAs L. For instance, $\mathcal{L}^{(t)}$ could contain either single-task LoRAs or merged LoRAs . Additionally, L is initialized to zeroes in the zero-shot case where no LoRAs are used.

3.3 Task Identification

In the settings we consider, it is typically easy to identify which task should be performed. The user can explicitly select the task or language, e.g., via UI elements. Alternatively, the task can be specified in the user prompt (e.g., "translate this sentence", or "summarize this passage"). In such cases, determining the appropriate adapter is straightforward as we can identify the task via matching to selected keywords. When task identity is not directly stated, lightweight task classifiers can be used. Since routing is not related to our contribution, we model this step abstractly and assume access to a perfect classifier for simplicity. This assumption isolates the evaluation of the merging method itself without conflating it with the performance of a separate routing module.

4 Proposed Method

To address the challenges of online continual merging, we propose a lightweight, data-free method that supports on-device continual merging within a given storage budget. An overview is shown in Fig. 2. At the core of our method, there are two key components: 1) **Similarity-based clustering** (Sec. 4.1): we identify the most suitable stored adapter to merge with for an incoming LoRA, based on an efficient similarity metric without using any training data. 2) **History-aware merging** (Sec. 4.2): we merge adapters using a weighted combination scheme that leverages information from the previous adapter merges.

These components are integrated into a decision-making framework to determine whether the new LoRA should be merged into an existing adapter or allocated to a new slot (subject to the storage budget). We introduce two variants:

- **K-Merge**, a lightweight solution that merges each new LoRA with its nearest stored adapter unless there is space available to store it separately.
- **K-Merge**++, an improved variant that selectively merges based on a similarity threshold, preserving space for future, more diverse LoRAs.

4.1 Cluster Assignment by Similarity Scoring

At the core of both variants is the ability to match the incoming LoRA $L^{(t)}$ to the most compatible stored adapter among $L_i \in \mathcal{L}^{(t-1)}, \forall i$ via similarity scoring.

Each LoRA L consists of a pair of low-rank update matrices $(A^{n,p}, B^{n,p})$ for each transformer layer $n \in \mathcal{N}$ and projection type

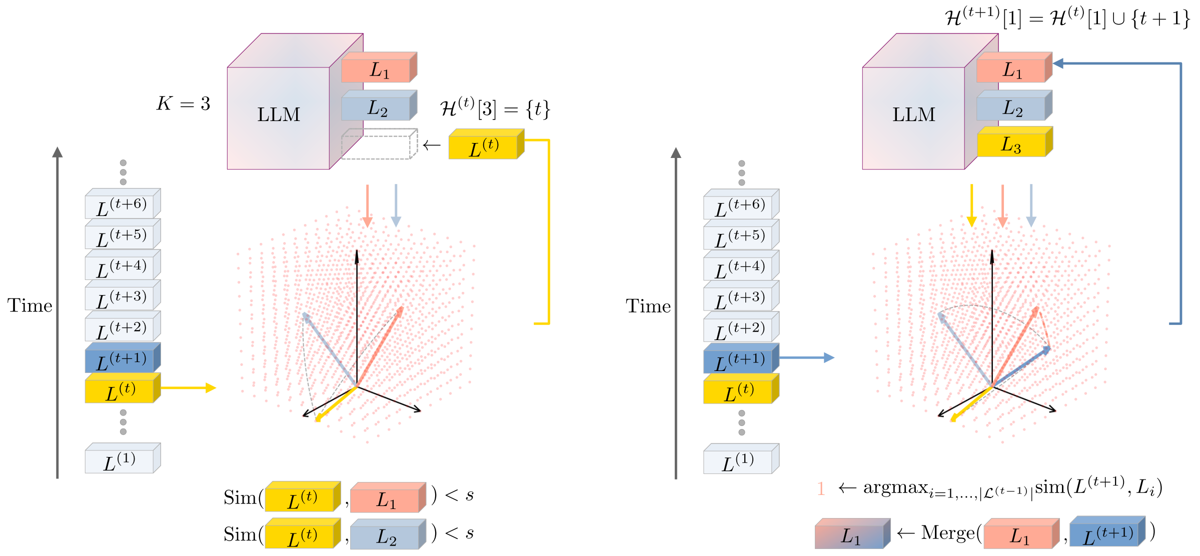


Figure 2: K-Merge++ outline. When the storage budget is not fulfilled and a new LoRA $L^{(t)}$ is downloaded, it is compared with the LoRAs in $\mathcal{L}^{(t-1)}$ using Eq. (5). If similarity is smaller than the threshold s, $L^{(t)}$ initializes a new cluster; otherwise, it gets merged with the closest LoRA following Eq. (6). After the budget limit has been reached, new LoRAs can only be merged.

$p \in \mathcal{P} = \{\texttt{key}, \texttt{query}, \texttt{value}, \texttt{output}\}$. Then, we perform the following steps:

1. Flatten each update matrix $\Delta W^{n,p} = B^{n,p}A^{n,p}$ into a vector via the flattening operation $f(\cdot)$.

2. Compute the cosine similarity between the flattened vector corresponding to the incoming LoRA and that of all the stored LoRAs, and average over all layers and projections:

$$\text{sim}(L^{(t)}, L_i) = \frac{\sum_{n\in\mathcal{N},p\in\mathcal{P}} \text{sim}(L^{(t)n,p}, L_i^{n,p})}{|\mathcal{N}| \cdot |\mathcal{P}|}, \tag{5}$$

$$\text{sim}(L^{(t)n,p}, L_i^{n,p}) = \cos(f(\Delta W^{(t)n,p}), f(\Delta W_i^{n,p})).$$

3. Determine the index c of the most-similar LoRA by:

$$c = \text{argmax}_{i=1,\ldots,|\mathcal{L}^{(t-1)}|}\text{sim}(L^{(t)}, L_i). \tag{6}$$

4.2 Iterative History-aware Model Merging

History Design: To retrieve the correct LoRA at the inference stage, we allocate some small storage (negligible in practice) to collect the time-varying history of merges as a map $\mathcal{H}^{(t)}$ with up to K keys, each linked to a set of task indices. The history is initialized as $\mathcal{H}^{(0)} = \emptyset$.

Save Incoming LoRA to Storage: Whenever an incoming adapter $L^{(t)}$ is selected to be stored on the on-device collection, a new key is added to the history and the task index is added to the value set, as:

$$\mathcal{H}^{(t)}[|\mathcal{L}^{(t-1)}| + 1] = \{t\}. \tag{7}$$

History-aware Merging: In case the incoming LoRA is selected to be merged with the most similar adapter L_c, we implement a history-aware merging. We implement the merge operation via a running average formulation, defined by:

$$\text{merge}(L_c, L^{(t)}) = \frac{L^{(t)} + |\mathcal{H}^{(t-1)}[c]| \cdot L_c}{|\mathcal{H}^{(t-1)}[c]| + 1}, \tag{8}$$

and the history of merges is updated accordingly as follows:

$$\mathcal{H}^{(t)}[c] = \mathcal{H}^{(t-1)}[c] \cup \{t\}. \tag{9}$$

This formulation enables efficient incremental merging without requiring storage of all LoRAs in each cluster. A key benefit of the formulation is also that it is order-invariant as all the LoRAs have the same contribution to the merged multi-tasking LoRA.

