
MobiEdit: Resource-efficient Knowledge Editing for Personalized On-device LLMs

Zhenyan Lu^{1,2}, Daliang Xu¹, Dongqi Cai^{1,4}, Zexi Li⁴,
Wei Liu⁵, Fangming Liu², Shangguang Wang¹, Mengwei Xu^{*1},
¹Beijing University of Posts and Telecommunications, ²Pengcheng Laboratory,
³Peking University, ⁴University of Cambridge, ⁵XiaoMi AI Lab

Abstract

Large language models (LLMs) are deployed on mobile devices to power killer applications such as intelligent assistants. LLMs pre-trained on general corpora often hallucinate when handling personalized or unseen queries, leading to incorrect or outdated responses. Knowledge editing addresses this by identifying and adjusting a small crucial portion of model weights, without compromising the general knowledge. However, prior knowledge editing methods are impractical to run on local devices due to the resource-heavy backpropagation (BP) needed for updates. We present MobiEdit, the first mobile knowledge editing framework that enables efficient LLM personalization on commercial off-the-shelf (COTS) mobile devices. MobiEdit replaces full-precision BP with quantized forward-only gradient estimation, thus compatible with the energy-efficient mobile neural processing units (NPUs). To further improve gradient estimation efficiency, we introduce two optimizations: an early stopping mechanism that adaptively terminates editing upon success and a prefix cache that reuses computation across steps. Our approach enables real-time editing of a 3B-parameter model (Qwen2.5-3B-Instruct) on COTS mobile devices with $7.6 \times$ less memory, $14.7 \times$ less energy and $3.6 \times$ less latency compared to previous knowledge editing methods.

1 Introduction

Mobile LLMs are transitioning from research to real-world deployment, empowering privacy-preserving and latency-sensitive applications such as personal agents [1, 11]. While mobile LLMs already embed general world knowledge, personalized knowledge is crucial for better understanding individual users. This user-specific information is typically absent or diluted during pre-training on large public corpora. As Figure 1 shows, if the user provides the address in one conversation, personalized LLM assistant memorizes the address and applies this information on the future request.

Knowledge editing [19] is gaining popularity for its effectiveness in model personalization. Different from retrieval-augmented generation [6], knowledge editing maintains fast inference speed and does not depend on the in-context learning capabilities, which is typically weak to those small LLMs deployed on mobile devices [12]. A prevailing knowledge editing paradigm is the locate-and-edit approach [4] [14][15] [8] [7], which first identifies influential parameters and then modifies them by introducing a perturbation optimized through backpropagation to produce the expected output.

Despite knowledge editing is effective in updating knowledge, current methods relying on backpropagation (BP) face three critical challenges for mobile deployment: (1) Substantial memory overhead. For example, editing a 3B-parameter LLM (Qwen2.5-3B-Instruct) with classic locate-and-edit method ROME [14] consumes over 10GB of memory to store activations for BP, which is excessively larger than smartphones's memory capacity, typically less than 8GB. (2) Incompatibility with mobile NPUs. Modern mobile phones are equipped with high-performance and energy-efficient NPUs, such as Google TPU and Qualcomm Hexagon. Mobile NPUs are designed and optimized

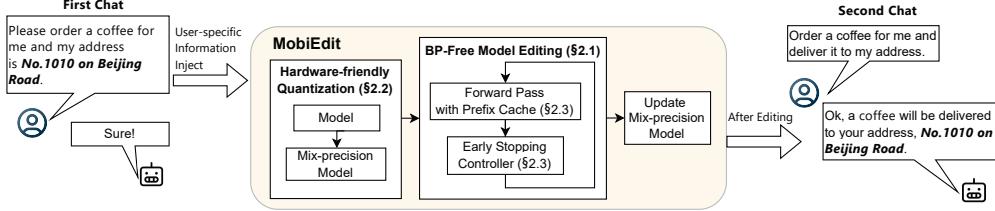


Figure 1: The on-device LLM remembers user information from the first interaction and applies it to subsequent requests.

exclusively for LLM inference, providing up to 60 \times speedup compared to CPUs [20]. However, training-specific operations are unsupported or poorly optimized by mobile NPUs [21], rendering existing BP-based knowledge editing methods infeasible. (3) Poor quantization support. BP-based training operations are often unstable or ineffective on fully quantized models (§2.2). The lack of quantization support exaggerates the previous two challenges: the memory footprint of model parameters remains prohibitively large, since full-precision weights must be stored and updated; it precludes efficient execution on mobile NPUs, which are specifically optimized for low-bit integer computation.

