ADAngel: Accelerating Arbitrary-Precision Quantized LLMs with Adaptive Computing Mapping

Abstract

Arbitrary-Precision Quantization (APQ), which uses asymmetric bit-widths for weights and activations (e.g., W4A8), is a prevalent technique for LLM inference because of its excellent accuracy-performance balance. APQ transforms the general matrix multiplications (GEMM), the core of LLM computation, into mixed-precision GEMM (mpGEMM) whose two operand matrices have different quantization bit-widths. However, we identify that the computation paradigms of mpGEMM in current APQ LLM inference systems are suboptimal because the shapes and bit-widths of mpGEMM tasks in APQ LLM are highly variable, whereas existing static and workload-unaware paradigms can only accelerate mpGEMM tasks with the same or similar shapes and bit-widths.

Based on this finding, we propose ADAngel, a framework for creating a workload-adaptive mpGEMM computation core for target LLMs. The theoretical foundation of ADAngel is the DPR (Decomposition-Partial Product-Reconstruction) computation model, which enables the systematic generation of a diverse portfolio of mpGEMM algorithms by specifying different bit-partition schemes. Guided by this model, ADAngel constructs a Computation Strategy Set comprising several highly-optimized mpGEMM kernels, and can exhaustively analyze the strategy set to create an oracle policy map, which enables a lightweight dispatcher to select and execute the optimal kernel for runtime tasks with negligible overhead. Our evaluation shows the ADAngel-specialized engine achieves up to a 5.36× speedup in decode throughput over llama.cpp; while in the prefill stage, it demonstrates its adaptivity by delivering speedups ranging from 1.29× to 1.86× over TensorRT-LLM in Time-To-First-Token (TTFT).

1 Introduction

Large Language Models (LLMs) [9, 21, 29, 34] are increasingly being deployed on edge devices [19, 20, 24] to enhance the capabilities of edge intelligent tasks and agents. Unfortunately, edge devices are constrained by design limitations in chip area, power consumption, and cost, resulting in significantly inferior computing and memory/storage capabilities compared to cloud-scale infrastructure. Deploying full-scale LLMs on edge devices is inefficient or often infeasible because of limited hardware resources. Post-training quantization (PTQ) [15, 30, 32, 35] is a mainstream model compression technique that is widely adopted for edge-side LLM deployment. It quantizes the weight or activation for trained LLMs to reduce data bit-width and computational complexity, thereby enabling efficient inference on edge devices. Arbitrary-Precision Quantization (APQ) [3, 14, 25] quantizes weights and activations into different bit-widths, such as W4A8 (4-bit weights and 8-bit activations), thereby achieving a fine-grained trade-off between inference speed and quality. APQ results in LLM's general matrix-matrix multiplication (GEMM) or general matrix-vector multiplication (GEMV) having operands with different bit-widths (mixed-precision bit-widths). However, existing edge computing units or accelerators lack native hardware support for operations with asymmetric bit-width operands.

To enable existing edge devices to support inference for APQ LLMs, padding [11, 17], lookup table (LUT) [18, 22, 31], and bit-disaggregation [38] techniques have been proposed. Padding techniques upcast low-bit-width weights to match the bit-width of activations, ensuring compatibility with existing edge computing units (e.g., executing W4A8 operations on W8A8 hardware). LUT-based approaches precompute the products of high-bit-width activations and low-bit-width weights, storing them in a lookup table. During inference, results are directly fetched from the LUT to avoid asymmetric GEMM and GEMV operations. Bit-disaggregation decomposes weights and activations of varying bit-widths into 1-bit representations, reconstructing the mixed-precision computation using 1-bit GEMM and GEMV kernels. Zeng et al. [38] leverage NVIDIA Ampere GPU's 1-bit Tensor Cores to accelerate bit-level GEMM and GEMV. Due to the memory overhead and hardware support requirements, LUT-based methods exhibit lower efficiency than padding and bit-disaggregation on edge devices.

However, our investigation reveals that relying solely on

padding or bit-disaggregation fails to accommodate the computational heterogeneity in LLM inference, thereby hindering the inference speed of APQ LLMs on edge devices. Specifically, (1) **Computational heterogeneity between prefill and decode:** LLM inference consists of two phases, prefill and decode. Prefill processes long-sequence prompts, and compute-intensive GEMM is the primary workload. Decode generates tokens sequentially based on Key-Value Cache (KV Cache), and memory-bound GEMV is the dominant operation. Padding and bit-disaggregation techniques demonstrate suboptimal adaptability for the computational and memory patterns of these two stages, resulting in unstable inference performance.

(2) **Intra-Prefill computational heterogeneity:** The arithmetic intensity of GEMM in the prefill phase increases with the sequence length. Our analysis indicates that bit-disaggregation incurs lower overhead for short sequences, while padding demonstrates superior efficiency for long sequences. A static strategy fails to adapt to these dynamic workload variations.

(3) **Intra-Decode computational heterogeneity:** Batching is a typical method adopted to improve decode parallelism. We observe that with increasing decode batch size, the attention layers remain memory-bound, whereas FFN layers transition towards higher arithmetic intensity. Consequently, attention consistently benefits from bit-disaggregation across batch sizes, but FFN requires an adaptive strategy selection.

(4) **Computational heterogeneity in quantization bit-width:** The quantization bit-widths of weights and activations affect the GEMM's arithmetic intensity. Bit-disaggregation outperforms padding at ultra-low bit-widths. Conversely, padding proves more effective for higher bit-widths.

The root cause of these suboptimal behaviors of existing algorithms is the fundamental mismatch between the static, one-size-fits-all nature and the computational heterogeneity inherent in APQ LLM inference. Substantially, this heterogeneity stems from both runtime dynamics and intrinsic model structures. Most importantly, dynamic shifts in sequence length and batch size drastically alter the M dimension. As shown in Figure 1, for a given N and K, the optimal algorithm changes as M varies. Furthermore, considering the tiling mechanisms employed in existing GEMM algorithms, variations in N and K also significantly influence the optimal strategy. Existing static paradigms fail because they cannot adapt to the shifting operational characteristics.

