Efficient and Adaptive Diffusion Model Inference Through Lookup Table on Mobile Devices

Qipeng Wang, Shiqi Jiang, Yifan Yang, Ruiqi Liu, Yuanchun Li, *Member, IEEE*, Ting Cao, and Xuanzhe Liu, *Senior Member, IEEE*

***Abstract*—Diffusion models have revolutionized image synthesis applications. Many studies focus on using approximate computation such as model quantization to reduce inference costs on mobile devices. However, due to their extensive model parameters and autoregressive inference fashion, the overhead of diffusion models remains high, which is challenging for mobile devices to handle. To reduce the inference overhead of diffusion models on mobile devices, we propose `LUT-Diff`, an algorithm-system co-design specifically tailored for mobile device diffusion model inference optimization. `LUT-Diff` optimizes using lookup tables and can efficiently generate a series of lookup table candidates for diffusion models without end-to-end training. During inference, `LUT-Diff` adaptively selects the best inference strategy based on the application/user's latency budget. Additionally, `LUT-Diff` includes a parallel inference engine that rapidly completes model inference through CPU-GPU co-scheduling. Extensive experiments demonstrate that `LUT-Diff` can generate images comparable to the original model, with an up to 0.012 MSE in generated images. `LUT-Diff` can also achieve up to 9.1× inference acceleration and reduce the inference memory footprint by up to 70.9% compared to baseline methods. Moreover, `LUT-Diff` can save at least 3281× the learning cost of lookup tables.**

***Index Terms*—Deep learning, diffusion model, lookup table.**

I. Introduction

DIFFUSION models [1], [2], [3] have achieved remarkable success in image synthesis, vital for a spectrum of mobile applications from real-time photo editing to personalized content creation. The need to deploy these advanced models directly on mobile devices is increasingly critical, driven by the demand for instant processing, enhanced user privacy, personalized digital experiences, and improved functionality of mobile applications by enabling sophisticated image generation capabilities on the go [4], [5], [6]. However, adapting diffusion models for mobile deployment presents significant challenges due to their extensive resource demands. To address these issues, researchers have explored several approximation computation techniques, such as model quantization [7], [8], [9], [10], by using low-precision data to approximate floating-point computations, which trades accuracy for improved inference efficiency.

Despite these advancements in approximation methods, such as quantizing models to int8 precision [7], [9], diffusion models remain exceedingly burdensome for mobile devices for two main reasons. First, diffusion models are typically large, often reaching gigabyte scale, with substantial memory requirements. Even with 8-bit quantization, the memory demands are heavy for mobile devices. Second, diffusion models function autoregressively by performing image denoising over multiple iterations, culminating in an escalated computational burden on mobile devices.

In this paper, we explore the potential for employing more aggressive approximation computation methods to further reduce inference overhead on mobile devices. Specifically, we intend to utilize the lookup table [11] to supplant the linear operations in diffusion models, such as convolution that can be converted to matrix multiplication (MM). Lookup table utilizes precomputed values stored in a table to quickly approximate results. A typical implementation of lookup table is based on product quantization (PQ) [12], [13], involving partitioning a high-dimensional vector into lower-dimensional subvectors, which are then independently encoded by discrete values called centroids. In PQ-based lookup table, the input is divided into multiple subvectors, whereupon each subvector is encoded through centroids. The products of centroids and weights could be precomputed and stored in a table. During inference, approximated results are retrieved without explicit computations, significantly accelerating inference and reducing memory footprints by adjusting lookup configurations, i.e., the number of centroids K and their lengths V.

We pinpoint a unique opportunity to apply lookup to diffusion models. Specifically, during the diffusion models inference, certain level of Gaussian noise is incorporated into the image at each denoising step [1] to ensure the diversity of the generated images. Intrinsically, diffusion models are tolerant to input noise under certain levels. We evaluated this tolerance in Section II.

However, applying lookup tables to reduce on-device diffusion model inference overhead faces two major challenges. The

Received 4 August 2024; revised 28 December 2024; accepted 31 March 2025. Date of publication 4 April 2025; date of current version 6 August 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62325201 and Grant 62272261 and in part by Tsinghua University (AIR)–AsiaInfo Technologies (China) Inc. Joint Research Center. Recommended for acceptance by A. A. Nayak. *(Corresponding authors: Xuanzhe Liu; Shiqi Jiang.)*

Qipeng Wang and Xuanzhe Liu are with the Key Laboratory of High Confidence Software Technologies, Ministry of Education, School of Computer Science, Peking University, Beijing 100871, China (e-mail: wangqipeng@stu.pku.edu.cn; liuxuanzhe@pku.edu.cn).

Shiqi Jiang, Yifan Yang, and Ting Cao are with the Microsoft Research, Redmond, WA 98052 USA (e-mail: shijiang@microsoft.com; yifanyang@microsoft.com; ting.cao@microsoft.com).

Ruiqi Liu is with the School of Software and Microelectronics, Peking University, Beijing 100871, China (e-mail: rich@stu.pku.edu.cn).

Yuanchun Li is with the Institute for AI Industry Research (AIR), Tsinghua University, Beijing 100190, China (e-mail: liyuanchun@air.tsinghua.edu.cn).

Digital Object Identifier 10.1109/TMC.2025.3558203

first challenge is how to obtain the tables efficiently. According to our preliminaries (Section II), directly applying lookup tables with uniform lookup configuration, i.e., subvector length V and centroid number K, introduces unacceptable errors. While using a finer-grained lookup table can lead to higher accuracy, it compromises acceleration. Efficiently obtaining lookup tables encompasses two issues: (1) Different granularities of configuration impact the accuracy of the lookup table and the gains in acceleration, and uniform lookup configurations misalign with varying informational content and volume across input features, and may have a dependency on others, such as RoI in an image [14]. Therefore, a finer-grained configuration is needed to trade accuracy and inference effenciency. Searching for optimal configurations exponentially increases the search space. (2) Learning objectives between centroid optimization misalign model training goals; the former typically optimizes the Euclid distance between inputs and centroids through clustering, while the latter optimizes the loss between model outputs and ground truth, which impacts table accuracy. While researchers proposed to reduce errors through end-to-end model training [13], it is impractical for diffusion models, requiring tens of thousands of GPU hours.

The second challenge is efficiently managing lookup table optimized inference on mobile devices, including two issues: (1) Table lookup introduces additional memory access, necessitating the full utilization of mobile hardware capacity to achieve efficient inference. (2) The system should meet diverse user demands for varying inference speeds while balancing generation quality. Specifically, better images should be generated when given a latency budget threshold. Latency budget refers to a specified time within which the model should generate an image or complete a single iteration of inference, such as 100 ms/iteration or 1 s/image, which is specified by the user.

Our design: We propose `LUT-Diff`, the first algorithm-system co-design that efficiently deploys diffusion models on mobile devices through lookup tables. `LUT-Diff` can quickly generate multiple lookup table candidates for each model layer, each lookup table candidate corresponds to varying performance. During the inference phase, `LUT-Diff` selects the optimal model inference strategy based on a given latency budget, choosing the appropriate lookup table. `LUT-Diff` leverages both the CPU and GPU on mobile devices for parallel model execution to fully utilize hardware capabilities, to accelerate on-device diffusion model inference. Specifically, `LUT-Diff` involves two techniques:

Efficient lookup table learning consists of two components: (1) *Layerwise lookup configuration searching*. To obtain finer-grained configurations and reduce the exponentially huge search space, we employ a layerwise lookup configuration search method to search the lookup configuration for each layer progressively. Our heuristic is that V and K should not be uniform to align with feature information, particularly that V should align with dependencies among features, hence we start with searching for V. During the search, we use the Fisher Information Matrix (FIM) to assess the quality of centroids, which better reflects the relationship between changes in a layer and the model output [15]. `LUT-Diff` progressively replaces the very beginning feature columns of input with centroids and selects the optimal V until all input columns are replaced. After determining V, `LUT-Diff` selects the same K for centroids with the same length and estimates all possibilities and corresponding FIM errors. Finally, `LUT-Diff` generates a set of lookup configuration candidates for each layer with different qualities. (2) *Weighted centroid learning*. To address the mismatch between the learning objectives of existing centroid clustering methods and model training optimization, `LUT-Diff` involves a weighted centroid learning method. This approach transforms the data space and learns the centroids in the output space, thereby reducing errors. To further reduce the memory footprint, `LUT-Diff` uses lower precision representation to encode the centroids due to the limited centroids number.

Adaptive parallel inference engine also includes two aspects: (1) *parallel inference execution engine*. To facilitate efficient on-device lookup table-based inference, `LUT-Diff` involves a parallel inference execution engine through CPU-GPU co-scheduling. For table lookup, we designed a grouped lookup strategy, grouping data according to the parallelism of hardware SIMD instructions [16]. Grouped lookup ensures that values within a group can be processed within a single instruction cycle. After the grouped lookup is finished, `LUT-Diff` merges the results of all groups. Grouped lookup and result merge are implemented through "shuffle" and "bitwise select" hardware instructions, respectively. To fully utilize hardware capacity, `LUT-Diff` partition the data according to the latency on CPU and GPU to accelerate the inference. The partition strategy is determined by minimizing the latency differences between the two types of hardware. The execution of different data partitions is independent, without introducing intra-operation synchronization overhead. (2) *Adaptive inference strategy selection*. Given a latency budget, `LUT-Diff` selects the optimal model-level inference strategy to achieve efficient on-device inference, i.e., choosing the lookup configuration for each layer to generate better images while meeting the target latency budget requirement. To address this, we first modeled the inference acceleration brought by the lookup table based on the profiled system performance. Then, we modeled the inference strategy search problem as a knapsack problem and used a dynamic programming approach to select the optimal model-level inference strategy under the given latency budget.

Implementation and evaluation: `LUT-Diff` is implemented based on the official latent diffusion codebase [17]. We deployed the models on 4 phones using the mobile-dedicated framework MNN [18]. We conducted extensive experiments on four models and compared `LUT-Diff` with different baselines, verifying the quality of generated images and the inference performance. The experiments show that images generated by `LUT-Diff` are comparable to the original model, with an average $4.85e^{-3}$ MSE for an image, and an average 0.48 FID increase compared with the original model. At the same time, `LUT-Diff` has higher inference efficiency, achieving up to $9.1\times$ inference acceleration compared to the baseline method. `LUT-Diff` can achieve more than $4\times$ acceleration for all models and devices, where $4\times$ acceleration is the upper bound of int8 quantization. `LUT-Diff` can achieve up to 80.2% inference memory footprint saving. In

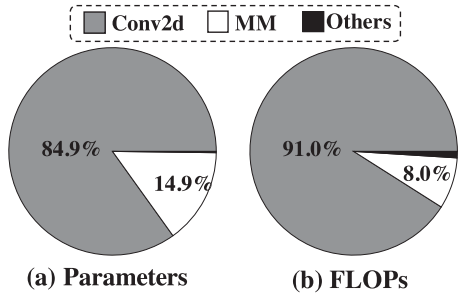

Fig. 1. Params. and FLOPs percentage of CelebA model.

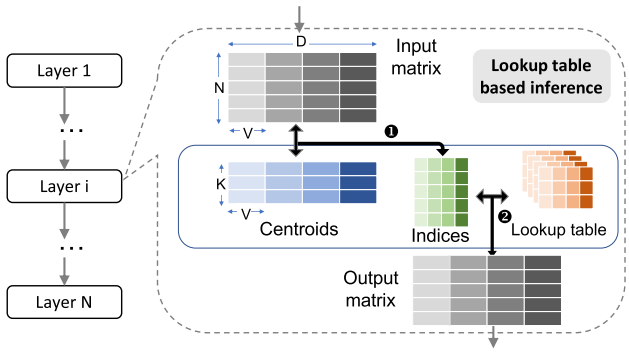

Fig. 2. Lookup table-based model inference.

