Enhancing On-Device LLM Inference with Historical Cloud-Based LLM Interactions

Yucheng Ding
Shanghai Jiao Tong University
Shanghai, China
yc.ding@sjtu.edu.cn

Chaoyue Niu*
Shanghai Jiao Tong University
Shanghai, China
rvince@sjtu.edu.cn

Fan Wu
Shanghai Jiao Tong University
Shanghai, China
wu-fan@sjtu.edu.cn

Shaojie Tang
University of Texas at Dallas
Richardson, Texas, United States
shaojie.tang@utdallas.edu

Chengfei Lyu
Alibaba Group
Hangzhou, Zhejiang, China
chengfei.lcf@alibaba-inc.com

Guihai Chen
Shanghai Jiao Tong University
Shanghai, China
gchen@cs.sjtu.edu.cn

Abstract

Many billion-scale large language models (LLMs) have been released for resource-constraint mobile devices to provide local LLM inference service when cloud-based powerful LLMs are not available. However, the capabilities of current on-device LLMs still lag behind those of cloud-based LLMs, and how to effectively and efficiently enhance on-device LLM inference becomes a practical requirement. We thus propose to collect the user's historical interactions with the cloud-based LLM and build an external datastore on the mobile device for enhancement using nearest neighbors search. Nevertheless, the full datastore improves the quality of token generation at the unacceptable expense of much slower generation speed. To balance performance and efficiency, we propose to select an optimal subset of the full datastore within the given size limit, the optimization objective of which is proven to be submodular. We further design an offline algorithm, which selects the subset after the construction of the full datastore, as well as an online algorithm, which performs selection over the stream and can be flexibly scheduled. We theoretically analyze the performance guarantee and the time complexity of the offline and the online designs to demonstrate effectiveness and scalability. We finally take three ChatGPT related dialogue datasets and four different on-device LLMs for evaluation. Evaluation results show that the proposed designs significantly enhance LLM performance in terms of perplexity while maintaining fast token generation speed. Practical overhead testing on the smartphone reveal the efficiency of on-device datastore subset selection from memory usage and computation overhead.

CCS Concepts

• **Information systems** → **Data mining**; • **Computing methodologies** → **Natural language processing**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

*Chaoyue Niu is the corresponding author.

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671679

Keywords

On-Device LLM Enhancement; Device-Cloud Hybrid Service; Datastore Subset Selection

ACM Reference Format:

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 (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671679

1 Introduction

Large language models (LLMs) have revealed remarkable text generation capability and have revolutionized the conversational user interfaces for intelligent services. On the side of cloud, the parameter scale of the LLMs normally reaches hundreds of billions, and massive users can interact with the cloud-based LLMs through API requests (e.g., ChatGPT, Bard, Claude, Qwen). On the side of mobile device, tech giants, smartphone manufacturers, and chipmakers have launched billion-scale LLMs (e.g., Qwen-1.8B, Phi-2.7B, ChatGLM3-6B, Baichuan2-7B, Gemini-Nano) to continuously provide local service when the cloud API is not available (e.g., when a user's mobile device has poor network connection, the rate limit of cloud API has been reached, or the cloud platform is overloaded with excessive requests or experiences an outage). Such a device-cloud hybrid service framework guarantees the completeness of LLM service. However, on-device LLMs exhibit markedly lower performance than cloud-based LLMs, and how to effectively and efficiently enhance local LLM inference on each resource-constraint mobile device becomes a new requirement in practice.

For on-device LLM enhancement, a basic way is to acquire high-quality data. We innovatively propose to exploit a user's historical interactions with the cloud API, because the user's queries to the cloud-based LLM tend to be similar to his/her queries to the on-device LLM, and meanwhile, the responses from the cloud-based powerful LLM can be regarded as reference answers. Another problem is how to enhance on-device LLM with these historical dialogues. Existing LLM enhancement work focused on resource-rich cloud rather than resource-constraint mobile device. One line of work resorted to tuning methods, including full model finetuning and parameter-efficient tuning (e.g., prefix tuning [35], adapter [29], LoRA [30]). However, tuning a billion-scale LLM for each user cannot be centralized on the cloud due to the user base of billions,

and also cannot be distributed on each mobile device because of the limited memory and computation resources. The other line of non-parametric work introduced an external datastore to assist LLM inference. Two celebrated methods are k-nearest neighbour language modeling (kNN-LM) [32] and retrieval-augmented generation (RAG) [34]. In particular, kNN-LM combines the pure LLM prediction result and the kNN search results over the datastore in the LLM embedding space, whereas RAG concatenates the retrieval results over the datastore and the user query as the new input of LLM. Considering the on-device scenario, kNN-LM is more suitable in terms of memory efficiency and user input length.

However, the fundamental design of on-device LLM enhancement with kNN-LM needs to limit the size of the external datastore, otherwise the token generation speed cannot meet the real-time requirement (e.g., 10 tokens per second). In particular, the full datastore constructed from an active user's all the dialogues with the cloud-based LLM within a long work cycle (e.g., 1 month) significantly exceeds the size limit. In addition, if directly take the dialogues in a short cycle (e.g., 1 day or 1 week), the size and the quality of the datastore are not sufficient for on-device LLM enhancement.

To balance the quality and the speed of on-device token generation, we propose to select an optimal subset, the size of which is within the limit, from the full datastore for on-device kNN-LM enhancement. Nevertheless, such a subset selection problem is NP-hard. We thus turn to a near-optimal solution that can be found efficiently on the mobile device. We first prove that the optimization objective is a monotone and submodular function. A naive algorithm, which greedily selects an element from the full datastore with the highest marginal gain each time, can provide theoretical performance guarantee, but has unaffordable time complexity. To improve efficiency, we propose an offline design and an online design[1], which execute selection after building the full datastore and over the online stream, respectively. In particular, the offline design overcomes the bottlenecks of the greedy algorithm by (1) estimating an element's marginal gain by its lower bound, which depends only on itself, rather than calculating the exact marginal gain related to the full datastore; and (2) selecting a batch of elements with the highest marginal gains from a randomly sampled subset, rather than one element from the full datastore each time. In addition, the offline design is quite salable with performance guarantee, and its time complexity is independent of the full datastore size. We also interpret the offline design from the perspective of pool-based active learning with training loss as the selection metric. Regarding the online design, it maintains a small buffer, selects elements incrementally once the buffer is full, and can be flexibly scheduled throughout a work cycle. The key difference from the offline setting is that the full length of the stream is unknown when doing selection. We thus propose to maintain several candidate datastore subsets with different selection ratios and return the best subset at the end of the work cycle.

We summarize the key contributions of this work as follows:

- We focus on the emerging device-cloud hybrid LLM service framework and consider the new problem of effectively and efficiently enhancing on-device LLM inference.

[1]The offline/online design in this paper refers to pool-based/stream-based algorithm, and does not mean the mobile devices being disconnected/connected to the Internet.

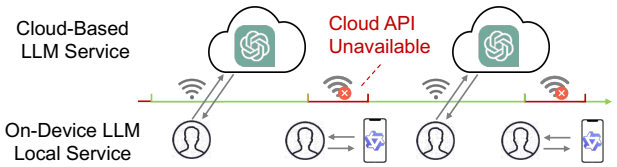


Figure 1: Device-cloud hybrid LLM service.

- We, for the first time, propose to take each user's historical high-quality interactions with the cloud-based LLM as an external datastore to enhance on-device LLM using kNN-LM.
- To balance on-device token generation quality and speed, we consider how to select an optimal subset of the full datastore. We propose both offline and online designs with near-optimal guarantees and low time complexities.
- We extensively evaluate the proposed designs using three ChatGPT related datasets, four different on-device LLMs, and the testbed of Honor V30 Pro. Key evaluation results include: (1) compared with pure on-device LLM, enhancement with an external datastore subset reduces perplexity by 16.07% on average; (2) compared with on-device LLM with the full datastore, the subset selection boosts the generation speed up to 2.3 × and achieves 10 tokens per second; and (3) the memory usage of the on-device datastore subset selection is within 1.37GB.

2 Background and Problem

To support the deployment of LLMs in various application scenarios with different computational resources, many companies have launched a series of LLMs with different parameter sizes, such as the open-source GPT-1, GPT-2, and the API-only GPT-3.5 or ChatGPT from OpenAI [19]; the API-only Gemini-Nano, Gemini-Pro, and Gemini-Ultra from Google [4]; the open-source 1.8B, 7B, 14B, and 72B versions of Qwen and the API-only Qwen-Max from Alibaba [15]; the open-source 7B, 13B, and 70B versions of Llama 2 from Meta [17]; the open-source 7B and 13B versions of Alpaca from Stanford [1]; the open-source 7B, 13B, and 33B versions of Vicuna from UC Berkeley [13]; and the open-source 7B and 13B versions of Baichuan2 and the API-only Baichuan3 [16].