To this end, we propose ADAngel, an adaptive computing mapping system designed to accelerate APQ LLM inference on edge devices. At the heart of ADAngel lies the DPR (Decomposition-Partial Product-Reconstruction) computation model, a unified theoretical abstraction that formalizes diverse mpGEMM software adaptation paradigms (e.g., padding, bit-disaggregation) into a single, generalized description. Guided by DPR, we systematically derive a diverse Computation Strategy Set to cover the spectrum of computational heterogeneity comprehensively. Then, in an offline phase, ADAngel exhaustively profiles these DPR-derived strategies across the target workload space defined by target LLM to construct an Oracle Policy Map. This map is then embedded into a lightweight dispatcher, enabling the engine to dynamically select and execute the optimal kernel for every task at runtime with negligible overhead. For Arbitrary-Precision Quantized Models (e.g., W4A8), ADAngel natively accelerates the underlying mixed-precision GEMM (mpGEMM) operations, delivering maximum performance. For Weight-Only Quantized Models (e.g., W4A16) where activations remain in FP16, activations can be quantized to integers when loading from global memory (e.g., llama.cpp), and benefit from ADAngel's low latency mpGEMM.

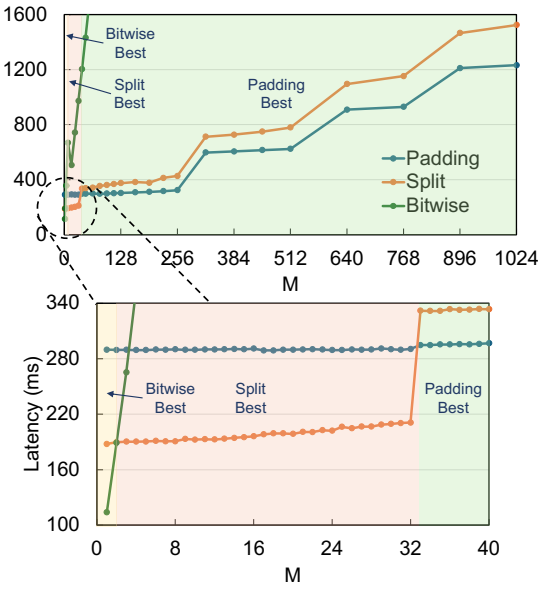


Figure 1: **The performance of three static computation strategies** (Split is our novel strategy, and Bitwise is based on bit-disaggregation) under different M for a typical W4A8 mpGEMM task in LLMs (N = 4096, K = 6144).

In summary, our main contributions are as follows:

(1) We are the first to perform a systematic analysis and characterization of the computational heterogeneity in APQ LLM inference, revealing the fundamental sub-optimality of static approaches.

(2) We propose the DPR (Decomposition-Partial Product-Reconstruction) computation model, a unified theoretical abstraction that establishes bit-level decomposition as a first principle to systematically generate a series of computation strategies for effective mpGEMM acceleration.

(3) We design and implement ADAngel, a methodology for constructing a workload-adaptive computation core for any given LLM through exhaustive offline profiling.

(4) We conduct an extensive evaluation demonstrating that the ADAngel-specialized engine significantly outperforms

state-of-the-art frameworks on both edge and data-center GPUs, achieving speedups of up to 336.78×.

2 Background

2.1 LLM Quantization

The rapid advancement in LLMs (Figure 2, left) enhances model accuracy while substantially increasing computing and storage requirements. For example, deploying DeepSeek-R1-70B with FP16 precision consumes over 140GB of memory, requiring at least two modern high-performance NVIDIA H100 [4] GPUs with 80GB capacity. This poses significant challenges for deploying LLM on low-cost edge devices with limited computing and memory/storage resources. Post-training quantization (PTQ) [15, 30, 32, 35] reduces memory/storage and computing costs by converting weights or activations from high-precision (FP32) to low-precision (INT8/INT4). It has become a mainstream approach for edge-side LLM deployment. Although quantizing model weights or activations enables LLM inference on edge devices, low-bit-width precision reduces model accuracy. Different quantization methods trade off LLM inference speed and quality. Quantization methods can be categorized by granularity into per-tensor, per-channel, and per-group quantization.

Existing studies [26, 36, 41] demonstrate that LLM weights exhibit high redundancy, making them amenable to low-bit-width quantization. In contrast, quantizing activations is more challenging, as low-bit-width activation representation significantly degrades LLM's quality. Arbitrary-precision quantization (APQ) [3, 14, 25] preserves LLM quality while minimizing resource demands by quantizing weights to low-bit-width representations, while maintaining high-precision formats for activations, as shown in Figure 2 (Middle). However, APQ introduces asymmetric bit-widths between weights and activations. Current edge devices cannot execute GEMM and GEMV with asymmetric bit-widths, requiring padding [11, 17], LUT [18, 22, 31], or bit-disaggregation [38].

2.2 Padding and LUT in APQ LLM Inference

As shown in Figure 2 (Right), existing solutions [11, 17] widely adopt padding to implement APQ LLM inference on edge devices. It transforms low-bit-width weights to match activation bit-widths, then performs computation using conventional GEMM/GEMV operators. Although padding enables APQ LLM inference on edge devices, it introduces additional overheads: (1) padding introduces non-negligible runtime bit manipulation overhead during LLM inference. (2) After padding, LLM inference still operates on high-bit-width weights, which increases the memory footprint and computational costs.

Some researchers [18, 22, 31] propose the LUT-based strategy for direct asymmetric-bit-width GEMM/GEMV execution to avoid padding. Specifically, it precomputes the products of high-bit-width activations and multiple low-bit-width weights, and stores them in the LUT. Results are retrieved directly from the LUT instead of performing actual computations during LLM inference. However, constructing a LUT for each weight-activation bit-width combination introduces a significant memory footprint. Moreover, edge hardware provides limited support for LUTs, making LUT-based methods inferior to padding.

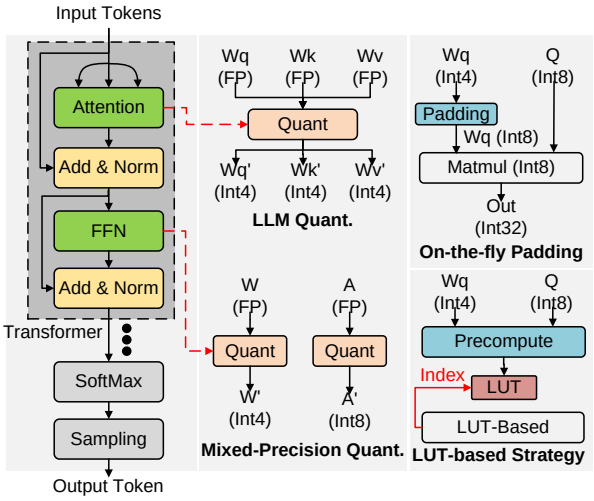


Figure 2: (1) Left: Decoder-only LLM architecture; (2) Middle: LLM quantization and arbitrary-precision quantization (APQ); (3) Right: Padding and the LUT-based strategy in APQ LLM inference.