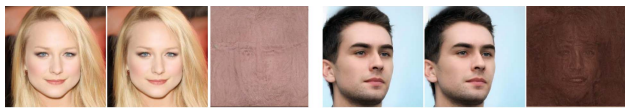

Fig. 3. Generated Images of the original Celeba-HQ model, lookup table-based with K = 256 and V = 1, and vanilla lookup table-based.

addition, LUT-Diff has higher learning efficiency, improving by at least 3281× compared to the baseline method, because of LUT-Diff 's efficient learning design.

Contributions of this paper are summarized as follows:

- We introduce LUT-Diff, the first framework designed for efficient inference of diffusion models on mobile devices, aimed at reducing overhead on mobile devices.
- LUT-Diff employs a co-design approach of algorithm and system integration. Specifically, LUT-Diff is the first to apply training-free lookup table learning, enabling efficient centroid learning and lookup configuration search. LUT-Diff integrates the first lookup table-based parallel inference engine, fully utilizing mobile hardware capacity to support efficient and adaptive diffusion model inference.
- We have implemented LUT-Diff and validated it on 4 commodity devices. Experiments demonstrate that LUT-Diff can generate comparable images and significantly reduce inference costs with minimal error trade-offs.

II. Background and Preliminaries

A. Diffusion Model

The diffusion models have been emerging for image generation.[1] The inference process of diffusion models involves autoregressively denoising a noisy image over multiple iterations, to refine the image towards a high-quality output. The diffusion models often involve U-Net [19] to estimate the noise in the input image and apply a sampling scheduler, like DDPM [1], to eliminate the noise, iteratively. However, diffusion models are notably large in terms of parameters [2]. Their autoregressive nature and the numerous FLOPs during their inference make them computation-intensive. Moreover, substantial memory is required for inference, which may be challenging for mobile devices with limited capacity. These traits highlight the need for efficient inference optimization in diffusion models on mobile devices [20], [21].

We conduct a breakdown analysis of the parameters and FLOPs percentage of CelebA-HQ [2], [22] model. The model size is 2303 MB. The results are shown in Fig. 1. We find that the linear operations, i.e., convolution and MM, account for most parameters and FLOPs. Such a result motivates us to optimize the performance of these operations in the diffusion model. Moreover, the default DDPM requires 1000 iterations for generation, and even optimized schedulers need tens of iterations to complete [23], making the overhead of diffusion model inference on mobile devices substantial.

B. Lookup Table in Model Inference

In model inference, lookup tables are commonly implemented via PQ to approximate MM. PQ is a data compression method that divides data into smaller, manageable subvectors and quantizes each independently using centroids of the same dimension. Fig. 2 illustrates the inference process using lookup tables in a linearly stacked model. The model inference is completed by matching the input with centroids (❶) and indexing the precomputed results (❷). In a PQ-based lookup table, an input matrix with dimensions $N \times D$ is divided into multiple subvectors each of length V, with each group of subvectors having dimensions $N \times V$. Each group of subvectors is then approximated and encoded using individual K centroids. The results of centroids and corresponding weight are precomputed. Therefore, for the encoded input matrix, the precomputed results can be directly used, replacing extensive computational MM operations with table lookups. The centroids are commonly obtained through clustering algorithms, such as KMeams.

C. Challenges and Opportunities

Challenges: We conduct a preliminary experiment on the diffusion model to explore the effectiveness of the vanilla lookup table [11], [24] and the generated images are shown in Fig. 3. We use the model trained on the CelebA-HQ dataset and apply the lookup table on all MMs and convolutions, which is converted to MM through *im2col*. We use the same hyperparameters as

[1] Many other diffusion models focus on other tasks, like audio synthesis. In this paper, we focus on image generation models.

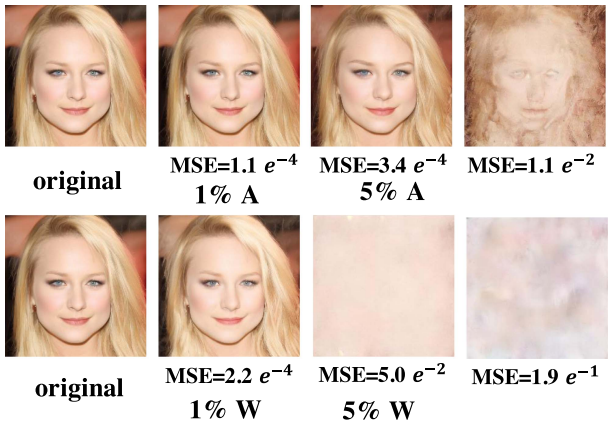

Fig. 4. Diffusion model is tolerant to noise. We add "$X\%$" noise ($X\% \cdot \mathcal{N}(\mathbf{0}, \mathbf{1})$) to weight ("W") and activation ("A"). $\mathcal{N}(\mathbf{0}, \mathbf{1})$ denotes standard Gaussian noise.

previous work, i.e., $K = 16$, and input is split into 64 subvectors. We find that the vanilla lookup table introduces non-trivial error and influences the generated image negatively. This is because the vanilla lookup table overlooks the correlations between data and the amount of information on each feature. Furthermore, the learning objectives for centroids in the lookup table differ from the optimization goals of model training, where the former optimizes the error between inputs and centroids in the input space, while the latter focuses on minimizing the error between model outputs and the ground truth in the output space, thereby introducing discrepancies. We also found that lookup tables with V = 1 and K = 256 (equivalent to int8 quantization) can generate high-quality images. Based on these results analysis, we conclude that the learning process of lookup tables should be optimized, considering more granular lookup configurations and accurate tables, to trade accuracy and acceleration.

Opportunities: Diffusion models are performed by eliminating noise in images. During denoising, the model adds random noise to each generated image for diversity [1] at each timestamp. This makes diffusion models naturally tolerant to noise in input. We show the changes in the image after adding a certain amount of random noise to each model layer in Fig. 4. We find that the diffusion model is more tolerant to input noise than weight noise. When we added 1% and 5% noise to each activation, the model could still generate reasonably good images, and the model only failed when the noise level reached 10% . However, when we added 5% noise to the weights, the model could no longer generate recognizable images. These results indicate that a certain level of noise can be tolerated, providing an opportunity to apply lookup tables in diffusion models for inference optimization.

III. LUT-Diff Overview

LUT-Diff aims to utilize lookup table to optimize the diffusion model inference on the mobile device. Specifically, given a latency threshold, LUT-Diff can automatically select the optimal inference strategy on a certain device. As illustrated in Fig. 5, LUT-Diff consists of 2 stages, i.e., offline stage

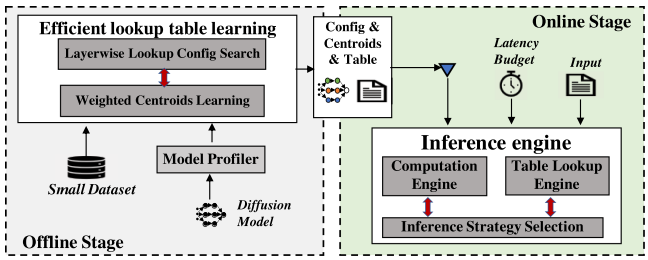

Fig. 5. LUT-Diff Architecture.

for efficient lookup table learning and online stage for adaptive inference execution.

Offline stage: In this stage, LUT-Diff learns the centroids for each layer and searches for the strategies needed for the online stage. Specifically, LUT-Diff includes the following steps. **(1)** First, LUT-Diff gets the layers where the lookup table can be applied through a "model profiler". **(2)** Then, LUT-Diff initialize the centroids through *weighted centroid learning (Section IV-A)* on a small dataset. Although the dataset size is much smaller than the whole training dataset, it is enough for clustering. **(3)** After centroids are initialized, LUT-Diff applies *layerwise lookup configuration searching (Section IV-B)*. Our insight is that it is necessary to use more granular lookup configurations, i.e. K and V, according to our preliminaries in Section II. Therefore, even for the same layer, the choice of K and V is not fixed. LUT-Diff searches for lookup configurations for each layer of the model. After completing the search for the lookup configuration, each obtained configuration corresponds to a different error, a set of centroids, and a lookup table.

Online stage: There are two key techniques in this stage. **(1)** To support efficient lookup table-based on-device diffusion model inference, LUT-Diff includes a *parallel inference execution engine (Section V-A)* that can quickly perform table lookup operations to complete the model inference. Specifically, when the inference strategy is specified given a latency threshold, LUT-Diff loads the learned centroids and completes the inference. During the inference process, LUT-Diff uses both CPU and GPU to perform inference in parallel to achieve inference acceleration by utilizing advanced *shuffle* and *bitwise select* SIMD instructions. **(2)** LUT-Diff includes an *adaptive inference strategy selection (Section V-B)* to determine the optimal inference strategy under a given latency budget. LUT-Diff first models the inference latency based on lookup tables, constructing the acceleration benefits brought by different configurations' corresponding lookup tables. Then, LUT-Diff selects a configuration and its corresponding lookup table for each layer to ensure the best model-level performance. Similar to the offline stage, we use FIM to evaluate the model-level quality. LUT-Diff models the aforementioned selection problem as a knapsack problem and solves it using dynamic programming to minimize FIM error while meeting the latency budget threshold.

IV. Offline Lookup Table Learning

In this section, we provide a detailed explanation of the offline stage of LUT-Diff about how it learns the lookup table. This

stage primarily involves two techniques: *Weighted Centroid Learning (Section IV-A)* and *Layerwise Lookup Configuration Searching (Section IV-B)*.

A. Weighted Centroid Learning

Existing methods for learning centroids mainly include two categories. The first is unsupervised learning, primarily represented by the KMeans method, which determines the Euclid distance between the input and centroids using euclidean distance or cosine similarity and clusters the centroids iteratively. This method is computation-efficient, as it only requires input data to obtain the centroids. However, this method introduces non-trivial error because its optimization goal is different from the model's, ultimately affecting the model's performance. To address this issue, researchers have proposed supervised centroid learning [13], which integrates clustering into end-to-end model training to reduce error. However, this requires backward propagation through the entire model, which is computationally inefficient. On the other hand, the end-to-end training overhead of the diffusion model is not affordable, which requires tens of thousands of GPU hours.

To address these problems, we propose a weighted centroid learning method to mitigate the large errors inherent in existing unsupervised methods and to avoid the inefficiencies of supervised methods. This method remains unsupervised but incorporates weights into the learning process. The key advantage of using a weighted approach is that it aligns the objectives of unsupervised learning more closely with the goals of model optimization by adjusting the importance assigned to different features based on their impact on the output. For instance, in MM, vectors that may be distant in input space could converge in output space after transformation, and vice versa. By applying weights to these vectors based on their transformational impact, the learning of centroids can more accurately represent their effect on the final output. Additionally, the weighted method enables efficient centroid learning, eliminating the extensive end-to-end training overhead traditionally required.

$$\underset{\{c_i\}_{i=1}^{K}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} \|x \otimes W - c_i \otimes W\|^2 \tag{1}$$

As formalized in Formula 1 ($\otimes$ denotes MM; c_i is the centroid of cluster S_i), we add the weight matrix W to the original method and use the distance between vectors in the output space to learn the centroids, reducing the error between outputs. Similar to the classic KMeans method, the goal of Formula 1 is to learn centroids $C = \{c_i\}_{i=1}^{K}$ such that the distance between the input and centroids in the output space is minimized. The rest of the learning process is consistent with the KMeans algorithm.