Among the series of LLMs, the largest model, the size of which normally reaches hundreds of billions, is deployed on the resource-rich cloud to serve massive users through online API requests (e.g., OpenAI's ChatGPT, Google's Bard, Anthropic's Claude, Alibaba's Qwen, Baidu's ERNIE Bot), whereas the light-weight model, the size of which is at the magnitude of billions, supports the deployment on the resource-constraint mobile devices for local real-time service (e.g., TinyLlama-1.1B, Qwen-1.8B, ChatGLM3-6B, Baichuan2-7B, Gemini-Nano). As shown in Figure 1, the cloud-based LLM and the on-device LLM serve users in a hybrid way to cover all the time periods, thereby ensuring the completeness of LLM service. In particular, when the cloud-based LLM service is available to a user with a mobile device, the mobile device directly sends requests, and the cloud returns responses; but when the cloud-based LLM service is not available, such as when the network condition of the mobile device is poor, the free-tier rate limit of cloud API has been reached

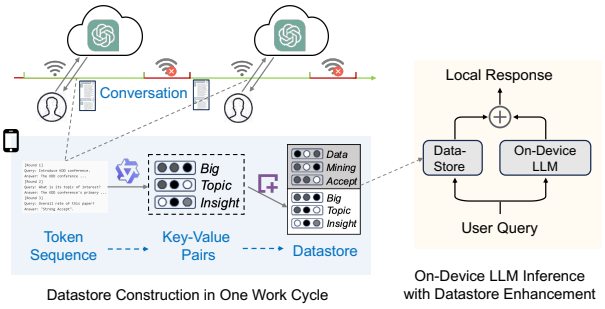


Figure 2: On-device LLM inference enhancement with local datastore constructed from cloud-based LLM interactions.

(e.g., 200 requests per day for ChatGPT, 1000 messages per day for Claude), or the cloud service platform experiences congestion or crashes, the on-device LLM needs to be invoked to serve the user.

However, compared with the cloud-based LLM, the on-device LLM is smaller in parameter size and limited in model performance. For example, according to OpenCompass 1.0 LLM Leaderboard [20], the average score and the language-related score of Qwen-1.8B are 16.9 and 12.3 lower than ChatGPT, respectively. To guarantee the quality of local service when the cloud-based API is not available, how to enhance the performance of on-device LLM under the device-cloud hybrid service framework becomes an important problem.

3 Fundamental Design of On-Device LLM Enhancement

To enhance on-device LLM service, the basic idea of our design is to build an external datastore from the historical high-quality interactions with cloud-based LLM to assist local LLM inference. The key insights include: (1) from query distribution, a certain user's historical queries to the cloud-based LLM are close to his/her queries to the on-device LLM; and (2) from response quality, the cloud-based LLM exhibits impressive capabilities, far stronger than the on-device LLM, and its responses can be regarded as reference answers to dramatically improve local LLM inference. In what follows, we introduce the details of the fundamental design.

3.1 Local Datastore Construction from Historical Cloud-Based LLM Interactions

As depicted in Figure 2, a user's new conversation with the cloud-based LLM will be stored on the mobile device. In particular, each conversation is first converted into a standard token sequence, denoted as $(w_1, w_2, \ldots, w_l)$, which can further be split into $(l-1)$ context-target pairs, denoted as $\{(w_{1:m}, w_{m+1}) | m = 1, 2, \ldots, l-1\}$. For each context-target pair $(w_{1:m}, w_{m+1})$, the context $w_{1:m}$ is first encoded by the on-device LLM to a representation vector $g(w_{1:m})$ for retrieval purpose, where $g(\cdot)$ denotes the encoding function; and $(g(w_{1:m}), w_{m+1})$ constitutes a key-value pair, added to the local datastore U on the mobile device.

The construction of a mobile device's local datastore spans a preset work cycle (e.g., 1 month) and comprises all the key-value pairs constructed from multiple user interactions with the cloud-based LLM in this work cycle. This datastore will be used to enhance the on-device LLM inference until the end of the next work cycle and then refreshed. We let $U = \{(g(x_i), y_i) | i = 1, 2, \ldots, N\}$ denote the latest local datastore with N key-value pairs in total, where $\mathcal{D} = \{(x_i, y_i) | i = 1, 2, \ldots, N\}$ denotes all the context-target pairs.

3.2 On-Device LLM Inference with Datastore

Given the external datastore U, we leverage kNN-LM [32] to assist on-device LLM inference. In essence, LLM inference is to execute the next token prediction task. We let $\hat{w}_{1:m}$ denote an input context and let $\hat{w}_{m+1}$ denote the next token to be predicted. Then, the probability distribution of $\hat{w}_{m+1}$ predicted by kNN-LM is the combination of the on-device LLM's output and the retrieval results from the datastore U, formulated as

$$\begin{aligned} &p(\hat{w}_{m+1} | \hat{w}_{1:m}, U) \\ &= (1-\alpha) p_{\text{DLM}}(\hat{w}_{m+1} | \hat{w}_{1:m}) + \alpha p_{k\text{NN}}(\hat{w}_{m+1} | \hat{w}_{1:m}, U), \end{aligned} \tag{1}$$

where $0 < \alpha < 1$ is a tuned parameter to adjust the weights of on-device LLM and datastore in the final prediction; $p_{\text{DLM}}(\hat{w}_{m+1} | \hat{w}_{1:m})$ denotes the output distribution of the on-device LLM over the next token; and $p_{k\text{NN}}(\hat{w}_{m+1} | \hat{w}_{1:m}, U)$ denotes the retrieval result from U. In particular, we let $\mathcal{K}$ denote the k-nearest neighbours in U according to the Euclidean distance between the datastore's key key and the input context's representation vector $g(\hat{w}_{1:m})$, denoted as $d(key, g(\hat{w}_{1:m}))$; and let (key, v) denote any key-value pair in $\mathcal{K}$ for brevity. Then, the retrieval result $p_{k\text{NN}}(\hat{w}_{m+1} | \hat{w}_{1:m}, U)$ can be computed as

$$p_{k\text{NN}}(\hat{w}_{m+1} | \hat{w}_{1:m}, U) \propto \sum_{(key,v) \in \mathcal{K}} \mathbb{1}_{\hat{w}_{m+1}=v} e^{-d(key, g(\hat{w}_{1:m}))}. \tag{2}$$

3.3 Feasibility Study

To validate the feasibility of the fundamental design, we first measure the time overhead of local datastore construction and introduce when it is reasonable to perform this task on a mobile device. For the token sequence $(w_1, w_2, \ldots, w_l)$, the most time-consuming step of datastore construction is to compute the representation vectors $\{g(w_{1:m}) | m = 1, 2, \ldots, l-1\}$, the encoding speed of which is similar to the on-device LLM generation speed. For example, the inference speed of Qwen-1.8B on the middle-end smartphone using MNN [39] is roughly 18 tokens per second. We also measure the response latency of the cloud-based Qwen API and find that the generation speed is roughly 10.3 tokens per second, which is slower than the on-device LLM generation speed. Therefore, after a mobile device sends a request and is waiting for the response from the cloud, the mobile device is almost idle and can perform datastore construction, not requiring any extra time.

We then reveal the performance improvement of on-device LLM enhancement with a datastore. We take a multi-turn dialogue dataset, called ShareGPT [11], which consists of the real-world conversations between users and ChatGPT. We filter out the datasets about 5 different topics from ShareGPT. Each topic-related dataset simulates the conversations of a user. For each user, we take 70% of conversations as the cloud-based LLM (i.e., ChatGPT) interactions to construct the local datastore. The number of conversations for each user's datastore construction is between 200 and 400 and

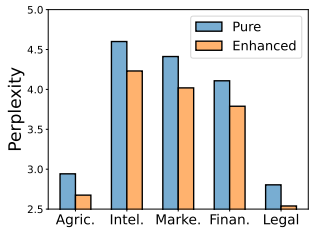


Figure 3: The perplexity of Qwen-1.8 with and without an external datastore.

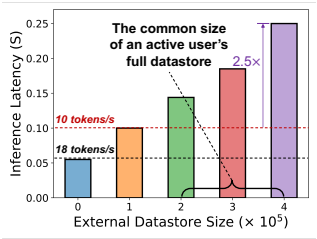


Figure 4: The on-device inference latency of Qwen-1.8 with different datastore sizes.

aligns with the practical number of ChatGPT conversations for an active user in one month. The size of key-value pairs in the datastore is between 2×10^5 and 4×10^5. We take the remaining 30% of conversations to evaluate the perplexity of on-device LLM (i.e., Qwen-1.8B) with and without datastore enhancement. From the results shown in Figure 3, we can observe that the datastore enhancement indeed improves the on-device LLM performance by sharply reducing perplexity.

We finally identify the bottleneck of on-device LLM generation speed with different sizes of an external datastore. We take the smartphone Honor V30 Pro as the testbed and take Qwen-1.8B as the on-device LLM. In particular, the dimension of the representation vector in the datastore is 2048, namely, the hidden size of Qwen-1.8B. In addition, the inference latency per token is estimated by adding up the original generation latency of Qwen-1.8B and the kNN-LM search time measured on the mobile device. We vary the size of datastore from 0 to 4×10^5, increasing by 10^5 each time, and show the results in Figure 4. We can see that (1) as the datastore size grows, the on-device inference latency increases; and (2) to achieve real-time token generation (i.e., at least 10 tokens per second), the size of datastore should be kept under 10^5. In particular, with 4×10^5 key-value pairs in the datastore, the on-device inference speed is only 4 tokens per second, 4.5 × slower than the pure on-device LLM inference, which seriously degrades user interactive experience.

