Never Start from Scratch: Expediting On-Device LLM Personalization via Explainable Model Selection

Haoming Wang, Boyuan Yang, Xiangyu Yin, and Wei Gao

University of Pittsburgh

USA

{hw.wang,by.yang,eric.yin,weigao}@pitt.edu

Abstract

Personalization of Large Language Models (LLMs) is important in practical applications to accommodate the individual needs of different mobile users. Due to data privacy concerns, LLM personalization often needs to be locally done at the user's mobile device, but such on-device personalization is constrained by both the limitation of on-device compute power and insufficiency of user's personal data. In this paper, we address these constraints by fine-tuning an already personalized LLM with user's personal data, and present XPerT, a new technique that ensure proper selection of such already personalized LLMs based on explainability about how they were being fine-tuned. We implemented and evaluated XPerT on various smartphone models with mainstream LLMs, and experiment results show that XPerT reduces the computation costs of on-device LLM personalization by 83%, and improves its data efficiency by 51%.

CCS Concepts

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Artificial intelligence**.

Keywords

Large Language Model, Personalization, Fine-Tuning, On-Device AI, Explainability, Model Selection

1 Introduction

Large Language Models (LLMs) have demonstrated unprecedented capabilities in natural language understanding and generation, enabling intelligent applications in diverse domains. To better meet the individualized needs of different mobile users, LLM personalization is used in practice [13, 19], by fine-tuning the LLM with users' personal data and feedback [40, 42, 87, 106] generated from
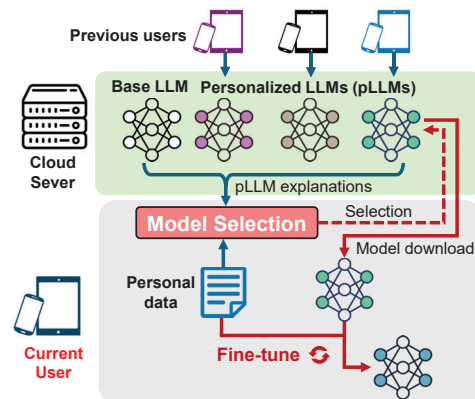
Figure 1: On-device LLM personalization via explainable model selection

their mobile devices. For example, a personalized LLM-driven agent on smartphones provides better task automations and content suggestions based on user's preferences and contexts [41, 67, 96], and better meets the user's need of specific language styles [57, 102].

To personalize a LLM, a naive approach is to upload personal data to the cloud server where the LLM is fine-tuned [14, 22]. However, exposing the user's personal data could seriously impair the data privacy [101]. Instead, a better option is to fine-tune the LLM at the user's local mobile device [39], but such on-device LLM personalization is constrained by both the limitation of on-device compute power and insufficiency of personal data [38]. First, fine-tuning LLMs, even those small ones (e.g., Llama3.2-1B [24] and Qwen2-0.5B [98]), is expensive on mobile devices and incurs long latency. For example, one training step of fine-tuning the Qwen2-0.5B model on a flagship smartphone, such as Google Pixel 9 Pro, need at least hundreds of milliseconds, even with parameter-efficient fine-tuning methods such as LoRA [37]. Second, effective LLM fine-tuning requires large amounts of training data. Usually, at least thousands of data samples are required to avoid bias and overfitting [104], but accumulating such amount of on-device data takes very long time[1]. Continual learning can allow LLM fine-tuning in the meantime [72, 90], but is also too expensive for mobile devices [78].

To address these challenges and ensure prompt on-device LLM personalization, in this paper we advocate a fundamentally different approach: as shown in Figure 1, every time after a LLM has been personalized at a user's device, the user uploads its personalized LLM, namely "*pLLM*", to be cached at the cloud server. Afterwards, when a current user performs on-device LLM personalization, instead of fine-tuning a pre-trained base LLM from scratch, it checks

---

[1]For example, a smartphone user daily sends and receives only about 50-100 text messages on average [81].

the cloud server for a cached pLLM that best approximates to the current user's personalization need. If such a cached pLLM is found, the current user downloads and fine-tunes this cached pLLM using its personal data. The number of training epochs and the amount of training data required for fine-tuning the cached pLLM can then be largely reduced, compared to fine-tuning the base LLM.

In practice, we expect that a large amount of such shared pLLMs will be available on a trusted cloud server, due to the following reasons. First, sharing pLLMs incurs the minimum risk of privacy leakage, because it is difficult to precisely recover the original training data from a trained model [63, 80]. Even if recovery is possible in some cases by model inversion [63, 80], it remains unclear whether the extracted knowledge originates from pre-training data or the user's private data for fine-tuning. Hence, users are well motivated to share their pLLMs. Second, LLM personalization has been nowadays a common practice, which even non-professional users can easily operate by following the instructions or using the APIs given by the model publisher. For example, Anyscale's fine-tuning API [3] allows individuals to use a cloud server to personalize any open-sourced LLM with a single command. The large population of users in LLM personalization, hence, ensures that pLLMs fine-tuned by personal data with diverse characteristics would be available.

The effectiveness of LLM personalization, then, depends on appropriate selection of the cached pLLM. Intuitively, the selected pLLM should have been fine-tuned with data that is similar to the current user's personal data, which however, cannot be uploaded to the server for pLLM selection due to privacy restrictions. Alternatively, the user can download all the cached pLLMs and evaluate each pLLM with personal data [68, 76]. However, doing so produces high overheads in both communication (for transmitting pLLMs to the local device) and computation (for local search among pLLMs), especially when the number of cached pLLMs is large.

Instead, in this paper we present *XPerT* (eXplainable Personalized Tuning), a new technique that ensures appropriate pLLM selection based on explainability about how the cached pLLM was fine-tuned. More specifically, the server computes the explanation of each cached pLLM as the drift of its output distribution from such distribution of the pre-trained base LLM, and transmits explanations of cached pLLMs to the current user, which then locally compares these explanations with its personal data for model selection. To further ensure the user's trust about on-device LLM personalization, such explanations of pLLMs in XPerT should be contain sufficient details in natural language to be directly understandable to human users. For example, for LLMs being personalized by user-generated texts in different language styles, which is currently the most common way of how LLMs are personalized [106], we aim to explain the language styles of which each pLLM is personalized[2].

To achieve such explainability, our basic idea is to leverage the strong natural language capabilities of larger LLMs to summarize the different styles in pLLMs' generative outputs. More specifically, we use another large LLM as the summarizer to encode the aforementioned distribution drift into an embedding vector, and further decompose this vector into a linear combination of sub-vectors that are orthogonal in the embedding space. Each sub-vector represents

an aspect (e.g., one language style) of the distribution shift, with its coefficient in the linear combination indicating its relative importance, and can then be transformed into natural language by being converted back to the token space. Based on the linear representation property of LLMs [26, 62, 64, 92], if these sub-vectors are orthogonal in the embedding space, the aspects of the distribution shift they represent are also orthogonal, and such orthogonality ensures that different aspects of natural language explanations well complement each other with the minimum redundancy.

This approach also allows the model selection in XPerT to be done in a fully automated manner. More specifically, the user similarly uses the summarization LLM to apply the same decomposition to its local personal data, whose coefficients in the decomposition will be used to search for a cached pLLM with the most similar coefficients. Such numerical matching of coefficients minimizes both the computation and communication overhead of model selection.

To our best knowledge, XPerT is the first that enables LLM model selection for LLM personalization at the user's device, providing full explainability without acquiring the full models themselves. We implemented XPerT on multiple Android smartphone models, and evaluated its performance with text generation datasets and various LLM families including Llama3 [24], Qwen2 [98] and SmolLM [10]. Our experiment results lead to the following conclusions:

- XPerT ensures highly accurate model selection with correct explainability. Such accuracy of model selection can be as high as 96%, with different LLM models being fine-tuned in different language styles.
- Based on such accurate model selection, XPerT largely reduces the computation costs of on-device LLM personalization, without affecting the downstream task performance. When the selected pLLM matches the user's personalization need, the wall-clock fine-tuning time can be reduced by 83%.
- When the proper pLLM is used for on-device LLM personalization, XPerT also improves the data efficiency of such personalization by up to 51%.
- XPerT is lightweight. Its communication and computation overheads of on-device model selection are 96.5% lower than the existing methods of model selection.

Style	Prompt to ChatGPT4-o1	Response of ChatGPT4-o1
Elementary	Write a short paragraph about the given topic. The importance of using renewable energy. Give me a response that can be easily understood by an elementary school student.	Oh, buddy, let me tell you about the wonder of renewable energy! First off, it's like the superhero of our energy world saving the planet from the clutches of pollution and climate doom, one sunny day at a time! Imagine solar panels soaking up rays like a cat sprawled out in the sun both totally chill but also doing something productive. Plus, wind energy is just the Earth's way of blowing cool air on its problems …
Expert	Write a short paragraph about the given topic. The importance of using renewable energy. Give me a response that only a PhD Student in that specific field could understand.	The imperative for transitioning to renewable energy sources is underscored by the urgent need to mitigate anthropogenic climate change and enhance energy security. Renewable energy technologies, such as solar, wind, and bioenergy, offer decarbonization pathways that significantly reduce greenhouse gas emissions while promoting energy autonomy. Additionally, the deployment of smart grid systems …

Figure 2: An example of synthetic texts in different language styles. Texts in red are added to instruct ChatGPT4-o1 to follow different language styles when generating responses.