During the learning process, we used a uniform sampled small dataset, significantly smaller than the original training set. This dataset can be generated in either of the following ways: by sampling from the complete training set, then processing as per the training procedure—adding random noise to the images and using a diffusion model to predict the noise while preserving the activations; or by randomly generating images and saving the activations produced during the inference. Since weighted centroid learning focuses on the distance relationships in the output space among vectors through clustering, a small dataset is sufficient to meet the algorithm's requirements. As for the text-conditional model and dataset, we learn the centroids for each class separately, where the class of the image refers to the subject of the image. This is because there are heterogeneous classes in the dataset and different content contains different subjects and features. In the learning process, the learning of each layer is independent. After obtaining the input x for each layer, we use the pre-trained weights W of the model to perform weighted learning of the centroids.

Finally, for each subvector set, we learn K centroids. Therefore, for all the centroids of a subvector set, there are at most K values in a column, indicating that the centroids can be quantized. Specifically, we apply asymmetric quantization [25], [26] to encode the centroids with $log_2 K$ bit integer, to further reduce the memory footprint of the centroids.

B. Layerwise Lookup Configuration Searching

1) Heuristic: Based on the preliminaries in Section II, our heuristic is that by using the appropriate lookup configuration, i.e., the centroid number K and the subvector length V, the lookup table can achieve good inference results. The question then is how to determine V and K. Because there are RoIs in the activations, our insight is that the configuration should align with them to be more accurate. The convolution operation can be transformed to MM through the *im2col* operation, which unfolds the input of the convolution into a matrix, where each region of the input is unfolded to a row vector. Therefore, the subvector should align with it and we search the subvector length first. Assuming the shape of the convolution input image is $[B, C_{in}, H_{in}, W_{in}]$,[2] where B represents the batch, C_{in} represents the input channel, H_{in} and W_{in} represent the width and height of the input image, respectively. After applying *im2col*, the input image is transformed into a matrix of shape $[B, H_{out} * W_{out}, L]$, where $H_{out} * W_{out}$ is the number of pixels in the output image, and L is the input feature dimension, related to the size of the convolution kernel, i.e., the region size. Specifically, $L = C_{in} * Kh_{in} * Kw_{in}$, where Kh_{in} and Kw_{in} is the size of the convolution kernel [27]. We utilize the shape information of the current layer's input and output during the search for V. Specifically, we divide adjacent areas of the image into the same subvector as much as possible to fully utilize the information of image locality. For example, for a common 3x3 2-D convolution kernel, choosing V=3 is more conducive to the utilization of locality than V=4, because V=3 aligns with one row of the convolution kernel in a channel; and V=9 is consistent with the size of a convolution kernel channel.

To implement adaptive inference, we need to obtain multiple configurations for each layer. Since subvector length V should correspond with the RoI, and in weighted centroids learning, more centroids will not lead to worse results, we will determine an optimal set of V for each layer and then explore multiple sets of K as configuration candidates.

[2] We focus on the Conv2D operation, which is commonly used in diffusion models for image generation.

Algorithm 1: V Configuration Search.

Input: *layer*, *Vcandidates*, *data*, *K*
Output: VConfiguration
1 Initialize centroids C_{init} with each v_i and K;
2 Convert *block* to `MatMul`; `// Convert Conv via im2col`
3 *VConfiguration* ← Dict();
4 **Function** `infer(`$vconfig_{tmp}$`)`**:**
5 | $L \leftarrow$ sum($vconfig_{tmp}$);
6 | Infer first L columns through table lookup;
7 | Infer remaining columns with original computation;
8 *Vconfig* ← [];
9 **while** sum(*Vconfig*) ¡ *layer.input.Nin* **do**
10 | **foreach** v_i in *Vcandidates* **do**
11 | | $Vconfig_{tmp} \leftarrow$ *Vconfig* + [v_i];
12 | | `Infer`($Vconfig_{tmp}$);
13 | | $fim_{v_i} \leftarrow$ FIM(v_i); `// calculate FIM for` v_i
14 | **end foreach**
15 | *Vconfig*.append($\arg\min_{v_i} fim$);
16 **end while**
17 *VConfiguration*[*layer*] ← *Vconfig*;

Algorithm 2: K Configuration Search.

Input: *layer*, *Kcandidates*, *data*, *VConfiguration*
Output: KConfiguration
1 Initialize centroids C_{init} with *VConfiguration* each k_i;
2 Convert *block* to `MatMul`; `// Convert Conv via im2col`
3 *KConfiguration* ← Dict(default=Dict());
4 **Function** `permute()`**:**
5 | *seq* ← [];
6 | **foreach** v_i in *layer* **do**
7 | | *seq*.append(*layer*.v_i);
8 | **end foreach**
9 | *permuted_kconfig* ← permute each k_p to each item in *seq*;
10 | **return** *permuted_kconfig*
11 **foreach** *kconfig* in `permute()` **do**
12 | infer *layer* with *kconfig* and *VConfiguration*;
13 | *KConfiguration*[*kconfig*] ← FIM(*kconfig*);
| `// calculate FIM error of kconfig`
14 **end foreach**

2) Searching for V: The first question is how to determine the subvector length V for the current layer. There are two challenges here. First, the error of a certain layer of the model does not reflect the overall error of the model, so how determining the impact of the current V on the whole model is a challenge. Second, since the amount of information contained in different regions in the image is different, V should be non-uniform to maximize the optimization brought by the lookup table. Because a smaller V means finer granularity of centroids, but at the same time, the corresponding acceleration will also be lower; vice versa [28]. The non-uniform V means a huge search space, and how to search for the appropriate V configuration in this huge space is a challenge.

For the first challenge, we used the second-order error FIM [15]. Compared to absolute errors like MSE, the second-order error FIM can better reflect the relationship between the changes in a model layer and the global changes in the model because the FIM quantifies how sensitive the model's predictions are to changes in each parameter [29], [30], meaning that this metric can estimate the impact of a parameter on the model's output, indicating which parameters are most critical for the network's performance. In lookup table-based model inference, the centroids can be treated as parameters. Therefore, we use FIM to address the first challenge.

For the second challenge, we perform a progressive search. Due to the model containing numerous layers, searching for subvector lengths for the entire model would result in an extremely large search space. Even for individual layers, if an exhaustive search method is used, the cost remains exponential. Therefore, we adopted a progressive replacement strategy, as detailed in the Algorithm 1. In the search process, we first initialize centroids according to Section IV-A based on the candidate set of V. In the initialization process, we chose a sufficiently large K for initialization to avoid the impact of a smaller centroid number (we used 4096 in the experiment). For each candidate $V = v_i$, we start from the first column of the input matrix, divide the first v_i columns into a subvector, perform inference of this part through a table lookup, and keep the rest unchanged (Line 10-12). After calculating the FIM of all candidate $V = v_i$ (Line 13), we can obtain the optimal length of the subvector at the current position (Line 15). The second loop starts at the position where the previous loop ended, continues to complete the subsequent replacement, and determines the lengths of the subsequent subvectors. Using such a progressive replacement method, the search for the V configuration of a layer can be completed with linear complexity. After completing the search process, the optimal subvector length can be obtained.

3) Searching for K: After completing the search for V, we can obtain a set of optimal subvector length configurations for the layer. Then, based on the determined V, we evaluate the impact of different choices of K on the overall performance of the layer, as shown in Algorithm 2. The algorithm first initializes the centroids based on the determined V (*VConfiguration*) and the candidate set of K (*Kcandidates*). For each subvector with length $V = v_i$, we permute $K = k_p$ from the candidate values and calculate the FIM error (Lines 11-13). It is important to note that when choosing K, we select the same K for subvectors of the same length in each layer to ensure parallelism in subsequent inference (Lines 6-9). Ultimately, Algorithm 2 will produce a set of configurations that includes the different choices of K for the layer and the corresponding FIM errors.

However, during the search process, we found that there are always some subvectors whose FIM only decreases when K is sufficiently large. These subvectors account for about 2% out of the total. When selecting K the same as other subvectors with the same length, their FIM error is significantly higher than that of other subvectors. Fig. 6 shows the CDF of FIM errors in different layers of the model trained on the CelebA-HQ dataset. These subvectors have a significant impact on the final model's performance. Therefore, for these subvectors, we choose not to replace them with lookup tables, but to retain the original computation.

For text-conditional models, we search the K and V for each class. Ultimately, Algorithm 1 and Algorithm 2 will generate

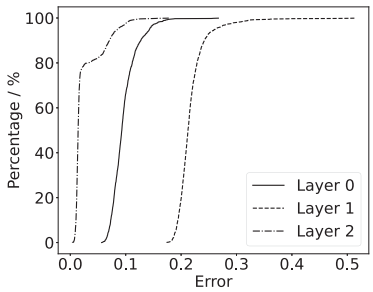

Fig. 6. Error CDF of 3 layers of CelebA-HQ model.

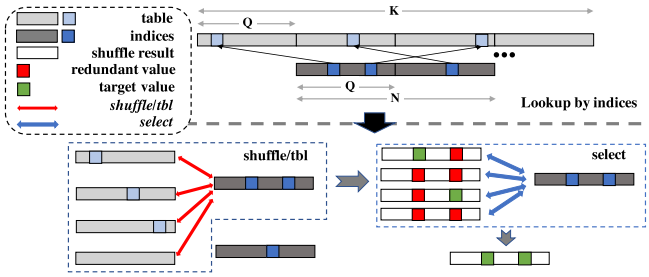

Fig. 7. Table lookup in the inference engine.

a series of lookup configurations, as well as the corresponding centroids, where V is fixed but K varies. Different configurations correspond to different errors. The lookup tables are also computed during the search process based on the centroids and the model weights.

V. Online Inference

In this section, we detail the online stage of `LUT-Diff`, explaining how it achieves efficient inference under varying latency budget constraints. There are two major techniques: the *Parallel Inference Execution Engine (Section V-A)* and *Adaptive Inference Strategy Selection (Section V-B)*.

A. Parallel Inference Execution Engine

We design an efficient inference engine that can efficiently implement parallel inference through CPU-GPU co-scheduling, to achieve optimal inference performance. Because utilizing both types of hardware can definitely achieve higher inference performance. Overall, we split the data between CPU and GPU, and the data partitioning in `LUT-Diff` is based on hardware capabilities, specifically the processing speeds of the CPU and GPU. Our goal is to balance the workload between the two hardware components as evenly as possible for the same operation. Since there are no dependencies or conflicts between the different features of the input data for the above three operations, they can be fully parallelized. Balancing the workload between the two hardware components minimizes the waiting overhead between hardware.