3.4 Newly Introduced Datastore Subset Selection Problem

To enhance the on-device LLM while guaranteeing the inference speed, it is necessary to select a subset of full datastore, denoted as $S \subset U$, such that the size of the subset is within the preset limit L, namely, $|S| \leq L$. In addition, given the full set of the context-target pairs $\mathcal{D} = \{(x_i, y_i) | i = 1, 2, \ldots, N\}$ and the full datastore $U = \{(g(x_i), y_i) | i = 1, 2, \ldots, N\}$, the optimal subset selection strategy should maximize the log-likelihood of all the targets given the corresponding contexts. We define the optimization objective as

$$S^* = \underset{S \subset U, |S| \leq L}{\arg\max} F(S) = \underset{S \subset U, |S| \leq L}{\arg\max} \sum_{(x_i, y_i) \in \mathcal{D}} \log p(y_i | x_i, S), \quad (3)$$

where $p(y_i|x_i, S)$ can be computed by equation 1 with x_i, y_i, and S corresponding to $\hat{w}_{1:m}$, $\hat{w}_{m+1}$, and U, respectively.

Unfortunately, directly solving equation 3 is infeasible since it is NP-hard. We need to consider how to efficiently find a near-optimal datastore subset on the resource-constraint mobile device.

4 Design of On-Device Datastore Subset Selection

We first derive the submodular property and then introduce efficient offline and online designs, where the difference is whether the subset selection is performed after the full datastore construction or directly over the stream of key-value pairs.

4.1 Submodularity of Subset Selection

For kNN-LM, we assume that $p_{k\text{NN}}(y_i|x_i, S)$ in $p(y_i|x_i, S)$ is determined by all the key-value pairs with the value $v = y_i$, denoted as $S_{y_i} \subset S$. Then, the on-device next token prediction function $p(y_i|x_i, S)$ becomes

$$\begin{aligned} p(y_i|x_i, S) &= (1-\alpha) p_{\text{DLM}}(y_i|x_i) \\ &\quad + \alpha \sum_{(key,v) \in S_{y_i}} c e^{-d(key, g(x_i))}, \end{aligned} \quad (4)$$

where c is a constant. Besides, to ensure the non-negativity of the objective, we define $f(S) \triangleq F(S) - F(\emptyset)$ as the objective function, where $F(\emptyset)$ is a constant. We then demonstrate its submodularity.

Theorem 1. *Under equation 4, the objective function $f(S)$ is non-negative, monotonically non-decreasing, and submodular.*

Proof. Obviously, $f(S) \geq 0$ and $f(\emptyset) = 0$. We first proof $f(S)$ is monotone non-decreasing. For any subset $A_1 \subseteq A_2 \subseteq U$ and context-target pair (x_i, y_i), we have

$$\begin{aligned} p_{k\text{NN}}(y_i|x_i, A_1) &= \sum_{(key,v) \in A_1^{y_i}} c e^{-d(g(x_i), key)} \\ &\overset{(a)}{\leq} \sum_{(key,v) \in A_2^{y_i}} c e^{-d(g(x_i), key)} \\ &= p_{k\text{NN}}(y_i|x_i, A_2), \end{aligned} \quad (5)$$

where (a) holds because $A_1^{y_i} \subseteq A_2^{y_i}$. Therefore, we further have $p(y_i|x_i, A_1) \leq p(y_i|x_i, A_2)$ and $f(A_1) \leq f(A_2)$.

We next prove $f(S)$ is submodular. For any $A_1 \subseteq A_2 \subseteq V$, key-value pair $r_i = (g(x_i), y_i) \notin A_2$, and $(x_k, y_k) \in \mathcal{D}$, we have

$$\begin{aligned} \Delta_k(r_i|S) &= \log \frac{p(y_k|x_k, S \cup \{r_i\})}{p(y_k|x_k, S)} \\ &= \log \left(1 + \frac{\alpha(p_{k\text{NN}}(y_k|x_k, S \cup \{r_k\}) - p_{k\text{NN}}(y_k|x_k, S))}{p(y_k|x_k, S)}\right) \\ &= \log \left(1 + \frac{\alpha c \mathbb{1}_{y_k = y_i} e^{-d(g(x_i), g(x_k))}}{p(y_k|x_k, S)}\right). \end{aligned} \quad (6)$$

where $\Delta_k(r_i|S)$ is defined in equation 8 and $S \in \{A_1, A_2\}$. Besides, by substituting equation 5 into equation 4, we have $p(y_k|x_k, A_1) \leq p(y_k|x_k, A_2)$. Therefore, we can derive that

$$\forall (x_k, y_k) \in \mathcal{D},\ \Delta_k(r_i|A_1) \geq \Delta_k(r_i|A_2), \quad (7)$$

showing that $\Delta(r_i|A_1) \geq \Delta(r_i|A_2)$. Therefore, $f(S)$ is submodular. □

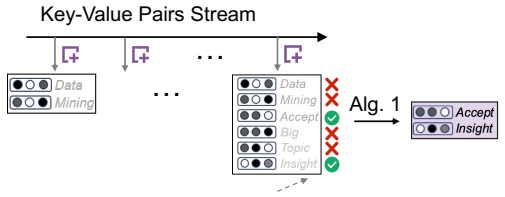


Figure 5: Workflow of offline datastore subset selection.

4.2 Offline Datastore Subset Selection

Given the properties of $f(S)$ in Theorem 1, a near-optimal datastore subset S with the approximation ratio of $(1 - 1/e)$ can be found by a naive greedy algorithm. In particular, for each step, the key-value pair $r_i \triangleq (g(x_i), y_i)$ with the highest marginal gain, denoted as $\Delta(r_i|S) := f(S \cup \{r_i\}) - f(S)$, is added to S until reaching the limit of subset size $|S| = L$. However, this greedy algorithm is inefficient on the resource-constraint mobile device for the following reasons: (1) finding the pair with the highest marginal gain among all the key-value pairs is time-consuming, as the size of the full datastore is large; (2) computing the exact $f(S)$ requires the time overhead proportional to the total number of tokens in the stored sequences; and (3) once adding a new key-value pair to the datastore subset, the marginal gains of all the remaining pairs need to be recomputed.

Specific to the above three problems, we propose an efficient offline datastore subset selection algorithm in Algorithm 1, which consists of three key optimization techniques: (1) **Random Subset Sampling** (line 3): To speed up submodular optimization with performance guarantee, a common method is to compute over a random subset rather than the full set [43]. Therefore, in each round t, we randomly sample only B key-value pairs from $U \setminus S$, denoted as R, for identifying the key-value pair with the highest marginal gain; (2) **Lower Bound on Marginal Gain** (line 4): Inspired by the widely adopted technique of maximizing the lower bound in the field of optimization, we turn to deriving the lower bound of the marginal gain that can be quickly computed as the selection metric. In particular, when adding a key-value pair $r_i = (g(x_i), y_i) \in R$ to the datastore subset S, the marginal gain can be expressed as

$$\begin{aligned}\Delta(r_i|S) &= \underbrace{\log p(y_i|x_i, S \cup \{r_i\}) - \log p(y_i|x_i, S)}_{\Delta_i(r_i|S)\text{: Internal gain over its own}} \\ &+ \sum_{k \neq i} \underbrace{[\log p(y_k|x_k, S \cup \{r_i\}) - \log p(y_k|x_k, S)]}_{\Delta_k(r_i|S)\text{: External gain over any other context-target pair}} \\ &\geq \Delta_i(r_i|S) = \log\left(1 + \frac{\alpha c}{p(y_i|x_i, S)}\right),\end{aligned} \tag{8}$$

where the inequality comes from the external gain $\Delta_k(r_i|S) \geq 0$ by equation 4. We also have experimentally found that the internal gain $\Delta_i(r_i|S)$ as the lower bound is a good estimation of the marginal gain $\Delta(r_i|S)$, since $\Delta_i(r_i|S)$ is far greater than the external gain $\Delta_k(r_i|S)$ in most cases; and (3) **Margin Gain Update After Batch Selection** (lines 4–5): From equation 4, the marginal gains of the key-value pairs with different values do not interfere with each other. Further considering the large target token space, the addition of a small number of key-value pairs has little impact on the marginal gains. Therefore, we propose to choose the top b key-value pairs from the random set R and then update the datastore subset S for the margin gain computation in the next round.

Algorithm 1 Offline Datastore Subset Selection

Require: The full datastore U; the size limit of the on-device datastore sutset L; the number of random sampling times T; the size of candidate key-value pairs per sampling B; the size of selected pairs per sampling b.
Ensure: $L = T \times b$.
1: $S \leftarrow \emptyset$;
2: **for** $t \leftarrow 1$ to T **do**
3: $R \leftarrow$ Randomly sample B key-value pairs from $U \setminus S$;
4: $R_t \leftarrow$ Select the top b key-value pairs from R according to the lower bound on the marginal gain;
5: $S \leftarrow S \cup R_t$;
6: **return** S

Algorithm Analysis. We first show that Algorithm 1 can return a near-optimal datastore subset S in expectation

$$\mathbb{E}[f(S)] \geq \left(1 - \frac{1}{e} - \gamma - \epsilon\right) f(S^*), \tag{9}$$

where γ, ϵ are related to b and B/b, respectively, and $\gamma, \epsilon \ll 1$ with appropriate settings. The details are deferred to Appendix A.

We also analyze the time complexity. In each round, it takes $O(B)$ time to compute the lower bound on the marginal gain for $r \in R$ and further takes $O(B)$ time to choose the top b key-value pairs. Therefore, the time complexity of Algorithm 1 is $O(BT)$, which is independent of the size of the full datastore N and is quite scalable.