4.3 K-Merge and K-Merge++

We now describe the two variants of our overall system, both designed to manage a fixed adapter budget of K slots. In contrast, existing merging

Algorithm 1 K-Merge++

Parameters: Storage budget K, similarity threshold s.
Initialize on-device collection by $\mathcal{L}^{(0)} = \emptyset$.
Initialize history by $\mathcal{H}^{(0)} = \emptyset$.
while new LoRA $L^{(t)}$ is received, $t > 0$ **do**
  if $|\mathcal{L}^{(t-1)}| > 0$ **then**
    Find index c of most similar LoRA via Eq. (6).
  end if
  if $|\mathcal{L}^{(t-1)}| = K$ or $(|\mathcal{L}^{(t-1)}| > 0$ and $\text{sim}(L^{(t)}, L_c) \geq s)$ **then**
    Update collection $\mathcal{L}^{(t)}$ via Eqs. (1), (8).
    Update history $\mathcal{H}^{(t)}$ via Eq. (9).
  else
    Update collection via Eq. (2).
    Update history $\mathcal{H}^{(t)}$ via Eq. (7).
  end if
end while
return $\mathcal{L}^{(t)}$

methods would use only one slot, while naïve storing of all single-task LoRAs would use one slot for each LoRA.

K-Merge: The basic strategy merges $L^{(t)}$ into its closest stored adapter L_c using Eq. (8), unless fewer than K adapters are currently stored, in which case $L^{(t)}$ is stored directly. This approach is simple and efficient, but has a key limitation: if early incoming LoRAs are similar, it may exhaust the storage budget prematurely by storing redundant adapters, thus leading to a sub-optimal clustering of adapters at later stages.

K-Merge++: To mitigate this issue, we introduce a more advanced approach relying on a similarity threshold s. A new LoRA $L^{(t)}$ is merged into L_c only when they are sufficiently similar to each other, i.e., $\text{sim}(L^{(t)}, L_c) \geq s$. Otherwise, it is stored separately if storage is still available. Once the storage is full, the merging is performed regardless of s. The threshold s is estimated empirically from an auxiliary dataset of LoRAs trained on held-out tasks. Specifically, we compute all pairwise similarities across LoRAs and set s to the median. This variant preserves storage for more diverse future LoRAs, improving adaptability across diverse deployment scenarios. Its only overhead is the need to estimate s. The procedure is detailed in Algorithm 1.

4.4 Inference Stage

We implement the mapping described in Eq. (3) as an inverse function of the history $\mathcal{H}^{(t)}$. To process a query input sample x belonging to task $\tau_i \in \mathcal{T}^{(t)}$, we look for the key $\hat{c} \in \{1, \dots, |\mathcal{L}^{(t-1)}|\}$ whose value set contains i, i.e., $i \in \mathcal{H}^{(t)}[\hat{c}]$. Then, we load the adapter $L_{\hat{c}}$ from storage, plug it into the model, and perform inference.

5 Experimental Evaluation

5.1 Experimental Details

Tasks and Metrics: We evaluate our method across a diverse set of tasks, each defined by a combination of a problem type and a target language, and each associated with a specific evaluation metric. We consider $\alpha = 5$ problem types: *Smart Reply*, *Summarization*, *Tone Adjustment*, *Question Answering*, and *Grammar Correction*. Each of these problem types is instantiated in $\beta = 8$ languages: English, Spanish, French, German, Italian, Korean, Japanese, and Chinese. As metrics, we report F-0.5 for Grammar Correction, F-1 for Question Answering, weighted ROUGE (Zhang et al., 2021) for Smart Reply, and ROUGE-L for both Text Summarization and Tone Adjustment.

Datasets: We use established benchmarks for each task: 1) *Smart Reply:* Persona-Chat Synthetic (Jandaghi et al., 2024). 2) *Summarization:* SAMSum (Gliwa et al., 2019). 3) *Tone Adjustment:* Sound Natural (Einolghozati et al., 2020) modified via a publicly available model fine-tuned for tone adjustment (Utsav, 2023). 4) *Question Answering:* SQuAD (Rajpurkar et al., 2016). Datasets for these four problem types are released in English, and we translated them into the other languages via the OPUS-MT model (Tiedemann and Thottingal, 2020) for Spanish, German, French, and Italian, and via the M2M100 model (Fan et al., 2021) for Korean, Chinese, and Japanese. 5) *Grammar Correction:* original-language datasets are critical here, as machine translation systems tend to correct grammatical errors (Luhtaru et al., 2024). We use Write & Improve for English (Bryant et al., 2019), Merlin for Italian (Boyd et al., 2014), ECSpell for Chinese (Lv et al., 2023), and the GitHub Typo Corpus (Hagiwara and Mita, 2020) for the other languages.

To calibrate the similarity threshold used in K-Merge++, we reserve a set of held-out tasks that include unseen problem types and languages. Specifically, we use: (i) *Translation to English* and *Title*

Generation as novel problem types, and (ii) Portuguese, Turkish, and Serbian as novel languages. TED Talks (Qi et al., 2018) serves as the dataset for translation, and XLSum (Hasan et al., 2021) for title generation.

All datasets are split into training, validation, and test partitions; see the Appendix for details.

Models: We experiment with two lightweight, instruction-tuned models designed for on-device deployment: Llama-3.2-1B-Instruct and Qwen-2.5-1.5B-Instruct.

Baselines: We compare our solutions with several alternative approaches. We include zero-shot and single-task LoRA as lower and upper performance boundaries. We also compare to standard model merging approaches (Linear, TIES, DARE, DARE-TIES) and continual merging (OPCM). All merging methods are adapted to our online setting by applying them to the pair consisting of the incoming LoRA $L^{(t)}$ and the most similar on-device LoRA L_c. Task descriptions are always provided via prompts (see the Appendix for examples).

Hyperparameters: All LoRAs are trained using AdamW with a learning rate of 5e-5, dropout rate of 0.05, and a mini-batch size of 3. LoRAs use rank 32 and scaling factor $\alpha = 128$. Linear merging uses equal weights (i.e., 0.5) for pairwise merging, while TIES, DARE, and DARE-TIES use unary weights and density 0.5. The similarity threshold is set using the held-out LoRA set to $s = 0.020$ and $s = 0.028$ for Llama-3.2-1B and Qwen-2.5-1.5B models, respectively. In our experiments, the number of LoRAs that can be stored on-device, K, varies from 1 to 8. In total, we simulate the sequential arrival of $\gamma = 40$ LoRAs. All results are averaged over 3 random permutations of LoRA arrival ordering (unless otherwise stated).