For a table lookup operation, we split it into CPU and GPU to fully utilize the device capacity. Specifically, the computation of subvectors with exceptionally large FIM errors (described in Section IV-B), the distance computation, and the remaining lookup, are split into CPU and GPU. These three operations are finished in sequence. To utilize both CPU and GPU simultaneously for executing an operator, we split the input data between the two hardware units. Since the different features in either operation are computationally independent, parallel execution on both types of hardware is feasible without the need to consider intra-operation synchronization issues during computation. We partition the input data to ensure the lowest inference latency for the current layer according to Formula 2, where $T_{CPU:OP}/T_{GPU:OP}$ denotes the latency of the certain operation on the certain hardware, such as MM and table lookup. The optimization goal is to minimize the latency gap between CPU and GPU. This is because of the varying lookup configurations across layers, making it challenging to identify an optimal model-level split strategy that minimizes end-to-end latency. Nevertheless, our optimization objective ensures that there is seamless synchronization between the CPU and GPU, thereby also reducing the overall latency to the greatest extent possible. The latency of each operation can be obtained according to our modeling method detailed in Section V-B. Finally, the optimal partition D_{CPU}/D_{GPU} indicates that in the matrix with input feature length D, subvectors with total length D_{XPU} are inferred on the corresponding hardware. $\sum_i^{n_{bad}} v_i$ denotes all sub-vectors whose FIM is extremely larger than others (Fig. 6). During the selection of subvectors, we prefer to place subvectors of the same length on the same hardware to maximize the parallelism. Specifically, we start from the initial features of the data and prioritize selecting the shortest subvectors, assigning them to the CPU. If the CPU's data allocation requirement D_{CPU} is not met at this point, we proceed to select the next shortest subvectors from the beginning of the features until D_{CPU} is satisfied. The remaining data is then allocated to the GPU.

$$D_{CPU}, D_{GPU} = \underset{D_{CPU}, D_{GPU}}{\arg\min} \; ||T_{CPU:OP} - T_{GPU:OP})||$$

$$D = D_{CPU} + D_{GPU} + \sum_i^{n_{bad}} v_i \tag{2}$$

We implement efficient parallel table lookup operations on both ARM CPU and GPU through "shuffle". Specifically, on ARM CPUs, we use the *TBL* instruction; on mobile GPUs, we use the *shuffle* function in OpenCL. The shuffle operation can take two fixed-length vectors as input (determined by the SIMD instruction length), representing the source data and indices, and can quickly swap multiple data points within the time of a single instruction. Since the number of lookups per instruction is limited, we design a grouped table lookup operation, as illustrated in Fig. 7. For example, if each table lookup operation can only look up Q numbers, the K centroids need to be grouped every Q centroids, and each group is looked up separately to form the result. After lookup operations, the results are merged through the "bitwise select" operation, which is implemented by leveraging the *select* function in OpenCL, and the *BSL* instruction on ARM CPU. This instruction/function can accept two fixed-length vectors as parameters: the source data and a mask. It then selects bits from the source data based on whether

each bit in the mask is set to 1. As illustrated in the figure, during the bitwise select operation, since each index selects only one target data, each final result value definitely and exclusively comes from the output corresponding to a specific group. After the lookup groups are defined, the mask can be calculated, which is then used in the bitwise select operation. Additionally, when implementing lookup operations on each hardware, we fully considered data access locality, including placing the index of shuffle operation in the inner loop to avoid redundant data loads and pre-transposing data to store it by grouped dimensions first, ensuring that the accessed data during loading is contiguous.

For other operations, we also split them into CPU and GPU according to the hardware capacity, and the goal is also to minimize the computation latency gap between the two types of hardware. Since other operations, such as distance computation through MM, do not have dependencies between different features during computation, it is also feasible to divide these operations between the CPU and GPU without the need to consider the effects of intra-operation synchronization. For these input data, we prioritize allocating the first D_{CPU} features to the CPU and then assign the remaining features to the GPU. Ultimately, the system can achieve efficient diffusion model inference through the parallel inference execution engine.

B. Adaptive Inference Strategy Selection

During the model inference in the online stage, different configurations correspond to different FIM errors and also different latency budgets. Since the model contains many layers, each with many different configurations, choosing the configurations is challenging.

$$\sum N \cdot v_i \cdot k_i + \frac{N}{Q} \cdot \frac{\sum k_i}{Q} \cdot M \cdot E \ll N \cdot D \cdot M \tag{3}$$

$$FIM^i_{LAT_j} = \{FIM^{i-1}_{LAT_j - lat_i} + fim_i | 1 \le i \le n\}_{\min} \tag{4}$$

We first modeled the latency of the system under different lookup configurations. Since the model is inferred layer by layer, it is only necessary to analyze the latency of one layer and then extrapolate to the latency of the entire model. We assume that the input with feature-length D is divided into n subvectors, with lengths $\{v_1, v_2, \ldots, v_n\}$, and the number of centroids for each sub-sector is $\{k_1, k_2, \ldots, k_n\}$. If the original model's input dimension is $N * D$ and the weight dimension is $D * M$, to ensure that the lookup table technique can bring acceleration, Formula 3 needs to be satisfied, where E denotes the normalized efficiency of lookup of a single value, and Q denotes the data length of SIMD instructions. For example, for 128-bit Neon instructions, if the data type operated on is int8, then $Q = 16$ (128/8=16). Lookup efficiency E can be profiled in Section V-A. In the formula, the first term on the left side represents the complexity of calculating the distance between the input data and the centroids; the second term represents the complexity of table lookup, and the right side represents the complexity of the original MM. Specifically, $\frac{N}{Q}$ indicates that we need to group the indices into $\frac{N}{Q}$ groups and perform the table lookup operation for each group; $\frac{\sum k_i}{Q}$ indicates that for the ith subvector set, we need to perform the lookup instruction for $\frac{k_i}{Q}$ times. Constrained by Formula 3, we can obtain the latency corresponding to each configuration on the target device.

Given the latency budget, the strategy selection is modeled as the problem of choosing the best combination of lookup configurations under the given budget. We model this problem as a knapsack problem, i.e., choosing a configuration for each layer so that the final FIM error is minimized and the latency requirement is satisfied. Since the number of layers in the diffusion model is constant, we use the classical dynamic programming algorithm, as formulated in Formula 4. In the formula, we calculate the minimal FIM error $FIM^i_{LAT_j}$ of computing i layers under latency LAT_j. lat_i and fim_i denote the latency and the FIM error of the ith layer, respectively. Because the FIM error can reflect the changes in the whole model, we used the added FIM error of each layer as the final error. After traversing all feasible combinations of configurations, we select a set of optimal configurations for the target latency budget with minimal FIM error as the inference strategy.

During the inference process, `LUT-Diff` selects the optimal inference strategy according to the given latency budget, loads the corresponding centroids and lookup tables, and performs inference. For the text-conditional models, the selection and loading are performed after the text prompt is processed by the text encoder. Specifically, `LUT-Diff` uses the text encoder as a "router". After the prompt is processed by the encoder, the subject of the image can be obtained according to the output. We select the optimal inference strategies of the corresponding subject. Then, `LUT-Diff` loads the required tables for inference. This is because of the heterogenous classes, requiring an "on-the-fly" loading manner. In case the class does not have learned tables, `LUT-Diff` just uses naive int8 quantization for inference and learns the needed table, and then the lookup table can be used in the subsequent image generation.

VI. Implementation

We implemented K-V configuration search, centroid learning, and inference strategy search based on the latent diffusion repository [2], [17] and PyTorch 2.0.1. We uses `MNN` [18] for deployment. We chose these because latent diffusion is the official code for diffusion models, and `MNN` provides a convenient solution for deploying models across different platforms, supporting direct conversion and deployment of PyTorch models. Our implementation includes 8.3 k lines of Python/C++/OpenCL code, including offline phase configuration and strategy search, as well as the weighted centroid learning algorithm, and operator implementation on various hardware for online inference.

Hyper-parameters: When searching for strategies, we exclude the first input layer and the last output layer because they are much more sensitive to the errors introduced by table lookup than other layers. We replace the remaining linear layers, including the MM in the transformer layer and the Conv2d operation, which is converted to MM through *im2col*. We set the candidate set for K as {8, 16, 32, 64, 96, 128}. This is due to the consideration of alignment with SIMD instructions, as well as the size of the input image for the model layer (ranging from

2x2 to 64x64). Moreover, it is necessary to ensure that the efficiency of using SIMD is faster than lookup values one by one, i.e., $\frac{N}{Q} \cdot \frac{K}{Q} < N$, which means that for the N input values, the number of table lookup operations should not be more than that of computation. The candidate set for V is set as {3, 6, 9} because 3x3 convolutions account for the largest proportion, and it is necessary to consider the locality of image regions so that the divided subvectors align the convolution kernels. For other linear operations, the correlation between different features of the input is relatively small, so we use the same V candidate set. After completing the layerwise lookup configuration search, we profile the inference engine based on the searched lookup configuration to obtain the lookup efficiency parameter E. During lookup table learning, we learn the centroids for one epoch and the batch size is 20. We use random seed 0 for image generation.

Baseline: We compare `LUT-Diff` with 4 baseline methods. To evaluate the quality of images generated by `LUT-Diff`, we compared `LUT-Diff` with `int8` and `original` model; to evaluate the inference performance of `LUT-Diff`, we compared `LUT-Diff` with `MNN` [18]; to evaluate the memory saving achieved by `LUT-Diff`, we compared `LUT-Diff` with `MNN`, and broke down the memory saving into parameter and activation memory saving; to evaluate the efficiency of `LUT-Diff` in generating configurations and learning centroids, we compared `LUT-Diff` with `LUT-NN`. Each baseline method is described as follows:

- `original`: the original diffusion model trained on the each dataset. We use the officially released model checkpoint [17].
- `int8`: We use `q-diffusion` [7] as the int8 quantization methods for comparison, a state-of-the-art quantization method that divides the model according to time steps and quantizes the model to int8 within each time step range separately.
- `MNN`: Deployment of the original full precision models on different devices based on the `MNN` [18] framework.
- `LUT-NN`: `LUT-NN` proposes a learnable lookup table technique, optimizing the learning of centroids through the end-to-end model and centroids training.

VII. Evaluation

A. Experiment Setting

Model and Dataset: We tested on 5 diffusion models. Four unconditional models were trained on the CelebA-HQ (*CelebA*) [22], LSUN-Churches256 (*Church*) [31], LSUN-Bedrooms256 (*Bedroom*) [31], and FFHQ256 (*FFHQ*) [32] datasets, respectively. One text-conditional model was trained on the Conceptual Captions (*txt2img*) [33] dataset. We use the official model checkpoints [17] for experiments. During lookup table learning, we use a subset with 1000 images uniformly sampled from the original training dataset for the unconditional models. For the text-conditional model, we categorized the dataset based on the main subject of the images and sampled 50 images per class. This is to reduce the learning overhead because of the heterogeneous classes contained in the *T2I* dataset. We use the original text encoder model contained in the diffusion model as the text classifier to split the dataset according to the image class. Initially, we learned tables for 50 classes. We obtain the centroids learning dataset by adding random noise, using the diffusion model to predict noise, and capturing intermediate activations through PyTorch hooks. The noise addition and training hyperparameters are the same as those used in the original paper [2], [17].

Testing Hardware: We conducted evaluations on 4 phones: (1) Xiaomi 14, equipped with SnapDragon 8Gen3 SoC, integrated with Adreno 750 GPU; (2) Redmi K60 Champion Edition (shorted as *Redmi K60*), equipped with SnapDragon 8Gen2 SoC, integrated with Adreno 740 GPU; (3) Samsung S21, equipped with SnapDragon 888, integrated with Adreno 660 GPU; (4) Samsung Note20, equipped with SnapDragon 865, integrated with Adreno 650 GPU.