View from Active Learning. Algorithm 1 can be regarded as a special case of pool-based active learning [52]. We recall that the lower bound on the r_i's marginal gain is monotonically decreasing with respect to $p(y_i|x_i, S)$. Therefore, for a given model (i.e., the on-device LLM plus the datastore subset), selecting the key-value pair with the largest lower bound on the marginal gain is actually choosing the data sample with the largest training loss (i.e., $-\log p(y_i|x_i, S)$). Analogous to active learning, which repeatedly collects unlabeled data samples with highest uncertainty for fine-tuning, Algorithm 1 repeatedly chooses the key-value pairs with the largest training loss and adds them into the datastore subset to "finetune" the on-device LLM in a non-parametric way.

4.3 Online Selection over Key-Value Stream

We propose an online extension of the datastore subset selection, which supports incremental selection directly over the key-value stream and can be flexibly scheduled at appropriate time throughout the entire work cycle. As illustrated in Figure 6, the basic idea is to introduce a small buffer of B key-value pairs. Once the buffer is full, the selection will be triggered. The key difference from the offline algorithm is that the length of stream (i.e., the full datastore size N) is unknown when performing selection. Therefore, how many key-value pairs should be selected from the buffer is also unknown. To ensure that regardless of N at least one feasible datastore subset can be constructed at the end of the work cycle, multiple candidate datastore subsets are maintained simultaneously to select the key-value pairs from the buffer with different ratios.

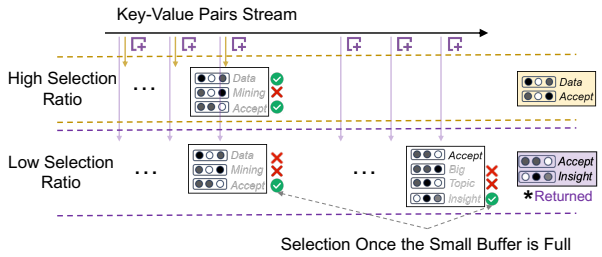


Figure 6: Workflow of online selection with different ratios.

Algorithm 2 Online Key-Value Buffer Selection

Require: The online stream of key-value pairs $\{r_1, r_2, \ldots, r_N\}$; the size limit of datastore subset L; the candidate datastore subsets $\mathbf{S} = \{S_1, S_2, \ldots, S_h\}$; the size of full buffer B; different numbers of selected key-value pairs over the buffer $\mathbf{b} = \{b_1, b_2, \ldots, b_h\}$.

1: For each $S_j \in \mathbf{S}$, $S_j \leftarrow \emptyset$;
2: $Buffer \leftarrow \emptyset$;
3: **for** $i \leftarrow 1$ to N **do**
4: Compute the lower bound $\Delta_i(r_i|S_j)$ on the marginal gain $\Delta(r_i|S_j)$ for each $S_j \in \mathbf{S}$;
5: **if** $|Buffer| < B$ **then**
6: $Buffer \leftarrow Buffer \cup \{r_i\}$;
7: **if** $|Buffer| = B$ **then**
8: **for** each $b_j \in \mathbf{b}$ **do**
9: $R_j \leftarrow$ Select top b_j key-value pairs from $Buffer$ according to the lower bound on marginal gain over S_j;
10: $S_j \leftarrow S_j \cup R_j$;
11: **if** $|S_j| > L$ **then**
12: Remove S_j from $\mathbf{S}$ and remove b_j from $\mathbf{b}$;
13: $Buffer \leftarrow \emptyset$;
14: $max \leftarrow \arg\max_j S_j \in \mathbf{S}$;
15: **return** S_{max}

Based on the design rationale, we present the details of the online design in Algorithm 2. h candidate datastore subsets are globally maintained, denoted as $\mathbf{S} = \{S_j | j = 1, 2, \cdots, h\}$. Each datastore subset S_j will select the top b_j key-value pairs from the buffer (line 9). In particular, $\{b_j | j = 1, 2, \ldots, h-1\}$ forms an exponentially growing integer array with $b_{j+1} = b_j \times a$ for $a \geq 2$ and $b_{h-1} < B$, and $b_h = B$. Once the size of S_j exceeds the limit L, it will become inactive (lines 11-12). Finally, the active datastore subset S_{max} with the maximum selection ratio is returned for on-device LLM enhancement.

Algorithm Analysis. We first demonstrate that Algorithm 2 can return a datastore subset with good performance. The key insight is that the collection of B key-value pairs in the buffer can be viewed as randomly sampling B pairs in the offline algorithm, while maintaining multiple candidate datastore subsets ensures that the returned subset is appropriate for any length of key-value stream. Formally, we have

$$f(S_{max}) \approx \max_{S \subseteq U, |S| \leq |S_{max}|} f(S) \geq \frac{f(S^*)}{a}, \tag{10}$$

The details are deferred to Appendix A.

Table 1: Statistics over the multi-turn dialogue datasets of ShareGPT and Personalized-Interactive-Conversation.

Datasets	Topic/User Identity	#Conversations
ShareGPT-Agriculture	Agriculture/Environment	324
ShareGPT-Intelligence	Artificial Intelligence	435
ShareGPT-Marketing	Business, Marketing	583
ShareGPT-Finance	Business, Finance	500
ShareGPT-Legal	Legal	422
PerIntConv-Biologist	I'm a marine biologist	423
PerIntConv-Chef	I'm a professional chef	318
PerIntConv-Student	I'm a full-time student	457
PerIntConv-Teacher	I'm a high school teacher	405
PerIntConv-Parent	I'm a stay-at-home parent	406

We then analyze the time complexity of Algorithm 2. When a new key-value pair comes, it requires to compute the lower bound on the marginal gain for all the active candidate datastore subsets (line 4). By equation 8, the time complexity is

$$\sum_{S_j \in \mathbf{S}} O(|S_j|) \leq O(L + \frac{L}{a} + \frac{L}{a^2} + \ldots) \leq O\left(\frac{aL}{a-1}\right), \tag{11}$$

which is only $O(L)$, same as one-time kNN-LM inference complexity. In practice, the mobile device can collect a batch of key-value pairs and then trigger computation in parallel, and the time overhead is still $O(L)$. Specifically, according to our evaluation, given $L = 10^5$ and $h = 5$ candidate datastore subsets, computing the lower bound on the margin value for 1000 key-value pairs on the smartphone Honor V30 Pro costs roughly 23 seconds. Therefore, the online design is quite efficient in practice and can flexibly scheduled at the idle time of a mobile device.

5 Evaluation

5.1 Evaluation Setups

Datasets and User-Level Partition. We take 3 datasets in total for 2 different application scenarios. (1) **Multi-Turn Dialogue:** The first dataset is **ShareGPT**, which has been introduced in the feasibility study (Section 3.3). The details on the 5 topics for 5 different users and the size of topic-related conversations are reserved in Table 1 in Appendix B. The second dataset is **Personalized-Interactive-Conversations** [8], which comprises the prompts about users' profiles and the conversations between the users and a cloud-based assistant. Based on the user information, we filter out the conversations of 5 different users. The number of one user's conversations is between 318 and 423. For the two multi-turn dialogue datasets, we take about 70% of the conversations for local datastore construction and leave the remaining 30% for testing the performance of on-device LLM inference. The statistics of the simulated multi-turn dialogue user datasets are recorded in Table 1; and (2) **Single-Turn QA:** The third dataset is **Chinese-Alpaca** [3], which translates the Alpaca's training dataset [1] to Chinese using ChatGPT API. We filter out low-quality answers and keep only the responses with the length longer than 128. We evenly divide the filtered data to 4 different users. Each user has roughly 2000 question-answer pairs, about 80% for building the local datastore and the left 20% for testing.

Table 2: Perplexity (lower is better) of the proposed design and baselines with different on-device LLMs over different datasets.