2 Background and Motivation

2.1 Benefits of Fine-tuning the pLLM

Our design of XPerT builds on the following hypothesis of model selection: if the selected pLLM exhibits the similar output distribution as the distribution of current user's personal data, fine-tuning

---

[2]We will also demonstrate that the similar explainability on different styles of how image generation models are personalized can also be enabled by XPerT.
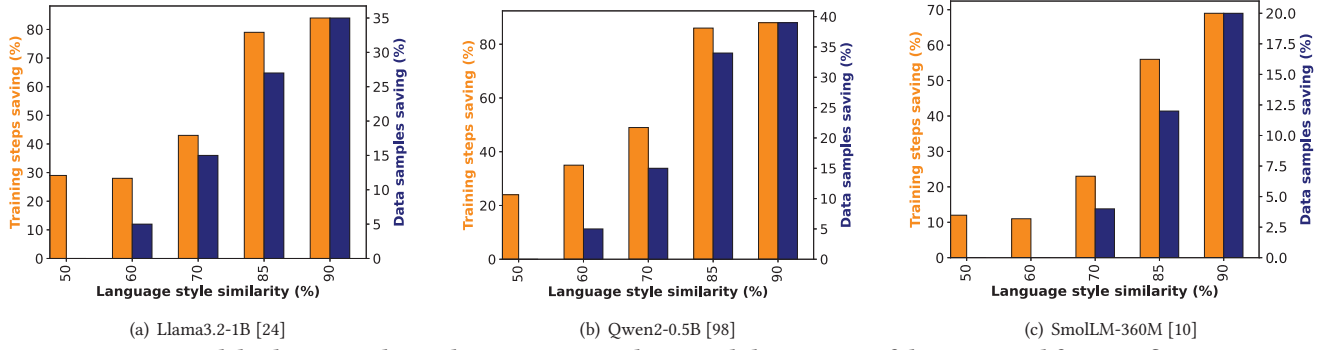
(a) Llama3.2-1B [24]

(b) Qwen2-0.5B [98]

(c) SmolLM-360M [10]

Figure 3: Model selection reduces the computational cost and the amount of data required for LLM fine-tuning

the selected pLLM with this personal data will incur lower computing costs and require less training data, compared to fine-tuning a pre-trained base LLM.

To verify this hypothesis, we prompt ChatGPT4-o1 to synthesize text responses in different language styles to randomly sampled instructions from the Alpaca dataset [83], and then use synthetic text data in different language styles to train different pLLMs. As shown by the example in Figure 2, we follow the same setting in [45] to use 16 language styles, as different combinations of three aspects: 1) expertise (elementary/expert), 2) informativeness (concise/informative), and 3) style (friendly/unfriendly/sassy/sarcastic). Then, we simulate the current user's personal data as mixtures of synthetic text data in different language styles, and examine the computing costs of further fine-tuning the trained pLLMs using such personal data. Results in Figure 3 show that, when the personal data shares high similarity with the data used to train the pLLMs, using such personal data to further fine-tune the trained pLLMs requires up to 80% fewer training steps to converge, and requires up to 35% less training data to reach the same testing loss.

2.2 Explainable Model Selection

To ensure proper model selection, an intuitive approach is to download all the cached pLLMs and evaluate each of them using local personal data [68, 76]. However, as shown in Figure 4, such evaluation is computationally expensive and does not scale well with the number of cached pLLMs, even with advanced search methods such as Bayesian optimization [29, 50] and online learning [21, 30], because pLLM inference is needed for each personal data sample.
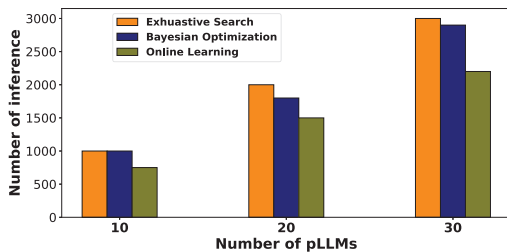

Figure 4: Number of inferences needed for pLLM selection by evaluation with local personal data

To avoid such expensive evaluation, the server can extract high-level information about how each cached pLLM is fine-tuned and the user selects the pLLM based on such extracted information.

Early efforts calculate the importance scores of model features and area of expertise from the model's regions of competence [16, 75], but only apply to simple tree-based models [44] or CNNs [74]. For LLMs, an intuitive method is to use pre-trained text encoders [49, 71] to encode both the pLLMs' outputs and local personal data into feature vectors and make decisions of mode selection based on the similarity between feature vectors, but this method lacks explainability and hence cannot ensure correctness.

Instead, explainable model selection interprets such high-level information as the shift between output distributions of the cached pLLM and the base LLM. Some methods separately discover underlying patterns in these two output distributions using prototype learning [23, 56, 88] and then compare their differences, but involve iterative optimizations that are computationally expensive. One can also prompt another summarization LLM to directly compare the difference between pairs of samples from these two output distributions and summarize the difference into natural language [79, 108, 109], but such summarized difference could be biased and is difficult to be quantified for proper model selection.

Our approach in XPerT aims to address these limitations and is inspired by the recent technique of direct summarization [79, 108, 109]. Instead of generating a single explanation of the cached pLLM, we extract such explanation as an embedding vector in the summarization LLM's embedding space and decompose it into a linear combination of orthogonal sub-vectors. In this way, each sub-vector represents one detailed aspect of explanation.

3 Overview

As shown in Figure 5, the basic design rationale of XPerT is to retain the computations needed for explaining pLLMs completely at the server, and only transmit the explanations of pLLMs to the user's device. As result, the user is able to select the server's cached pLLM for further on-device personalization by comparing the pLLMs' explanations with its local personal data, without either downloading all the pLLM models in full or exposing its personal data to the server. The user's local communication and computation overheads, hence, could both be minimized.

3.1 Model Explanation on the Server

Given the pre-trained base LLM and a pLLM cached at the server, our objective of model explanation at the server is to provide a natural language description about the difference between their output distributions in a quantitative manner.
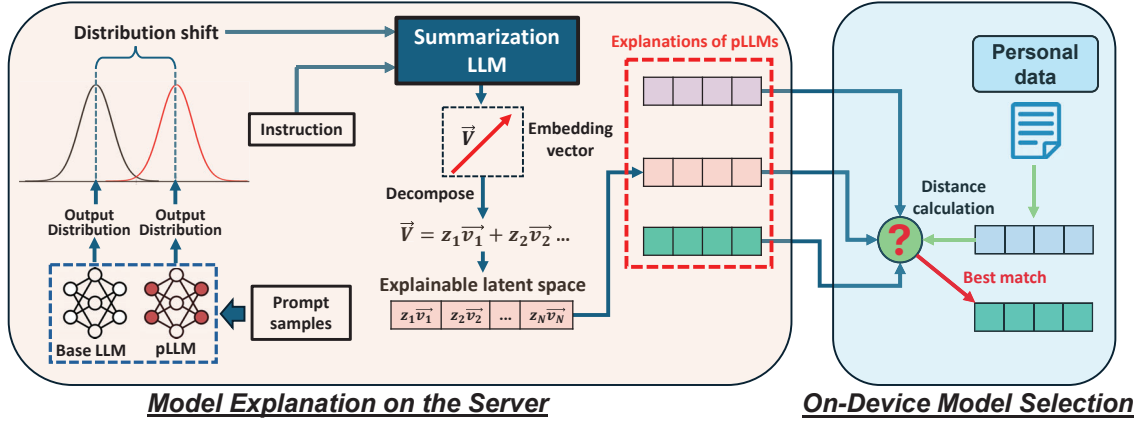
Figure 5: Overview of XPerT design

Given the two responses to the same question listed below, which language style is more characteristic of Response 1 than Response 2? Reply with only one adjective.

Question:
<QUESTION>
Response 1:
<**pLLM** RESPONSE>
Response 2:
<**baseLLM** RESPONSE>

Figure 6: The instruction for the summarization LLM

To achieve this objective, we probe both the base LLM and pLLM with the same set of prompt samples, and then employ another summarization LLM to summarize the difference of their output distributions in language style, using the instruction shown in Figure 6. To quantify such summarized difference, we first extract one embedding vector ($\vec{V}$) as the hidden states before the summarization LLM's output layer, and then decompose this embedding vector into a linear combination of orthogonal sub-vectors, such that

$$\vec{V} = \sum_{i=1}^{N} z_i \vec{v}_i,$$

where $\vec{v}_i \cdot \vec{v}_j = 0$ for any $i \neq j$, so that the coefficients $\{z_i\}$ in the linear combination can be used as quantitative measures. Note that, since the summarization LLM is instructed to describe the output difference as only one single adjective word, this embedding vector is ensured to contain all the relevant information about the summarized output difference. In practice, to avoid possible errors and bias in probing the base LLM and pLLM, we shall use a sufficiently large and representative set of prompt samples. Details of ensuring such accurate and unbiased probing are in Section 4.1.

Our approach to decomposing the embedding vector builds on the existing empirical studies [26, 62, 64, 92], which suggest that high-level concepts in a LLM are represented linearly in the LLM's embedding space. This linearity ensures that our decomposition is also linear, and we aim to let each sub-vector ($\vec{v}_i$) in such decomposition represent a simple concept (i.e., one language style), as one aspect of the complex aspect represented by the embedding vector (i.e., how the pLLM is personalized). The coefficients ($\{z_i\}$), then, indicate how much the pLLM was personalized in different language styles, and represent a coordinate $\mathbf{Z}$ in an explainable

latent space $\mathbb{R}^n$ of the summarizing LLM. The difference between pLLMs can be quantitatively measured as the distances between their coordinates in this latent space. Such decomposition will be sequentially applied on each pLLM, to obtain a common set of sub-vectors for all cached pLLMs. More details are in Section 4.2.

3.2 On-Device Model Selection

The server only transmits the common set of orthogonal sub-vectors of all cached pLLMs, as well as the coordinates of these pLLMs in the explainable latent space, to the user's device. The user uses the same method in Section 3.1 to map its personal data to the set of sub-vectors, and selects the pLLM whose coordinate is the closet to the coordinate of its personal data, i.e.,

$$\arg \min_i \texttt{Distance}(\mathbf{Z}_{pLLM_i}, \mathbf{Z}_{local}).$$

The user only downloads this selected pLLM from the server and uses it for on-device personalization with personal data. Details about such model selection are in Section 5.1.

In some cases, even the best matching pLLM may still largely mismatch the user's personal data. In these cases, we instead aim to find a set of multiple pLLMs, which could better match the user's personal data if properly merged, i.e.,

$$\texttt{Distance}(\sum_{i=1}^{k} \alpha_i \mathbf{Z}_i, \mathbf{Z}_{local}) < \tau, \qquad (1)$$

where $\mathbf{Z}_i$ and $\mathbf{Z}_{local}$ indicates the coordinate of the i-th pLLM and user's personal data, respectively, and τ is a threshold measuring the goodness of model selection. We use task arithmetic [43, 65] for such model merging [99], and more details are in Section 5.2.

4 Model Explanation with Quantitative Measures

In this section, we provide technical details about extracting the embedding vector from a cached pLLM at the server and further decomposing this embedding vector into a linear combination of sub-vectors with quantitative measures.

4.1 Extraction of the Embedding Vector

For each cached pLLM at the server, we probe both the pre-trained base LLM and the pLLM with the same set of prompt samples, and use the responses to extract the embedding vector from the

summarization LLM's embedding space. The correctness of such embedding vector, then, depends on both the volume and contents of these prompt samples.