Evaluation Metrics: We evaluated the efficiency of `LUT-Diff` from different aspects, including the quality of generated images, the acceleration of image generation, the inference memory footprint, and the efficiency of the training-free centroids learning. In evaluating image quality, we compared the MSE between images generated by different methods and those generated by the original model to compare their similarity. At the same time, we tested the Frechet Inception Distance (FID) [34], [35] of generated images compared with the original training dataset. The acceleration is measured by the normalized inference speedup in one denoising iteration. The memory footprint is measured by the normalized memory saving. The efficiency of lookup table learning is measured in GPU hours. Because of the significant variations in inference latency across different models and devices, we use the acceleration ratio as the evaluation metric for demonstration in subsequent experiments. The acceleration ratio refers to the acceleration brought by `LUT-Diff` compared with the naive deployment method, which is equivalent to the latency budget. We chose the three inference strategies with three different acceleration ratios. We chose the theoretical upper bound of `int8`, i.e., $4\times$ acceleration, as the baseline to ensure that `LUT-Diff` consistently outperformed it. The lowest was the one that is just faster than `int8`, denoted as "Low"; the highest was the one achieving the maximum acceleration ratio without a sharp deterioration in images, denoted as "High"; the medium was a balance between the lowest and the highest ratio, denoted as "Medium". As for the granularity of different strategies, "Low" corresponds to a finer granularity with larger K and correspondingly slower inference, while "High" corresponds to a coarser granularity with smaller K and correspondingly faster inference, and "Medium" is a compromise one.

B. Quality of Generated Images

We evaluated the quality of generated images by `LUT-Diff` and compared them with `int8` and `original`. We exclude `LUT-NN` because it requires end-to-end model and centroids training through backward propagation, which is unaffordable. We will show these results in Section VII-E. We used Redmi K60 in the experiment.

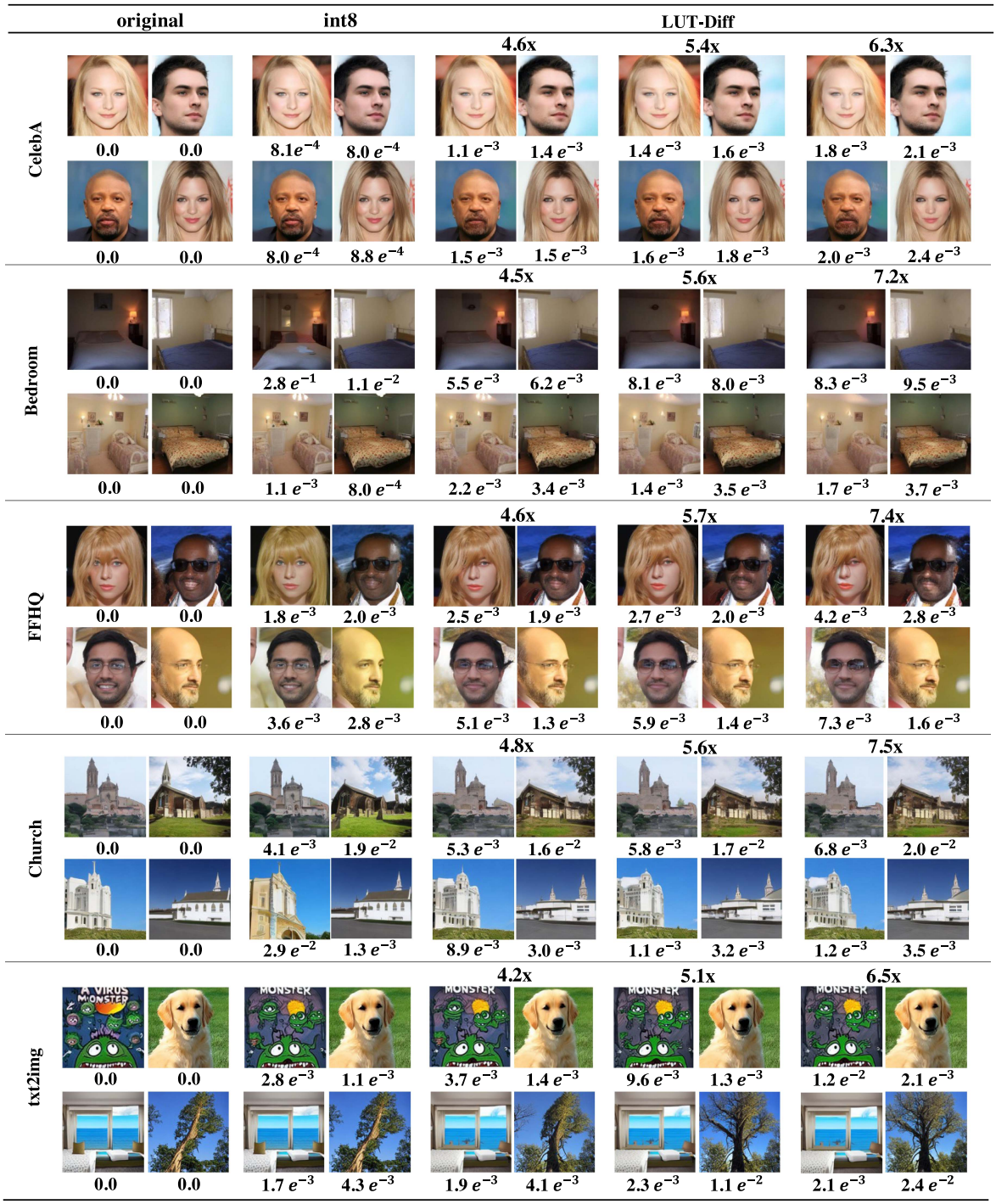

Fig. 8. Images generated by baselines and LUT-Diff, under strategies with different acceleration ratios. The value below each image is the MSE of the generated image compared to the baseline, with the image data normalized to [-1, 1]. For the text-conditional model, we use the average acceleration ratio.

Fig. 8 shows the images generated by each method. We find that LUT-Diff can generate images very similar to those of the original model. In Fig. 8, the MSE produced by our method is lower than $2.4e^{-2}$. In the best case, it reached $1.1e^{-3}$ (the first image of the CelebA model). Compared to the int8 method, the difference in images produced by our method is also very small. For instance, on the CelebA model, the MSE of images generated by our method ranges from a low of $1.1e^{-3}$ to a high of $2.4e^{-3}$, whereas the MSE for the int8 method is $8.2e^{-4}$. This error is minimal, making the visual differences in the generated images almost imperceptible to the naked eye. Additionally, we find that in some cases, the MSE of our method is even better than that of int8, such as the first image of the Bedroom model, where the highest MSE of images generated by our method under different acceleration ratios is up to $8.3e^{-3}$, but the MSE for the int8 is $2.8e^{-1}$. The gap between them reaches $33.7\times$, with an

TABLE I
FID RESULTS

	original	int8	LUT-Diff		
			Low	Medium	High
Church	2.32	2.50	2.68	2.91	2.79
Bedroom	3.57	4.38	4.49	4.74	4.91
CelebA	0.80	0.82	0.88	0.86	0.92
FFHQ	2.29	2.85	2.11	2.07	2.17
txt2img	9.53	10.22	11.04	12.26	13.11

Lower is better.

absolute error of 0.27. Correspondingly, the images generated by int8 also show significant visual differences from the original images. The reason is that the lookup table is expressive with fine-grained configuration and accurate centroids. LUT-Diff can capture the internal information of different feature dimensions and minimize the centroids error through the proposed lookup table learning approach. As for the text-conditional model, LUT-Diff can also generate images similar to baseline methods. We use the prompts to generate "a virus monster", "a golden retriever on the grass", "a room with an ocean view", and "a towering tree under the sky". Although there are some scenarios that exhibit discrepancies compared to the original model, for instance, when generating virus monster images under a high acceleration ratio, with the MSE reaching 0.012, LUT-Diff is still capable of generating high-quality images. This is primarily because LUT-Diff employs a multi-lookup table design approach, generating a corresponding lookup table for each class, thereby improving the model accuracy.

Table I lists the FID of images generated by each method. Taking the "High" strategy as the example, on the Church model, the FID of our method surpasses int8 by up to 0.29, on the Bedroom model by 0.53, and on the CelebA model by up to 0.1. Even compared to original, the increase in FID caused by our method does not exceed 1.34. Interestingly, on the FFHQ model, we find that LUT-Diff 's FID is at least 0.12 lower than that of original, and at least 0.68 lower than that of int8. One reason we analyze is that the differences between these generated images are small, but the features of the images generated by LUT-Diff in the high-dimensional space are closer to the training data. Our method achieves better results than int8, which also demonstrates the high accuracy of the lookup tables learned by LUT-Diff . For text-conditional models, we observed that LUT-Diff increases more FID compared to the unconditional models. This is mainly because text-conditional models are more complex, with more diverse inputs, which increases the error of lookup table. However, it is also acceptable, as LUT-Diff is still capable of generating high-quality images, with only subtle differences in the appearance of the images. In summary, LUT-Diff 's performance on FID is comparable with original.

In summary, LUT-Diff has comparable model performance with the original model under various acceleration ratios. Such results are inseparable from a specific component in the system. First, the offline lookup table learning we proposed accurately captures the distribution characteristics of different subvectors, trading a minor accuracy loss for significant savings in latency and memory footprint. Additionally, the CPU-GPU parallel inference engine we developed provides systemic support for inference acceleration, ensuring efficient inference. We will conduct a breakdown analysis in Section VII-F to demonstrate the impact of each component.

C. Inference Acceleration

We evaluated the acceleration ratio of end-to-end inference of LUT-Diff under different inference strategies and devices. We also use 3 strategies and denote them as "Low", "Medium" and "High", respectively, the same as the previous section. We also exclude LUT-NN in this experiment due to the same reason. The results are shown in Fig. 9.

Our key observation and conclusion are that across all devices, LUT-Diff consistently and remarkably achieves a higher acceleration ratio than MNN and outperforms the theoretical acceleration upper bound of int8. On all devices, compared with MNN, LUT-Diff brings 4.8-8.7× acceleration on the Church model, 4.6-8.5× acceleration on the CelebA model, 4.5-9.1× acceleration on the Bedroom model, 4.6-8.6× acceleration on the FFHQ model, and 4.1-7.7× acceleration on the txt2img model. We observed that on the text-conditional model, LUT-Diff does not perform as well in terms of acceleration ratio compared to other models. This is mainly because text-conditional models have more complex input, requiring LUT-Diff to adopt a more fine-grained lookup configuration with smaller subvector lengths V and a larger number of centroids K. This results in larger tables, increasing inference time. However, LUT-Diff is still able to achieve faster acceleration compared to int8 quantization. On the four devices, LUT-Diff consistently and remarkably outperforms the upper bound of the acceleration of int8 w.r.t. all strategies, as shown in the red dashed line in the figure. Such a promising result comes from the computation reduction, hardware co-scheduling, and efficiency of lookup operation than computation. LUT-Diff can achieve up to a 9.1× acceleration compared to the baseline method (Bedroom model, Samsung S21), because of LUT-Diff 's ability to select the optimal inference strategy under a specific budget, maximizing the benefits of the lookup table while minimizing errors. On resource-constrained mobile devices, using lookup tables to reduce computation yields significant benefits. LUT-Diff 's inference engine fully utilizes the CPU and GPU for parallel processing without the need for intra-operation synchronization. On mobile devices, the unified memory architecture on the chip ensures high data transfer efficiency, reducing the synchronization overhead after each operation.