Datasets	On-Device LLMs	Pure	Random	Merging	Full	Subset		Offline vs.		
						Offline	Online	Pure	Random	Merging
ShareGPT-Agriculture	Qwen-1.8B	3.5326	2.9415	2.8613	2.6753	2.7428	2.7661	↓ 22.36%	↓ 6.76%	↓ 4.14%
ShareGPT-Intelligence	Qwen-1.8B	5.2997	4.6004	4.4595	4.2308	4.3608	4.3809	↓ 17.72%	↓ 5.21%	↓ 2.21%
ShareGPT-Marketing	Qwen-1.8B	4.6277	4.4120	4.2534	4.0194	4.1786	4.1585	↓ 9.70%	↓ 5.29%	↓ 1.76%
ShareGPT-Finance	Qwen-1.8B	4.3579	4.1084	3.9814	3.7885	3.8857	3.8804	↓ 10.84%	↓ 5.42%	↓ 2.40%
ShareGPT-Legal	Qwen-1.8B	3.9492	2.8039	2.7421	2.5386	2.6336	2.7365	↓ 33.31%	↓ 6.07%	↓ 3.96%
PerIntConv-Biologist	TinyLlama-1.1B	3.9138	3.0852	3.1379	2.9343	3.0795	3.0850	↓ 21.32%	↓ 0.18%	↓ 1.86%
	OPT-1.3B	4.2855	3.6492	3.7104	3.5461	3.6175	3.6549	↓ 15.59%	↓ 0.87%	↓ 2.50%
	Pythia-1.1B	4.6530	3.9049	3.9537	3.7138	3.8290	3.8373	↓ 17.71%	↓ 1.94%	↓ 3.15%
PerIntConv-Chef	TinyLlama-1.1B	3.7845	3.1547	3.2212	3.0216	3.1386	3.1680	↓ 17.07%	↓ 0.51%	↓ 2.56%
	OPT-1.3B	4.1334	3.6720	3.7556	3.5860	3.6068	3.6406	↓ 12.74%	↓ 1.78%	↓ 3.96%
	Pythia-1.1B	4.4137	3.9276	4.0672	3.7910	3.7848	3.8322	↓ 14.25%	↓ 3.64%	↓ 6.94%
PerIntConv-Student	TinyLlama-1.1B	4.2198	3.5195	3.5352	3.2918	3.4605	3.4638	↓ 17.99%	↓ 1.68%	↓ 2.11%
	OPT-1.3B	4.3181	3.8757	3.9048	3.7174	3.7891	3.7944	↓ 12.25%	↓ 2.23%	↓ 2.96%
	Pythia-1.1B	4.7743	4.2940	4.3464	4.0531	4.1213	4.1295	↓ 13.68%	↓ 4.02%	↓ 5.18%
PerIntConv-Teacher	TinyLlama-1.1B	4.3038	3.4549	3.5327	3.3037	3.4445	3.4543	↓ 19.97%	↓ 0.30%	↓ 2.50%
	OPT-1.3B	4.3785	3.8069	3.8362	3.6795	3.7621	3.7667	↓ 14.08%	↓ 1.18%	↓ 1.93%
	Pythia-1.1B	4.8305	4.1705	4.2105	3.9761	4.0632	4.0708	↓ 15.88%	↓ 2.57%	↓ 3.50%
PerIntConv-Parent	TinyLlama-1.1B	4.5374	3.7662	3.7978	3.552	3.7056	3.7091	↓ 18.33%	↓ 1.61%	↓ 2.43%
	OPT-1.3B	4.6678	4.1543	4.2432	4.0396	4.0509	4.1068	↓ 13.22%	↓ 2.49%	↓ 4.53%
	Pythia-1.1B	5.2705	4.6980	4.7869	4.4629	4.4926	4.5239	↓ 14.76%	↓ 4.37%	↓ 6.15%
CN-Alpaca (Average)	Qwen-1.8B	13.3281	12.8401	12.8463	12.5659	12.7050	12.7425	↓ 4.68%	↓ 1.05%	↓ 1.10%

Cloud-Based LLM and On-Device LLMs. We take ChatGPT as the cloud-based LLM, which generates the three datasets by interacting with different users. For on-device LLMs, we take Qwen-1.8B [45] for ShareGPT, take TinyLlama-1.1B [12, 53], OPT-1.3B [54], and Pythia-1.1B [22] for Personalized-Interactive-Conversations, and take Qwen-1.8B for Chinese-Alpaca.

Baselines. To compare with the proposed on-device LLM enhancement designs with full datastore ("Full"), offline selected datastore subset ("Offline"), and online selected datastore subset ("Online"), we introduce the following baselines: (1) **Pure On-Device LLM** ("Pure"), which does not introduce any external datastore; (2) **Random Selection** ("Random"), which randomly selects L key-value pairs from the full datastore for enhancement; and (3) **Merging** [28], which heuristically compresses the full datastore by merging neighboring key-value pairs with the same value. The number of merging neighbours is set to 100, and only the top L pairs with the highest weights after merging are kept for enhancement. We note that the baselines of random selection and merging are executed in an offline manner after the construction of the full datastore.

Implementation Settings. The size limit on the datastore subset L is set to 10^5. For the offline design, the size of candidate key-value pairs per sampling B is set to 50,000, the size of selected pairs per sampling b is set to 10,000, and the number of sampling times $T = L/b$ is 10. For the online design, the buffer size is set to 10,000, a is set to 2, and $h = 5$ different numbers of selected key-value pairs over the buffer are $\mathbf{b} = \{1000, 2000, 4000, 8000, 10000\}$. For on-device kNN-LM, the number of neighbours k is set to 100, the weight of the external datastore in the final prediction α is set to 0.15, 0.25, 0.25, and 0.3 by default for Qwen-1.8B, TinyLlama-1.1B, OPT-1.3B, and Pythia-1.1B, respectively. The testbed of mobile device takes Honor V30 Pro with Kirin 990 CPU and 8GB memory, and the on-device deployment is through Termux.

5.2 Evaluation Results and Analysis

On-Device LLM Inference Performance and Latency. We report in Table 2 the perplexity of the proposed design and the baselines. We also show in Figure 7 the inference latency with the full datastore and the selected datastore subset.

By comparing the enhancement with the full datastore with the pure on-device LLM, we can see that the perplexity decreases by 18.11% on average. We can draw that the historical interactions with the cloud-based LLM indeed can significantly improve the quality of on-device LLM generation.

By comparing the on-device LLM enhancement with the offline and the online selected datastore subsets and with the full datastore, we can observe that the offline and the online designs averagely increase the perplexity by 2.49% and 3.07% (i.e., 0.09 and 0.11 in absolute value of perplexity), respectively. The slight perplexity gap comes from dropping up to 74% of key-value pairs. As a result, the on-device LLM inference speed using the offline and the online designs is up to 2.3× faster than using the full datastore, where the latter speed is much lower than the desired 10 tokens per second.

By comparing the on-device LLM enhancement with the proposed offline and online designs with the three baselines, we can observe that (1) compared with the pure on-device LLM, the perplexities of the offline and the online designs decrease by 16.07% and

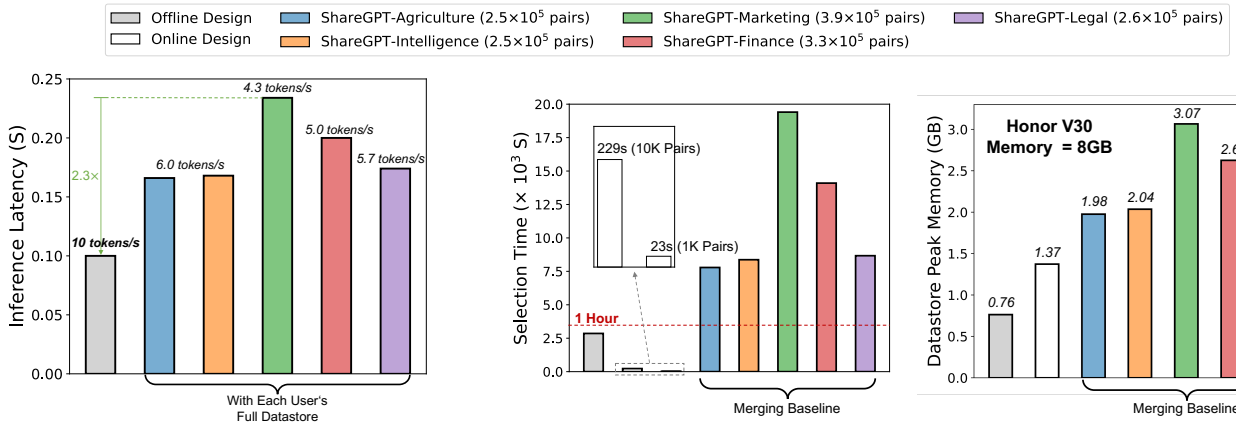
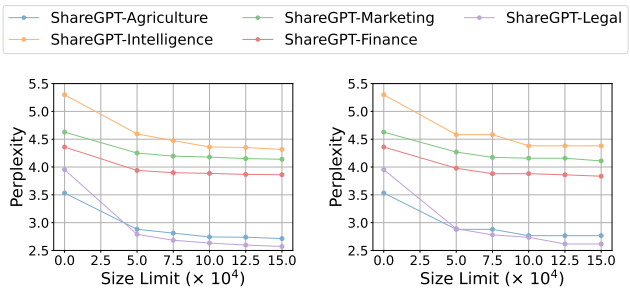


Figure 7: The on-device inference latency with the full datastore and with the offline selected datastore subset.

(a) Selection Time (b) Peak Memory

Figure 8: The selection time and peak memory of the proposed offline and online designs as well as the merging baseline.



(a) Offline Design (b) Online Design

Figure 9: The perplexity of the offline and the online designs with varying limits on the datastore subset size.

15.59% on average, respectively; (2) compared with the random selection policy, which is the strongest baseline over the Personalized-Interactive-Conversations and CN-Alpaca datasets, the perplexities of the offline and the online designs decrease by 2.82% and 2.64% on average, respectively; and (3) compared with the merging baseline, which is the strongest baseline over the ShareGPT dataset, the perplexities of the offline and the online designs decrease by 3.23% and 2.65% on average, respectively. These results reveal that the datastore subset selection policy based on the metric of the marginal gain is more effective in finding high-quality key-value pairs for on-device LLM enhancement.

Additionally, the results over the ShareGPT and Personalized-Interactive-Conversations datasets demonstrate the consistent enhancement of using the datastore (subset) over pure on-device LLM under different levels of in-dataset topic similarities. Specifically, each ShareGPT simulated user dataset follows the same topic, a simulated user dataset from Personalized-Interactive-Conversations involves highly diverse topics, and the evaluation results over both datasets in Table 2 show the effectiveness of taking the datastore (subset) for enhancement.