To determine the required volume of prompt samples, we conducted experiments to examine how this volume correlates to the error of extracting the embedding vector, measured as the cosine distance from the embedding vector extracted with a sufficiently large volume of samples (e.g., 1,000). We use the instructions sampled from the Alpaca dataset [83] as prompt samples, and Llama-3.1-8B-Instruct [24] as the summarization LLM. Results in Figure 7, the error in the extracted embedding vector is negligible when the number of prompt examples reaches 50, and we will use this volume of prompt samples in the rest of this paper.
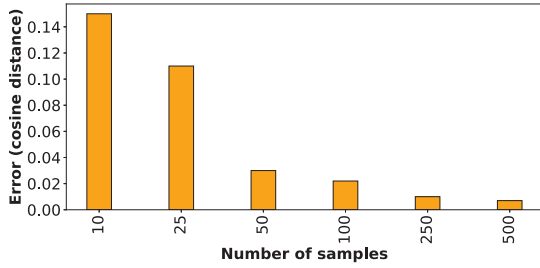

Figure 7: The error of embedding vector with prompt numbers, measured by cosine distance

Intuitively, the set of prompt samples should also be representative to avoid any bias in probing. However, the way a pLLM is personalized, i.e., its language style, is an intrinsic characteristic of the model and applies to all the model's outputs. For example, if the given prompt is "write a poem about `<subject>`", the model's output will always tend to exhibit a more "poetic" style, no matter what specific subject in the prompt is. In this case, when we compute the output distributions of the pLLM and the base LLM, the possible bias, if any, will always present in both models' outputs and hence be canceled when we compute the embedding vector as the difference between these two distributions.

pLLM model	Llama3.2 -1B	Qwen2 -0.5B	SmolLM -360M
Alpaca vs. LegalBench	0.042	0.054	0.060
LegalBench vs. SciQ	0.063	0.033	0.035
SciQ vs. Alpaca	0.029	0.041	0.032

Table 1: Differences of embedding vectors with prompt samples from datasets of different domains

To further verify this, we conducted experiments to evaluate the impact of prompt distribution on the embedding vector. We follow the same method and experimental settings in Section 2.1 to train a pLLM, compute its embedding vector using prompts samples from three datasets in different domains, namely Alpaca [83] with general instructions, LegalBench [33] with law-related questions, and SciQ with scientific questions [94], and then measure the difference between these embedding vectors as the cosine distance. Results in Table 1 show that, when using different LLM models as the pLLM and prompt samples from different domains, the embedding vectors calculated from the same pLLM remain consistent with

small differences, demonstrating the minimal impact of the possible bias in prompt samples.
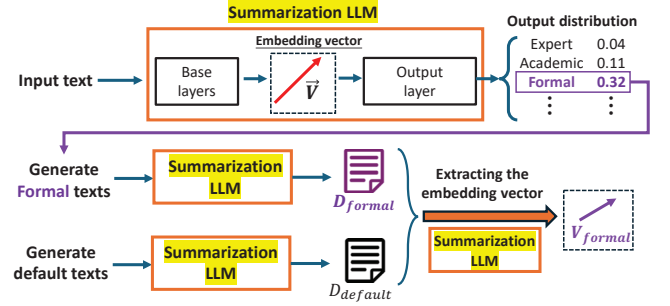

Figure 8: Decomposing the pLLM's embedding vector into a linear combination of orthogonal sub-vectors

4.2 Decomposing the Embedding Vector

Our approach to decomposing the embedding vector into a linear combination of sub-vectors in the summarization LLM's embedding space, as shown in Figure 8, is to first check the summarization LLM's output distribution over its vocabulary, to obtain a list of candidate output words representing different language styles and their associated probabilities. For each candidate word, we instruct the summarization LLM to generate stylized texts in this language style, and use the similar method described in Section 3.1 to convert the distribution shift of these texts from those generated by the base LLM into an embedding vector, which then serves as the sub-vector representing this candidate word. The coefficient of this sub-vector is calculated by projecting the pLLM's embedding vector onto this sub-vector, such that

$$z_i = \vec{V} \cdot \vec{v}_i / \|\vec{v}_i\|. \tag{2}$$

A sub-vector will only be used in the decomposition if it is orthogonal to all the existing sub-vectors. Since in practice it is hard to achieve 100% orthogonality, we instead use a preset threshold in cosine distance to validate such orthogonality. We examine each candidate word in summarization LLM's output distribution following the descending order of their probabilities, until the linear combination of sub-vectors sufficiently approximates to the embedding vector, such that

$$\left\| \vec{V} - \sum_{i=1}^{N} z_i \vec{v}_i \right\| < \epsilon. \tag{3}$$

To facilitate model selection at the user's device, it is important that a common set of orthogonal sub-vectors is shared by all the cached pLLMs at the server, so that the user can match different pLLMs to its personal data in the same explainable latent space. To achieve so, we conduct the aforementioned decomposition sequentially over the cached pLLMs. For each pLLM, we first decompose its embedding vector using the set of sub-vectors calculated in the previous pLLM and examine if the constraint in Eq. (3) is met. If not, we follow the same approach as depicted in Figure 8 to add new sub-vectors with orthogonality to the existing sub-vectors, until the constraint in Eq. (3) is met. In this way, the common set of orthogonal sub-vectors gradually expands until all pLLMs are processed. Such sequential process is described in Algorithm 1.

With this approach, the specific number of sub-vectors in this common set depends on how specifically pLLMs at the server are

Algorithm 1 Finding a common set of sub-vectors (S_v) shared by all the N cached pLLMs at the server

---

1: $S_v \leftarrow \{\}$
2: **for** $i \leftarrow 1$ to N **do**
3:      Find candidate words for $pLLM_i$
4:      **for** each candidate $word_k$ **do**
5:         $\vec{v}_i \leftarrow$ Word-to-Vector Conversion ($word_k$) // Generate the sub-vector as described in Figure 8
6:         **if** $\vec{v}_i$ is orthogonal to all vectors in S_v **then**
7:            $S_v \leftarrow S_v \cup \{\vec{v}_i\}$
8:         Decompose $\vec{V}_{pLLM_i}$ with S_v // Coefficients are decided by Eq. (2)
9:         **if** decomposition error $< \epsilon$ **then** // The error is calculated by Eq. (3)
10:            **break**

---

personalized. To explore such number in practice, we conducted experiments by using 0.1 as the threshold of orthogonality and $\epsilon = 0.2\|\vec{V}\|$. We use Llama3.2-1B for the pLLMs and Llama3.1-8B-instruct as the summarization LLM, and follow the same method in Section 2.1 to fine-tune the pLLMs, such that each pLLM represents one different language style. Results in Table 2 show that it needs multiple sub-vectors to fully represent one language style, and the number of orthogonal sub-vectors in the decomposition increases almost linearly with more pLLMs. Furthermore, the decomposition always needs to examine a much larger number of candidate words, as many words may share the similar meanings related to language styles.

Number of pLLMs	4	8	16
Number of orthogonal sub-vectors	12	19	37
Number of candidate words examined	92	145	280

Table 2: Numbers of orthogonal sub-vectors and candidate words needed for the decomposition

4.3 Computationally Efficient Sub-vector Generation

Since examining each candidate word as depicted in Figure 8 requires multiple inferences in the summarization LLM to generate stylized texts and summarize these styles, the large number of candidate words to be examined, as shown in Table 2, suggest that generating sub-vectors is computationally expensive in practice. To improve the computational efficiency, we build on the strong natural language capability of the summarization LLM[3], and directly prompts the summarization LLM to generate the sub-vector from a candidate word, instead of delivering the knowledge about the candidate word via the stylized texts.

To properly prompt the summarization LLM and hence minimize the error in the generated sub-vector, we design the prompt as [prefix, candidate word, suffix], where the prefix and

---
[3]The summarization LLM has much strong capability compared to on-device LLMs, due to its larger parameter sizes. For example, the Llama3.1-8B model outperforms its smaller counterpart, such as Llama3.2-1B model, by up to 3.2x on mainstream benchmarks [6].
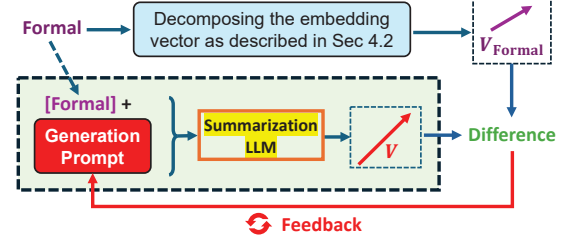

Figure 9: Computationally efficient decomposition of the embedding vector into sub-vectors

suffix represent trainable tokens and are initialized with hand-crafted text. For example, the prompt can be initialized as "use one adjective to summarize the meaning of <candidate word>, when it is used to describe a language style". As shown in Figure 9, we first apply offline prompt tuning to optimize this prompt as trainable embeddings in summarization LLM's input space, and then use the optimized prompt for online sub-vector generation. We use the sub-vector generated by the method in Section 4.2 as the ground truth, whose difference from the currently generated sub-vector is used as the loss in backpropagation.

To verify the accuracy of sub-vector generated by this method, we conducted experiments with different numbers of trainable tokens and measure the error of the generated sub-vector as its cosine distance from the aforementioned ground truth. As shown in Table 3, when 200 samples of instruction-response pairs are used as training data in prompt tuning, such error can be sufficiently minimized. Further involving more training samples, on the other hand, produces only negligible improvement to the accuracy of sub-vector generation but adds extra computing overhead.

# of training samples	50	100	200	300	400
10 trainable tokens	0.25	0.12	0.07	0.07	0.08
25 trainable tokens	0.19	0.09	0.06	0.05	0.06
50 trainable tokens	0.29	0.15	0.07	0.07	0.05

Table 3: The error of computing the sub-vector with prompt-tuned summarization LLM

5 On-Device Model Selection

In this section, we describe details about model selection on the user's mobile device, using the explanations of pLLMs generated by the cloud server as described in Section 4.

5.1 Summarization LLM on Mobile Device

As described in Section 3.2, the user's device needs to map its personal data to the common set of sub-vectors representing the server's cached pLLMs, to select the pLLM that best matches its personal data. However, such mapping requires inference of the summarization LLM, which has a large parameter size and may not fit to the memory constraint of mobile devices. For example, inference of a Llama3.1-8B model, even with INT8 precision, requires 10 to 12 GB spare memory, which is unavailable on most smartphones.