D. Memory Footprint

We evaluated the memory footprint during inference with LUT-Diff, comparing it with original. We also excluded LUT-NN in this experiment. We used a batch size of 1, and the device is Redmi K60. We chose the same strategies with Section VII-B. Fig. 10 shows the memory footprint of inference using inference strategies with different theoretical acceleration ratios. Since int8 brings up to 75% memory saving, we exclude it in the figure. We find that LUT-Diff can bring

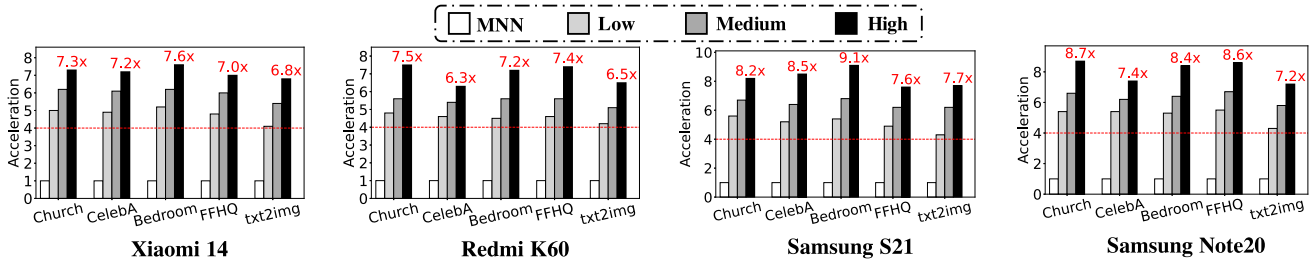

Fig. 9. Normalized acceleration results. The red dashed line is the upper bound of int8, i.e., 4×.

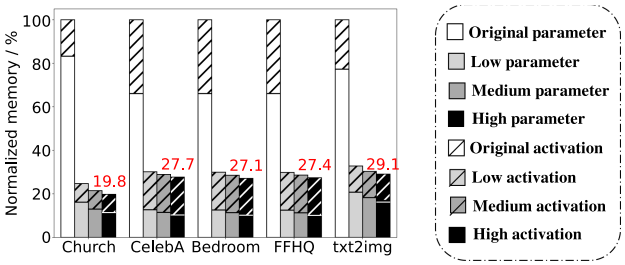

Fig. 10. Normalized memory on Redmi K60.

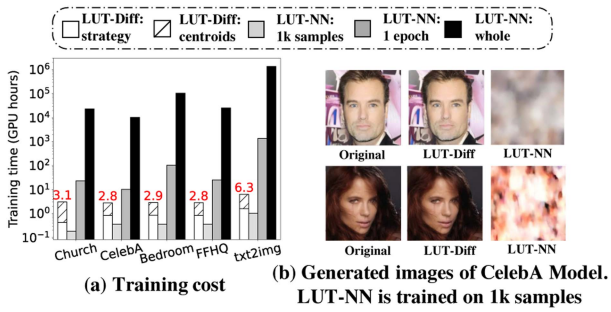

Fig. 11. Lookup table learning efficiency of LUT-Diff .

different degrees of memory savings during inference. When choosing the "High" strategy, LUT-Diff can bring the maximum memory footprint savings, reaching up to 80.2% on the Church model. The memory saving is at least 70.9% in all cases. LUT-Diff can reduce the memory of both model parameters and activation. Specifically, LUT-Diff reduces up to 86.2% parameter memory and 50.0% activation memory. This is because LUT-Diff does not need to store the complete model parameters but instead the centroids and corresponding pre-computed result table. Given a MM with input size $N * D$ and weight size $D * M$, LUT-Diff can reduce the parameter memory footprint from $O(D * M) * O(float32)$ to at most $O(\sum(k_i * v_i + k_i * M)) * O(log_2 k_i)$, where k_i and v_i is the centroid number and length of each subvector. We encode the centroids and table non-uniformly to reduce the memory footprint further so that the memory overhead for each value becomes $O(log_2 k_i)$, as described in Section IV-A. Because $k_i \ll D$, LUT-Diff can achieve promising parameter memory saving. Besides, the memory footprint of the encoded centroid indices during inference is as low as $N * \frac{D}{V} * O(log_2 k_i)$. We found that on the text-conditional model, LUT-Diff performs less effectively in terms of memory savings compared to other models. This is primarily because LUT-Diff requires larger lookup tables to store information. Nevertheless, the memory-saving performance of LUT-Diff remains very significant, reaching 70.9% .

E. Efficiency of Lookup Table Learning

We evaluated the efficiency of centroid learning and configuration searching, i.e., the offline stage in LUT-Diff . The baseline we compared with is LUT-NN, which requires end-to-end training of centroids. Since the cost of training diffusion models is unaffordable to us, with the default setting requiring training on the complete dataset for 1000 epochs, we used the same settings as LUT-Diff when training LUT-NN, i.e., training 1000 images for 1 epoch. We estimated the training cost of LUT-NN when training the whole dataset for 1000 epochs by multiplying the cost of one epoch by 1000. Other hyper-parameters of model training, such as learning rate, as well as K and V in LUT-NN, were kept consistent with the original paper. At the same time, we evaluated the quality of images generated by LUT-Diff and LUT-NN after learning the same amount of data, i.e., 1000 images for 1 epoch. We used one P100 GPU in the experiment.

The results are shown in Fig. 11. Fig. 11(a) shows the training time of each method. For all models, LUT-Diff can complete all centroid learning and strategy searches within 3.1 hours. In contrast, although LUT-NN outperforms LUT-Diff in training 1000 images, LUT-NN will be slower than LUT-Diff if it undergoes training for one epoch on the complete dataset. If training for 1000 epochs, LUT-NN would require at least 10173 P100 hours (CelebA model), which is 3281× of LUT-Diff and is unacceptable. We conducted a breakdown analysis of the learning efficiency of LUT-Diff, dividing it into the costs of centroid learning and lookup configuration search. We found that the overhead of centroid learning averages 25.9% . These results indicate the efficiency of the proposed centroid learning methods. LUT-Diff can finish the configuration search within 2.6 hours, indicating that the layerwise search algorithm can significantly reduce the search space. For text-conditional models, we observed that the training time for LUT-Diff significantly increased. This is mainly because we trained separate lookup tables for multiple categories. In this experiment, we used a total of 2500 images, far more than those used for

TABLE II
FID RESULTS WITHOUT WEIGHTED CENTROIDS LEARNING OR LOOKUP CONFIGURATION SEARCH

	LUT-Diff w/o centroids learning			LUT-Diff with fixed V			LUT-Diff with fixed K		
	Low	Medium	High	Low	Medium	High	Low	Medium	High
Church	3.48 (0.80)	4.10 (1.19)	4.52 (1.73)	10.00 (7.32)	13.33 (10.42)	19.31 (16.52)	30.06 (27.38)	30.06 (27.15)	30.06 (27.27)
Bedroom	7.09 (2.60)	7.63 (2.89)	8.35 (3.44)	16.39 (11.90)	19.81 (15.07)	37.02 (32.11)	42.59 (38.10)	42.59 (37.85)	42.59 (37.68)
CelebA	1.23 (0.35)	1.21 (0.35)	1.40 (0.48)	3.55 (2.67)	5.12 (4.26)	7.28 (6.36)	18.12 (17.24)	18.12 (17.26)	18.12 (17.20)
FFHQ	3.00 (0.89)	3.13 (1.06)	3.45 (1.28)	8.27 (6.16)	10.16 (8.09)	12.48 (10.31)	35.29 (33.18)	35.29 (33.22)	35.29 (33.12)
txt2img	13.92 (2.88)	17.81 (5.55)	21.62 (8.51)	49.91 (38.87)	61.23 (48.97)	81.37 (68.26)	186.03 (174.99)	186.03 (173.77)	186.03 (172.92)

Value in the bracket is the FID increase compared with lut-diff in Table I.

other models. Moreover, we found that the time for centroid learning hardly increased. This is because the centroid learning for each layer is independent and can be performed in parallel. In contrast, the search configuration requires the involvement of the entire model, which results in a higher proportion of time consumption. Due to the larger model size, the training time for LUT-NN increases significantly. In comparison, our method is more efficient than LUT-NN. At the same time, we showcased a comparison of images generated by LUT-Diff and LUT-NN on the CelebA model, where LUT-NN was trained for one epoch on 1000 images. The results are shown in Fig. 11(b). We find that after learning the same amount of data, LUT-NN cannot generate recognizable images, whereas our method produces images very close to original. In summary, LUT-Diff can learn centroids more efficiently than LUT-NN. With the same amount of training, LUT-Diff generated better images than LUT-NN. Considering the high training cost of LUT-NN, we do not consider this baseline in subsequent experiments.

F. Ablation Study

We conducted an ablation study to explore the specific benefits each technology provides. Specifically, we examined the benefits of weighted centroid learning, lookup configuration searching, and the parallel inference engine. We used Redmi K60 in this experiment.

Benefit of weighted centroid learning: We explored the FID of generated images after learning the centroids in the input space. The results, shown in the first 3 columns of Table II, indicate that the FID in the input space is higher than in the output space. The results reveal that the FID in the input space under various acceleration ratios increases by 1.42 on average compared with learning in the output space, and the average FID is 1.51× that of LUT-Diff on average for unconditional models. Taking the *Bedroom* model as the example, the FID increased by up to 3.44. This is primarily due to the differences between the learning targets in the input space and the optimization objectives of model training in the output space. The weighted centroid learning more closely aligns with the optimization objectives of model training, as both are completed in the output space. In contrast, the input space learning disregards the impact of model weights, thereby affecting the FID results. For the text-conditional model, we found that learning centroids in the input space leads to a more significant increase in FID. For instance, under the "High" inference strategy, the FID increased by 8.51. This is primarily due to the larger parameter size and the more complex inputs of the model. Without processing through the model layers, the input space contains more diverse features with higher data variance, making centroid learning in the input space significantly more challenging.

Benefit of lookup configuration searching: We evaluated the FID under uniform lookup configurations. Specifically, after searching for the lookup configurations, we updated the configurations and fixed either the number of centroids or the sub-vector length. We used the same values as LUT-NN when fixing K or V. We used the same searched configurations as Section VII-B for comparison convenience. The results are presented in the last 6 columns of Table II.

After we fixed the sub-vector length V, using the Bedroom model as an example, the FID of the generated images was significantly higher than that of LUT-Diff across all configurations. For instance, for the unconditional models, in the "High" configuration, the model's FID increased by 32.11, reaching 37.02, which is 10.37× that of the original model. Even in the "Low" configuration, the FID reached 16.39, which is 4.59× that of the original model, an increase of 11.90 compared to the LUT-Diff configuration where both V and K are searched. Across all models and configurations, the average FID is 5.26× that of LUT-Diff, with an average increase of 10.92 compared to LUT-Diff . These results indicate that fixing the sub-vector length makes the lookup table less capable of capturing local correlations in the data, leading to a significant decline in model performance. Thus, using a fixed V substantially negatively impacts model performance. For the text-conditional model, we found that fixing V also leads to an increase in FID, and the increase is more pronounced compared to the unconditional models. Specifically, under the "High" inference strategy, the FID increased by 68.26. This further demonstrates that fixing V hinders the ability of subvectors to capture the correlation information between data points, highlighting the advantages of the LUT-Diff algorithm.