The above challenges confine current knowledge editing methods to cloud-based computation, undermining two key advantages of mobile LLMs: privacy preservation and offline availability. In this paper, we propose MobiEdit, the first mobile knowledge editing system for efficient LLMs personalization. MobiEdit is designed to be memory-efficient, NPU-friendly, and compatible with quantization, targeting practical deployment on resource-constrained COTS mobile devices.

Our solution To unleash the power of mobile NPUs, MobiEdit builds atop ROME, a widely locate-and-edit scheme, with a few key building blocks renovated: (1) BP-free training. Instead of calculating standard gradients using BP, MobiEdit uses the differences of loss to estimate the gradients—a classical zeroth-order optimization approach. MobiEdit operates entirely through forward passes, which is memory-efficient and well-suited for mobile NPUs. (2) NPU-friendly training-time quantization. MobiEdit introduces a new quantization paradigm for efficient knowledge editing on mobile NPUs. Unlike previous BP-based low-precision training, our forward-only gradient updating is more stable under quantized computation. We thereby quantize all model parameters except the critical projection weights essential for knowledge editing. Only a small portion of weights undergoes full-precision computation to conduct precise gradient estimation.

To compensate for the slower convergence performance caused by BP-free training [5], we introduce two optimizations to further improve system efficiency: a *prefix cache* that reuses static intermediate results across editing steps, and a *early stopping controller* that adaptively terminates editing once success criteria are met for different knowledge. Together, these design choices make MobiEdit efficient and practical for mobile deployment.

Results We test our method on three COST mobile phones, Xiaomi K60 pro, K70 and Oneplus 13. MobiEdit achieves comparable edit quality on ZsRE and CountFacts datasets while reducing memory usage by 7.6 \times (from 46GB to 6.2GB), editing latency by 3.6 \times , and energy consumption by 14.7 \times compared to ROME[14], MEMIT[15], WISE[18] and AlphaEdit[7]. To our best knowledge, MobiEdit is the first system to make LLM knowledge editing feasible on commercial smartphones.

2 Method

2.1 BP-Free Knowledge Editing

MobiEdit consists of two main stages: (1) Subject-key localization. MobiEdit first identifies the model’s internal representation of the subject; (2) Target value injection. MobiEdit then inserts new factual associations by adjusting internal activation. Specifically, the MLP layers in Transformer-based language models can be interpreted as a key-value memory [14], where the down-projection matrix $W \in \mathbb{R}^{d_v \times d}$ maps hidden representations (keys) to output activations (values). Given this view, the editing objective is to insert a new association (k_*, v_*) into the MLP such that:

$$W k_* \approx v_*. \quad (1)$$

Follow previous literature [14], MobiEdit constructs the **key** k_* by averaging the MLP-layer activation of the final subject token across a set of randomly sampled prompts:

$$k_* = \frac{1}{N} \sum_{j=1}^N \phi(x_j + s), \quad (2)$$

where $x_j + s$ denotes sampled prefix sequences followed by the subject, and $\phi(\cdot)$ denotes post-activation outputs from a selected MLP layer.

To ensure that the model recalls the target object o_* when prompted with the edited subject, we optimize a value vector $v \in \mathbb{R}^{d_v}$ such that, when substituted as the activation at the MLP output, the model generates o_* . The optimization objective is defined as follow:

$$\mathcal{L}(v) = \frac{1}{N} \sum_{j=1}^N \left[-\log \mathbb{P}_{G(v)}(o_* | x_j + p) + D_{KL}(\mathbb{P}_{G(v)}(\cdot | x_j + p') \| \mathbb{P}_G(\cdot | x_j + p')) \right], \quad (3)$$

where p is a factual prompt (e.g., “my address is a”), and p' is a neutral or essence-preserving prompt. The first term encourages correctness; the second term discourages undesired semantic drift.

Different from traditional methods that minimize this loss via computing BP-based gradients, instead, MobiEdit estimate the gradients using only forward passes. Specifically, we estimate gradients using central differences along sampled directions. Given a perturbation direction $u \sim \mathcal{N}(0, I)$, the directional gradient estimate is computed as[2]:

$$\hat{\nabla}_v \mathcal{L} = \frac{\mathcal{L}(W + \mu u) - \mathcal{L}(W - \mu u)}{2\mu} \cdot u, \quad (4)$$

where $\mu > 0$ is a small scalar step size. To further reduce variance and stabilize training, we average over N independently sampled directions $u_i \sim \mathcal{N}(0, I)$:

$$\hat{\nabla}_v \mathcal{L} = \frac{1}{N} \sum_{i=1}^N \frac{\mathcal{L}(v + \mu u_i) - \mathcal{L}(v - \mu u_i)}{2\mu} \cdot u_i. \quad (5)$$

We update $v \leftarrow v - \eta \cdot \hat{\nabla}_v \mathcal{L}$, and repeat until convergence. The resulting estimator enables approximate gradient descent solely based on forward passes. We apply this process iteratively to update v . Once the optimal v^* is found, we apply the closed-form rank-one update:

$$\widehat{W} = W + \Lambda (C^{-1} k_*)^\top, \quad \text{where } \Lambda = \frac{(v^* - W k_*)}{(C^{-1} k_*)^\top k_*}. \quad (6)$$

Here $C = K K^\top$ is an estimated key covariance matrix computed from a sample of the model’s activation statistics. This update “inserts” (k_*, v^*) into the memory, enabling the model to recall the new factual association.