2.3 Bit-Disaggregation GEMM and GEMV

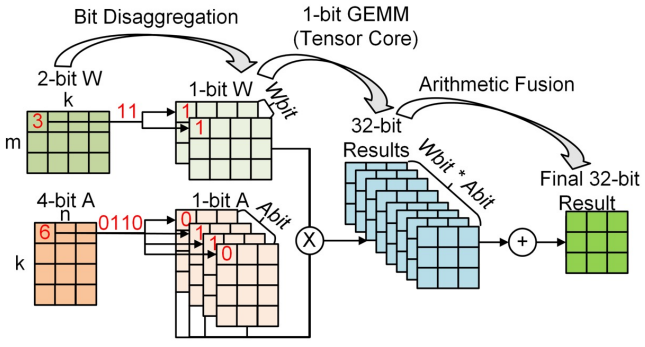
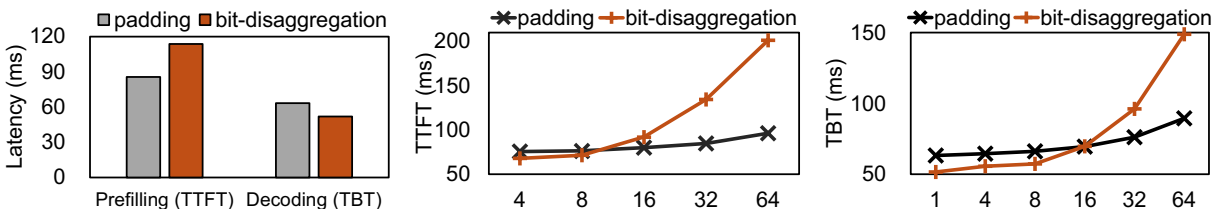


Figure 3: W2A4 GEMM bit-disaggregation workflow.

Bit-disaggregation is a mathematical approach for directly executing GEMM and GEMV with asymmetric bit-widths. As shown in Figure 3, it first disaggregates the weight matrix $[M,K]$ and activation matrix $[K,N]$ into $WBit \times [M,K]$



(a) Prefill and Decode (b) Prompt Length (c) Decode Batch Size

Figure 4: The computational heterogeneity between prefill and decode, intra-prefill computational heterogeneity, and intra-decode computational heterogeneity.

and $ABit \times [K,N]$ 1-bit matrices via bit-level disaggregation. $WBit$ and $ABit$ represent the bit-width of weights and activations. Next, pairwise multiplications are performed between the 1-bit weight and activation matrices, yielding $WBit \times ABit$ result matrices. Finally, the $WBit \times ABit$ result matrices are combined into the final output. Equation 1 formally captures the bit-disaggregation process described above. It increases the number of GEMM and GEMV operations by a factor of $WBit \times ABit$, while also introducing additional fusion overhead. Zeng et al. [38] leverage 1-bit Tensor Cores in NVIDIA Ampere architecture GPUs to accelerate 1-bit GEMM and GEMV, thereby optimizing the inference speed of APQ LLMs that adopt the bit-disaggregation strategy.

$$Result = \sum (2^0 W^{WBit-1} + ... + 2^{WBit-1} W^0) \times (2^0 A^{ABit-1} + ... + 2^{ABit-1} A^0) \quad (1)$$

3 Motivation

Although padding and bit-disaggregation enable existing edge devices to support APQ LLM inference, we observe that using either alone fails to effectively capture the computational heterogeneity in LLM inference, which leads to suboptimal inference performance. Next, we provide a detailed discussion to present our insights on the computational heterogeneity in APQ LLM inference.

3.1 Computational Heterogeneity Between Prefill And Decode

First, we deploy Llama2-7B on the edge GPU of NVIDIA Jetson AGX Orin [20] using FasterTransformer [6]. The model adopts W4A8 quantization (4-bit weights, 8-bit activations). For W4A8 support, FasterTransformer's GEMM and GEMV operations are reimplemented using padding and bit-disaggregation techniques. Figure 4a presents the prefill and decode latency of the model when using padding and bit-disaggregation approaches, under the following configuration: sequence length (seq_len) = 32, generated sentence length (gen_len) = 10, and batch size (batch) = 1. Prefill latency is measured as Time To First Token (TTFT), while decode latency is recorded as Time Between Tokens (TBT).

The results show that prefill TTFT with the padding method outperforms bit-disaggregation. In contrast, the decode TBT with bit-disaggregation is better than padding. The underlying reason is that prefill involves compute-intensive GEMM operations, where computing is the primary bottleneck. Bit-disaggregation significantly increases computational overhead, leading to poor performance. In contrast, decode is I/O-intensive GEMV, where Streaming Multiprocessors (SMs) in the GPU are underutilized. The additional computing introduced by bit-disaggregation improves SM utilization, resulting in optimized performance. **Observation 1: Using only padding or bit-disaggregation fails to address the computational heterogeneity between prefill and decode, leading to suboptimal APQ LLM inference performance.**

3.2 Intra-Prefill Computational Heterogeneity

Based on the configuration in Section 3.1, we increase the prefill length (seq_len) from 4 to 64 and measure the TTFT for Llama2-7B's prefill. As shown in Figure 4b, the TTFT exhibits a monotonic increase with growing prefill lengths for both padding and bit-disaggregation approaches. We observe a crossover at a prefill length of 8: bit-disaggregation achieves a lower TTFT for short lengths (≤ 8), while padding demonstrates superior efficiency beyond this threshold. This is because GEMM operations in Llama2-7B prefill exhibit insufficient parallelism for prefill lengths ≤ 8, leaving GPU SMs underutilized. Bit-disaggregation mitigates this limitation by increasing computational intensity, which raises SM occupancy and reduces TTFT. For prefill lengths $>$ 8, SM resources become saturated, and the added computations from bit-disaggregation exacerbate this bottleneck, increasing latency. padding, with its lower computational overhead, improves TTFT in such cases. Additionally, although the performance crossover point between padding and bit-disaggregation emerges at prefill length $= 8$ in our experiments, this point demonstrates substantial variability across model architectures, quantization bit-widths, and batch sizes. **Observation 2: Relying solely on padding or bit-disaggregation individually cannot fully account for the computational heterogeneity within prefill in APQ LLM inference.**

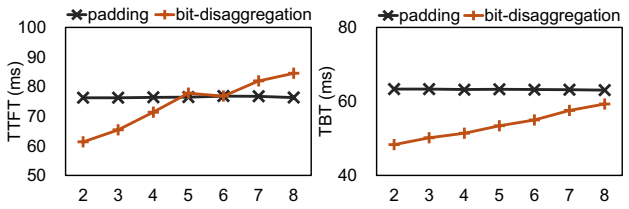
3.3 Intra-Decode Computational Heterogeneity

Next, we investigate the computational heterogeneity within decode. Figure 4c presents the TBT of Llama2-7B decode with padding and bit-disaggregation strategies for batch sizes 1∼64. All other parameters follow the settings in Section 3.1.