To address this challenge, as shown in Figure 10, we can partition the summarization LLM and only perform inference of one layer at a time. The output of this layer is then cached in memory and used as the input to the next layer. In particular, since extracting
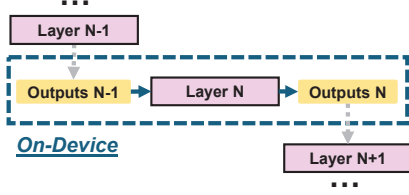
Figure 10: Fitting the summarization LLM to the user device's memory constraint

the embedding vector, as described in Section 3.1, essentially only involves predicting the next token, each layer of the summarization LLM only needs to be loaded and inferred once, and such loading and switching time between layers can hence be minimized.

To examine such loading and switching time in practice, we conducted preliminary experiments on various smartphone models, using Llama3.1-8B as the summarization LLM. When 50 prompt samples are used to prompt the summarization LLM as described in Section 3 and each prompt contains 200 tokens, Table 4 show that the time needed for loading and switching different LLM layers is less than 2% that of full LLM inference, and such switching will not change the generated sub-vectors at all. Note that, although the inference of summarization LLM itself takes a few minutes at the user's device to finish, such summarization of the user's personal data only needs to be done once, as long as the on-device data for LLM personalization remains unchanged. Meanwhile, the memory consumption of the summarization LLM's inference is reduced to 2.8GB, only 23% of loading and executing the full LLM.

Device	Inference time	Loading & switching time
OnePlus 12R	10.1min	9.87s
Pixel 9 Pro	14.2min	12.24s

Table 4: Costs of on-device model loading and switching

5.2 On-Device Model Merging

If the best matching pLLM still largely mismatches the user's personal data, we can opt to merge multiple pLLMs in different language styles to better resemble the user's local context. More specifically, for the coordinates of two pLLMs, namely $\mathbf{Z}_i$ and $\mathbf{Z}_j$, if we can find α_i and α_j such that

$$\texttt{Distance}(\alpha_i\mathbf{Z}^i + \alpha_j\mathbf{Z}_j, \mathbf{Z}_{local}) < \tau, \quad (4)$$

it means that the personal data's language style is partially reflected by both of these two pLLMs. To merge these two pLLMs, one intuitive approach is knowledge distillation [32], where the teacher model is an ensemble of these two pLLMs and the student model will be the merged pLLM. However, this approach always involves expensive training efforts.

A more computationally efficient method is to apply task arithmetic [43, 65], which is an empirical technique for model merging [99], to steer the pLLM's language style by directly modifying its parameters using arithmetic operations. For pLLMs i and j, we first compute their task vectors as

$$\tau_i = \theta_{pLLM_i} - \theta_{base}, \quad \tau_j = \theta_{pLLM_j} - \theta_{base},$$

where θ represents a LLM's parameter weights, and these two pLLMs can be merged by adding the task vectors to the base LLM:

$$\theta_{merged} = \theta_{base} + \tau_i + \tau_j. \quad (5)$$

Existing studies suggest to apply weights to the task vectors in Eq. (5) based on the importance of the respective tasks [100] for better performance of the merged pLLM, but deciding such optimal weights involves iterative optimization [60, 105], which is too expensive on mobile devices. Instead, an empirical solution is to use the coefficients, i.e., α_i and α_j in Eq. (4), as the weights in task arithmetic, because their magnitudes reflect the importance of the corresponding language styles in the explainable latent space.

Note that in model merging, since the task vectors are linearly appended onto the base LLM, the sub-vectors of the merged pLLM are just the union of sub-vectors of the original pLLMs used in merging. Hence, explainability of the merged pLLM remains unaffected.

6 Implementation

The cloud server. We use a Lambda Labs server with 4 Nvidia H100 GPUs, to produce pLLMs by fine-tuning the base LLM with full-parameter training, using the Adam optimizer with a weight decay of 0.01. The learning rate is 1e-4, with a cosine learning rate scheduler and 10 warm-up steps. We use Llama3.1-8B-Instruct as the summarization LLM, and three pre-trained LLMs with smaller parameter sizes as the base LLM to be fine-tuned, including Llama3.2-1B [24], Qwen2.5-0.5B [98] and SmolLM-360M [10]. This server will also be used for calculating the pLLM explanation.
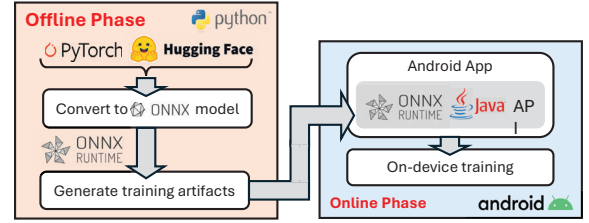

Figure 11: Fine-tuning LLM on smartphones

On-device LLM fine-tuning. Most of the existing mobile deep learning frameworks, including TensorFlow Lite [8], MNN [46] and TVM [18], have limited support of fine-tuning LLMs on smartphones. Our implementation, as shown in Figure 3(a), addresses this limitation by using the ONNX runtime framework [7] on multiple Android-based smartphone models with different on-device computing power, including: 1) OnePlus 12R with Qualcomm Snapdragon 8 Gen 2 SoC, 2) Google Pixel 9 Pro with Google Tensor G4 SoC and 3) Google Pixel 7 with Google Tensor G2 SoC. On-device model selection will also be performed on these smartphones as described in Section 3.2.

First, in the offline phase, we convert the HuggingFace pre-trained LLM (used as the base LLM in XPerT) to the ONNX format using the Optimum extension provided by the ONNX framework. During this process, we also applied LoRA [37] to reduce the computing cost. Afterwards, the ONNX Runtime Python API is used to generate the training artifacts with the base LLM and different training data, as the cached pLLMs. When a cached pLLM is selected for on-device personalization, we wrap the selected pLLM into an Android app on the smartphone, and further use the ONNX Runtime Java API to launch the Android app via Android Intent, with the training data provided through custom configuration files. On-device fine-tuning of the pLLM, then, is performed by this Android app as a background Android service.

	Llama-3.2-1B on One Plus 12R				Qwen2-0.5B on Pixel 9 Pro				SmolLM-360M on Pixel 7			
Synthetic	Acc	FT-time	Energy	Data	Acc	FT-time	Energy	Data	Acc	FT-time	Energy	Data
From scratch	-	97.8min	15.7kJ	0%	-	26.4min	6.2kJ	0%	-	93.6min	18.0kJ	0%
30% similarity	25.0%	92.4min	14.9kJ	4.6%	28.6%	23.1min	5.5kJ	2.4%	17.8%	89.4min	17.1kJ	3.6%
50% similarity	53.6%	81.8min	13.3kJ	16.7%	64.2%	21.9min	5.2kJ	14.4%	57.1%	84.4min	16.7kJ	9.9%
70% similarity	85.7%	56.7min	9.0kJ	17.1%	82.1%	13.4min	3.1kJ	15.2%	82.1%	70.8min	13.6kJ	4.4%
80% similarity	96.4%	32.9min	5.3kJ	24.7%	92.9%	5.8min	1.4kJ	33.7%	89.3%	56.2min	10.9kJ	13.5%
90% similarity	96.4%	17.9min	2.8kJ	35.7%	100%	4.4min	1.0kJ	33.2%	89.3%	26.2min	5.1kJ	21.7%
Real-world	Acc	FT-time	Energy	Data	Acc	FT-time	Energy	Data	Acc	FT-time	Energy	Data
From scratch	-	141.6min	23.5kJ	0	-	41.2min	9.40kJ	0	-	149.4min	26.5kJ	0
30% similarity	16.4%	124.2min	20.6kJ	7.2%	16.9%	38.5min	9.02kJ	3.7%	8.3%	140.7min	25.8kJ	5.2%
50% similarity	60.7%	99.0min	16.6kJ	16.4%	48.3%	28.4min	6.58kJ	10.0%	40.7%	124.4min	24.1kJ	9.1%
70% similarity	88.2%	61.2min	10.1kJ	29.0%	80.5%	22.7min	4.34kJ	30.6%	76.3%	96.6min	17.2kJ	15.1%
80% similarity	88.9%	37.8min	6.27kJ	42.1%	84.7%	19.0min	19.0kJ	38.2%	88.1%	74.4min	12.9kJ	27.2%
90% similarity	96.7%	27.5min	5.29kJ	47.4%	92.4%	7.1min	1.61kJ	51.4%	92.8%	56.7min	10.7kJ	28.3%

Table 5: XPerT's performance of on-device LLM personalization with different levels of similarity between pLLMs and on-device personal data. Performance is measured by model selection accuracy ('Acc'), wall-clock time ('FT-time') and energy consumption ('Energy') of on-device fine-tuning, and the improvement of data efficiency ('Data')

7 Performance Evaluation

In our evaluations, we use the following two text generation datasets with diverse language styles, for fine-tuning the base LLM into pLLM and on-device personalization of the selected pLLM:

1) Synthetic Data: We prompt ChatGPT4-o1 to generate texts with 28 different language styles in three dimensions: 1) Expertise (elementary / expert), 2) Informativeness (concise / informative), and 3) Style (friendly/ unfriendly/ sassy/ sarcastic / persuasive / neutral / poetic), by sampling instructions from the Alpaca dataset [83] and applying them to the prompt in Figure 2, which asks for answers in a specific language style. For each language style, the dataset contains 52,000 question-answer (QA) pairs, and ChatGPT4-o1's default outputs without any language style are also included.

2) Real-World Data: We combine three real-world datasets for more language styles, including 1) CDS (Corpus of Diverse Styles) [51], containing 15 million sentences in styles of poetry, lyrics, tweets and Shakespeare; 2) Gutenberg3 [12], containing extracts from 629 fiction novels in three styles, i.e., fantasy, romance, and sci-fi; and 3) ScientificPapers [20], containing two sets of long and structured documents obtained from the ArXiv and PubMed repositories as the academic style. We also mix data in two different styles to create intermediate styles, expanding the total to 25.

The personal data used for on-device LLM personalization and the training data for fine-tuning a pLLM will both contain only one language style, and hence the number of pLLMs cached at the server equals to the number of language styles in the datasets listed above. However, to mimic various levels of similarity between the on-device personal data and pLLMs, the training data for fine-tuning the pLLMs will also contain default samples without any style, and we then measure such similarity as the percentage of stylized samples in the pLLM's training data that have the same language style as the on-device personal data. All experiment results are then averaged over personal data with all language styles.