Benefits of BP-free editing BP-free editing is well-suited for mobile NPUs that only support forward passes. In terms of memory efficiency, activations, which takes more than 40% of BP-based memory consumption, can be invalidated immediately after the forward pass, because only the final output is required to compute the estimated gradients.

2.2 NPU-friendly Quantization for BP-Free Editing

Despite no longer needing to store activations, the large size of LLM weights still exhausts the memory capacity of mobile devices. For example, storing the 12GB of full-precision weights from Qwen-2.5-3B often causes out-of-memory (OOM) errors on newest COTS mobile devices like the Xiaomi 15, which typically offer 16GB RAM. Besides, mobile NPUs are best suited for accelerating INT matrix multiplication with 1024-bit INT8 vector arithmetic. Their floating-point computation capabilities are relatively weak compared to mobile GPUs. To avoid frequent memory swapping, which severely harms mobile SSD lifespan, and leverage the integer computation advantages of NPUs, we propose a NPU-friendly quantization workflow for BP-free editing.

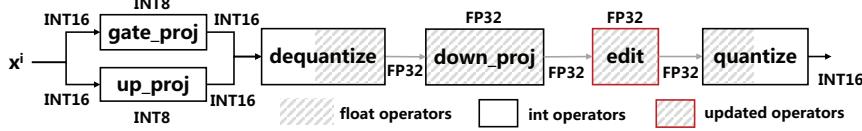


Figure 2: MobiEdit quantization workflow and strategy. MobiEdit quantizes all activation functions, with only the editing layer and its preceding layer executed in floating-point format.

Quantization workflow and strategy Figure 2 illustrates the quantization workflow of MobiEdit. Due to the hardware constraints of mobile NPUs, MobiEdit employs a static quantization strategy. The static scales for quantization are determined using representative corpora data. To balance efficiency and accuracy, MobiEdit adopts a mixed-precision editing approach: the editing vector and its preceding linear layer are executed in floating-point format; while all other weights are quantized to 8/16-bit integers. This design is informed by two key observations: (i) MobiEdit modifies only a small set of parameters proportional to the hidden size. Even minor quantization errors in this context can significantly impact editing accuracy. Furthermore, since the editing module modifies the knowledge stored in its preceding linear layer [14], floating-point precision in this layer is also crucial to maintain accuracy. (ii) The edited vector, being of limited size (equal to the hidden size), results in a negligible computational cost when floating-point precision is used for the editing module and its preceding linear layer. For example, in the Qwen-2.5-3B model, these computations account for only 0.89% of the overall computation, making the performance overhead of using floating-point precision minimal.

Advantage of our quantization MobiEdit is more robust to quantization errors than BP-based editing. Consider an L -layer Transformer network where all weights and activations are quantized:

$$W_\ell^q = W_\ell + \epsilon_{W,\ell}, \quad a_\ell^q = a_\ell + \epsilon_{a,\ell}, \quad (7)$$

where $\epsilon_{W,\ell}$ and $\epsilon_{a,\ell}$ denote quantization errors (zero mean, variance σ^2 , i.i.d. for different layers and forward passes). In each forward pass, quantization noise is recursively accumulated. If $f_\ell(x) = x$ (i.e., the network is linear), then by expanding the recursion, we have

$$a_L^q = W_L W_{L-1} \cdots W_1 x + \sum_{j=1}^L \mathcal{N}_j, \quad (8)$$

where \mathcal{N}_j denotes noise terms arising from combinations of the quantization errors $\epsilon_{W,k}$ and $\epsilon_{a,k}$ for $k \leq L$. Thus, each network output contains all previous layers' quantization noise, with total noise growing linearly or even exponentially with L . Backpropagation will amplify the noise along the chain rule. Let Δ be a small edit in layer l . The gradient w.r.t. Δ is

$$\frac{\partial \mathcal{L}}{\partial \Delta} = \frac{\partial \mathcal{L}}{\partial a_L^q} \prod_{j=l+1}^L \frac{\partial a_j^q}{\partial a_{j-1}^q} \cdot \frac{\partial a_l^q}{\partial \Delta}. \quad (9)$$