The performance crossover point between padding and bit-disaggregation occurs at batch size 16. This crossover point is non-deterministic—it varies with model architectures and quantization bit-widths. While the self-attention in decode retains the GEMV pattern (favoring bit-disaggregation, Section 3.1), the FFN shifts from GEMV to GEMM as batch sizes increase, significantly increasing computational overhead. For small batch sizes, the GEMV in self-attention and GEMM in FFN leave SM resources underutilized. Bit-disaggregation improves SM occupancy, thereby optimizing TBT. However, as the batch size increases, the GEMM in FFN gradually saturates the SMs, making the computational overhead of bit-disaggregation the bottleneck. In contrast, padding achieves lower TBT. **Observation 3: Both padding and bit-disaggregation fail to comprehensively handle the computational heterogeneity within decode in APQ LLM inference.**

3.4 Computational Heterogeneity in Quantization bit-width

Finally, with the activation bit-width fixed at 8 bits and the weight bit-width varying from 2 to 8 bits (other settings as in Section 3.1), we evaluate the prefill TTFT and decode TBT for both padding and bit-disaggregation methods. Figure 5 demonstrates that TTFT and TBT remain stable with the padding method as the weight bit-width increases. This stability occurs because padding scales all weights to 8-bit precision (matching the activations) and processes them using INT8 Tensor Cores, introducing a constant computational overhead independent of weight bit-width. In contrast, bit-disaggregation introduces progressively more 1-bit matrix computations as weight bit-width increases, resulting in increased TTFT and TBT. During prefill, bit-disaggregation achieves lower TTFT than padding at small weight bit-widths by exploiting underutilized GPU SMs. However, increasing the bit-width makes bit-disaggregation's computational overhead dominant, ultimately yielding higher TTFT than padding. Since GEMV dominates decode computations, bit-disaggregation's computational cost scales more slowly with weight bit-width than prefill's GEMM, leading to superior TBT performance over padding. **Observation 4: Neither padding nor bit-disaggregation alone can capture the computational heterogeneity introduced by the quantization bit-width in APQ LLMs.**



(a) Prefill (b) Decode

Figure 5: **Computational heterogeneity in quantization bit-width**. The activations are fixed at 8-bit quantization, while the weights vary from 2-bit to 8-bit. The x-axis represents the weight quantization bit-width.

4 Design

4.1 Overall Architecture

Our preceding analysis reveals that workload-unaware computation paradigms are fundamentally sub-optimal for APQ LLM inference, failing to address the widespread computation heterogeneity of mpGEMM in LLM inference. To achieve high-performance inference, therefore, a new paradigm of adaptivity to both the computational workload and quantization bit-widths is needed, whose adaptivity is predicated on two core principles: (1) **Optionality**, the availability of a diverse portfolio of kernels that leverage hardware resources differently, and (2) **Optimality**, the capability to select the optimal kernel for each specific task.

To realize this paradigm, we propose **ADAngel**, a workload-aware framework, which is designed to accelerate APQ LLM inference, as shown in Figure 6. The theoretical foundation of ADAngel is the ❶ **DPR** (Decomposition–Partial Product–Reconstruction) computation model, which provides a unified form to both express and implement a diverse spectrum of mixed-precision GEMM strategies. Guided by the DPR model and hardware characteristics, we first define a set of three canonical strategies (Padding, Split, Bitwise) and perform deep, manual optimizations on their kernel implementations to form the high-performance ❷ **Computation Strategy Set**. Subsequently, the ❸ **Strategy Dispatcher** exhaustively evaluates these strategies offline on every potential workload within any given target LLM. This yields an oracle policy map which maps workloads to strategies, enabling the Strategy Dispatcher to perform zero-overhead, adaptive strategy selection at runtime when the LLM inference engine needs to execute mpGEMM/mpGEMV.

4.2 DPR Framework

The performance upper bound of an adaptive system is fundamentally determined by the quality and diversity of its available strategies. Based on this principle, our framework requires a rich portfolio of high-performance computation