We compare XPerT with the existing methods of model selection. In particular, our scenario of model selection is related to the existing methods in AutoML [29] and hyperparameter optimization [35], which are used as the baselines and listed below.

- **Exhaustive Search** [68], which evaluates each pLLM's output with the personal data and selects the best one.
- **Bayesian Optimization** [29, 50], which treats the selection of pLLM as a new hyperparameter to be optimized by Bayesian optimization.
- **HyperBand** [28, 53] builds on the bandit principle to identify the best hyperparameter configurations and uses early stopping to eliminate bad ones.

Note that, XPerT is orthogonal to other parameter-efficient fine-tuning methods such as quantization [25] and LoRA [37], which can be applied to on-device personalization of the selected pLLM. These methods, hence, are not directly comparable with XPerT. Instead, LoRA has been applied to all our experiments of on-device LLM personalization.

7.1 Performance of On-Device LLM Personalization

We first evaluated XPerT's performance of on-device LLM personalization in the following three aspects:

- **Accuracy of model selection** with various levels of similarity between the pLLMs and the on-device personal data, and a model selection is considered as accurate if the stylistic samples in the selected pLLM's training data shared the same language style with the on-device personal data.
- **On-device LLM personalization costs**, measured by the percentage of wall-clock time reduction (FT-time) and the amount of energy consumed (Energy).
- **Data efficiency**, measured by how much fewer data samples are required to achieve the same testing loss in on-device LLM fine-tuning.

As shown in Table 5, XPerT can ensure highly accurate model selection when the similarity between pLLMs and on-device personal data reaches 70%, and can retain such accuracy at 60-65% even if the similarity drops to 50%. Based on such accurate model selection, XPerT can significantly reduce the cost of on-device LLM personalization. When the similarity is 70%, XPerT can reduce the computing time needed for on-device LLM personalization by 55%

	Llama-3.2 1B on One Plus 12R			Qwen2-0.5B on Pixel 9 Pro			SmolLM-360M on Pixel 7		
Synthetic	BLEU	ROUGE-1	ROUGE-L	BLEU	ROUGE-1	ROUGE-L	BLEU	ROUGE-1	ROUGE-L
From scratch	0.13	0.32	0.23	0.14	0.33	0.25	0.09	0.35	0.23
30% similarity	0.13	0.33	0.21	0.15	0.33	0.21	0.09	0.32	0.22
70% similarity	0.12	0.33	0.21	0.13	0.32	0.24	0.10	0.34	0.21
90% similarity	0.15	0.33	0.22	0.13	0.32	0.25	0.11	0.35	0.23
Real-world	BLEU	ROUGE-1	ROUGE-L	BLEU	ROUGE-1	ROUGE-L	BLEU	ROUGE-1	ROUGE-L
From scratch	0.18	0.39	0.30	0.07	0.31	0.27	0.06	0.29	0.26
30% similarity	0.18	0.39	0.30	0.07	0.31	0.27	0.06	0.29	0.26
70% similarity	0.19	0.39	0.28	0.07	0.31	0.25	0.06	0.28	0.27
90% similarity	0.20	0.38	0.29	0.07	0.31	0.28	0.06	0.28	0.29

Table 6: XPerT's performance in text generation task, when using personalized LLM fine-tuned from the selected pLLM and from scratch, measured by the BLEU and ROUGE scores

	Llama-3.2-1B on One Plus 12R			Qwen2-0.5B on Pixel 9 Pro			SmolLM-360M on Pixel 7		
Synthetic	t_{comm}	t_{comp}	Power	t_{comm}	t_{comp}	Power	t_{comm}	t_{comp}	Power
Exhaustive Search	1.26h	48.3min	13.1kJ	29.9min	19.6min	4.51kJ	23.8min	12.7min	1.54kJ
Bayesian Optimization	1.26h	45.5min	12.1kJ	27.8min	17.3min	4.00kJ	22.9min	12.1min	1.46kJ
HyperBand	1.26h	40.0min	10.3kJ	29.9min	16.2min	3.76kJ	23.8min	10.6min	1.29kJ
XPerT	2.7min	13.8min	3.7kJ	1.07min	10.8min	2.50kJ	0.85min	10.1min	1.22kJ
Real-world	t_{comm}	t_{comp}	Power	t_{comm}	t_{comp}	Power	t_{comm}	t_{comp}	Power
Exhaustive Search	1.12h	42.8min	11.4kJ	26.7min	16.2min	3.75kJ	21.2min	11.3min	1.54 kJ
Bayesian Optimization	1.03h	39.3min	10.7kJ	24.6min	15.6min	3.75kJ	20.4min	10.7min	1.37kJ
HyperBand	1.12h	35.9min	9.6kJ	26.7min	14.4min	3.25kJ	21.2min	10.3min	1.12kJ
XPerT	2.7min	13.8min	3.7kJ	1.07min	10.8min	2.50kJ	0.85min	10.1min	1.22kJ

Table 7: Communication cost (t_{comm}), computation cost (t_{comp}) and power consumption of on-device model selection

and the energy consumption by 57%. When the similarity reaches 90%, such reduction will further increase to 82% and 83%, respectively. On the other hand, even if the similarity is only 30%, we can still reduce such cost by up to 15%.

Meanwhile, Table 5 shows that fine-tuning the selected pLLM requires up to 51% fewer data samples for on-device LLM personalization. Such improvement of data efficiency is important for mobile devices, where accumulating large amounts of personal data is usually very time consuming.

We also evaluated whether on-device personalization using the selected pLLMs would affect the personalized LLM's performance in text generation tasks. Since the cross-entropy loss may not align well with real-world evaluation for text generation tasks [34], we instead used generation-based metrics including the BLEU [66] and ROUGE [59] scores, which assess the quality of text generation by comparing the N-gram matches between generated texts and the ground truth. As shown in Table 6, even with 30% similarity, personalization using the selected pLLMs can achieve almost identical performance in downstream tasks, compared to fine-tuning the base LLM from scratch using the same training data.

7.2 Cost of On-Device Model Selection

In XPerT, selecting the proper pLLM for on-device personalization is not for free, and instead incurs both communication and computation overhead to the user's mobile device. We investigated such model selection cost in XPerT and compared such cost with those in the existing methods of model selection. More specifically, the communication cost (t_{comm}) covers the time of downloading all

the necessary data, models and model explanations from the cloud server, and the computing cost (t_{comp}) represents the wall-clock time for all the computations of model selection on smartphone devices. For fair comparisons, we use the same network bandwidth in all baselines and do not take the concurrent network traffic into account, because pLLMs can be transmitted only when the mobile device is idle to minimize the impact of such concurrent traffic. We also measured the end-to-end power consumption of on-device model selection, including data downloading, data loading, summarization LLM's inference, calculation of sub-vectors and coordinate search for the best pLLM.

As shown in Table 7, compared to baseline methods, XPerT significantly reduces both the communication and computation costs on the user's smartphone device, by up to 96.5% and 71.4%, respectively. For communication cost, XPerT avoids downloading any of the cached pLLM itself from the server, but instead only needs to transmit the set of extracted sub-vectors shared by all pLLMs and the pLLMs' coefficients in the corresponding explainable latent space. The computing cost, on the other hand, depend on the size of pLLM. For relatively large models such as the LLaMA3.2 1B, the computing cost of Exhaustive Search is approximately 3.5x higher than that of XPerT, because it needs to evaluate all the pLLMs with the on-device personal data via inference. Similarly, Bayesian Optimization achieves negligible savings in the computing cost, because the indices of pLLMs are randomly one-hot encoded and users lack information about how these pLLMs are fine-tuned. HyperBand can reduce the computing cost by 15-20% via early
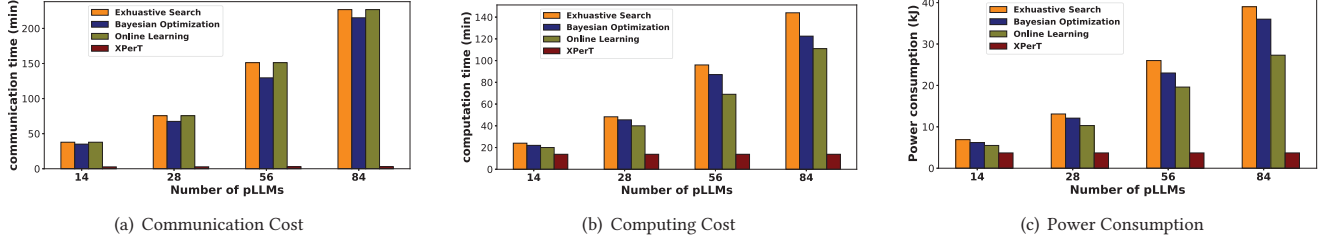
(a) Communication Cost

(b) Computing Cost

(c) Power Consumption

Figure 12: Scalability of XPerT's on-device model selection with different numbers of cached pLLMs on the server

stopping, but such saving is limited because a sufficient number of samples still should be evaluated to ensure accurate model selection.

XPerT's savings of computing costs become smaller when the model size is smaller, because the computing cost of summarization LLM inference becomes dominant. Nevertheless, such savings can still be up to 25% with the synthetic dataset. The explainability of model selection in XPerT, further allows users to use the cloud server's cached pLLMs for on-device personalization with trust.

We also evaluated how the model selection cost varies with the number of cached pLLMs at the server. Results in Figure 12 show that, while such costs in baseline methods linearly increase with the number of pLLMs, XPerT can retain a constantly low level of such cost, because all the pLLMs share a common set of sub-vectors that is used for on-device model selection. This scalability is important in many real-world scenarios that involve a large population of mobile users with much diverse language styles.
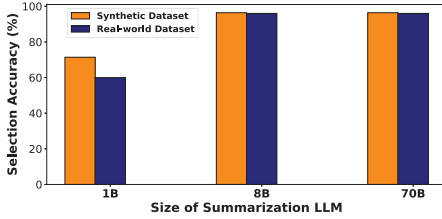

Figure 13: Accuracy of model selection with variants of Llama3.1/3.2 models as summarization LLM

7.3 Using Different Summarization LLMs

The performance of XPerT highly depends on the natural language capabilities of the summarization LLM, which generally relates to the parameter size of the summarization LLM. We evaluate XPerT's selection accuracy using different summarization LLMs, in a simulated environment on a server equipped with 4 NVidia H100 GPUs. As shown in Figure 13, when we use the Llama3.2-1B model as the summarization LLM, the accuracy of XPerT's model selection is noticeably affected, due to the model's limited capabilities of summarizing the language styles and differences in pLLMs' output. However, when the model's parameter size increases to 8B, such impact on the model selection accuracy can be effectively eliminated. Further using a larger summarization LLM, such as the Llama3.1-70B-instruct model, brings little improvement on the model selection accuracy, but would be very expensive and infeasible on smartphones.