Each chain rule factor is computed on quantized variables and thus noisy; their product amplifies noise multiplicatively. Assuming each derivative factor introduces independent noise of variance σ^2 , the total gradient noise variance is approximately

$$\text{Var} \left[\nabla_{\Delta}^{\text{backprop}} \right] \sim O \left(\sigma^2 \prod_{j=l+1}^L \|W_j\|^2 \right), \quad (10)$$

wherein deeper networks or larger weights lead to rapid escalation of noise. In contrast, MobiEdit only accumulates noise throughout the forward pass. The zeroth-order estimate of the gradient is

$$g = \frac{(\mathcal{L}_\Delta + \eta_+) - (\mathcal{L}_{-\Delta} + \eta_-)}{2\Delta}, \quad (11)$$

where \mathcal{L}_\pm are two forward passes with perturbed weights $W_l^q \pm \Delta$, each with independent quantization noise. η_+ , η_- denoting output noise for the two passes.

The variance of the estimate is

$$\text{Var}[g] = \frac{\text{Var}[\eta_+] + \text{Var}[\eta_-]}{(2\Delta)^2} = \frac{2\sigma_L^2}{4\Delta^2} = \frac{\sigma_L^2}{2\Delta^2}, \quad (12)$$

where σ_L^2 is the per-pass output noise variance. This variance does not grow with network depth L .

In quantized networks, backpropagation suffers from noise that is recursively and multiplicatively propagated along the gradient chain, leading to an exponential or polynomial increase in gradient variance with depth. In contrast, the centered difference estimator’s variance is solely determined by the difference of two output noise instances, remaining bounded and independent of model depth. Thus, zeroth-order methods are provably more robust to quantization noise than backpropagation in deep or low-bit networks.

2.3 Further Optimizations

Despite a single step of MobiEdit is efficient on mobile NPUs, the remaining challenge is that it takes significant more steps to stabilize the gradient estimation for similar convergence performance. For example, MobiEdit take $20\times$ on average more steps than BP-based model editing for ZsRE and CounterFact datasets, eliminating its efficiency benefits in the wall clock time.

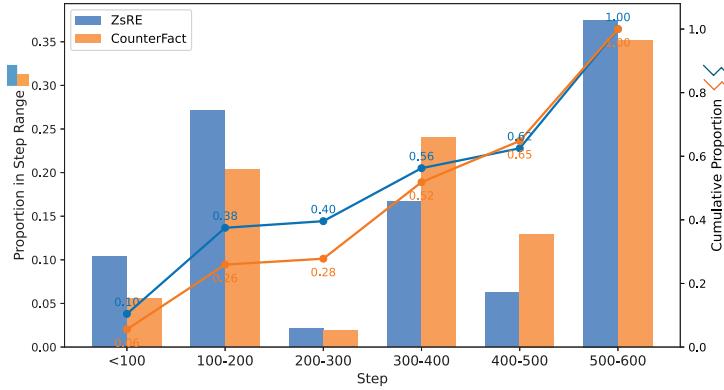


Figure 3: The edit success step number.

Early stopping controller To address this, we first analyze the successful editing step distribution of various knowledge. As shown in Figure 3, we find that different knowledge has different editing difficulty. Based on this observation, we introduce a lightweight early stopping controller that adaptively determines the editing horizon based on runtime success feedback. Specifically, during editing, we periodically evaluate the model’s response to the edited fact every M steps (e.g., every 20 steps). The editing process is terminated early once the model satisfies a pre-defined success criterion—typically, when it produces the desired target output with a confidence above a given threshold m (could be other mathematical letter), and explicitly describe the threshold we used in the eval setup. This early stopping controller automatically adjusts the editing steps to the complexity of each edit instance, avoiding unnecessary forward passes for easy-to-edit facts and reducing overfitting risk by stopping at the point of first success.

Prefix cache In our knowledge editing setting, each optimization step uses the same set of input examples constructed by combining a fixed set of randomly sampled prefixes with the fact to be edited. Formally, for a target fact f , we define a set of editing inputs as:

$$\mathcal{X}_{\text{edit}} = \{[p_1 + f], [p_2 + f], \dots, [p_n + f]\}, \quad (13)$$

where p_1, p_2, \dots, p_n are different randomly sampled prefixes. These inputs are used repeatedly across all editing steps.

We observe that in each step, the prefix tokens in the input do not change, and therefore their corresponding activations are recomputed redundantly. To reduce this overhead, we introduce a simple

optimization: during the first step, similar to KV cache, we cache the intermediate activations, corresponding to the prefix tokens. In subsequent steps, we directly reuse the cached prefix activations and only recompute the activations for the fact tokens. This greatly reduces compute without changing the model architecture or input format. Though model parameters are updated during editing, and the correct prefix activations shift across editing steps, we empirically find that reuse stale activations from the first step does not negatively affect the editing outcome.