Interestingly, when we fixed the number of centroids K, we found that the performance across different configurations was identical. Whether the strategy is "Low", "medium", or "High", the FID of the generated images was the same. This is primarily due to the way LUT-Diff searches lookup configurations. LUT-Diff first searches for V to capture local data correlations and then fixes V while searching for different K values. Therefore, if a uniform K is used, the differences among configurations are eliminated, resulting in the same model performance. We found that after fixing K, the FID of the generated images increased by 17.20-38.10, which is 8.67-21.07× that of LUT-Diff for unconditional models. This is mainly because using a uniform number of centroids ignores

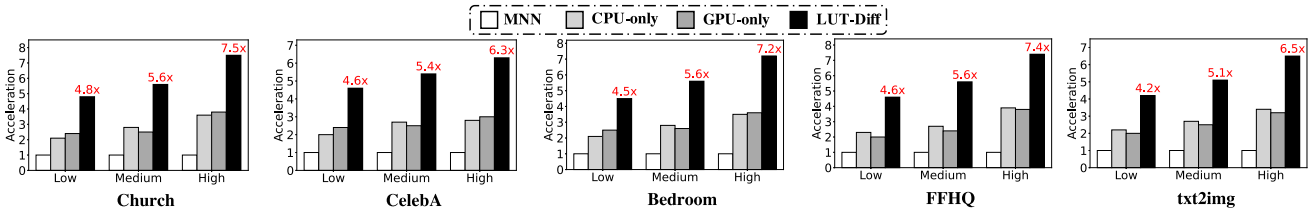

Fig. 12. Ablation results of the parallel inference engine of LUT-Diff . The device is Redmi K60.

the diversity characteristic of different sub-vector distributions. If the distribution is more dispersed, smaller K can lead to larger errors, affecting model performance. Therefore, using a uniform number of centroids not only negatively impacts model performance but also undermines the adaptiveness of LUT-Diff . As for text-conditional models, we observed a significant increase in FID, reaching 186.03, when K was fixed. Upon examining the generated images, we found that the entities in the images became barely recognizable. This is primarily because LUT-NN uses a fixed K of 16, which is far from sufficient to meet the demands of this model. In contrast, LUT-Diff effectively reduces lookup errors by dynamically searching for the optimal K, further demonstrating its efficiency and adaptability.

These results indicate that both the number of centroids K and the sub-vector length V, are crucial for the accuracy of the lookup table, and optimal performance can only be achieved by considering both factors together.

Benefit of parallel inference engine: We evaluated the inference speed using only CPU or GPU. The results, as shown in Fig. 12, demonstrate that using only one type of hardware does not achieve optimal inference speed. Specifically, using only CPU achieved accelerations of 2.0-3.9×; using only GPU achieved accelerations of 2.0-3.8×. Both scenarios performed worse than using the parallel inference engine on both CPU and GPU. For instance, for the *Bedroom* model, when using only the CPU, LUT-Diff could only achieve a maximum acceleration of 3.6×; on the GPU, the maximum acceleration was 3.7×. While using both types of hardware, the acceleration reaches up to 7.2×. This is primarily because the engine can utilize both CPU and GPU for cooperative processing, ensuring that computations on both pieces of hardware do not depend on each other. Additionally, LUT-Diff minimizes the latency difference between CPU and GPU, reducing the additional synchronization overhead and enhancing the inference performance. For the text-conditional model, we observed that the acceleration ratio of LUT-Diff is lower compared to unconditional models. Through a detailed analysis of the strategies, we found that, unlike unconditional models, LUT-Diff searches finer-grained V configurations for the text-conditional model, resulting in more sub-vectors of length 3. Additionally, the number of centroids K is also higher. This leads to an increased number of operations required for table lookup, resulting in higher latency and reduced acceleration. However, LUT-Diff consistently outperforms the int8 method, achieving 4.2× acceleration even under the "Low" strategy.

G. Breakdown Analysis

1) Inference Strategy Breakdown: We analyzed the inference strategies obtained through the search. Since the differences between strategies lie in the number of centroids K, we evaluated the proportion of different K values for each model. Additionally, we assessed the proportion of subvector lengths V in the strategies for each model. We used Redmi K60in this experiment. The results are shown in Table III.

As for the sub-vector length V, we observed that $V = 3$ accounted for the largest proportion across all strategies and, with smaller V values being more frequent. This is because larger V introduces greater errors. For example, in the CelebA model, the proportions of sub-vectors with lengths 3, 6, and 9 were 62.3% , 29.9% , and 7.8% , respectively in the "High" strategy. The presence of $V = 9$ indicates there are dependencies in the image data, which would be overlooked if a uniform sub-vector length were used, leading to higher errors. For the number of centroids K, we found that as the acceleration ratio increases, smaller K values become more dominant. This phenomenon aligns with the modeling in Section Section V-B, where K is positively correlated with computation and lookup latency. Furthermore, we conducted a more detailed analysis of K distribution and found that smaller K values were concentrated in the middle layers of the model, while larger K values appeared near the input and output layers. This is because input and output layers process larger image dimensions and are more sensitive to errors.

2) Inference Latency Breakdown: We evaluated the latency breakdown of different operations during inference. The table lookup latency was divided into three categories: (1) encoding latency, representing the latency for matching inputs with centroids to obtain lookup indices; (2) computing latency, for calculating the sub-vectors with extremely high FIM error; and (3) LUT latency, for retrieving values using indices. We used Redmi K60 in this experiment. The results are shown in Table III.

We find that encoding latency accounts for the largest proportion in the "Low" and "Medium" strategies, exceeding 50% . As for the "High" strategy, the encoding latency also accounts for a large proportion, reaching 41% . This is primarily because matching inputs with centroids involves MM, as described by Formula 3, which depends on the input size and the number of centroids. On the other hand, memory access latency is lower than computation latency, particularly multiplication latency, making encoding latency dominant. Additionally, we observed that as the acceleration ratio increases, the proportion of encoding latency decreases. This is because the acceleration ratio

TABLE III
BREAKDOWN OF INFERENCE STRATEGY (THE LEFT 9 COLUMNS) AND INFERENCE LATENCY (THE RIGHT 3 COLUMNS)

	%	K=8	K=16	K=32	K=64	K=96	K=128	V=3	V=6	V=9		%	Enc.	Comp.	LUT
Low	Church	10.1	33.3	30.2	17.6	5.2	3.6	61.4	29.7	8.9	Low	Church	57.0	25.7	17.3
	Bedroom	11.4	31.5	34.1	15.4	4.1	2.5	60.2	31.9	7.9		Bedroom	55.3	27.7	17.0
	CelebA	10.9	30.7	33.5	16.4	5.0	3.5	62.3	29.9	7.8		CelebA	56.2	26.7	17.1
	FFHQ	10.8	30.2	32.1	16.8	4.8	5.3	61.9	31.1	7.0		FFHQ	55.1	27.8	17.1
	txt2img	8.2	28.2	35.3	16.5	5.1	6.7	66.2	29.9	3.9		txt2img	58.6	28.1	13.3
Medium	Church	12.6	35.5	31.4	14.1	3.5	3.2	61.4	29.7	8.9	Medium	Church	50.6	33.4	16.0
	Bedroom	12.8	34.8	35.4	13.3	2.1	1.6	60.2	31.9	7.9		Bedroom	51.3	32.5	16.2
	CelebA	13.0	34.7	35.2	12.9	2.4	1.8	62.3	29.9	7.8		CelebA	49.8	34.3	15.9
	FFHQ	13.2	34.1	34.7	12.3	3.2	2.5	61.9	31.1	7.0		FFHQ	52.1	31.6	16.3
	txt2img	9.8	31.1	34.3	15.2	5.0	4.1	66.2	29.9	3.9		txt2img	54.3	35.3	10.4
High	Church	13.1	38.4	28.1	13.9	3.5	3.0	61.4	29.7	8.9	High	Church	42.5	43.0	14.5
	Bedroom	13.0	39.3	29.0	12.8	3.1	2.8	60.2	31.9	7.9		Bedroom	43.1	41.6	14.7
	CelebA	14.6	39.1	29.3	13.1	2.5	1.4	62.3	29.9	7.8		CelebA	43.7	42.3	14.0
	FFHQ	14.1	37.6	30.0	13.8	3.2	1.3	61.9	31.1	7.0		FFHQ	41.3	44.5	14.2
	txt2img	10.8	34.0	32.7	15.1	4.8	2.6	66.2	29.9	3.9		txt2img	49.1	41.5	9.4

"Enc.": encoding latency; "Comp.": computing latency for sub-vectors with large error; "LUT": LUT latency. The device is Redmi K60.

is influenced by the number of centroids, i.e., K, and a higher acceleration ratio corresponds to a smaller K. Both encoding latency and LUT latency decrease accordingly; however, since multiplication latency is higher, the absolute reduction in encoding latency is more significant, leading to a lower proportion. For computing latency, its value remains relatively constant across different acceleration ratios since the computation is unaffected by K. However, as other latencies decrease, the proportion of computing latency increases.

VIII. RELATED WORK

A. Approximate Computation in Model Inference

Model quantization: Model quantization, including scalar quantization (SQ), PQ, and vector quantization (VQ), is to reduce the memory footprint and computational requirements of model inference. SQ [36], [37], [38], [39], [40] quantize each parameter value with lower precision representation, e.g., from float32 to int8. Octo [40] quantizes CNN to INT8 and accelerates on-device inference. XNOR-Net [41] quantizes CNN with binary values for image classification tasks. PQ [11], [12], [13], [28] divides each vector into several smaller subvectors and quantizes them independently. Chen et al. proposed differentiable PQ for embedding layer [13]. MADDNESS [11] appies PQ-based lookup table to approximate the MM. LUT-NN [28] applies differentiable PQ for table lookup through end-to-end model training. To our best knowledge, there does not exist work that explores the lookup table to the whole deep learning model without end-to-end training. VQ [42], [43], [44], [45], [46] encodes the whole vector in with centroids. VQ-VAE [47], [48] integrates VQ with variational autoencoders to encode images, excelling in generating high-quality outputs for various data types like images and audio. Gong et al. [49] leverage VQ to quantize the CNN model.

Intermediate results reuse: Several studies have also explored reusing intermediate results during model inference to reduce computations. DeepCache [5] reuses computational results from similar regions within images, which is a cache design that enhances deep learning inference efficiency by exploiting temporal locality in video streams and integrating video compression heuristics to manage data reuse within models. SMTM [50] is a semantic memory design for CNN inference that features a hierarchical memory architecture and novel techniques to adaptively adjust cache size according to scene dynamics, thus speeding up inference on mobile devices. MCDNN [51] reuses results from other apps in continuous mobile vision domain model serving scenarios to enable sharing.

In contrast, we, for the first time, enable efficient inference of diffusion models on mobile devices through lookup table. Unlike existing methods that focus on model quantization or intermediate result reuse, our approach replaces computationally intensive MM with lightweight lookup table-based indexing, fundamentally reducing computational overhead on mobile devices.

B. Diffusion Model

Diffusion models generate high-quality and diverse images by predicting the noise in the image and eliminating it. DDPM [1] gradually adds noise to data during training to learn the diffusion process. It reverses such a process by gradually eliminating noise at each step. DDIM [23] accelerates the denoising process by applying a continuous-time Ordinary Differential Equation, allowing the model to generate samples with fewer timesteps. Stable diffusion stabilizes the diffusion process in latent space [2]. Many other optimizations focus on the diffusion model architecture and sampling algorithm [3], [52], [53], [54], [55], [56], [57]. For example, ScaleCrafter [55] uses re-dilated convolution to enlarge the perception field of the filter. These efforts mainly focus on the denoising algorithm and model architecture, which are orthometric to our approach. Besides image synthesis, many other diffusion models focus on video generation [58], [59], [60] and audio generation [61], [62], [63]. In contrast, our method does not alter the model architecture or the noise elimination process; instead, it focuses on optimizing the inference process. As a result, our approach is orthogonal to existing optimizations of diffusion models, making it compatible and complementary to them.

C. Lookup Table in Deep Learning

Several studies have explored lookup table applications in deep learning. LUT-NN [28] proposed an end-to-end training method to learn centroids and tables. Akshay et al. [64] introduced a lookup table-based Processing-In-Memory technique to accelerate inference by caching the bitline computation results. DeepGEMM [65] applies lookup table to accelerate 2-bit quantized model inference. T-MAC [66] applies lookup table for low-bit quantized large language model deployment on mobile CPU. In contrast, LUT-Diff proposed a training-free table learning method and adaptively performs optimized inference under a given latency budget for the diffusion model. LUT-Diff can be applied to different layer types, as demonstrated in the experiments, including transformer, linear, and convolution layers. In contrast, our method generates the lookup table using a training-free process, eliminating the need for complex end-to-end training and making it significantly more practical for diffusion models.