On-Device Datastore Subset Selection Time. We first compare the on-device subset selection time of the proposed offline design and the merging baseline over the ShareGPT dataset. The results are shown in Figure 8. Specifically, the primary time consuming contributor is the evaluation of marginal gain, which is estimated by equation 8. As $p_{\text{DLM}}(y|x)$ is already computed when encoding the interaction data, the extra time consumed for evaluating is approximately equal to that for computing $p_{k\text{NN}}(y|x, S)$, which further approximates to the retrieval time. Therefore, we use the measured retrieval time as the estimation of the execution time of the algorithms. More specifically, for Algorithm 1, we measure the total retrieval time; and for Algorithm 2, we measure the time required for one step of evaluation, which retrieves a batch of 10,000 or 1,000 key-value pairs based on the setting, and plot the longest time consumption in Figure 8. One key observation is that the time overhead of the proposed offline design is 47.6 minutes, reducing up to 6.79 × than the merging baseline. The second key observation is that as the size of a user's local full datastore grows, the time overhead of the offline design does not change, whereas the time overhead of the merging baseline significantly grows. This is because the time complexity of the offline design is $O(BT)$, which is independent of the full datastore size N, while the complexity of the merging baseline is $O(N^2)$.

We then show the efficiency of online selection design over the key-value stream. When accumulating 1,000 and 10,000 key-value pairs in a batch, the most time-consuming step of computing the lower bound on the margin gain in parallel costs roughly 23 seconds and 229 seconds, respectively.

On-Device Peak Memory. We compare the memory usages of the proposed offline and online designs as well as the merging baseline. Specifically, we record the highest total memory consumption of the active datastore subsets, and plot the results in Figure 8. We can see that the peak memories of our two designs are consistently lower than that of the merging baseline, reducing up to 4 ×. We can also observe that the memory usage of the merging baseline is proportional to the user's full datastore size, whereas the memory usages of the offline and online designs depend only on the size limit L on the datastore subset. We can further see that the peak

memory of the online design is higher than that of the offline design, because several candidate datastore subsets need to be maintained. Nevertheless, by equation 11, the peak memory of the online design does not exceed $2\times$ of the offline design's memory.

Impact of Datastore Subset Size. We vary the limit on the datastore subset size L from 5×10^4 to 15×10^4. From the perplexity depicted in Figure 9, we can observe that the enhancement with the offline or the online selected datastore subset significantly outperforms the pure on-device LLM (i.e., $L = 0$). We can also see that as L grows, the perplexity of the offline design consistently decreases, and the PPL of the online design shows a step-wise descent, since different L's may return the same datastore subset. Of course, a larger L will lead to a slower on-device LLM inference speed, implying a trade-off between effectiveness and efficiency.

6 Related Work

6.1 On-Device LLM Inference

The LLMs on the mobile devices are good complements to the cloud-based LLMs with the nice properties of local service, low response latency, low cost, service customization, and good privacy. Tech giants (e.g., Alibaba [45], Google [26], Microsoft [36]), smartphone manufacturers (e.g., Apple [2], Xiaomi [7], vivo [14], Huawei [5]), and smartphone chipmakers (e.g., MediaTek [6], Qualcomm [9]) are racing to launch light-weight LLMs and integrate them into mobile operating systems and mobile applications. To deploy LLMs on heterogeneous mobile devices, MLC-LLM [18] provided a high-performance native deployment solution with machine learning compilation technique. MNN [39] reduced the memory usage by segmenting LLM and loading each segment from storage one at a time for on-device inference. Flash-LLM [21] enabled on-device LLM inference by storing the model parameters in large flash memory and bringing them on demand to small DRAM. At the algorithm level, some existing work considered how to accelerate token generation. For example, speculative decoding [24, 33, 50] employed a light-weight LLM for fast token generation and leveraged a more powerful LLM for parallel verification. In addition, the sparse model architecture of mixture of experts was adopted to selectively activate a few model blocks to reduce inference overhead [41, 51].

Parallel to the above work, we propose to enhance the token generation quality of on-device LLM with the user's historical interactions with the cloud-based LLM.

6.2 Device-Cloud Collaborative Learning

To integrate the natural advantages and the resources of both mobile devices and the cloud, the new paradigm of device-cloud collaborative learning emerges. The celebrated federated learning framework [23, 31, 37, 42, 44] considered deploying the same model on the cloud and each mobile device. The model size was limited by the resource constraints of the mobile device. Later, some work turned to the collaboration of a powerful model on the cloud and light-weight models on the mobile devices. Specific to LLMs, model multiplexing [56] invoked the cloud-based LLM only when the query is not in the cache and the on-device LLM cannot provide a satisfying answer, the purpose of which is to reduce the overhead of cloud-based LLM inference. For collaborative training, one line of work (e.g., CD-CCA [48]) proposed to let the cloud train user-specific LLMs, while the mobile devices are responsible for collecting training data and performing inference. Such centralized design is practically infeasible when the number of mobile devices is large (e.g., billions in practice). In contrast, this work proposes to leverage the user's interactions with the cloud as a data provider and distribute the LLM enhancement task on each mobile device without the need of local training. The other line of existing work (e.g., [25, 49]) proposed an on-device emulator-based finetuning technique. These work focused on enhancing cloud-based LLMs rather than on-device LLMs.

6.3 LLM Enhancement with External Datastore

As introduced in Section 1, RAG [27, 34] and kNN-LM [32, 55] are two typical methods for enhancing LLM inference with an external datastore. RAG has been widely used in natural language tasks [38, 40], and can effectively eliminate LLM hallucinations [46]. Compared with RAG, kNN-LM is memory efficient [10] and allows longer user-side input sequence, which are important for on-device LLM inference. One line of the following work of kNN-LM considered how to improve inference efficiency by compressing the datastore in the area of machine translation. He et al. [28] proposed the greedy merging method to prune the datastore, which has been introduced as an offline baseline for comparison in our evaluation, and leveraged principal component analysis (PCA) to reduce the dimension of the keys in the datastore. Wang et al. [47] proposed to train a mapping model to compress the dimension of the representation vector and then perform cluster-based pruning.

Existing work on kNN-LM efficiency focused on compressing the datastore on the cloud. In contrast, this work requires the datastore subset selection be executed on the mobile device, imposing the constraints on the memory usage and the time complexity. As a result, the techniques of PCA and model training adopted in [28, 47] are no longer applicable. In contrast, the proposed designs require only the estimation of marginal gain for key-value pairs selection.

7 Conclusion

In this work, we have proposed to construct a local high-quality datastore from the user's historical interactions with the cloud-based powerful LLM to effectively and efficiently enhance on-device LLM inference. To balance the quality and the speed of token generation, we have proposed both offline and online designs to select a subset of the full datastore with near-optimal performance guarantees and low time complexities. Extensive evaluation results have demonstrated the overall superiority of the proposed designs over the pure on-device LLM, the enhancement with full datastore, the random selection policy, and the merging method, from perplexity, generation speed, memory usage, and selection time.

Acknowledgements

This work was supported in part by National Key R&D Program of China (No. 2022ZD0119100), China NSF grant No. 62025204, No. 62202296, and No. 62272293, Tencent WeChat Research Fund, and SJTU-Huawei Explore X Gift Fund. The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government.

References

[1] 2023. Alpaca Dataset. https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json.

[2] 2023. Apple Language Model Research Report. https://www.theverge.com/2023/9/6/23861763/apple-ai-language-models-ajax-gpt-training-spending.

[3] 2023. Chinese Alpaca Dataset. https://github.com/LC1332/Chinese-alpaca-lora.

[4] 2023. Google Gemini. https://cloud.google.com/blog/products/ai-machine-learning/gemini-support-on-vertex-ai.

[5] 2023. Huawei Noah. https://github.com/huawei-noah/Pretrained-Language-Model.

[6] 2023. MediaTech On-Device LLM Report. https://www.mediatek.com/blog/mediatek-research-launches-the-worlds-first-ai-llm-in-traditional-chinese.

[7] 2023. MiLM. https://github.com/XiaoMi/MiLM-6B.

[8] 2023. Personalized Interactive Conversations Dataset. https://huggingface.co/datasets/erbacher/personalized-interactive-conversations.

[9] 2023. Qualcomm On-Device LLM Report. https://www.qualcomm.com/news/releases/2023/07/qualcomm-works-with-meta-to-enable-on-device-ai-applications-usi.

[10] 2023. Retrieval-based Language Models and Applications. https://acl2023-retrieval-lm.github.io/.

[11] 2023. ShareGPT Dataset. https://huggingface.co/datasets/shareAI/ShareGPT-Chinese-English-90k.

[12] 2023. TinyLlama. https://huggingface.co/PY007/TinyLlama-1.1B-Chat-v0.1.

[13] 2023. UC Berkeley Vicuna. https://github.com/eddieali/Vicuna-AI-LLM.

[14] 2023. vivo BlueLM. https://developers.vivo.com/product/ai/bluelm.

[15] 2024. Alibaba Qwen. https://github.com/QwenLM/Qwen.

[16] 2024. Baichuan2. https://github.com/baichuan-inc/Baichuan2.

[17] 2024. Meta Llama. https://github.com/facebookresearch/llama.

[18] 2024. MLC-LLM. https://llm.mlc.ai/.

[19] 2024. OpenAI ChatGPT. https://chat.openai.com/.

[20] 2024. Openompass LLM Leaderboard. https://rank.opencompass.org.cn/leaderboard-llm.

[21] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C. Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *CoRR* abs/2312.11514 (2023).

[22] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *ICML (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 2397–2430.

[23] Dongqi Cai, Yaozong Wu, Shangguang Wang, Felix Xiao, zhu Lin, and Mengwei Xu. 2023. Efficient Federated Learning for Modern NLP. In *MobiCom*. ACM, Madrid, Spain, 14 pages.