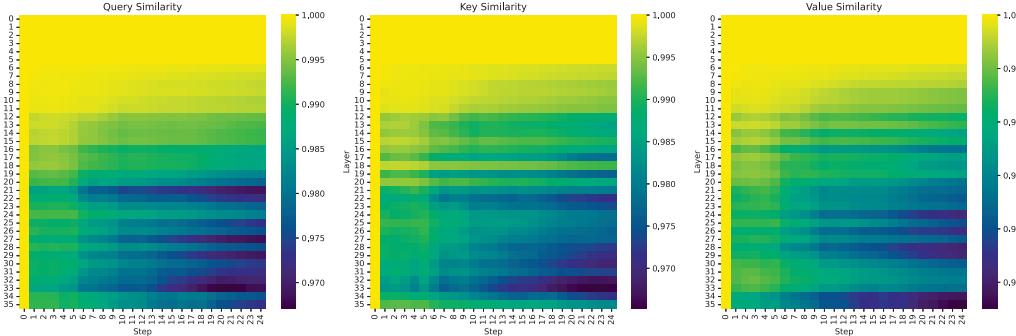


Figure 4: The cosine similarity of QKV representations at each step and layer, comparing runs with and without prefix caching.

In Figure 4, the cosine similarity gradually decreases as the number of layers and editing steps increases. However, it remains as high as 0.9 even in the deepest layers and later steps. In fact, in some cases, it slightly improves editing success and stability. We hypothesize that this is because each step in the editing process updates only a small portion of the model parameters. As a result, the overall shift in the activation distribution is minimal, and the cached prefix representations remain sufficiently aligned with the evolving model. Moreover, since the role of the prefix is primarily to introduce stochastic variation and encourage generalization, rather than to encode semantically critical content, minor discrepancies in the prefix activation have negligible impact on the final edit.

To avoid the staleness accumulates from halting the editing process, we re-compute the prefix cache as long as the editing loss does not decrease by 0.001 over 3 steps.

3 Experiments

3.1 Setup

Baselines We compare MobiEdit against four representative locate-and-edit methods: ROME [14], MEMIT [15], AlphaEdit [7], and WISE [18]. These methods follow the same paradigm of identifying key activations and injecting new knowledge into MLP layers, but differ in target granularity and update mechanism. ROME performs single-layer editing. MEMIT extends it to multi-fact scenarios. AlphaEdit uses null-space projections for preservation, and WISE incorporates dynamic routing to FFNs that store facts.

Datasets and model We evaluate MobiEdit on two standard datasets widely used in factual knowledge editing: ZsRE [10], a zero-shot relation extraction dataset derived from WikiRE, and CounterFact [14], a curated benchmark of factual edits targeting named entities (e.g., people, locations), with truth and counterfactual contexts. These two datasets jointly assess *edit success*, *locality*, and *portability* – the three key metrics for knowledge editing. We use Qwen2.5-3B-Instruct [17] as our target model, a recent open-source transformer-based LLM with approximately 3 billion parameters.

Implementation details To assess on-device feasibility, we run all editing procedures on three COTS mobile devices, as shown in Table 1. We perform all experiments using local inference engines `mllm-npu` [20] optimized for NPU execution. The latency on CPU is obtained by running on mobile phones using `llm.c` [9]. We use memory swapping while reaching the memory limit. MobiEdit uses W8A16 quantization (INT8 weights, INT16 activations), a format widely supported

Table 1: Device list.

Device	SoC	RAM	NPU
Xiaomi K60 Pro	Snapdragon 8 Gen 2	16GB LPDDR5	Hexagen NPU V73
Xiaomi K70	Snapdragon 8 Gen 3	16GB LPDDR5	Hexagen NPU V75
OnePlus 13	Snapdragon 8 Elite	24GB LPDDR5	Hexagen NPU V79

by mobile NPUs and inference engines, ensuring compatibility and throughput. The metric of memory usage in this paper is defined as the total memory required, assuming sufficient memory is available. For a simple comparison, the system efficiency values are first normalized to the range [40, 100] using min-max normalization, and then inverted.