5.2 Experimental Results

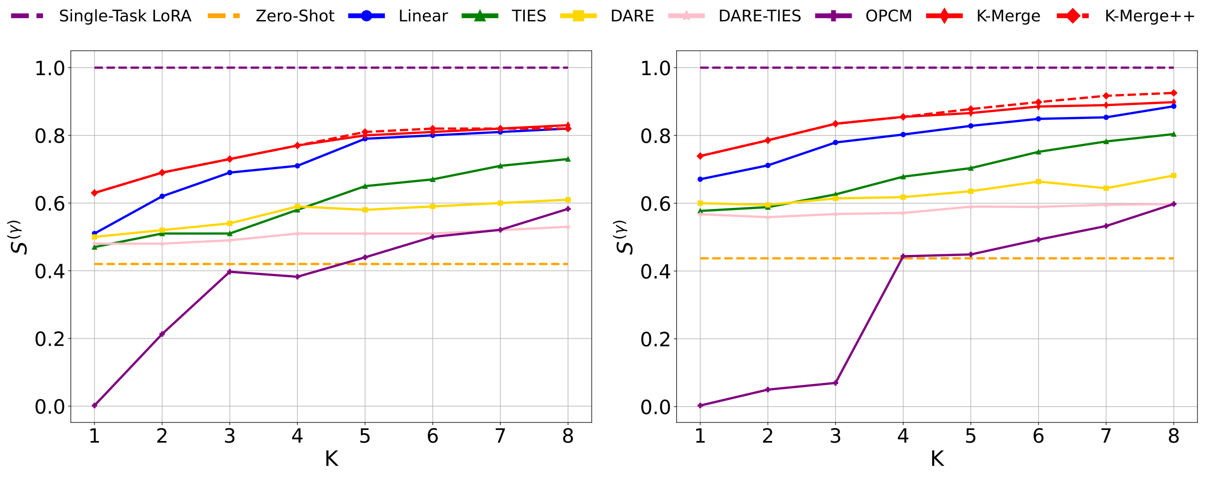
Results with Random Task Ordering: Fig. 3 illustrates the average score of the methods for the two LLMs at varying numbers of clusters over three random task orderings. Additionally, we plot zero-shot and single-task LoRA performance as a reference. First, we note the great benefit of LoRA fine-tuning for all LLMs, as indicated by the improvement of single-task LoRAs over zero-shot performance. Second, we observe DARE, DARE-TIES, and OPCM are unable to offer competitive performance, being significantly lower than single-task LoRAs and other approaches, although the corresponding performance gap shrinks slightly as the number of clusters increases. TIES generally outperforms these methods; however, the Linear merge proves to be the most effective baseline. Remarkably, our proposed methods outperform Linear at all storage budgets. In particular, with Llama-3.2-1B our method scales better than competitors for small numbers of clusters. With Qwen-2.5-1.5B the gain is stable across different values of K.

Method	Ordering	$K=3$	$K=5$	$K=7$
Linear	Random	0.69	0.79	0.81
Linear	Problem Types	-	0.80	-
Linear	Worst	0.62	0.69	0.73
TIES	Random	0.51	0.65	0.71
TIES	Problem Types	-	0.72	-
TIES	Worst	0.52	0.56	0.67
K-Merge	Random	0.73	0.80	0.82
K-Merge	Problem Types	-	0.84	-
K-Merge	Worst	0.73	0.75	0.78
K-Merge++	Random	0.73	0.81	0.82
K-Merge++	Worst	0.73	0.80	0.82

Table 1: Robustness of approaches across different ordering of tasks evaluated by means of score $S^{(\gamma)}$ on Llama-3.2-1B.

Finally, K-Merge++ improves the performance of K-Merge even further by utilizing the similarity threshold mechanism, especially for larger K, where there is more scope for more distinct multi-tasking LoRAs. However, the main benefit of K-Merge++ is robustness against worst-case task orderings, as we analyse later. Using our methods, the overall score reaches about 80-90% of the single-task performance with just 8 clusters, showing the usefulness of our solution.

Robustness to Task Ordering: We investigate the robustness of our methods to different task orderings in Table 1. In particular, we compare the average score obtained via random task ordering (*Random*) against a clustering of LoRAs done according to problem types (*Problem Types*) and an ordering that we empirically found to lead to low performance for our setup (*Worst*). In particular, in the latter, we force LoRAs of the same problem type to appear consecutively. We remark that we could only test *Problem Types* for $K = \alpha = 5$ since we allocate one problem type for each on-device LoRA cluster. Additionally, evaluating K-Merge++ on the *Problem Types* case is not meaningful, as the clustering is pre-determined and the



(a) Llama-3.2-1B-Instruct.

(b) Qwen-2.5-1.5B-Instruct.

Figure 3: Score $S^{(\gamma)}$ of compared methods at variable storage budget K. Averaged over 3 random task orderings.

similarity threshold is therefore not employed.

First, the scores for the *Problem Types* case are higher than for *Random*, being a favorable case where similar LoRAs are merged together (see Appendix Fig. A.3). Nonetheless, this ordering makes three unrealistic assumptions: (i) the device knows *a priori* all future problem types; (ii) the device has exactly $K = \alpha$ on-device LoRA storage slots; and (iii) LoRAs always have the highest similarity with other LoRAs from the same problem type rather than from different ones. In practice, these assumptions are difficult to meet, hence motivating the usefulness of our approach. Even more, K-Merge outperforms the baselines also in this setup.

Second, under the *Worst* case, we observe K-Merge++ in particular significantly outperforms K-Merge and shows strong robustness thanks to its threshold mechanism. Linear and TIES degrade significantly. This setting complicates the initialization of empty slots, often causing the system to fill multiple slots with similar LoRAs (e.g., targeting the same problem type across different languages). These results highlight the robustness of our K-Merge++ approach in handling *a-priori* unknown and adversarial task orderings on the device side.

Varying Timestep: Fig. 4 shows that K-Merge initially outperforms K-Merge++, but is quickly overtaken as the latter benefits from saving storage slots at early stages thus having them available when more diverse LoRAs arrive. Linear matches our methods at initial timesteps, but is then consistently surpassed as more adapters arrive.

Efficiency: K-Merge is lightweight and fast: the integration of a new incoming LoRA takes between 0.04s and 0.18s when there are 2 to 8 stored multi-tasking LoRAs, demonstrating its practicality for on-device deployment. K-Merge++ exhibits comparable runtime. Since memory is dominated by text generation, our approach adds only negligible extra memory overhead. Full details are presented in the Appendix.