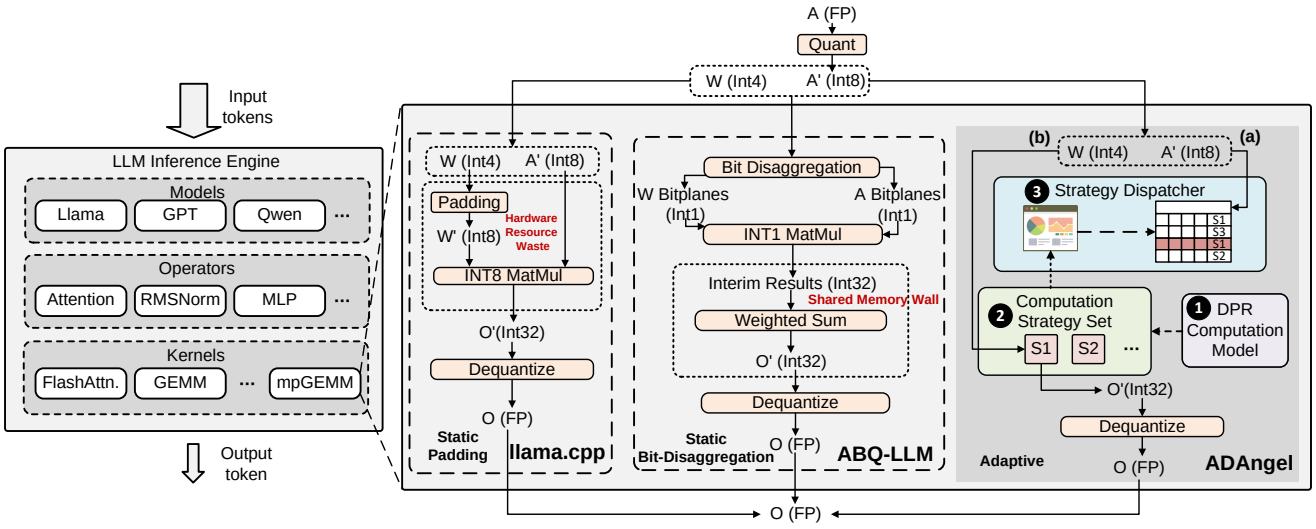


Figure 6: **The Overview of ADAngel**. Contrasting with traditional static approaches, namely llama.cpp (which incurs hardware resource waste due to weight upcasting) and ABQ-LLM (which hits a shared memory wall during prefill due to bit-disaggregation, detailed in Section 6.2), ADAngel is built upon the (1) DPR Computation Model. By analyzing hardware architecture, a (2) Computation Strategy Set can be constructed based on DPR constructs, and a (3) Strategy Dispatcher is employed to map each mpGEMM task to the optimal computation strategy during APQ LLM inference. As indicated by the dashed paths (offline stage), ADAngel designs the strategy set under DPR guidance and evaluates these strategies on the target LLM to generate a dispatch table. As indicated by the solid arrows (runtime stage), ADAngel (a) retrieves the appropriate strategy based on the problem size and bit-width of the mpGEMM task, and (b) invokes the corresponding kernel to complete the computation.

strategies to be effective. The **DPR** (Decomposition–Partial Product–Reconstruction) computation model serves as the theoretical cornerstone for this task. That is, the strategies in our Computation Strategy Set are designed under the guidance of the DPR model, and the computation kernels dispatched and executed at runtime are concrete instantiations of it. DPR formally deconstructs any mixed-precision matrix multiplication into three logical stages: the framework first decomposes the input matrices into multiple components based on their bit-level representation; next, it performs matrix multiplication on these components to obtain a series of partial products; finally, it accurately reconstructs the final result via a weighted summation.

We now formally define this three-stage process. Given an activation matrix $X \in \mathbb{Z}_2^{M\times K}$ with bit-width $XBITS$ and a weight matrix $W \in \mathbb{Z}_2^{K\times N}$ with bit-width $WBITS$, our DPR framework computes the mpGEMM result $Y = XW \in \mathbb{Z}_2^{M\times N}$ as follows:

Decomposition. Its goal is to transform matrices X and W to corresponding physical tensors $\{X_{i,phys}\}$ and $\{W_{j,phys}\}$, which are uniformly typed and ready for computation by hardware accelerators like Tensor Cores in the next stage. It encapsulates two sub-steps, logical bit-partitioning and physical representation mapping.

We begin with defining an operation **logical bit-partition**: For a matrix A with $ABITS$ bit-width, based on a given bit-partitioning schema:

$$P_A = (q_1, q_2, \ldots, q_m), \text{where} \sum_{i=1}^{m} q_i = ABITS, \tag{2}$$

the value of any element x in A is precisely expressed as a weighted sum of the corresponding elements x_i:

$$x = \sum_{i=1}^{n} x_i \cdot 2^{\text{shift}(i)}, \text{ where } \text{shift}(i) = \sum_{j=1}^{i-1} q_j. \tag{3}$$

Thus matrix A can be precisely expressed as a weighted sum of matrices A_i:

$$A = \sum_{i=1}^{n} A_i \cdot 2^{\text{shift}(i)}, \text{ where } \text{shift}(i) = \sum_{j=0}^{i-1} q_j, \tag{4}$$

then $\{A_i\}$ is the **logical bit-partition** of A based on the given bit-partitioning schema P_A.

In the logical bit-partitioning step, respectively, $P_X = (q_{X1}, q_{X2}, \ldots, q_{Xm})$ and $P_W = (q_{W1}, q_{W2}, \ldots, q_{Wn})$ are applied to X and W, generating their logical partitions $\{X_s\}_{s=1}^{m}$ and $\{W_t\}_{t=1}^{n}$. By applying different bit-partitioning schemes, this step provides a high degree of flexibility, allowing us to conceptually decompose matrices into logical components of arbitrary bit-widths. However, the underlying hardware acceleration units, particularly Tensor Cores, are not general-purpose multipliers. They are highly specialized computation

engines, optimized for a fixed set of symmetric input precision pairs. This creates a mismatch between the flexibility of the logical decomposition at the software level and the rigid constraints of the hardware execution units. For instance, a 4-bit logical component and an 8-bit logical component cannot be multiplied directly by any single, native Tensor Core instruction.

To bridge this gap, the Physical Representation Mapping stage is an essential step. Its core mission is to unify all potentially heterogeneous logical matrices onto a single, homogeneous computation data type that is natively supported by the hardware. This is achieved through our Hardware-Aware Promotion Principle. The process first determines the maximum bit-width across all logical partitions, $b_{\max}$. It then promotes this value to the smallest, natively supported hardware precision that can accommodate it. Let $H = \{1, 4, 8\}$ be the set of discrete bit-widths supported by the hardware. The final target bit-width can be expressed as

$$b_{\text{target}} = \min\{h \in H | h \geq \max\{q_{X1}, \ldots, q_{Xm}, q_{W1}, \ldots, q_{Wn}\}\}. \tag{5}$$

Subsequently, every logical matrix A_i (representing any W_s or X_t) is physically realized into a tensor $A_{i,phys}$ by promoting it to this target bit-width. This ensures that all tensors entering the computation stage are of a uniform, hardware-supported precision.

The resulting physical tensors from the decomposition stage are stored contiguously in memory, which maximizes bandwidth through coalesced accesses and allows us to fuse multiple partial product computations into a single kernel launch (e.g., computing $X_1 \times W_1$ and $X_2 \times W_1$ simultaneously), significantly reducing dispatch overhead.

Partial Product Computation. The output of the Decomposition stage is a set of hardware-ready physical tensors, $\{W_{s,phys}\}$ and $\{X_{t,phys}\}$, which all share a uniform bit-width, b_{target}. The goal of the Partial Product Computation stage is to perform the actual matrix multiplication on these prepared tensors.

Specifically, this stage executes $m \times n$ multiplications on the pairs of physical tensors to yield a series of partial product matrices, Y_{st}:

$$Y_{st} = \text{ComputeOp} < b_{target} > (W_{s,phys}, X_{t,phys}). \tag{6}$$

This stage's computational homogeneity ensures all operations are mapped to a single type of native hardware instruction (e.g., IMMA or BMMA) on the Tensor Cores, which maximizes throughput by eliminating functional unit contention and maintaining a saturated execution pipeline.