3.2 End-to-end Performance

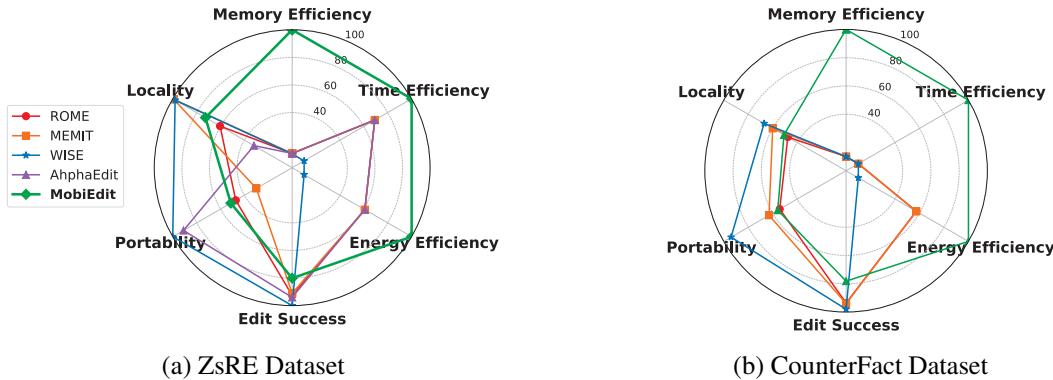


Figure 5: The comprehensive performance comparison of knowledge editing methods on the ZsRE and CounterFact datasets. System efficiency is obtained as the average of three devices.

Editing performance. We compare MobiEdit against all baselines across six dimensions: edit success, locality, portability, time efficiency, memory efficiency, and energy efficiency, as shown in Figure 5, showing that MobiEdit **achieves a balance of high accuracy and low resource cost**.

MobiEdit achieves an 80.1 edit success score, 72.6 locality score, and 51.4 portability score while requiring only 25.5 minutes, 0.018 kJ energy, and 6.2 GB memory. Although slightly behind in quality (13.9 reduction in edit success loss compared to MEMIT and AlphaEdit), MobiEdit significantly outperforms in efficiency, reducing memory usage by more than 7× and energy consumption by over 10×. These substantial resource savings make MobiEdit the only realistically viable solution for mobile devices. The significant performance improvement is attributed to MobiEdit leveraging a bp-free editing method, which reduces the memory and computational overhead caused by back-propagation. Additionally, MobiEdit incorporates mobile hardware-friendly quantization and two optimizations specifically designed for the bp-free training step. These enhancements not only maximize the performance potential of mobile NPUs but also minimize the number of training steps and eliminate redundant computations. The reduction in accuracy occurs because MobiEdit modifies only the value vector of a single MLP layer, a more resource-efficient and lightweight editing approach compared to the multi-layer editing methods employed by MEMIT and AlphaEdit.

System cost. Table 2 provides a detailed comparison of memory, latency, and energy usage across three commercial smartphones on ZsRE and CounterFact datasets. All baseline methods, including ROME, MEMIT, AlphaEdit, and WISE, demand excessive resources—over 46GB of memory because the `llm.c` lacks memory optimization on training part of parameters. And their per-edit energy is ranging from 0.18J to 0.63J. For instance, WISE consumes 0.63J and takes 11,359 seconds on K60 for a single edit. Such workloads not only exceed memory budgets but impose intense thermal and scheduling pressure on mobile hardware.

In practical deployment, this level of energy usage often leads to SoC thermal throttling, causing time-dependent slowdowns and inconsistent performance. Moreover, resource contention causes

Table 2: Performance comparison of our method with NPU and other knowledge editing methods with CPU on different devices.

(a) ZsRE Dataset							
Method	Memory (GB)	K60		K70		OnePlus	
		Time (s)	Energy (J)	Time (s)	Energy (J)	Time (s)	Energy (J)
ROME	46.14	4543.78	0.25	4276.49	0.24	3252.81	0.18
MEMIT	46.14	4543.78	0.25	4276.49	0.24	3252.81	0.18
WISE	46.30	11359.44	0.63	8552.99	0.47	6505.63	0.36
AhphaEdit	46.14	4543.78	0.25	4276.49	0.24	3252.81	0.18
MobiEdit	6.20	1902.88	0.023	1477.67	0.018	1211.83	0.014

(b) CounterFact Dataset							
Method	Memory (GB)	K60		K70		OnePlus	
		Time (s)	Energy (J)	Time (s)	Energy (J)	Time (s)	Energy (J)
ROME	46.14	4416.66	0.24	4156.86	0.23	3161.82	0.17
MEMIT	46.14	4416.66	0.24	4156.86	0.23	3161.82	0.17
WISE	46.30	11041.65	0.61	8313.72	0.46	6323.63	0.35
AhphaEdit	46.14	4416.66	0.24	4156.86	0.23	3161.82	0.17
MobiEdit	6.20	1983.17	0.024	1546.77	0.019	1271.71	0.016

foreground processes (like UI response or system services) to stall during editing tasks lasting over 1.5 to 3 hours, effectively rendering the device unusable during editing.