7.4 The Impact of Empirical Thresholds

As described in Section 4.2, we used two empirical thresholds when decomposing the embedding vector of a pLLM into sub-vectors:

one threshold for deciding orthogonality between sub-vectors and another threshold for the linear combination of sub-vectors to approximate the embedding vector. We examined the performance of XPerT with different values of these two thresholds. Results in Tables 8 and 9 show that, using low values in these thresholds result in some redundant calculations of sub-vectors and impair the computing efficiency. In contrast, higher values of these thresholds may miss important sub-vectors and affect the accuracy of model selection. Therefore, in our experiments, we set the orthogonality threshold to 0.1 and the decomposition threshold to $0.2 \cdot |\vec{V}|$, where $\vec{V}$ is the embedding vector.

Orthogonality threshold	0.01	0.1	0.5
Number of candidate words examined	∞	372	185
Model selection accuracy (%)	-	96.4	60.7

Table 8: Different values of the threshold for sub-vector orthogonality

Decomp. threshold as a fraction of $|\vec{V}|$	0.02	0.2	0.6
Number of candidate words examined	∞	372	78
Model selection accuracy (%)	-	96.4	14.3

Table 9: Different values of the threshold for decomposing the embedding vector into sub-vectors

7.5 Validating the Explainable Latent Space

As stated in Section 3, XPerT builds on the assumption that closer distances in the explainable latent space correspond to more similar language styles. We verify this assumption with a synthetic dataset that ensembles language styles at different levels. For instance, for a language style of "expertise", the dataset includes samples of four levels: elementary school students, middle school students, undergraduates, and PhD students. We then test the assumption by checking if the coordinates of two consecutive levels are closer.

Style	Level 1	Level 2	Level 3	Level 4
Elementary	Elementary school students	Middle school students	Undergraduates	PhD students in the field
Formality	Slang, casual expressions	Everyday language, for friendly chat	Professional but with a more conversational tone	Professional language, used in corporate settings
Persuasiveness	Strictly factual with no personal opinion or bias	Presenting facts with slight persuasion or bias	Appeals to logic and emotions, but supported by facts	Convince the reader, with emotional appeal

Figure 14: Language styles with multiple levels

We prompt ChatGPT4-o1 to synthesize this dataset with language styles shown in Figure 14, and then use data samples of different levels of a language style to train different pLLMs. The coordinates of these pLLMs in the latent space are then extracted
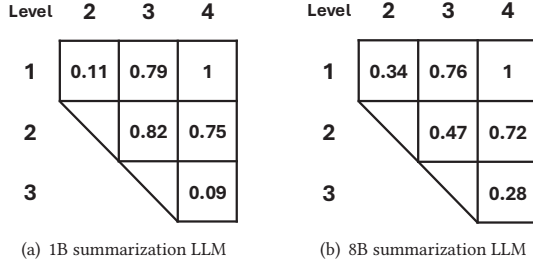
Level	2	3	4
1	0.11	0.79	1
2		0.82	0.75
3			0.09

(a) 1B summarization LLM

Level	2	3	4
1	0.34	0.76	1
2		0.47	0.72
3			0.28

(b) 8B summarization LLM

Figure 15: Distance between pLLMs coordinates fine-tuned with data at different levels of a language style

from two summarization LLMs (Llama3.2-1B and Llama3.1-8B), and their distances are measured as the L1-norm. Results in Figure 15 show that the consistency between language styles and coordinates in the latent space relates to the capability of summarization LLM. With Llama3.2-1B as the summarization LLM, we can observe noticeable inconsistency, which however, can be effectively eliminated by using the Llama3.1-8B as the summarization LLM.

7.6 On-Device Model Merging

It is possible that none of the cached pLLMs can well match the user's personal data, and Section 5.2 suggested to create a new pLLM with better matching by merging multiple pLLMs. To investigate the effectiveness of such on-device model merging, we consider the user's personal data as combinations of different language styles, so that none of the cached pLLM that represents one language style can well match the personal data. Results in Figure 16 show that, when using Llama3.2-1B as the pLLM, as long as the user's personal data contains more than one language style, using the merged pLLM always gains extra reduction on the cost of on-device LLM personalization, by up to 25%.
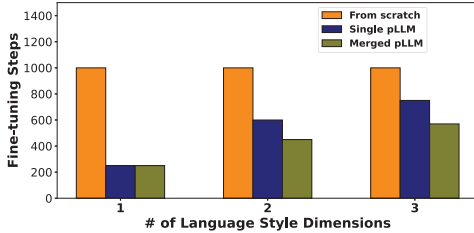

Figure 16: Effectiveness of on-device model merging

7.7 XPerT on Other Data Modalities

To demonstrate the generalizability of XPerT, we further apply XPerT to the modality of image data. More specifically, we target the image generation task using text-to-image diffusion models, and fine-tune the U-Net structures of the Stable Diffusion v2.1 model [73] using the Adam optimizer with a learning rate of 1e-4. In such fine-tuning, we use multiple real-world image datasets, including WikiArt [9], DiffusionDB [93], Midjourney-Detailed [5] and TinySketch [4][4], to incorporate 9 distinct artistic image styles as exemplified in Figure 17, and randomly select 500 images in each style for fine-tuning the diffusion model.

---

[4]For datasets without textual descriptions of images such as WikiArt [9], we use a pre-trained BLIP2 image captioning model [52] to generate the textual descriptions for image samples.
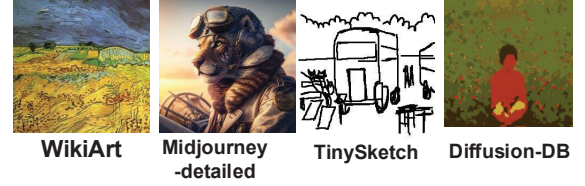

WikiArt   Midjourney-detailed   TinySketch   Diffusion-DB

Figure 17: Examples of images in different datasets with diverse artistic styles

In such experiments on fine-tuning the diffusion model, for XPerT, the summarization LLM must not only be capable of converting differences in images into text, but also generating synthetic image data to compute sub-vectors as described in Section 4.2. However, most open-source models that meet such multimodal requirements have only limited representation power. As a result, we use two separate models: the Qwen2-VL-7B-Instruct model [89] for summarizing the differences in images, and the Stable Diffusion v3.5 model [27] for generating synthetic image data.

Experiment results are shown in Figure 18. XPerT achieves over 77.8% accuracy of pLLM selection, when the similarity between the device's local data and the training data used in generating pLLMs exceeds 70%. Besides, we also implemented the baseline methods by using the FID score [36] to measure the distance between the pLLM's training data and the user's local data. Similar to results in the text modality, the selection cost of baseline methods is proportional to the number of pLLMs and is hence much higher than that of XPerT, which is independent from the number of pLLMs.
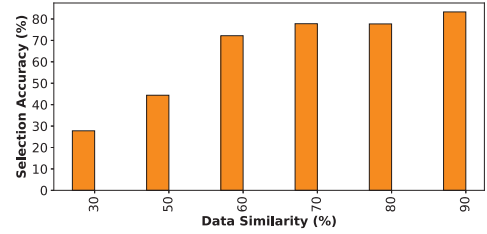

Figure 18: Accuracy of pLLM selection on the image generation task

7.8 Comparing XPerT with In-Context Learning

Besides fine-tuning, another option to improve the model's performance for downstream tasks is in-context learning (ICL), which allows the model to learn tasks from the few in-context examples given through the prompt [13, 82, 95, 97]. However, compared to fine-tuning, ICL usually exhibits limited capability in complex or domain-specific tasks, due to the absence of model training efforts that are essential to teach new concepts and knowledge to the model [86]. For example, ICL has been found to perform significantly worse than fine-tuning on instruction-tuning datasets, especially on small LLMs with less than 1B parameters [69].

To further verify this, we compared XPerT with ICL by using the Qwen2.5-1.5B-Instruct model [84] and 4 domain-specific datasets. These datasets include: 1) *MedDialog* [17], which contains 0.26 million dialogues between doctors and patients from healthcaremagic.com and icliniq.com; 2) *DISC-Law-SFT* [103], a QA dataset covers different legal scenarios; 3) *Educhat-sft* [1], which contains 4M QA samples in education domain; and 4) *FinanceAlpaca* [2], which contains 70k QA samples collected from multiple

datasets in the finance domain. For XPerT, we use each dataset to fine-tune the model using the same settings as above. For ICL, we assume that users are not experts and lack the professional knowledge to select and curate ICL examples, as described in [58]. Instead, they merely adopt vanilla ICL with randomly chosen examples. We set the number of in-context examples to 3, and run the experiment 5 times for each dataset to obtain the averaged results.

Experiment results in Table 10 show that the performance of ICL is significantly lower than that of XPerT in different data domains. Specifically, in tasks requiring domain-specific knowledge such as the medical domain, ICL learns very little from in-context samples, resulting in substantially lower performance. In contrast, for the education domain which does not require extensive domain-specific knowledge, ICL can achieve a non-trivial improvement by learning the format from in-context examples. However, its performance is still lower than that of XPerT.

Metric	BLEU		ROUGE-1	
	ICL	XPerT	ICL	XPerT
MedDialog [17]	0.042	0.128	0.069	0.152
DISC-Law-SFT [103]	0.156	0.239	0.078	0.308
Educhat-sft [1]	0.229	0.255	0.207	0.332
FinanceAlpaca [2]	0.137	0.278	0.114	0.289

Table 10: Performance of XPerT and ICL in different domains

Additionally, ICL's performance in instruction-following tasks is highly dependent on the specific samples used in the prompt [107], making it unreliable in practice, particularly for non-expert users. Beyond its limited capability, ICL requires the model to process additional tokens and incurs extra computational cost in inference.