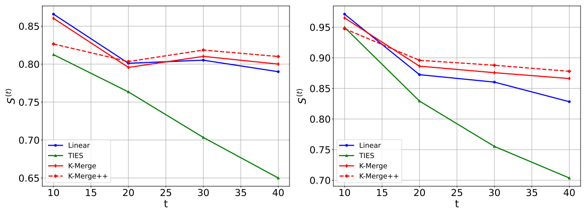


Figure 4: Performance $S^{(t)}$ on discovered tasks at varying timesteps for $K = 5$ using Llama-3.2-1B (left) and Qwen-2.5-1.5B (right).

Model	Parameters per LoRA	Storage per LoRA
Llama-3.2-1B	23M	27MB
Qwen-2.5-1.5B	37M	34MB

Table 2: Number of parameters and storage used by one LoRA adapter. Storing all adapters (40) is not practical as it would take a significant amount of storage (1GB).

Storage Analysis: We report the number of parameters and storage used per adapter in Table 2. Storing, e.g., 5 adapters is lightweight, consuming 135-170MB of storage in total. Storing all 40 adapters would require about 1GB of storage, a rather substantial amount.

Similarity Metric: We have evaluated different similarity measures, with cosine similarity (i.e.,

Method	L^1	L^2	L^∞	Ours (cos)
		$K = 3$		
K-Merge	0.66	0.66	0.66	0.75
Linear	0.51	0.51	0.51	0.66
TIES	0.47	0.47	0.50	0.51
DARE	0.53	0.52	0.51	0.53
		$K = 5$		
K-Merge	0.69	0.67	0.67	0.78
Linear	0.50	0.52	0.57	0.77
TIES	0.52	0.51	0.52	0.59
DARE	0.52	0.51	0.51	0.58

Table 3: Comparison of similarity metrics. Score $S^{(\gamma)}$ for different values of K (single seed).

Method	History-aware	Constant
	$K = 1$	
Linear	0.63	0.45
TIES	0.44	0.46
DARE	0.43	0.50
DARE-TIES	0.43	0.48
	$K = 3$	
Linear	0.75	0.66
TIES	0.46	0.51
DARE	0.44	0.53
DARE-TIES	0.44	0.48

Table 4: Effectiveness of our history-aware merging. Score $S^{(\gamma)}$ for different values of K (single seed).

our selected option) giving the best performance. In particular, we have compared it with the L^1, L^2 and L^∞ measures. In our experiments, the cosine similarity yielded the best performances, as can be seen from Table 3. It shows how our approach achieves $S^{(\gamma)}$ scores of 0.75 and 0.78 with $K = 3$ and $K = 5$, respectively. The alternative similarity measures led to significantly lower scores than cosine similarity.

History-Aware Merging: Note that the Linear approach combined with history-aware merging corresponds to our K-Merge. For this reason, we also apply similarity-based clustering to each merging approach that we report in the results. We have conducted additional experiments that integrate history-aware merging into the diverse baselines. The results in Table 4 show that they obtain worse performance than our solution (linear with history-aware merging).

Further Analyses: We include further analyses studying LoRA similarity, similarity threshold, clustering consistency and out-of-domain LoRA performance in the Appendix. The analyses show, for example, there is more similarity between LoRAs from the same problem type rather than from the same language.

6 Conclusion

We introduced the novel task of on-device continual LoRA merging, where incoming adapters must be merged over time due to storage constraints, with the goal of supporting new capabilities while preserving existing ones. We proposed two methods: K-Merge, which merges incoming adapters with their closest stored counterpart when storage is full, and K-Merge++, which uses a similarity threshold to decide between merging or allocating a new slot. Both methods have been evaluated on 40 tasks across 5 domains and 8 languages, achieving strong performance and high efficiency, while requiring no data. Average performance is similar, but K-Merge++ is notably more robust to the task arrival order.

Limitations

We restrict our study to LLMs, leaving extensions to multimodal tasks for future work. We only studied LoRA adapters as it is the current industry standard, but other adapter types could be considered. We evaluated LLMs that are in sizes suitable for on-device deployment as that is the desired use case. Our solution could also be evaluated for larger LLMs, which would, however, require significantly larger amount of compute.

Ethical Considerations

Our work focuses on text generation, which is an area where ethical considerations are important due to the potential for large-scale real-world impact. Our work in particular focuses on efficiency and as a result it does not introduce new capabilities that could be misused. However, merging of adapters can lead to weakening of safeguarding mechanisms. As a result, it is important to conduct experiments that explore how effective earlier safeguarding mechanisms remain before deploying the solution.

References

Ondrej Bohdal, Taha Ceritli, Mete Ozay, Jijoong Moon, Kyeng-Hun Lee, Hyeonmok Ko, and Umberto Michieli. 2026. Data-driven clustering and merging of adapters for on-device large language models. In *ICASSP*.

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*.

Adriane Boyd, Jirka Hana, Lionel Nicolas, Detmar Meurers, Katrin Wisniewski, Andrea Abel, Karin Schöne, Barbora Stindlová, and Chiara Vettori. 2014. The merlin corpus: Learner language and the cefr. In *LREC*.

Christopher Bryant, Mariano Felice, Øistein E Andersen, and Ted Briscoe. 2019. The BEA-2019 shared task on grammatical error correction. In *ACL Workshop*.

Dave Burke. 2023. A new foundation for ai on android.

Taha Ceritli, Ondrej Bohdal, Mete Ozay, Jijoong Moon, Kyenghun Lee, Hyeonmok Ko, and Umberto Michieli. 2025. Hydraopt: Navigating the efficiency-performance trade-off of adapter merging. In *EMNLP*.

Eric Nuertey Coleman, Luigi Quarantiello, Julio Hurtado, and Vincenzo Lomonaco. 2024. Adaptive lora merging for efficient domain incremental learning. In *NeurIPS Workshop*.

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 TIOT*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, and 1 others. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Sebastian Dziadzio, Vishaal Udandarao, Karsten Roth, Ameya Prabhu, Zeynep Akata, Samuel Albanie, and Matthias Bethge. 2025. How to merge your multimodal models over time? In *CVPR*.

Arash Einolghozati, Anchit Gupta, Keith Diedrick, and Sonal Gupta. 2020. Sound natural: Content rephrasing in dialog systems. In *EMNLP*.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, and 1 others. 2021. Beyond english-centric multilingual machine translation. *JMLR*.

Thomas Gauthier-Caron, Shamane Siriwardhana, Elliot Stein, Malikeh Ehghaghi, Charles Goddard, Mark McQuade, Jacob Solawetz, and Maxime Labonne. 2024. Merging in a bottle: Differentiable adaptive merging (dam) and the path from averaging to automation. *arXiv preprint arXiv:2410.08371*.

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

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, and 1 others. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*.

Masato Hagiwara and Masato Mita. 2020. GitHub typo corpus: A large-scale multilingual dataset of misspellings and grammatical errors. In *LREC*.