[24] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. *CoRR* abs/2302.01318 (2023).

[25] Yucheng Ding, Chaoyue Niu, Fan Wu, Shaojie Tang, Chengfei Lyu, and Guihai Chen. 2023. DC-CCL: Device-Cloud Collaborative Controlled Learning for Large Vision Models. *CoRR* abs/2303.10361 (2023).

[26] Gemini Team Google. 2023. Gemini: A Family of Highly Capable Multimodal Models. *CoRR* abs/2312.11805 (2023).

[27] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-Augmented Language Model Pre-Training. *CoRR* abs/2002.08909 (2020).

[28] Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient Nearest Neighbor Language Models. In *EMNLP (1)*. Association for Computational Linguistics, 5703–5714.

[29] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 2790–2799.

[30] 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*. OpenReview.net, Virtual, 13 pages.

[31] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *ICML*, Vol. 119. PMLR, Virtual, 5132–5143.

[32] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through Memorization: Nearest Neighbor Language Models. In *ICLR*. OpenReview.net.

[33] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *ICML (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 19274–19286.

[34] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[35] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *ACL/IJCNLP (1)*. Association for Computational Linguistics, 4582–4597.

[36] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks Are All You Need II: phi-1.5 technical report. *CoRR* abs/2309.05463 (2023).

[37] Youpeng Li, Xuyu Wang, and Lingling An. 2023. Hierarchical Clustering-based Personalized Federated Learning for Robust and Fair Human Activity Recognition. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 1 (2023), 20:1–20:38.

[38] Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. 2021. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN. In *ICLR*. OpenReview.net.

[39] Chengfei Lv, Chaoyue Niu, Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, Tao Huang, Hui Shu, Jinde Song, Bin Zou, Peng Lan, Guohuan Xu, Fei Wu, Shaojie Tang, Fan Wu, and Guihai Chen. 2022. Walle: An End-to-End, General-Purpose, and Large-Scale Production System for Device-Cloud Collaborative Machine Learning. In *OSDI*. USENIX, Carlsbad, CA, USA, 249–265.

[40] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-Augmented Retrieval for Open-Domain Question Answering. In *ACL/IJCNLP (1)*. Association for Computational Linguistics, 4089–4100.

[41] Saeed Masoudnia and Reza Ebrahimpour. 2014. Mixture of experts: a literature survey. *Artif. Intell. Rev.* 42, 2 (2014), 275–293.

[42] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*. JMLR, Fort Lauderdale, USA, 1273–1282.

[43] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. 2015. Lazier Than Lazy Greedy. In *AAAI*. AAAI Press, 1812–1818.

[44] Jaeyeon Park, Kichang Lee, Sungmin Lee, Mi Zhang, and JeongGil Ko. 2023. AttFL: A Personalized Federated Learning Framework for Time-series Mobile and Embedded Sensor Data Processing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 3 (2023), 116:1–116:31.

[45] Alibaba Group Qwen Team. 2023. Qwen Technical Report. *CoRR* abs/2309.16609 (2023).

[46] S. M. Towhidul Islam Tonmoy, S. M. Mehedi Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. 2024. A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models. *CoRR* abs/2401.01313 (2024).

[47] Dexin Wang, Kai Fan, Boxing Chen, and Deyi Xiong. 2022. Efficient Cluster-Based k-Nearest-Neighbor Machine Translation. In *ACL (1)*. Association for Computational Linguistics, 2175–2187.

[48] Guanqun Wang, Jiaming Liu, Chenxuan Li, Junpeng Ma, Yuan Zhang, Xinyu Wei, Kevin Zhang, Maurice Chong, Ray Zhang, Yijiang Liu, and Shanghang Zhang. 2023. Cloud-Device Collaborative Learning for Multimodal Large Language Models. *CoRR* abs/2312.16279 (2023).

[49] Guangxuan Xiao, Ji Lin, and Song Han. 2023. Offsite-Tuning: Transfer Learning without Full Model. *CoRR* abs/2302.04870 (2023).

[50] Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. 2023. LLMCad: Fast and Scalable On-device Large Language Model Inference. *CoRR* abs/2309.04255 (2023).

[51] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. 2023. EdgeMoE: Fast On-Device Inference of MoE-based Large Language Models. *CoRR* abs/2308.14352 (2023).

[52] Xueying Zhan, Huan Liu, Qing Li, and Antoni B. Chan. 2021. A Comparative Survey: Benchmarking for Pool-based Active Learning. In *IJCAI*. ijcai.org, 4679–4686.

[53] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. *CoRR* abs/2401.02385 (2024).

[54] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *CoRR* abs/2205.01068 (2022).

[55] Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. Training Language Models with Memory Augmentation. In *EMNLP*. Association for Computational Linguistics, 5657–5673.

[56] Banghua Zhu, Ying Sheng, Lianmin Zheng, Clark W. Barrett, Michael I. Jordan, and Jiantao Jiao. 2023. On Optimal Caching and Model Multiplexing for Large Model Inference. *CoRR* abs/2306.02003 (2023).

A Analysis and Proofs

A.1 Analysis of Algorithm 1

First, we show the feasibility of updating $\Delta(r|S)$ after batch selection by proving that the probability of key-value pairs with the same label appearing among R_t is very low. More specifically, we assume that the values (target tokens) in R_t follow a distribution d_t, with $p_{d_t}(v = y) \le p_0$ for any y, where p_0 is very small and at the magnitude of $\frac{1}{\text{Size of Token Space}}$. Then the probability that the values of different key-value pairs in R_t are all different satisfies

$$\begin{aligned}&\Pr(\forall(g(x_i), y_i), (g(x_j), y_j) \in R_t \ \ (i \neq j), y_i \neq y_j) \\ &\ge \prod_{i=1}^{b}(1 - p_0(i-1)) \ge (1 - p_0(b-1))^{b/2} \ge e^{-\frac{p_0 b^2/2}{1-p_0 b}}\end{aligned} \tag{12}$$

Let $\gamma \triangleq \frac{p_0 b^2/2}{1-p_0 b}$, and γ could be far less than 1 with appropriate setting of b. Specifically, for any $0 < \hat{\gamma} < 1$, we have

$$\gamma \le \hat{\gamma} \ when \ 0 < b \le \min\left\{\frac{1}{p_0}, \sqrt{\hat{\gamma}^2 + \frac{2\hat{\gamma}}{p_0}} - \hat{\gamma}\right\}. \tag{13}$$

With an appropriate b we can have $\gamma \ll 1$, and further have

$$\Pr(\forall(g(x_i), y_i), (g(x_j), y_j) \in R_t \ (i \neq j), y_i \neq y_j) \ge e^{-\gamma} \ge (1 - \gamma) \tag{14}$$

By equation 14, we show that it is likely for the key-value pairs in R_t to have different values, thus the updating the marginal gain after batch selection can significantly improve efficiency while generally keeping the performance. In practice, as p_0 is extremely small, the appropriate range of setting b is actually wide for a small γ. In addition, although in our experiments we choose some very large b that may exceed the theoretical range for efficiency, Algorithm 1 still maintains good performance.

We next analyze the approximation ratio of Algorithm 1. Let S_t denote the selected datastore subset before round t. By equation 4, when all the key-value pairs in R_t have distinct values, we have

$$\Delta(R_t|S_t) = \sum_{r \in R_t} \Delta(r|S_t) \tag{15}$$

Based on equations 14 and 15, we make the following assumption for the brevity of the analysis.

Assumption 1. *For any S_t and R_t at step t in Algorithm 1, we have $\Delta(R_t|S_t) \ge (1-\gamma)\sum_{r \in R_t}\Delta(r|S_t)$.*

We further make an assumption about computing $\Delta(r|S_t)$.

Assumption 2. *For any S_t at step t, we can obtain $\Delta(r|S)$ (by exactly computing it or estimating it by its lower bound).*

Under the assumptions, we bound the approximation ratio.

Theorem 2. *Under Assumptions 1 and 2, we can have a near-optimal datastore subset in expectation*

$$\mathbb{E}[f(S)] \ge (1 - \frac{1}{e} - \gamma - \epsilon) f(S^*),$$

where $\epsilon = e^{-\frac{BL}{bN}}$ and S^* denotes the optimal subset with $|S^*| \le L$.