It generates $m \times n$ partial product matrices, each of dimension $M \times N$. Crucially, the elements of these matrices are 32-bit integers (int32). This high precision is dictated by the architecture of the Tensor Cores, and exposes a fundamental performance trade-off: a more fine-grained decomposition (larger m or n) results in a proportionally larger memory footprint for these int32 intermediate results. This increased pressure on on-chip resources (especially shared memory) can constrain the achievable parallelism and occupancy of the GPU, a cost that can completely offset the inherent latency and throughput advantages of lower-precision Tensor Cores.

Reconstruction. In the final stage, our objective is to compute the result matrix Y. With the distributive property of multiplication, the product $Y = XW$ can be expanded from their logical decompositions into a weighted sum of partial products:

$$\begin{aligned} Y = XW &= (\sum_{s=1}^{m} X_s \cdot 2^{\text{shift}(s)})(\sum_{t=1}^{n} W_t \cdot 2^{\text{shift}(t)}) \\ &= \sum_{s=1}^{m}\sum_{t=1}^{n} X_s W_t \cdot 2^{\text{shift}(t)+\text{shift}(s)}. \end{aligned} \tag{7}$$

In summary, the DPR computation model provides a formal and principled foundation for mixed-precision GEMM. It establishes a clear methodology to deconstruct a complex, heterogeneous computation into a series of simple, hardware-aligned homogeneous operations.

4.3 Computation Strategy Set

The purpose of the Computation Strategy Set is to provide ADAngel with a broad and high-performance portfolio of computation strategies. Although the representational capacity of our DPR model provides a vast theoretical design space for this set, a significant portion of these strategies are practically inefficient. This inefficiency stems from the redundant promotion overhead introduced by our Hardware-Aware Promotion Principle when a strategy's logical bit-width does not align with the discrete precisions natively supported by the GPU's Tensor Cores.

To make this concrete, consider a W4A8 matrix multiplication where the activation partition is fixed at $P_A = (8)$. A weight decomposition of $P_W = (2,2)$ is provably suboptimal compared to the trivial $P_W = (4)$ scheme, because the $(2,2)$ scheme decomposes the work into two separate W8A8 GEMM operations, effectively doubling the computational workload.

Therefore, our method is to prune the design space by minimizing the promotion overhead, which is achieved by aligning the decomposed bit-widths with the native Tensor Core precisions. Based on this principle, we designed three families of computation strategies.

Throughput-Oriented: Padding. This strategy's philosophy is to achieve the highest possible computational throughput. It is formally expressed in our DPR model by a trivial partitioning scheme (i.e., $P_A = (ABITS)$) which maps the computation to the lowest-precision native hardware unit capable of accommodating the largest operand (e.g., INT8 Tensor

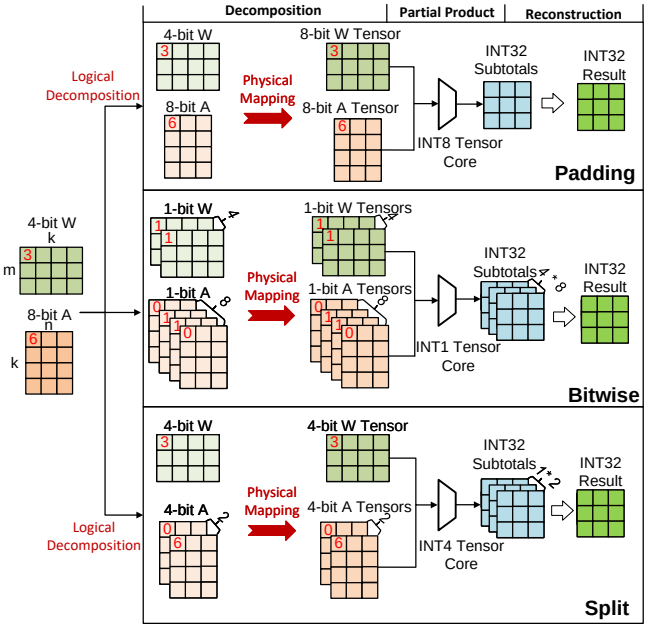


Figure 7: **Visualizing the DPR Framework**. For a canonical W4A8 task, how different Logical Decompositions and Physical Mappings, guided by our DPR model, result in three distinct hardware-aligned computation strategies.

Cores), making the entire operation into a single, powerful hardware instruction.

Latency-Oriented: Bitwise. This strategy is designed to achieve the lowest possible operational latency. It corresponds to the finest-grained decomposition scheme in our DPR model (i.e., $P = (1, 1, \ldots, 1)$), replacing conventional arithmetic with a massive number of simple bit-logic operations.

Balanced: Split. This novel strategy provides a balanced solution between the two extremes. It is enabled by an intermediate, hardware-aligned partitioning scheme within our DPR model (e.g., $P_A = (4, 4)$ for an 8-bit activation). This provides a crucial sweet spot, avoiding the wasted memory read of the Padding approach, while not incurring the instruction and shared memory overhead of a Bitwise approach.

To make the application of our three strategies concrete, we now instantiate them for a W4A8 mixed-precision task. The selection of the bit-partitioning scheme for each archetype, and the resulting hardware computation path dictated by our DPR model, are summarized in Figure 7.

For the above three strategy prototypes, to achieve the highest performance, it is also necessary to implement them at the kernel level. We have highly optimized all kernels in our policy set, including: (1) Fusion, which aims to reduce expensive global memory traffic. We selectively fuse adjacent operations within the DPR pipeline. A key optimization in all our strategies is the fusion of the final Reconstruction and Dequantization stages into a single CUDA kernel. This ensures that the intermediate int32 accumulator results are never written back to global memory, but are instead converted directly to the final floating-point output. (2) Weight Swizzling, where weights are pre-processed into a hardware-friendly memory layout to maximize memory coalescing; and (3) Contiguous Data Layouts for all decomposed tensors.

In summary, guided by the theoretical principles of our DPR computation model, we have designed and implemented three distinct, highly-optimized computation strategies that constitute our Computation Strategy Set. This portfolio of kernels provides the rich and diverse action space that is the essential foundation for ADAngel's adaptivity.