Hasan Abed Al Kader Hammoud, Umberto Michieli, Fabio Pizzati, Philip Torr, Adel Bibi, Bernard Ghanem, and Mete Ozay. 2024. Model merging and safety alignment: One bad model spoils the bunch. In *EMNLP Findings*.

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*.

Tahmid Hasan, Abhik Bhattacharjee, Md. Saiful Islam, Kazi Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. 2021. XL-sum: Large-scale multilingual abstractive summarization for 44 languages. In *ACL-IJCNLP Findings*.

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*.

Pegah Jandaghi, Xianghai Sheng, Xinyi Bai, Jay Pujara, and Hakim Sidahmed. 2024. Faithful persona-based conversational dataset generation with large language models. In *ACL Findings*.

Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. VeRA: Vector-based random matrix adaptation. In *ICLR*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024a. Dora: Weight-decomposed low-rank adaptation. In *ICML*.

Yixin Liu, Kejian Shi, Katherine S He, Longtian Ye, Alexander R Fabbri, Pengfei Liu, Dragomir Radev, and Arman Cohan. 2024b. On learning to summarize with large language models as references. In *NAACL*.

Agnes Luhtaru, Elizaveta Korotkova, and Mark Fishel. 2024. No error left behind: Multilingual grammatical error correction with pre-trained translation models. In *EACL*.

Qi Lv, Ziqiang Cao, Lei Geng, Chunhui Ai, Xu Yan, and Guohong Fu. 2023. General and domain-adaptive chinese spelling check with error-consistent pretraining. *ACM TALLIP*.

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*.

Daniel Marczak, Bartłomiej Twardowski, Tomasz Trzciński, and Sebastian Cygert. 2024. Magmax: Leveraging model merging for seamless continual learning. In *ECCV*.

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.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 2024. Tied-LoRA: Enhancing parameter efficiency of LoRA with weight tying. In *NAACL*.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A simple recipe for multilingual grammatical error correction. In *ACL*.

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*.

Lei Shu, Liangchen Luo, Jayakumar Hoskere, Yun Zhu, Yinxiao Liu, Simon Tong, Jindong Chen, and Lei Meng. 2024. Rewritelm: An instruction-tuned large language model for text rewriting. In *AAAI*.

Ghada Sokar, Gintare Karolina Dziugaite, Anurag Arnab, Ahmet Iscen, Pablo Samuel Castro, and Cordelia Schmid. 2025. Continual learning in vision-language models via aligned model merging. *arXiv preprint arXiv:2506.03189*.

Abigail Sticha, Norbert Braunschweiler, Rama Sanand Doddipatla, and Kate M Knill. 2024. Advancing faithfulness of large language models in goal-oriented dialogue question answering. In *ACM CUI*.

George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. 2025. Model merging with SVD to tie the knots. In *ICLR*.

Anke Tang, Enneng Yang, Li Shen, Yong Luo, Han Hu, Lefei Zhang, Bo Du, and Dacheng Tao. 2025. Merging on the fly without retraining: A sequential approach to scalable continual model merging. In *NeurIPS*.

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

Kumar Utsav. 2023. Redpajama-incite-base-3b-v1 model finetuned for paraphrasing and changing the tone. `https://huggingface.co/llm-toys`.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and 1 others. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Xingrun Xing. 2024. LM-Cocktail: Resilient tuning of language models via model merging. In *ACL Findings*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2023. 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, and 40 others. 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*.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *ICML*.

Mozhi Zhang, Wei Wang, Budhaditya Deb, Guoqing Zheng, Milad Shokouhi, and Ahmed Hassan Awadallah. 2021. A dataset and baselines for multilingual reply suggestion. In *ACL*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Ziyu Zhao, Tao Shen, Didi Zhu, Zexi Li, Jing Su, Xuwu Wang, and Fei Wu. 2025. Merging LoRAs like playing LEGO: Pushing the modularity of loRA to extremes through rank-wise clustering. In *ICLR*.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2024a. Multilingual machine translation with large language models: Empirical results and analysis. In *NAACL Findings*.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024b. A survey on model compression for large language models. *TACL*.

Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. 2023. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*.

Appendix

This document provides additional material that could not be included in the main paper due to space constraints. Additional details for the employed datasets, prompts, and hardware are reported in Appendix A.1. Detailed results for single-task LoRAs and continual merging are reported in Appendix A.2. Additional analyses describing computation time overhead of our approach, cosine similarity histograms, clustering consistency and out-of-domain performance are presented in Appendix A.3.

A.1 Additional Details

A.1.1 Dataset Statistics

Table A.2 shows the number of training, validation, and test samples for the various problem types and languages.

A.1.2 Prompts

Table A.8 shows the prompts that have been used for the different problem types, which are taken from (Ceritli et al., 2025). Different tones have separate prompts, which are also detailed in Table A.8.

A.1.3 Hardware and Software Details

For all our experiments, we have used Python 3.9.21 with PyTorch 2.6.0+cu124. We have run our experiments on an Ubuntu 20.04.6 LTS machine with kernel version 5.4.0-205-generic, equipped with 10 NVIDIA A40 GPUs, and two Intel Xeon Gold 5218 processors (16 cores per socket). Note that the continual merging approach (i.e., our contribution) is very lightweight and fast, the hardware resources have been used mostly to train the LoRAs and perform inference.

A.2 Detailed Results

A.2.1 Single-task LoRA Performance

Table A.1 shows the performance of zero-shot (i.e., without any LoRA adapter) inference and the ones of the single-task LoRA for all combinations of problem types and languages. It is possible to notice a large gap between the two options as using LoRAs typically doubles the score. Qwen-2.5-1.5B has better performances than Llama-3.2-1B, possibly due to the larger number of parameters. The scores vary across languages, with English outperforming the others. We provide a similar analysis in Table A.3 for the additional languages and problem types used in the held-out set. Note that this data has been used only to select the similarity threshold s in the K-Merge++ version of the approach.