8 Related Work

Linear representation in LLM. Well-trained LLMs are believed to encode linear representations of human-interpretable concepts, such as gender [26, 62, 64, 92]. If the direction of a concept in the embedding space is known, the semantics of the concept representation can be adjusted using simple linear operations. This linearity was also observed in other domains including computer vision models [85, 91] and intelligent systems [77]. Hence, in XPerT we build on this observation and achieve explainability of the embedding vector by linearly decomposing it using concept representations.

Extraction of text features. Our main approach in XPerT is the leverage the natural language capability of LLM to encode the difference between pLLMs' outputs into the embedding vector. On the other hand, to convert LLMs' output texts into such a feature vector, the simplest way is to use a pre-trained text encoder, such as CLIP [71]. However, most existing text encoders are trained to extract the text's semantic meaning [49] rather than high-level concepts such as the language style. An alternative approach is to train a language style classification model that can directly extract language style features, but the existing pre-trained models are limited to a small number of language styles [47], such as formality and toxicity. Additionally, the feature vectors extracted by these methods lack sufficient explainability.

Model selection. Our design of XPerT is related to the existing work in model selection, which selects the model from a set of candidates that best satisfies the task requirement or user context

[30, 31, 48, 55]. The candidates could be models with different architectures or the same model with different hyperparameters. In our scenario, the candidate models are the same LLM with different initial weights, which can also be seen as a kind of hyperparameter. Our design of XPerT, however, is fundamentally different because we extract high-level information from each candidate model for selection, instead of directly evaluating the candidate's performance.

9 Discussions

Other ways of personalization. Besides different language styles, an LLM can also be personalized in many other aspects. For example, a code generation LLM can be personalized for different programming languages [15], and a QA LLM can be customized for specific domains such as history or physics [11]. XPerT can also be applied to these aspects, by tailoring the prompt in Figure 6 accordingly. Unlike stylized text generation, tasks of code generation and domain-specific QA require more explicit methods of reflecting the personalized aspects in the LLM outputs, and it is better the prompt the summarization LLM with more task-specific instructions.

Untrusted cloud sources. Users may hesitate to use a pLLM downloaded from the cloud for on-device personalization, because such as download pLLM from untrusted cloud sources may be backdoored by malicious model publishers [54]. To mitigate such risks, the users should be advised to only use pLLMs from trusted cloud sources. Furthermore, fine-tuning a backdoored LLM with benign data (i.e., the user's personal data) has been verified to effectively mitigate the backdoor attack [61], and lightweight backdoor detection methods are also available for use on mobile devices [70].

Generalizability of personalized LLM. We evaluate the model's performance only on datasets of specific downstream tasks, rather than on standard benchmarks for general assessment. This is because, by definition, fine-tuning improves the model's performance on a specific downstream task. Hence, generalizability is naturally reduced in a fine-tuned model compared to the pre-trained model, and this loss of generalizability is orthogonal to XPerT's design.

Connectivity to the server. In XPerT, the user's device needs to download the selected pLLM from the server, and we assume that mobile devices have an almost always-on connection to the server via wireless links. This assumption is the cornerstone of many smartphone services and apps, including cloud storage, AI assistants, and streaming platforms. Additionally, XPerT only communicates with the server once to download the pLLM; afterwards, local fine-tuning can proceed without any further communications.

10 Conclusion

In this paper, we present XPerT, a new technique that expedites on-device LLM personalization by fine-tuning the proper pLLM cached at the cloud server with on-device personal data. XPerT provide detailed and quantitative explainability of pLLMs, so as to ensure proper model selection.

Acknowledgments

References

[1] [n. d.]. Educhat-SFT. https://huggingface.co/datasets/ecnu-icalk/educhat-sft-002-data-osm.

[2] [n. d.]. Finance-alpaca. https://huggingface.co/datasets/gbharti/finance-alpaca.

[3] [n. d.]. Fine-tuning open-weight LLMs with Anyscale. https://www.anyscale.com/use-case/llm-fine-tuning

[4] [n. d.]. FS-COCO: Towards Understanding of Freehand Sketches of Common Objects in Context.

[5] [n. d.]. Midjourney: Detailed Prompts. https://huggingface.co/datasets/MohamedRashad/midjourney-detailed-prompts.

[6] [n. d.]. NousResearch on Llama3.2-1B. https://huggingface.co/NousResearch/Llama-3.2-1B.

[7] [n. d.]. Open Neural Network Exchange. https://github.com/onnx/onnx.

[8] [n. d.]. TensorFlow Lite. https://www.tensorflow.org/lite.

[9] [n. d.]. Wikiart. https://huggingface.co/datasets/huggan/wikiart.

[10] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. 2024. SmolLM - blazingly fast and remarkably powerful.

[11] Avinash Anand, Janak Kapuriya, Chhavi Kirtani, Apoorv Singh, Jay Saraf, Naman Lal, Jatin Kumar, Adarsh Raj Shivam, Astha Verma, Rajiv Ratn Shah, et al. 2024. MM-PhyRLHF: Reinforcement Learning Framework for Multimodal Physics Question-Answering. *arXiv preprint arXiv:2404.12926* (2024).

[12] billray110. 2018. gutenberg3-generalfiction-scifi-fantasy-romance-adventure-dpo. https://huggingface.co/datasets/sam-paech/gutenberg3-generalfiction-scifi-fantasy-romance-adventure-dpo.

[13] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[14] Fenglong Cai, Dong Yuan, Zhe Yang, and Lizhen Cui. 2024. Edge-llm: A collaborative framework for large language model serving in edge computing. In *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 799–809.

[15] Doga Cambaz and Xiaoling Zhang. 2024. Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 172–178.

[16] Vítor Cerqueira, Luís Torgo, Fábio Pinto, and Carlos Soares. 2017. Arbitrated ensemble for time series forecasting. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part II 10*. Springer, 478–494.

[17] Shu Chen, Zeqian Ju, Xiangyu Dong, Hongchao Fang, Sicheng Wang, Yue Yang, Jiaqi Zeng, Ruisi Zhang, Ruoyu Zhang, Meng Zhou, Penghui Zhu, and Pengtao Xie. 2020. MedDialog: a large-scale medical dialogue dataset. *arXiv preprint arXiv:2004.03329* (2020).

[18] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. TVM: An automated End-to-End optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[19] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research* 25, 70 (2024), 1–53.

[20] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)* (2018). https://doi.org/10.18653/v1/n18-2097

[21] Ashok Cutkosky and Kwabena Boahen. 2017. Online learning without prior information. In *Conference on learning theory*. PMLR, 643–677.

[22] Yucheng Ding, Chaoyue Niu, Fan Wu, Shaojie Tang, Chengfei Lyu, and Guihai Chen. 2024. Enhancing on-device llm inference with historical cloud-based llm interactions. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 597–608.

[23] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A Efros. 2015. What makes paris look like paris? *Commun. ACM* 58, 12 (2015), 103–110.

[24] 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* (2024).

[25] Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. 2024. Exploiting LLM Quantization. *arXiv preprint arXiv:2405.18137* (2024).

[26] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. 2022. Toy models of superposition. *arXiv preprint arXiv:2209.10652* (2022).

[27] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. 2024. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*.

[28] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*. PMLR, 1437–1446.

[29] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research* 23, 261 (2022), 1–61.

[30] Dylan J Foster, Satyen Kale, Mehryar Mohri, and Karthik Sridharan. 2017. Parameter-free online learning via model selection. *Advances in Neural Information Processing Systems* 30 (2017).

[31] Dylan J Foster, Akshay Krishnamurthy, and Haipeng Luo. 2019. Model selection for contextual bandits. *Advances in Neural Information Processing Systems* 32 (2019).

[32] Takashi Fukuda, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. 2017. Efficient Knowledge Distillation from an Ensemble of Teachers.. In *Interspeech*. 3697–3701.

[33] Neel Guha, Julian Nyarko, Daniel E. Ho, Christopher Ré, Adam Chilton, Aditya Narayana, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel N. Rockmore, Diego Zambrano, Dmitry Talisman, Enam Hoque, Faiz Surani, Frank Fagan, Galit Sarfaty, Gregory M. Dickinson, Haggai Porat, Jason Hegland, Jessica Wu, Joe Nudell, Joel Niklaus, John Nay, Jonathan H. Choi, Kevin Tobia, Margaret Hagan, Megan Ma, Michael Livermore, Nikon Rasumov-Rahe, Nils Holzenberger, Noam Kolt, Peter Henderson, Sean Rehaag, Sharad Goel, Shang Gao, Spencer Williams, Sunny Gandhi, Tom Zur, Varun Iyer, and Zehua Li. 2023. LegalBench: A Collaboratively Built Benchmark for Measuring Legal Reasoning in Large Language Models. arXiv:2308.11462 [cs.CL]

[34] Zishan Guo, Renren Jin, Chuang Liu, Yufei Huang, Dan Shi, Linhao Yu, Yan Liu, Jiaxuan Li, Bojian Xiong, Deyi Xiong, et al. 2023. Evaluating large language models: A comprehensive survey. *arXiv preprint arXiv:2310.19736* (2023).

[35] Maxime Heuillet, Benoit Debaque, and Audrey Durand. 2021. Sequential Automated Machine Learning: Bandits-driven Exploration using a Collaborative Filtering Representation. In *8th ICML Workshop on Automated Machine Learning (AutoML)*.

[36] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).

[37] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[38] Kai Huang and Wei Gao. 2022. Real-time neural network inference on extremely weak devices: agile offloading with explainable ai. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 200–213.

[39] Kai Huang, Boyuan Yang, and Wei Gao. 2023. Elastictrainer: Speeding up on-device training with runtime elastic tensor selection. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 56–69.

[40] Kai Huang, Hanyun Yin, Heng Huang, and Wei Gao. 2024. Towards Green AI in Fine-tuning Large Language Models via Adaptive Backpropagation. *ICLR* (2024).

[41] Kai Huang, Xiangyu Yin, Tao Gu, and Wei Gao. 2024. Perceptual-Centric Image Super-Resolution using Heterogeneous Processors on Mobile Devices. In *ACM MobiCom*.

[42] Kai Huang, Xiangyu Yin, Heng Huang, and Wei Gao. 2025. Modality Plug-and-Play: Runtime Modality Adaptation in LLM-Driven Autonomous Mobile Systems. In *ACM MobiCom*.

[43] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089* (2022).

[44] Matthias Jakobs and Amal Saadallah. 2023. Explainable Adaptive Tree-based Model Selection for Time-Series Forecasting. In *2023 IEEE International Conference on Data Mining (ICDM)*. IEEE, 180–189.