MobiEdit consumes only 6.2GB memory under 0.03J energy per edit across all devices, completing edits in 1200 to 2000 seconds. This 10× energy reduction allows MobiEdit to run editing workloads unobtrusively in the background without interrupting the user experience or triggering thermal limits. Such sustainability is critical for real-world mobile applications, where knowledge editing may be triggered interactively under tight system constraints.

3.3 Ablation Studies

Figure 6 presents an ablation study of the key algorithmic and system-level optimizations in our framework. The basic zeroth-order method (zo) achieves moderate edit success but incurs excessive time cost, often over 4000 seconds per edit. Introducing early stopping (dynamic step controller) alone reduces average editing time by over 40%, without sacrificing accuracy. The early stopping module effectively eliminates redundant optimization once the target knowledge has been successfully fitted. Adding prefix cache further accelerates editing by another 20–30%, as observed across all devices. For each fact, the prefix cache reduces computation proportionally to the ratio of the prefix length to the total input length. MobiEdit, which incorporates both optimizations, reduces editing time to nearly one-third of the baseline zo and achieves the best balance of edit success and efficiency. By leveraging quantization and NPUs, our approach significantly reduces memory usage and editing time by 7× and 10×, with only a slight reduction in editing success rate. This enables efficient and practical deployment on commercial mobile devices. This significant performance improvement mainly stems from the usage of floating-point calculations exclusively for a small number (less than 1%) zhof accuracy-sensitive editing parameters, while the majority of non-trainable parameters are efficiently processed using low-precision integer computations on the mobile NPU. MobiEdit outperforms the ROME baseline (CPU-based) in terms of speed, underscoring the advantages of quantized, forward-only editing on mobile hardware. These

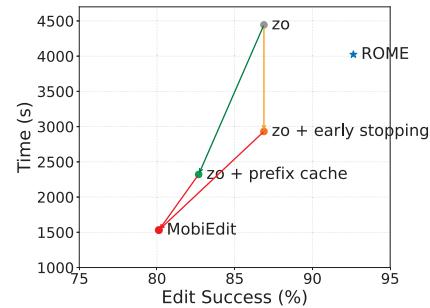


Figure 6: Edit success vs. time on ZsRE. Time is averaged across all devices.

results highlight the necessity and complementarity of each system-level optimization in achieving practical on-device knowledge editing, delivering substantial improvements in both hardware and algorithmic efficiency.

4 Related Work

4.1 Knowledge Editing Methods

Recent work on knowledge editing in language models spans several paradigms. Locate-and-editing methods like ROME[14] and MEMIT[15] directly modify internal weights to update factual knowledge, offering high precision. ROME[14] operates by identifying a “critical layer” and modifying a specific activation vector, achieving high accuracy in single-fact edits with minimal disruption to the model’s behavior. MEMIT[15] extends ROME to support the simultaneous editing of multiple factual associations by modifying activations across several layers. AlphaEdit[7], also based on ROME and MEMIT, introduces null space projection techniques to preserve the model’s original knowledge during editing. WISE[18] augments the FFN module with a dynamic routing mechanism to accommodate new factual knowledge. In contrast, RAG-based methods like RECIPE[3] avoid modifying model parameters by retrieving external knowledge or prompts, allowing efficient and continual updates. MeLLO[22] reveals limitations in generalization under multi-hop reasoning, a challenge for both direct and retrieval-based methods. Finally, meta-learning approaches like MEND[16] learn how to edit from examples, striking a balance between flexibility and generalization but requiring higher setup costs. These approaches trade off between precision, efficiency, and scalability in different ways.

While highly effective in terms of edit success and locality, these approaches depend on multi-step backpropagation, resulting in substantial memory and latency overhead. This makes such methods fundamentally incompatible with mobile deployment, where memory is typically limited to 16GB or less, and compute is constrained to forward-only inference on NPUs. In light of these limitations, we focus on memory-efficient, NPU-friendly, and compatible with quantization editing methods that better align with the resource constraints of mobile hardware.

4.2 BP-Free Methods

Zero-order optimization (ZO) techniques have gained renewed interest as scalable alternatives to backpropagation, particularly in scenarios where gradient computation is expensive. In the context of model tuning, methods such as FwdLLM [21] and MeZO [13] demonstrate that high-quality adaptation can be achieved using purely forward computations, typically through numerical estimation of gradient signals via perturbation. However, they primarily focus on downstream task adaptation, often requiring thousands of adaptation steps, and have not been applied to factual memory injection.

To the best of our knowledge, no prior work enables reliable factual editing under both forward-only and quantized constraints. Our method addresses this gap by introducing a backpropagation-free, quantization-aware framework designed specifically for edge deployment.