		Task Performance						
Lang	Method	TC	QA	RP	Sum	Tone	AVG	S
		Llama-3.2-1B-Instruct						
en	Zero-Shot	13.1	16.2	5.1	23.4	27.6	17.1	0.39
	LoRA	35.1	63.5	23.0	38.2	58.1	43.6	1
		Qwen 2.5 1.5B						
	Zero-shot	19.6	24.6	5.0	27.5	43.5	24.0	0.51
	LoRA	38.6	68.0	22.3	38.6	59.0	45.3	1
		Llama-3.2-1B-Instruct						
de	Zero-Shot	7.6	8.4	2.2	15.8	17.2	10.3	0.33
	LoRA	26.0	36.5	10.7	28.6	43.8	29.1	1
		Qwen 2.5 1.5B						
	Zero-shot	7.2	10.1	2.3	18.7	33.9	14.4	0.43
	LoRA	23.0	42.4	12.8	28.9	44.0	30.2	1
		Llama-3.2-1B-Instruct						
es	Zero-Shot	6.7	9.1	3.5	16.8	25.4	12.3	0.36
	LoRA	34.0	42.3	13.2	31.3	44.4	33.0	1
		Qwen 2.5 1.5B						
	Zero-shot	12.1	13.0	2.7	21.2	38.5	17.5	0.47
	LoRA	29.7	48.7	14.7	32.6	46.0	34.3	1
		Llama-3.2-1B-Instruct						
fr	Zero-Shot	5.7	7.6	3.9	18.0	26.6	12.4	0.40
	LoRA	20.5	34.7	12.2	30.2	46.2	28.8	1
		Qwen 2.5 1.5B						
	Zero-shot	10.4	12.3	3.1	20.8	30.3	15.4	0.42
	LoRA	36.8	38.7	14.3	31.3	47.5	33.7	1
		Llama-3.2-1B-Instruct						
it	Zero-Shot	23.3	7.2	2.5	17.8	21.3	14.4	0.49
	LoRA	27.2	37.1	8.8	28.3	42.8	28.8	1
		Qwen 2.5 1.5B						
	Zero-shot	22.9	9.9	1.8	18.7	32.2	17.1	0.49
	LoRA	31.6	42.0	11.6	30.2	43.7	31.8	1
		Llama-3.2-1B-Instruct						
ja	Zero-Shot	3.7	6.8	4.3	18.4	34.6	13.6	0.52
	LoRA	12.9	22.5	9.1	27.5	39.4	22.3	1
		Qwen 2.5 1.5B						
	Zero-shot	5.2	7.1	3.2	18.7	32.8	13.4	0.45
	LoRA	26.4	27.9	11.6	26.7	39.4	26.4	1
		Llama-3.2-1B-Instruct						
ko	Zero-Shot	2.0	3.2	1.2	8.1	23.2	7.6	0.40
	LoRA	8.2	18.2	5.0	14.0	30.5	15.2	1
		Qwen 2.5 1.5B						
	Zero-shot	3.3	4.5	0.5	8.1	16.8	6.7	0.30
	LoRA	21.5	23.3	6.2	15.2	31.5	19.5	1
		Llama-3.2-1B-Instruct						
zh	Zero-Shot	3.0	7.5	2.0	18.1	25.8	11.3	0.44
	LoRA	37.9	22.6	7.5	22.7	35.2	25.2	1
		Qwen 2.5 1.5B						
	Zero-shot	11.9	7.4	1.9	18.3	27.4	13.4	0.43
	LoRA	63.2	29.1	6.6	26.1	37.3	32.5	1

Table A.1: Test performance on the main tasks (%). F-0.5 score for Text Correction (TC), F-1 for Question Answering (QA), Weighted ROUGE for Smart Reply (RP) and ROUGE-L for Text Summarization (Sum) and Tone Adjustment (Tone).

Problem Type	Dataset	Language	# Train. Samples	# Val. Samples	# Test Samples
Grammar Error Correction	Write & Improve	English	23,523	2,526	2,639
	Merlin	Italian	572	79	81
	ECSpell	Chinese	6,680	750	750
	GitHub Typo Corpus	French	616	240	227
	GitHub Typo Corpus	German	412	119	132
	GitHub Typo Corpus	Japanese	1,043	325	321
	GitHub Typo Corpus	Korean	255	75	93
	GitHub Typo Corpus	Spanish	348	137	116
Smart Reply	Persona-Chat Synthetic	All	225,061	1,000	1,000
Text Summarization	SAMSum	All	14,732	818	819
Tone Adj.	Sound Natural	All	2,245	321	642
Question Answering	SQuAD	All	65,699	1,000	1,000

Table A.2: Summary of the statistics for the employed datasets.

Lang	Method	Translation	Title generation
pt	Zero-Shot	21.6	19.4
	LoRA	41.4	21.4
tr	Zero-Shot	13.2	19.9
	LoRA	17.4	21.2
sr	Zero-Shot	9.7	12.1
	LoRA	27.2	10.8

Table A.3: Test performance on the held-out tasks (%). Languages: Portuguese, Turkish, and Serbian. Problem types: translation (to English) and Title generation. BLEU score is used for translation and ROUGE-L for title generation.

A.2.2 Detailed Continual Merging Results

We analyzed our continual merging experiments in Fig. 3 in the main text, while in this section, we report the corresponding results in more detail, including the mean and standard deviation across three random task orderings in Table A.4 and Table A.5. The first table shows the results for the Llama-3.2-1B model, while the second shows them for the Qwen-2.5-1.5B model.

A.3 Additional Analyses

A.3.1 Integration Time of Incoming LoRAs

Fig. A.1 shows the time that it takes to integrate a new incoming LoRA using K-Merge. The time is a function of the number of stored LoRAs (clusters), and it scales roughly linearly with the number of clusters. The proposed approach is lightweight, as the time varies between 0.04 and 0.18 seconds. The time required by K-Merge++ is similar to K-Merge because all additional operations take negligible time. Memory usage is dominated by text generation.

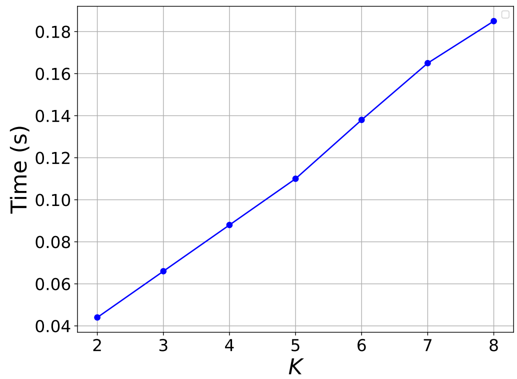


Figure A.1: Analysis of the time it takes to integrate a new LoRA for the K-Merge approach (values for K-Merge++ are roughly the same). The time is proportional to the number of stored LoRAs (clusters).

A.3.2 Cosine Similarities Analysis

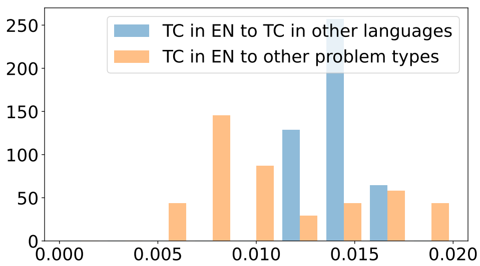
Fig. A.2 shows the histograms of LoRA cosine similarities for text correction (TC) and question answering (QA) with respect to 1) the other problem types in the same language, 2) the same problem type in different languages. We use the Llama-3.2-1B model and English as the reference language for this analysis. The results indicate that there is less similarity between different problem types in the same language than for the same problem type in different languages. Fig. A.3 reports the cross-task LoRA similarity, $\mathrm{sim}(L^{(i)}, L^{(j)}), \forall i, j = 1, \ldots, \gamma, i \neq j$, across both tasks and languages. We observe that LoRAs tend to naturally cluster per problem type (red solid boxes), while similarity per language is weaker. Nonetheless, European languages display higher