Proof. We first introduce Lemma 1:

Lemma 1. *[43] Given the current datastore subset S, the expected gain of selecting the key-value pair with the highest marginal gain among m random pairs is at least $\frac{1-e^{-\frac{mL}{N}}}{L}\sum_{e \in S^* \setminus S}\Delta(e|S)$.*

Proof of Lemma 1. Following [43], we examine the probability that $R \cap (S^* \setminus S)$ and have

$$\begin{aligned}\Pr(R \cap (S^* \setminus S) = \emptyset) &= \left(1 - \frac{|S^* \setminus S|}{|V \setminus S|}\right)^m \\ &\le e^{-m\frac{|S^* \setminus S|}{|V \setminus S|}} \\ &\le e^{-m\frac{|S^* \setminus S|}{N}}.\end{aligned} \tag{16}$$

By the fact that $0 \le |S^* \setminus S| \le L$, we have

$$\Pr(R \cap (S^* \setminus S) \neq \emptyset) \ge 1 - e^{-m\frac{|S^* \setminus S|}{N}} \ge \frac{1 - e^{-\frac{mL}{N}}}{L}|S^* \setminus S|. \tag{17}$$

When $R \cap (S^* \setminus S) \neq \emptyset$, let R denote the sampled pairs and r_R^* denote the element with the highest marginal gain among R. Overall, R is equally likely to contain each element in $S^* \setminus S$, so we have

$$\begin{aligned}\mathbb{E}[\Delta(r|S)] &\ge \Pr(R \cap (S^* \setminus S) \neq \emptyset)\frac{\sum_{r \in S^* \setminus S}\Delta(r|S)}{|S^* \setminus S|} \\ &\ge \frac{1 - e^{-\frac{mL}{N}}}{L}\sum_{r \in S^* \setminus S}\Delta(r|S).\end{aligned} \tag{18}$$

□

We further view that the process of sampling B key-value pairs in Algorithm 1 as repeatedly sampling B/b key-value pairs by b times, where the sampling sets are recorded as $R^1, R^2, \cdots, R^b$ (we have neglected the small differences between the two, which is practically insignificant when $B \ll N$, while considering such differences would make the analysis extremely tedious), and by the top b selection operation, we have

$$\sum_{r \in R_t}\Delta(r|S_t) \ge \sum_{i=1}^{b}\max_{r \in R^i}\Delta(r|S_t). \tag{19}$$

By equation 19 and Lemma 1, we further have

$$\mathbb{E}\left[\sum_{r \in R_t}\Delta(r|S_t)\right] \ge \frac{b(1 - e^{-\frac{BL}{bN}})}{L}\sum_{r \in S^* \setminus S_t}\Delta(r|S_t). \tag{20}$$

By Assumption 1 and equation 20, we have

$$\mathbb{E}[\Delta(R_t|S_t)] \ge \frac{b(1-\gamma)(1 - e^{-\frac{BL}{bN}})}{L}\sum_{r \in S^* \setminus S_t}\Delta(r|S_t). \tag{21}$$

When $B/b = \frac{N}{L}\log\frac{1}{\epsilon}$, we have

$$\mathbb{E}[\Delta(R_t|S_t)] \ge \frac{b(1-\gamma)(1-\epsilon)}{L}\sum_{r \in S^* \setminus S_t}\Delta(r|S_t). \tag{22}$$

By the submodularity of $f(S)$, we have

$$\sum_{r \in S^* \setminus S_t}\Delta(r|S_t) \ge f(S^*) - f(S_t). \tag{23}$$

Therefore,

$$\begin{aligned}\mathbb{E}[f(S_{t+1}) - f(S_t)|S_t] &= \mathbb{E}[\Delta(R_t|S_t)] \\ &\geq \frac{b(1-\gamma)(1-\epsilon)}{L}\left(f(S^*) - f(S_t)\right).\end{aligned} \tag{24}$$

By taking expectation, we have

$$\mathbb{E}[f(S_{t+1}) - f(S_t)] \geq \frac{b(1-\gamma)(1-\epsilon)}{L}\mathbb{E}[f(S^*) - f(S_t)]. \tag{25}$$

By induction and $L = bT$, we have

$$\begin{aligned}\mathbb{E}[f(S_T)] &\geq \left(1 - \left(1 - \frac{b(1-\gamma)(1-\epsilon)}{L}\right)^T\right) f(S^*) \\ &\geq \left(1 - e^{-(1-\gamma)(1-\epsilon)}\right) f(S^*) \\ &\overset{(a)}{\geq} \left(1 - \frac{1}{e} - \gamma - \epsilon\right) f(S^*),\end{aligned} \tag{26}$$

where (a) follows from $1 - e^{x-1} \geq 1 - 1/e - x$ for any $0 \leq x \leq 1$. □

A.2 Analysis of Algorithm 2

We first analyze the performance of Algorithm 2. We consider the collection of key-value pairs as a stochastic process. Therefore, the collection of B key-value pairs by the buffer can be viewed as randomly sampling B pairs. Thus, the selection of S_j in Algorithm 2 can be seen as repeating randomly sampling B pairs and choosing the top b_j with the highest marginal gain for N/B times. This is equivalent to Algorithm 1 to select the near-optimal subset with $L_j := b_j N/B$ key-value pairs. In Theorem 1, we have demonstrated the effectiveness of Algorithm 1. Besides, we let $S^*_{size} \triangleq \max_{|S| \leq size} f(S)$ and introduce the following theorem for further analysis.

Theorem 3. *For a non-negative, non-decreasing, and submodular objective function $f(S)$ and the full set U, when $(L_1 \mid L_2)$ we have*

$$f(S^*_{L_1}) \geq \frac{L_1}{L_2} f(S^*_{L2}), \tag{27}$$

Proof. We divide $S^*_{L_2}$ into L_2/L_1 mutually exclusive subsets, denoted as $S^*_{L_2}(1), S^*_{L_2}(2), \cdots, S^*_{L_2}(L_2/L_1)$.

By the definition of $S^*_{L_1}$, we have

$$f(S^*_{L_1}) \geq f(S^*_{L_2}(i)),\ \forall i \in \{1, 2, \cdots, L_2/L_1\}. \tag{28}$$

Besides, by the submodularity of $f(S)$, we have

$$\sum_{i=1}^{L_2/L_1} f(S^*_{L_2}(i)) \geq f(S^*_{L_2}). \tag{29}$$

Therefore, by equations 28 and 29, we have

$$\frac{L_2}{L_1} f(S^*_{L_1}) \geq f(S^*_{L_2}), \tag{30}$$

which corresponds to Theorem 3. □

In Algorithm 2, by Theorem 3 and $f(S)$'s submodularity,

$$f(S_{max}) \approx f(S^*_{|S_{max}|}) \geq \frac{f(S^*_{a|S_{max}|})}{a} \overset{(*)}{\geq} \frac{f(S^*_L)}{a} \tag{31}$$

where $(*)$ follows from $|S_{max}| \geq |S_{max+1}|/a \geq L/a$ when $N > L$. Therefore, Algorithm 2 can return a datastore subset with good performance compared to $f(S^*_L) = f(S^*)$.

Table 3: Perplexity of the proposed design with varying distribution differences between training and test data.

Test Set	Pure	Random	Full	Offline Design	Online Design
Domain B	5.14	4.72	4.39	4.54	4.53
Domain B&C Mixed	4.51	4.29	4.19	4.22	4.22

B Supplementary Experimental Notes

B.1 Implementation Details

In this section, we introduce some details in our experimental implementation of the selection algorithms and kNN-LM. In the implementation of Algorithm 2, we stipulate that if the size of S_{max} is greater than $0.6L$, then return S_{max}; otherwise, return S_{max+1}.

For the distance function $d(key, g(\hat{w}_{1:m}))$ in kNN-LM, we adopt scaled Euclidean distance. Formally,

$$d(key, g(\hat{w}_{1:m})) = \frac{||key - g(\hat{w}_{1:m})||^2}{\theta}, \tag{32}$$

where θ is set to 10^4, 500, 500, and $10^4/6$ for Qwen-1.8B, TinyLlama-1.1B, OPT-1.3B, and Pythia-1.1B, respectively. Such a small modification has no impact on our analysis.

For the datastore subset returned by the selection algorithms, the kept key-value pairs are typically with small LLM probability, i.e., $p_{\text{DLM}}(y|x)$ is small for each $(g(x), y)$ in the subset. This is because Algorithms 1 and 2 generally selects key-value pairs that are not learned well by the on-device LLM. Therefore, kNN-LM with the subset may not help with predicting the next token for contexts where the on-device LLM can already make accurate predictions. With this observation, we design an adaptive $p_{k\text{NN}}$ weight given an input context. Specifically, let $\hat{w}_{1:m}$ denote the input context, and the predicted next token distribution is

$$\begin{aligned}p(\hat{w}_{m+1}|\hat{w}_{1:m}) = (1 - \alpha(1 - \beta \cdot \text{MaxProb}))p_{\text{DLM}}(\hat{w}_{m+1}|\hat{w}_{1:m}) \\ + \alpha(1 - \beta \cdot \text{MaxProb})p_{k\text{NN}}(\hat{w}_{m+1}|\hat{w}_{1:m}),\end{aligned} \tag{33}$$

where MaxProb:= $\max_{y \in \text{Token Space}} p_{\text{DLM}}(y|\hat{w}_{1:m})$. β is set to 1 as default, set to 0.5 for experiments with TinyLlama-1.1B, and set to 0 for experiments over ShareGPT-Agriculture, ShareGPT-Intelligence, and ShareGPT-Legal. We do not adopt the design for other baselines (i.e., $\beta = 0$) as the key-value pairs kept in their subset is not with small $p_{\text{DLM}}(y|x)$ and such a design even increases their PPL.

B.2 Additional Evaluation Results

In practice, a user's interest may change over time, causing the discrepancy between the training set and the test set. Therefore, we add experiments when there is a difference in topics between the training set and the test set. More specifically, the training set is composed of 150 conversations from ShareGPT-Agriculture (domain A) and 150 conversations from ShareGPT-Intelligence (domain B), and the full test set is composed of another 100 conversations from ShareGPT-Intelligence (domain B) and 100 conversations from ShareGPT-Marketing (domain C). We evaluate the perplexity of the proposed methods, where β (in equation 33) is set to 1 as default, over the test data with domain B only and domains B & C mixed. The evaluation results are shown in Table 3. We can observe that the proposed methods can still enhance on-device LLM for varying distribution differences between training and test data.