5 Conclusion

We have introduced **MobiEdit**, the first mobile-compatible framework for efficient knowledge editing in large language models. By replacing backpropagation with a quantized, forward-only gradient estimation technique, **MobiEdit** aligns with the compute and memory constraints of commercial mobile NPUs. Two further optimizations—prefix cache and early stopping—enhance editing efficiency without sacrificing edit quality. Experimental results demonstrate that **MobiEdit** enables real-time editing of Qwen2.5-3B entirely on-device, reducing memory usage by 7.6 \times , energy consumption by 14.7 \times , and latency by 72% compared to prior methods. These results highlight **MobiEdit** as a promising step toward practical, user-driven LLM personalization in mobile and edge scenarios.

6 Limitations

While **MobiEdit** achieves efficient on-device knowledge editing, it has two main limitations. First, its edit success is lower than gradient-based methods, especially for more challenging or ambiguous knowledge that lacks clear signal from the target output. Second, **MobiEdit** currently supports edits in the form of simple subject–object factual pairs. Complex prompts involving multi-hop reasoning, conditional logic, or indirect references are beyond the scope of the current framework.

Acknowledgments and Disclosure of Funding

Use unnumbered first level headings for the acknowledgments. All acknowledgments go at the end of the paper before the list of references. Moreover, you are required to declare funding (financial activities supporting the submitted work) and competing interests (related financial activities outside the submitted work). More information about this disclosure can be found at: <https://neurips.cc/Conferences/2025/PaperInformation/FundingDisclosure>.

Do **not** include this section in the anonymized submission, only in the final paper. You can use the `ack` environment provided in the style file to automatically hide this section in the anonymized submission.

References

- [1] Apple. Apple intelligence. <https://www.apple.com/apple-intelligence/>, 2024. URL <https://www.apple.com/apple-intelligence/>.
- [2] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.
- [3] Qizhou Chen, Taolin Zhang, Xiaofeng He, Dongyang Li, Chengyu Wang, Longtao Huang, and Hui Xue'. Lifelong knowledge editing for LLMs with retrieval-augmented continuous prompt learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13565–13580, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.751. URL [https://aclanthology.org/2024.emnlp-main.751/](https://aclanthology.org/2024.emnlp-main.751).
- [4] Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.581. URL <https://aclanthology.org/2022.acl-long.581/>.
- [5] John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.
- [6] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '24*, page 6491–6501, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671470. URL <https://doi.org/10.1145/3637528.3671470>.
- [7] Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. Alphaedit: Null-space constrained model editing for language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=HvSytvG3Jh>.
- [8] Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. Model editing harms general abilities of large language models: Regularization to the rescue. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16801–16819. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.934. URL <https://aclanthology.org/2024.emnlp-main.934/>.
- [9] karpathy. llm.c. <https://github.com/karpathy/llm.c>, 2013.
- [10] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In Roger Levy and Lucia Specia, editors, *Proceedings of the 21st*

Conference on Computational Natural Language Learning (CoNLL 2017), pages 333–342, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-1034. URL <https://aclanthology.org/K17-1034/>.

- [11] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. arXiv preprint arXiv:2401.05459, 2024.
- [12] Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D Lane, and Mengwei Xu. Small language models: Survey, measurements, and insights. arXiv preprint arXiv:2409.15790, 2024.
- [13] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In Thirty-seventh Conference on Neural Information Processing Systems, 2023. URL <https://openreview.net/forum?id=Vota6rFhBQ>.
- [14] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. Advances in neural information processing systems, 35:17359–17372, 2022.
- [15] Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In The Eleventh International Conference on Learning Representations, 2023. URL <https://openreview.net/forum?id=MkbcAHTYgyS>.
- [16] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast model editing at scale. CoRR, 2021. URL <https://arxiv.org/pdf/2110.11309.pdf>.
- [17] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.
- [18] Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. Advances in Neural Information Processing Systems, 37:53764–53797, 2024.
- [19] Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. Knowledge editing for large language models: A survey. ACM Comput. Surv., 57(3), November 2024. ISSN 0360-0300. doi: 10.1145/3698590. URL <https://doi.org/10.1145/3698590>.
- [20] Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. Fast on-device llm inference with npus. In Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, pages 445–462, 2025.
- [21] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. {FwdLLM}: Efficient federated finetuning of large language models with perturbed inferences. In 2024 USENIX Annual Technical Conference (USENIX ATC 24), pages 579–596, 2024.
- [22] Zexuan Zhong, Zhengxuan Wu, Christopher Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 15686–15702. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.971. URL <https://aclanthology.org/2023.emnlp-main.971/>.