Method	K							
	1	2	3	4	5	6	7	8
Linear	0.51 ± 0.06	0.62 ± 0.06	0.69 ± 0.03	0.71 ± 0.03	0.79 ± 0.02	0.80 ± 0.04	0.81 ± 0.00	0.82 ± 0.01
TIES	0.47 ± 0.05	0.51 ± 0.03	0.51 ± 0.02	0.58 ± 0.00	0.65 ± 0.06	0.67 ± 0.01	0.71 ± 0.02	0.73 ± 0.02
DARE	0.50 ± 0.03	0.52 ± 0.02	0.54 ± 0.03	0.59 ± 0.03	0.58 ± 0.04	0.59 ± 0.03	0.60 ± 0.02	0.61 ± 0.03
DARE-TIES	0.48 ± 0.02	0.48 ± 0.01	0.49 ± 0.01	0.51 ± 0.02	0.51 ± 0.02	0.51 ± 0.02	0.52 ± 0.01	0.53 ± 0.01
OPCM	0.00 ± 0.00	0.21 ± 0.14	0.40 ± 0.04	0.38 ± 0.06	0.44 ± 0.06	0.50 ± 0.06	0.52 ± 0.03	0.58 ± 0.03
K-Merge	$\mathbf{0.63 \pm 0.00}$	$\mathbf{0.69 \pm 0.02}$	$\mathbf{0.73 \pm 0.01}$	$\mathbf{0.77 \pm 0.00}$	0.80 ± 0.02	0.81 ± 0.02	$\mathbf{0.82 \pm 0.01}$	$\mathbf{0.83 \pm 0.02}$
K-Merge++	$\mathbf{0.63 \pm 0.00}$	$\mathbf{0.69 \pm 0.02}$	$\mathbf{0.73 \pm 0.01}$	$\mathbf{0.77 \pm 0.00}$	$\mathbf{0.81 \pm 0.02}$	$\mathbf{0.82 \pm 0.03}$	$\mathbf{0.82 \pm 0.03}$	0.82 ± 0.03

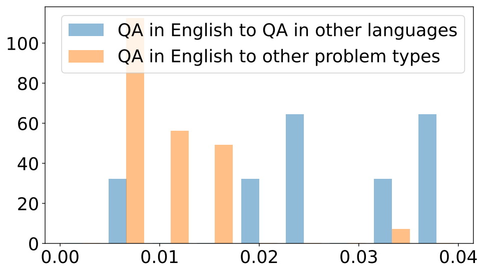
Table A.4: Llama-3.2-1B-Instruct: score $S^{(\gamma)}$ of compared merging methods at variable LoRA storage budget K. Average and standard deviation across three random task orderings. Best in bold.

Method	K							
	1	2	3	4	5	6	7	8
Linear	0.67 ± 0.01	0.71 ± 0.01	0.78 ± 0.05	0.80 ± 0.05	0.83 ± 0.05	0.85 ± 0.06	0.85 ± 0.06	0.89 ± 0.03
TIES	0.58 ± 0.01	0.59 ± 0.01	0.63 ± 0.01	0.68 ± 0.02	0.70 ± 0.01	0.75 ± 0.02	0.78 ± 0.04	0.80 ± 0.02
DARE	0.60 ± 0.01	0.60 ± 0.01	0.61 ± 0.02	0.62 ± 0.01	0.64 ± 0.02	0.66 ± 0.01	0.64 ± 0.05	0.68 ± 0.03
DARE-TIES	0.57 ± 0.01	0.56 ± 0.00	0.57 ± 0.00	0.57 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.60 ± 0.01	0.60 ± 0.01
OPCM	0.00 ± 0.00	0.05 ± 0.03	0.07 ± 0.01	0.44 ± 0.13	0.45 ± 0.15	0.49 ± 0.21	0.53 ± 0.17	0.60 ± 0.13
K-Merge	$\mathbf{0.74 \pm 0.00}$	$\mathbf{0.79 \pm 0.01}$	$\mathbf{0.83 \pm 0.01}$	$\mathbf{0.85 \pm 0.01}$	0.87 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.90 ± 0.02
K-Merge++	$\mathbf{0.74 \pm 0.00}$	$\mathbf{0.79 \pm 0.01}$	$\mathbf{0.83 \pm 0.01}$	$\mathbf{0.85 \pm 0.01}$	$\mathbf{0.88 \pm 0.01}$	$\mathbf{0.90 \pm 0.01}$	$\mathbf{0.92 \pm 0.02}$	$\mathbf{0.93 \pm 0.01}$

Table A.5: Qwen-2.5-1.5B-Instruct: score $S^{(\gamma)}$ of compared merging methods at variable LoRA storage budget K. Average and standard deviation across three random task orderings. Best in bold.



(a) Text correction



(b) Question answering

Figure A.2: LoRA cosine similarities histograms for different cases with Llama-3.2-1B. In general, there is more similarity for the same problem type in different languages rather than between problem types in the same language.

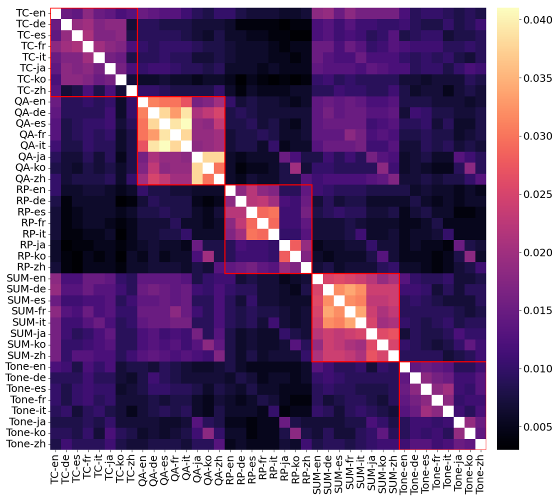


Figure A.3: Similarity $\text{sim}(L^{(i)}, L^{(j)}), \forall i, j = 1, \ldots, \gamma, i \neq j$ between LoRAs trained on Llama-3.2-1B. LoRAs tend to cluster per problem type (red solid boxes) naturally.

similarity between themselves rather than to Asian ones, as expected. We remark that the system is unaware of the overall task sequence at any time and, therefore, could not perform clustering based on this information.

A.3.3 Clustering Consistency (at $K = \alpha$)

To further confirm that LoRAs tend to cluster per problem type, we take the history of merges and analyse the frequency of problem types within each cluster. Specifically, for each cluster, we identify the most common problem type and compute the fraction of items in that cluster that belong to that problem type. The overall clustering consistency is then calculated as the total number of items matching their cluster's dominant problem type, divided by the total number of items. We restrict this analysis to the case where the number of on-device LoRA slots equals the number of problem types, i.e., $K = \alpha = 5$.