IX. Discussion

Application to other models: In this paper, we propose LUT-Diff, an optimization approach for diffusion model deployment on mobile devices. LUT-Diff leverages the diffusion model's inherent tolerance to noise in input data during inference, using an approximation approach for inputs. For other models, such as CNNs, which do not add random Gaussian noise during inference, their tolerance to input noise is lower. Therefore, LUT-Diff could lead to a decrease in accuracy. Some additional effort may be needed to apply LUT-Diff to other models, such as end-to-end fine-tuning. Nevertheless, as our experiments have already demonstrated the efficiency of LUT-Diff across different models and datasets, we believe LUT-Diff can promote the deployment of diffusion models on mobile devices.

On-device lookup table learning: For text-conditional models, we leveraged the text encoder as a router to select different lookup tables, adopting the concept of a mixture-of-experts model. In case the input prompt lacks a corresponding table, LUT-Diff first performs fast inference using a quantized model while offloading the lookup table learning to a cloud GPU server. Since LUT-Diff follows a training-free paradigm, on-device centroid learning can be applied during the device is idle. We leave this part as our future work.

X. Conclusion

In this paper, we have designed and implemented LUT-Diff, an algorithm-system co-design that aims at optimizing on-device diffusion model inference through lookup table. LUT-Diff can efficiently learn the lookup table and adaptively perform diffusion model inference according to different budgets. Extensive experiments show that LUT-Diff can efficiently learn centroids and lookup tables (at least 3281× faster), generate high-quality images (less than 0.012 MSE), achieve promising inference acceleration (up to 9.1× acceleration), and reduce inference memory footprint (up to 70.9%).

References

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 6840–6851.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proc. IEEE/CVF Conf. Comput. Vis. pattern Recognit.*, 2022, pp. 10684–10695.

[3] Y. Li et al., "SnapFusion: Text-to-image diffusion model on mobile devices within two seconds," 2023, *arXiv:2306.00980*.

[4] S. A. Osia et al., "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4505–4518, May 2020.

[5] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: Principled cache for mobile deep vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 129–144.

[6] Y. Wang et al., "A survey on deploying mobile deep learning applications: A systemic and technical perspective," *Digit. Commun. Netw.*, vol. 8, no. 1, pp. 1–17, 2022.

[7] X. Li et al., "Q-diffusion: Quantizing diffusion models," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 17535–17545.

[8] Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan, "Post-training quantization on diffusion models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 1972–1981.

[9] Y. He, L. Liu, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "PTQD: Accurate post-training quantization for diffusion models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 13237–13249.

[10] J. So, J. Lee, D. Ahn, H. Kim, and E. Park, "Temporal dynamic quantization for diffusion models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 48686–48698.

[11] D. Blalock and J. Guttag, "Multiplying matrices without multiplying," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 992–1004.

[12] B. Klein and L. Wolf, "End-to-end supervised product quantization for image search and retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5041–5050.

[13] T. Chen, L. Li, and Y. Sun, "Differentiable product quantization for end-to-end embedding compression," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1617–1626.

[14] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–16.

[15] Y. Li et al., "BRECQ: Pushing the limit of post-training quantization by block reconstruction," 2021, *arXiv:2102.05426*.

[16] "Single instruction, multiple data," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Single_instruction,_multiple_data

[17] "latent-diffusion," 2024. [Online]. Available: https://github.com/CompVis/latent-diffusion

[18] X. Jiang et al., "MNN: A universal and efficient inference engine," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 1–13, 2020.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Med. Image Comput. Comput.-Assisted Interv.: 18th Int. Conf.*, Munich, Germany, Springer, 2015, pp. 234–241.

[20] Q. Wang et al., "Melon: Breaking the memory wall for resource-efficient on-device machine learning," in *Proc. 20th Annu. Int. Conf. Mobile Syst., Appl. Serv.*, 2022, pp. 450–463.

[21] I. Gim and J. Ko, "Memory-efficient DNN training on mobile devices," in *Proc. 20th Annu. Int. Conf. Mobile Syst., Appl. Serv.*, 2022, pp. 464–476.

[22] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," 2017, *arXiv: 1710.10196*.

[23] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2020, *arXiv: 2010.02502*.

[24] D. W. Blalock and J. V. Guttag, "Bolt: Accelerated data mining with fast vector compression," in *Proc. 23 rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 727–735.

[25] X. Li and P. Li, "Random projections with asymmetric quantization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 10858–10867.

[26] Y. Zhang et al., "AFPQ: Asymmetric floating point quantization for LLMs," 2023, *arXiv:2311.01792*.

[27] "Pytorch unfold," 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Unfold.html

[28] X. Tang et al., "LUT-NN: Empower efficient neural network inference with centroid learning and table lookup," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–15.

[29] R. Karakida and K. Osawa, "Understanding approximate fisher information for fast convergence of natural gradient descent in wide neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 10891–10901.

[30] D. Deng, G. Chen, Y. Yu, F. Liu, and P.-A. Heng, "Uncertainty estimation by fisher information-based evidential deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 7596–7616.
[31] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop," 2015, *arXiv:1506.03365*.
[32] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4401–4410.
[33] "Google's conceptual captions," 2024. [Online]. Available: https://ai.google.com/research/ConceptualCaptions/
[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[35] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6629–6640.
[36] E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 580–595.
[37] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," 2020, *arXiv: 2008.05000*.
[38] S. Zhu, L. H. Duong, and W. Liu, "XOR-Net: An efficient computation pipeline for binary neural network inference on edge devices," in *Proc. IEEE 26th Int. Conf. Parallel Distrib. Syst.*, 2020, pp. 124–131.
[39] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.
[40] Q. Zhou et al., "Octo:{INT8} training with loss-aware compensation and backward quantization for tiny on-device learning," in *Proc. 2021 USENIX Annu. Tech. Conf.*, 2021, pp. 177–191.
[41] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2016, pp. 525–542.
[42] E. Agustsson et al., "Soft-to-hard vector quantization for end-to-end learning compressible representations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1141–1151.
[43] K. Chen and C.-G. Lee, "Incremental few-shot learning via vector quantization in deep embedded space," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–16.
[44] C. Gong, Y. Chen, Y. Lu, T. Li, C. Hao, and D. Chen, "VecQ: Minimal loss DNN model compression with vectorized weight quantization," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 696–710, May 2020.
[45] K. L. Oehler and R. M. Gray, "Combining image compression and classification using vector quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 5, pp. 461–473, May 1995.
[46] A. Fan et al., "Training with quantization noise for extreme model compression," 2020, *arXiv: 2004.07320*.
[47] A. Van Den et al., "Neural discrete representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6309–6318.
[48] H. Wu and M. Flierl, "Vector quantization-based regularization for autoencoders," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 6380–6387.
[49] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014, *arXiv:1412.6115*.
[50] Y. Li et al., "Boosting mobile CNN inference through semantic memory," in *Proc. 29th ACM Int. Conf. Multimedia*, 2021, pp. 2362–2371.
[51] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2016, pp. 123–136.
[52] Y. Shi, C. Xue, J. Pan, W. Zhang, V. Y. Tan, and S. Bai, "DragDiffusion: Harnessing diffusion models for interactive point-based image editing," 2023, *arXiv:2306.14435*.
[53] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein, and K. Aberman, "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 22500–22510.
[54] C. Si, Z. Huang, Y. Jiang, and Z. Liu, "Freeu: Free lunch in diffusion U-Net," 2023, *arXiv:2309.11497*.
[55] Y. He et al., "Scalecrafter: Tuning-free higher-resolution visual generation with diffusion models," 2023, *arXiv:2310.07702*.
[56] O. Avrahami, D. Lischinski, and O. Fried, "Blended diffusion for text-driven editing of natural images," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 18208–18218.
[57] Q. Zhang, M. Tao, and Y. Chen, "gddim: Generalized denoising diffusion implicit models," 2022, *arXiv:2206.05564*.
[58] W. Harvey, S. Naderiparizi, V. Masrani, C. Weilbach, and F. Wood, "Flexible diffusion modeling of long videos," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 27953–27965.
[59] J. Ho et al., "Imagen video: High definition video generation with diffusion models," 2022, *arXiv:2210.02303*.
[60] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 8633–8646.
[61] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro, "Diffwave: A versatile diffusion model for audio synthesis," 2020, *arXiv: 2009.09761*.
[62] Y. Wang et al., "Audit: Audio editing by following instructions with latent diffusion models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024.
[63] L. Ruan et al., "Mm-diffusion: Learning multi-modal diffusion models for joint audio and video generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 10219–10228.
[64] A. K. Ramanathan et al., "Look-up table based energy efficient processing in cache support for neural network acceleration," in *Proc. 53 rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2020, pp. 88–101.
[65] D. C. Ganji et al., "Deepgemm: Accelerated ultra low-precision inference on cpu architectures using lookup tables," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 4656–4664.
[66] J. Wei et al., "T-mac: Cpu renaissance via table lookup for low-bit LLM deployment on edge," 2024, *arXiv:2407.00088*.

Qipeng Wang received the BS degree in the School of Software from Beihang University. He is currently working toward the PhD degree in the School of Computer Science, Peking University. His main research interests lie in edge intelligence and deep learning system. Web page: https://qipengwang.github.io.

Shiqi Jiang received the bachelor's degree in computer engineering from Zhejiang University and the PhD degree in computer science from Nanyang Technological University, in 2018. He is a senior researcher with Systems and Networking Research Group, Microsoft Research Asia (MSRA). His research interests broadly fall in edge computing, mobile sensing, Internet-of-Things (IoT), and wearables.

Yifan Yang received the bachelor's degree from Tongji University and the master's degree from Peking University, both in computer science. He is a research engineer with Microsoft Research Asia. Yifan's current research focuses on multimodal learning, with specific interests in multimodal representation learning, large multimodal models, and image/video generation. His publication covers relevant top conferences such as CVPR, ICCV, ECCV, NeurIPS, and AAAI, and he serves as a reviewer for these conferences.

Ruiqi Liu is with the School of Software and Microelectronics at Peking University, enrolled, in 2022. His major is Software Engineering. His main interests lie in software development and engineering.

Yuanchun Li (Member, IEEE) received the BS and PhD degrees in computer science from Peking University. He was a senior researcher with Microsoft Research Asia. He is a research assistant professor with the Institute for AI Industry Research (AIR), Tsinghua University. His research interests lie in the efficiency and reliability of edge AI systems. His work won the UbiComp Honorable Mention Award and ISEUD Best Paper Award, and the related systems and tools are widely used in the open-source community. He is a member of ACM.

Ting Cao received the PhD degree from the Australian National University. She is a Principal research manager of the Heterogeneous Extreme Computing group (HEX) with Microsoft Research. Her research interests include deep learning system and algorithm design, novel efficient hardware design, tensor/language compiler design, etc. Before joining MSR, she had been working with the Chinese Academy of Sciences and Huawei Technologies.

Xuanzhe Liu (Senior Member, IEEE) is a full professor and Endowed Boya Distinguished professor with the School of Computer Science with Peking University, Beijing, China. His research interests mainly fall in systems software. Most of his recent efforts have been published at prestigious conferences including SOSP, OSDI, SIGCOMM, NSDI, MobiCom, etc. He is a distinguished member of the ACM and the CCF. Web page: http://www.liuxuanzhe.com/.