4.4 Strategy Dispatcher

The final component of ADAngel framework is responsible for fulfilling the Optimality requirement: selecting the best-performing kernel from our Strategy Set for any given task. Our approach is grounded in a key insight: for a specific LLM, the variety of its constituent GEMM operations is highly constrained. This observation makes an exhaustive offline profiling approach not only feasible but also optimal. Therefore, our design consists of two phases: an offline policy construction phase that performs the analysis to generate an oracle policy map, and a lightweight Online Static Dispatch phase that executes mpGEMM tasks with the best strategy selected by the policy map.

We analyze the GEMM workload space within a target LLM. While the M dimension of a GEMM is dynamic, the N and K dimensions are static, determined by the model's architecture (e.g. hidden dimension). Consequently, the number of unique GEMM "types" (defined by static N and K dimensions) is small and finite. For instance, our analysis shows that an entire Llama-3-8B model inference involves only about 7 unique types of GEMM operations.

This bounded problem space makes an exhaustive, offline profiling approach conceivable, but its practical feasibility depends on the storage cost of the resulting lookup table. We quantify this cost as follows: storing function pointers for 7 unique GEMM types across a comprehensive range of 2048 sequence lengths (requiring 8 bytes per pointer) results in a total size of approximately 112 KB. This trivial memory footprint confirms that our empirical performance oracle approach is not only theoretically sound but also eminently practical.

The oracle policy map is constructed through an exhaustive, empirical profiling process executed once per target LLM and hardware platform. To achieve this, we first define the complete workload space by performing a static analysis of the target model's architecture to identify its finite set of unique GEMM "types", and then sweeping across a comprehensive range of the dynamic M dimension. For every point in this space, we benchmark each kernel from our Computation Strategy Set, using high-precision timers after a sufficient warm-up period to ensure accuracy. The kernel with the lowest average latency for each specific task is recorded, creating the oracle

policy map: a simple key-value structure that maps a workload to a function pointer for its empirically proven optimal kernel.

The runtime behavior of the Strategy Dispatcher is simple and efficient. Upon receiving a GEMM task, it performs a direct lookup in the pre-computed oracle policy map using the task's shape as a key. This retrieves and immediately invokes a pointer to the empirically proven optimal kernel.

This dispatch mechanism, being a simple table lookup, is what provides ADAngel's two decisive advantages: guaranteed optimality (relative to our offline analysis) and negligible runtime overhead.

5 Implementation

We implemented the ADAngel framework in C++ and CUDA 12.6 with over 15k LOC. Our implementation first involved engineering the Computation Strategy Set: a portfolio of kernels where Padding and Split are built upon CUTLASS [8], and Bitwise is a custom BMMA-based implementation. With this optimized portfolio established, we then constructed the Strategy Dispatcher's policy through an automated, exhaustive offline profiling process. Finally, we integrated these components as an mpGEMM backend into a state-of-the-art inference framework for end-to-end evaluation.

High-Performance Kernel Engineering. To ensure our strategy portfolio is highly competitive, we engineered each kernel family for maximum performance. The arithmetic-based strategies (Padding, Split) were developed using the CUTLASS 3.1.0 library as a foundation, and the Bitwise strategy was implemented as a custom high-performance CUDA kernel based on BMMA instructions, including selective operator fusion (e.g., fusing Reconstruction and Dequantization) and weight pre-processing into hardware-friendly memory layouts.

Offline Policy Construction. The Strategy Dispatcher's policy was constructed by an automated Python script that orchestrates the entire empirical profiling workflow. It benchmarked every kernel in our strategy set across a comprehensive range of sequence lengths in target LLMs, using CUDA events for precise latency measurement. By selecting the lowest-latency kernel for each potential workload, this process generated the Optimal Policy Map.

End-to-End Framework Integration. To evaluate real-world performance, we integrated our ADAngel computation core into NVIDIA's FasterTransformer. We further implemented support for the widely used model Llama3-8B within our FasterTransformer-based testbed. The integration is modular, requiring only redirecting the framework's standard mpGEMM calls to our ADAngel dispatcher.

6 Evaluation

6.1 Experiment Setup

Platform. Experiments were conducted on an NVIDIA Jetson AGX Orin developer kit 64GB, featuring an Ampere-architecture GPU (sm_87) and 64GB unified memory. We used the NVIDIA JetPack 6.1 [7] software stack (CUDA 12.6). The power mode of Orin is MODE_50W. Our entire testbed is built upon FasterTransformer, and we implemented W4A8 quantized versions of Llama3 within this testbed. To validate the generality of the ADAngel methodology, we applied ADAngel to a data-center-class NVIDIA A100 GPU [5], generating a new, A100-specific engine.

Models. We used a representative modern LLM: Llama3-8B, quantized to W4A8.

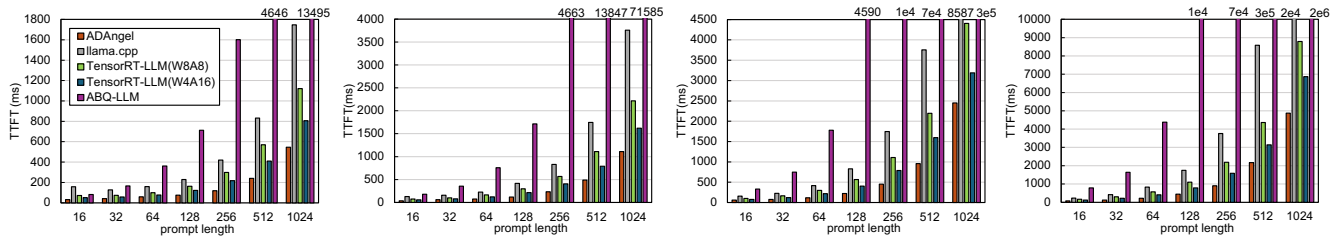
Baselines and Our System: (1) ABQ-LLM [38], serves as a representative framework for bit-disaggregation-based mpGEMM. We integrated its specialized kernels into our testbed, replacing the standard computation routines. (2) llama.cpp [17]. This serves as a highly optimized, end-to-end baseline utilizing online padding mpGEMM routines. (3) TensorRT-LLM. As a recognized high-performance inference framework, TensorRT-LLM serves as a strong baseline. Since the current version for the Orin platform lacks native W4A8 support, we employ W8A8 (SmoothQuant) and W4A16 (AWQ) quantization methods for a comprehensive comparison. (4) QServe. Its W4A8KV4 configuration is a powerful solution for cloud inference but is currently unsupported on Orin. To address this, we compare ADAngel with QServe on an A100 80GB GPU to evaluate the scalability of our approach. (5) ADAngel. This is our main proposal. All mpGEMM operations in our testbed are handled by our ADAngel-specialized computation core.