Linear has the lowest clustering consistency (82.5%). K-Merge has a higher score (83.3%) which is further improved by K-Merge++ (88.3%, which corresponds to the highest result). Interestingly, TIES has a high clustering consistency (86.7%); however, it underperforms other approaches.

A.3.4 Similarity Threshold

Table A.6 shows the performance for different values of the threshold s. We selected $s = 0.02$ as it is the median value of pairwise similarities across LoRAs on held-out tasks, but it also performs the best compared to other options.

s	0.010	0.015	0.020	0.025
K-Merge++	0.68	0.74	0.81	0.80

Table A.6: Ablation of threshold s on Llama-3.2-1B for $K = 5$. Value of $s = 0.020$, as also selected via our median rule, performs the best.

A.3.5 Larger Models

In Table A.7, we report the performance of the compared methods using Llama-3.2-3B model (instruction tuned) as the base model, confirming that our approach outperforms the competing approaches also when employing larger models.

Method	$k = 1$	$k = 3$	$k = 5$	$k = 7$
K-Merge	0.62	0.71	0.77	0.82
Linear	0.53	0.65	0.77	0.81
TIES	0.48	0.53	0.61	0.72
DARE	0.52	0.54	0.58	0.59

Table A.7: Llama-3.2-3B-Instruct: score $S^{(\gamma)}$ of compared merging methods at variable LoRA storage budget K (single seed).

A.3.6 Out-of-Domain Performance

We analyse out-of-domain generalization performance of each English LoRA to other problem types in Fig. A.4. The scores are normalized with respect to the ones of using the corresponding problem-type LoRA (i.e., the problem-type LoRA performance corresponds to a score of 1.0). Problem type pairs such as tone adjustment and text correction, or summarization and question answering, share more similarity and have relatively high scores, while, for example, reply and tone adjustment, or reply and text correction have low scores.

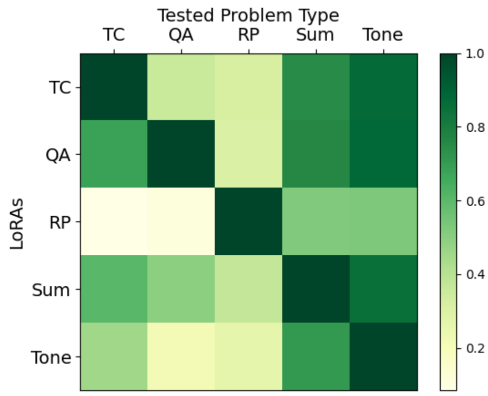


Figure A.4: Generalization performance of each English LoRA to the other problem types, normalized.

Problem Type	Language	Prompt
Grammar Error Correction	English	Remove all grammatical errors from this text:
	Spanish	Quita todos los errores gramaticales de este texto:
	French	Supprimez tous les erreurs grammaticales de ce texte:
	German	Verbessere alle grammatischen Fehler in diesem Text:
	Italian	Rimuovi tutti gli errori grammaticali da questo testo:
	Chinese	除文本中的所有法:
	Korean	주어진 사용자의 입력에 오타나 문법 오류가 있으면 고친다:
	Japanese	このテキストからすべての文法エラを削除する:
Smart Reply	English	Suggest a reply for the following text:
	Spanish	Sugiera una respuesta para el texto siguiente:
	French	Propose une réponse pour le texte suivant:
	German	Schlagen Sie eine Antwort für den folgenden Text vor:
	Italian	Suggerisci una risposta per il seguente testo:
	Chinese	建以下文本行回:
	Korean	다음 텍스트에 대한 답변을 제안하시오:
	Japanese	次のテキストにする返信を提案します:
Text Summarization	English	Summarize the following text:
	Spanish	Resume el siguiente texto:
	French	Résume le texte suivant:
	German	Zusammenfassen Sie den folgenden Text:
	Italian	Riassumi il seguente testo:
	Chinese	一下下面的文字:
	Korean	다음 텍스트를 요약하시오:
	Japanese	次の文章を要約します:
Tone Adj. (Professional)	English	Changes a given user's input sentence or text to the Professional style:
	Spanish	Cambia la oración o el texto introducido por un usuario al estilo Profesional:
	French	Transforme la phrase ou le texte saisi par un utilisateur en style Professionnel:
	German	ändert die Eingabe eines bestimmten Benutzers in einen Professionellen Stil:
	Italian	Cambia la frase o il testo immesso da un utente in stile Professionale:
	Chinese	定用的入句子或文本更改格:
	Korean	주어진 사용자의 입력을 전문적인 문체로 변경한다:
	Japanese	指定されたユザの入力文またはテキストを プロフェッショナル スタイルに更する:
Tone Adj. (Casual)	English	Changes a given user's input sentence or text to the Casual style:
	Spanish	Cambia la oración o el texto introducido por un usuario al estilo Informal:
	French	Transforme la phrase ou le texte saisi par un utilisateur en style Informel:
	German	ändert die Eingabe eines bestimmten Benutzers in einen Freundlichen Stil:
	Italian	Cambia la frase o il testo immesso da un utente in stile Informal:
	Chinese	定用的入句子或文本更改日常格:
	Korean	주어진 사용자의 입력을 평범한 문체로 변경한다:
	Japanese	指定されたユザの入力文またはテキストを カジュアル スタイルに更する:
Tone Adj. (Witty)	English	Changes a given user's input sentence or text to the Witty style:
	Spanish	Cambia la oración o el texto introducido por un usuario al estilo Ingenioso:
	French	Transforme la phrase ou le texte saisi par un utilisateur en style Spirituel:
	German	ändert die Eingabe eines bestimmten Benutzers in einen Witziger Stil:
	Italian	Cambia la frase o il testo immesso da un utente in stile Spiritoso:
	Chinese	定用的入句子或文本更改机智格:
	Korean	주어진 사용자의 입력을 재치있는 문체로 변경한다:
	Japanese	指定されたユザの入力文またはテキストを ウィットに富んだ スタイルに更する:
Tone Adj. (Paraphrase)	English	Paraphrase the following text:
	Spanish	Parafrasea el siguiente texto:
	French	Paraphraser le texte suivant:
	German	Fassen Sie den folgenden Text zusammen:
	Italian	Parafrasare il testo seguente:
	Chinese	解以下文字:
	Korean	다음 텍스트를 의역하세요:
	Japanese	次のテキストを言い換えてください:
Question Answering	English	Answer the following question:
	Spanish	Responde a la siguiente pregunta:
	French	Réponds à la question suivante:
	German	Beantworten Sie die folgende Frage:
	Italian	Rispondi alla seguente domanda:
	Chinese	回答以下:
	Korean	다음 질문에 답하시오:
	Japanese	次の質問に答えましょう:

Table A.8: Prompts for each problem type and language, from (Ceritli et al., 2025).