[45] Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. 2023. Personalized soups: Personalized large language model alignment via post-hoc parameter merging. *arXiv preprint arXiv:2310.11564* (2023).

[46] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lv, and Zhihua Wu. 2020. MNN: A Universal and Efficient Inference Engine. In *Proceedings of MLSys*.

[47] Di Jin, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea. 2022. Deep learning for text style transfer: A survey. *Computational Linguistics* 48, 1 (2022), 155–205.

[48] Mohammad Reza Karimi, Nezihe Merve Gürel, Bojan Karlaš, Johannes Rausch, Ce Zhang, and Andreas Krause. 2021. Online active model selection for pre-trained classifiers. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 307–315.

[49] Hyunjae Kim, Seunghyun Yoon, Trung Bui, Handong Zhao, Quan Tran, Franck Dernoncourt, and Jaewoo Kang. 2024. Fine-tuning CLIP Text Encoders with

Two-step Paraphrasing. *arXiv preprint arXiv:2402.15120* (2024).

[50] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research* 18, 25 (2017), 1–5.

[51] Kalpesh Krishna, John Wieting, and Mohit Iyyer. 2020. Reformulating Unsupervised Style Transfer as Paraphrase Generation. In *Empirical Methods in Natural Language Processing*.

[52] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*. PMLR, 19730–19742.

[53] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52.

[54] Yige Li, Hanxun Huang, Yunhan Zhao, Xingjun Ma, and Jun Sun. 2024. Backdoorllm: A comprehensive benchmark for backdoor attacks on large language models. *arXiv preprint arXiv:2408.12798* (2024).

[55] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Efficient automatic CASH via rising bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4763–4771.

[56] Yao Li, Lingqiao Liu, Chunhua Shen, and Anton van den Hengel. 2015. Mid-level deep pattern mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 971–980.

[57] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459* (2024).

[58] Bill Yuchen Lin, Abhilasha Ravichander, Ximing Lu, Nouha Dziri, Melanie Sclar, Khyathi Chandu, Chandra Bhagavatula, and Yejin Choi. 2023. The unlocking spell on base llms: Rethinking alignment via in-context learning. *arXiv preprint arXiv:2312.01552* (2023).

[59] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[60] Deyuan Liu, Zecheng Wang, Bingning Wang, Weipeng Chen, Chunshan Li, Zhiying Tu, Dianhui Chu, Bo Li, and Dianbo Sui. 2024. Checkpoint Merging via Bayesian Optimization in LLM Pretraining. *arXiv preprint arXiv:2403.19390* (2024).

[61] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 45–48.

[62] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*. 746–751.

[63] John X Morris, Wenting Zhao, Justin T Chiu, Vitaly Shmatikov, and Alexander M Rush. 2023. Language model inversion. *arXiv preprint arXiv:2311.13647* (2023).

[64] Neel Nanda, Andrew Lee, and Martin Wattenberg. 2023. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941* (2023).

[65] Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. 2024. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems* 36 (2024).

[66] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. 311–318.

[67] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*. 1–22.

[68] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813* (2023).

[69] David Ponce and Thierry Etchegoyhen. 2025. In-context Learning vs. Instruction Tuning: The Case of Small and Multilingual Language Models. *arXiv preprint arXiv:2503.01611* (2025).

[70] Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2020. Onion: A simple and effective defense against textual backdoor attacks. *arXiv preprint arXiv:2011.10369* (2020).

[71] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[72] Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. 2023. Progressive prompts: Continual learning for language models. *arXiv preprint arXiv:2301.12314* (2023).

[73] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10684–10695.

[74] Amal Saadallah, Matthias Jakobs, and Katharina Morik. 2021. Explainable online deep neural network selection using adaptive saliency maps for time series forecasting. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 404–420.

[75] Amal Saadallah, Matthias Jakobs, and Katharina Morik. 2022. Explainable online ensemble of deep neural network pruning for time series forecasting. *Machine Learning* 111, 9 (2022), 3459–3487.

[76] Julian Salazar, Davis Liang, Toan Q Nguyen, and Katrin Kirchhoff. 2019. Masked language model scoring. *arXiv preprint arXiv:1910.14659* (2019).

[77] Lisa Schut, Nenad Tomasev, Tom McGrath, Demis Hassabis, Ulrich Paquet, and Been Kim. 2023. Bridging the human-ai knowledge gap: Concept discovery and transfer in alphazero. *arXiv preprint arXiv:2310.16410* (2023).

[78] Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. 2024. Continual learning of large language models: A comprehensive survey. *arXiv preprint arXiv:2404.16789* (2024).

[79] Chandan Singh, Aliyah R Hsu, Richard Antonello, Shailee Jain, Alexander G Huth, Bin Yu, and Jianfeng Gao. 2023. Explaining black box text modules in natural language with language models. *arXiv preprint arXiv:2305.09863* (2023).

[80] Adrians Skapars, Edoardo Manino, Youcheng Sun, and Lucas C Cordeiro. 2024. Was it Slander? Towards Exact Inversion of Generative Language Models. *arXiv preprint arXiv:2407.11059* (2024).

[81] A Smith. 2019. How Americans Use Text Messaging, Pew Research Center: Internet, Science & Tech.

[82] Jifeng Song, Kai Huang, Xiangyu Yin, Boyuan Yang, and Wei Gao. 2024. Achieving Sparse Activation in Small Language Models. *arXiv preprint arXiv:2406.06562* (2024).

[83] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[84] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/.

[85] Matthew Trager, Pramuditha Perera, Luca Zancato, Alessandro Achille, Parminder Bhatia, and Stefano Soatto. 2023. Linear spaces of meanings: compositional structures in vision-language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15395–15404.

[86] Johannes Treutlein, Dami Choi, Jan Betley, Samuel Marks, Cem Anil, Roger B Grosse, and Owain Evans. 2024. Connecting the dots: Llms can infer and verbalize latent structure from disparate training data. *Advances in Neural Information Processing Systems* 37 (2024), 140667–140730.

[87] Yu-Min Tseng, Yu-Chao Huang, Teng-Yun Hsiao, Yu-Ching Hsu, Jia-Yin Foo, Chao-Wei Huang, and Yun-Nung Chen. 2024. Two tales of persona in llms: A survey of role-playing and personalization. *arXiv preprint arXiv:2406.01171* (2024).

[88] Nanne Van Noord. 2023. Prototype-based dataset comparison. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1944–1954.

[89] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024. Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. *arXiv preprint arXiv:2409.12191* (2024).

[90] Xiao Wang, Yuansen Zhang, Tianze Chen, Songyang Gao, Senjie Jin, Xianjun Yang, Zhiheng Xi, Rui Zheng, Yicheng Zou, Tao Gui, et al. 2023. TRACE: A Comprehensive Benchmark for Continual Learning in Large Language Models. *arXiv preprint arXiv:2310.06762* (2023).

[91] Zihao Wang, Lin Gui, Jeffrey Negrea, and Victor Veitch. 2023. Concept algebra for score-based conditional model. In *ICML 2023 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*.

[92] Zihao Wang, Lin Gui, Jeffrey Negrea, and Victor Veitch. 2024. Concept algebra for (score-based) text-controlled generative models. *Advances in Neural Information Processing Systems* 36 (2024).

[93] Zijie J. Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. 2022. DiffusionDB: A Large-Scale Prompt Gallery Dataset for Text-to-Image Generative Models. *arXiv:2210.14896 [cs]* (2022). https://arxiv.org/abs/2210.14896

[94] Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209* (2017).

[95] Zhiyong Wu, Yaoxiang Wang, Jiacheng Ye, and Lingpeng Kong. 2022. Self-adaptive in-context learning: An information compression perspective for in-context example selection and ordering. *arXiv preprint arXiv:2212.10375* (2022).

[96] Junlin Xie, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. 2024. Large multimodal agents: A survey. *arXiv preprint arXiv:2402.15116* (2024).

[97] Qiyao Xue, Xiangyu Yin, Boyuan Yang, and Wei Gao. 2025. Phyt2v: Llm-guided iterative self-refinement for physics-grounded text-to-video generation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2025).

[98] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2

technical report. *arXiv preprint arXiv:2407.10671* (2024).

[99] Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024. Model Merging in LLMs, MLLMs, and Beyond: Methods, Theories, Applications and Opportunities. *arXiv preprint arXiv:2408.07666* (2024).

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

[101] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* (2024), 100211.

[102] Xiangyu Yin, Kai Huang, Erick Forno, Wei Chen, Heng Huang, and Wei Gao. 2023. PTEase: Objective Airway Examination for Pulmonary Telemedicine using Commodity Smartphones. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 110–123.

[103] Shengbin Yue, Wei Chen, Siyuan Wang, Bingxuan Li, Chenchen Shen, Shujun Liu, Yuxuan Zhou, Yao Xiao, Song Yun, Xuanjing Huang, and Zhongyu Wei. 2023. DISC-LawLLM: Fine-tuning Large Language Models for Intelligent Legal Services. arXiv:2309.11325 [cs.CL]

[104] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. 2024. When scaling meets llm finetuning: The effect of data, model and finetuning method. *arXiv preprint arXiv:2402.17193* (2024).

[105] Frederic Z Zhang, Paul Albert, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. 2024. Knowledge composition using task vectors with learned anisotropic scaling. *arXiv preprint arXiv:2407.02880* (2024).

[106] Zhehao Zhang, Ryan A Rossi, Branislav Kveton, Yijia Shao, Diyi Yang, Hamed Zamani, Franck Dernoncourt, Joe Barrow, Tong Yu, Sungchul Kim, et al. 2024. Personalization of Large Language Models: A Survey. *arXiv preprint arXiv:2411.00027* (2024).

[107] Hao Zhao, Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. 2024. Is in-context learning sufficient for instruction following in LLMs? *arXiv preprint arXiv:2405.19874* (2024).

[108] Ruiqi Zhong, Charlie Snell, Dan Klein, and Jacob Steinhardt. 2022. Describing differences between text distributions with natural language. In *International Conference on Machine Learning*. PMLR, 27099–27116.

[109] Ruiqi Zhong, Peter Zhang, Steve Li, Jinwoo Ahn, Dan Klein, and Jacob Steinhardt. 2023. Goal driven discovery of distributional differences via language descriptions. *Advances in Neural Information Processing Systems* 36 (2023), 40204–40237.