Evaluation Metrics: We report (1) Time-To-First-Token (TTFT) to measure the latency of the Prefill stage, and (2) Tokens Per Second (TPS) to measure the throughput of the Decode stage. All presented results are the average of multiple runs following a warm-up period to ensure stable and accurate measurements.

6.2 End-to-end Evaluation

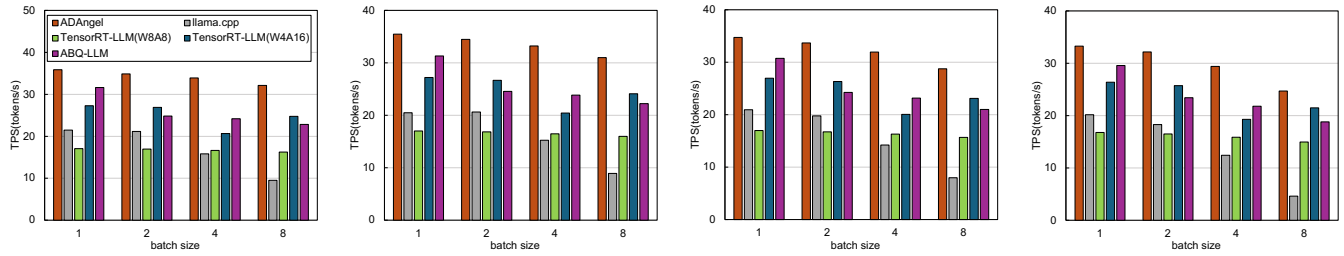
Prefill. Figure 8 illustrates the end-to-end prefill latency (TTFT) across varying prompt lengths from 16 to 1024. ADAngel consistently establishes superior performance, achieving a peak speedup of 336.78× over ABQ-LLM and a speedup ranging from 1.29× to 1.86× over TensorRT-LLM.

Compared to TensorRT-LLM configurations, ADAngel surpasses the W4A16 baseline by leveraging the superior arithmetic throughput of INT8 Tensor Cores over FP16, and exceeds the W8A8 baseline by virtue of a lightweight system architecture that eliminates the significant runtime overheads inherent in the TensorRT-LLM framework.



(a) Llama3-8B, batch size = 1 (b) Llama3-8B, batch size = 2 (c) Llama3-8B, batch size = 4 (d) Llama3-8B, batch size = 8

Figure 8: **Prefill Stage Performance Comparison**. ADAngel significantly reduces the TTFT across various models and input sequence lengths, demonstrating its superiority in compute-bound scenarios.



(a) Llama3-8B, pl = 128 (b) Llama3-8B, pl = 256 (c) Llama3-8B, pl = 512 (d) Llama3-8B, pl = 1024

Figure 9: **Decode Stage Performance Comparison**. ADAngel achieves substantially higher TPS during the auto-regressive generation phase, showcasing its efficiency in memory-bound, low-latency operations.

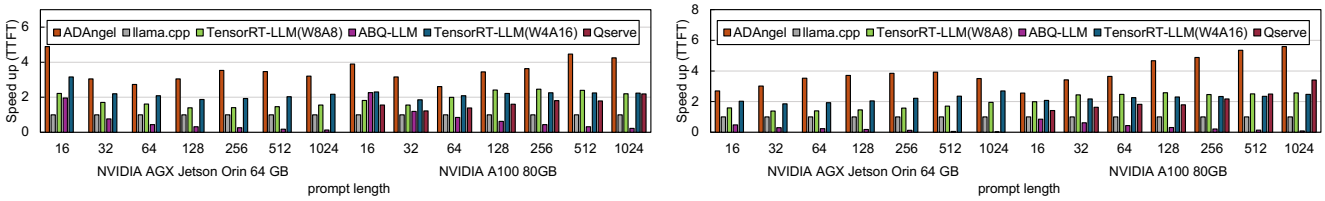
In short prompt length regimes, static padding-based methods suffer from inherent bandwidth redundancy due to upcasting 4-bit weights. Conversely, ADAngel activates the Bitwise and Split strategies to preserve the compressed storage format during memory access, effectively bypassing the bandwidth bottleneck. For instance, at bs $= 1$ and pl $= 32$, ADAngel reduces latency by 67.5% compared to llama.cpp (41 ms vs. 126 ms).

As the workload transitions to the compute-bound long-sequence regime, the bit-disaggregation approach (ABQ-LLM) consumes up to 32$\times$ more shared memory per thread block compared to padding-based approaches. This bottleneck, identified as the Shared Memory Wall, severely restricts GPU occupancy due to excessive intermediate accumulation states. Consequently, although ABQ-LLM is the SOTA framework for decoding and short-sequence prefill, it becomes practically unusable in this setting: at bs $= 8$ and pl $= 1024$, its TTFT surges to approximately 27.3 minutes. The root cause is the Shared Memory Wall induced by the storage of int32 intermediate states. Our profiling reveals that this heavy shared memory consumption saturates the SM resources, restricting the hardware scheduler to launch only 3 thread blocks per SM. Such low occupancy effectively nullifies the massive parallelism advantage of the GPU. In contrast, ADAngel adaptively switches to the appropriate kernel to circumvent this latency explosion, maintaining the latency within seconds (4.87 s) and achieving a substantial 336.78$\times$ speedup.

Decode. Figure 9 reports the decode throughput (TPS, tokens per second) across batch sizes from 1 to 8, where ADAngel consistently establishes superior performance by effectively managing the trade-offs between redundancy and resource consumption. In the critical single-batch regime ($M = 1$), ADAngel achieves a 2.10$\times$ speedup over padding-based method TensorRT-LLM W8A8 by eliminating two fundamental inefficiencies: it bypasses the physical bandwidth redundancy of upcasting 4-bit weights to 8-bit, and simultaneously removes computational redundancy, where static padding kernels constrained by fixed INT8 Tensor Core shapes (typically $M = 16$) leave $\frac{15}{16}$ (93.75%) of compute cycles idle on zero-padding. As the batch size scales ($M = 2 \sim 8$), while pure bit-disaggregation (ABQ-LLM) also circumvents these redundancies, it does so at the cost of excessive shared memory usage and instruction bloat; in contrast, ADAngel's Split strategy strikes an optimal balance between eliminating redundancy and minimizing memory cost, thereby avoiding the resource bottlenecks encountered by bitwise methods and ultimately surpassing TensorRT-LLM W8A8 by 1.97$\times$ on average.

6.3 Generality Evaluation

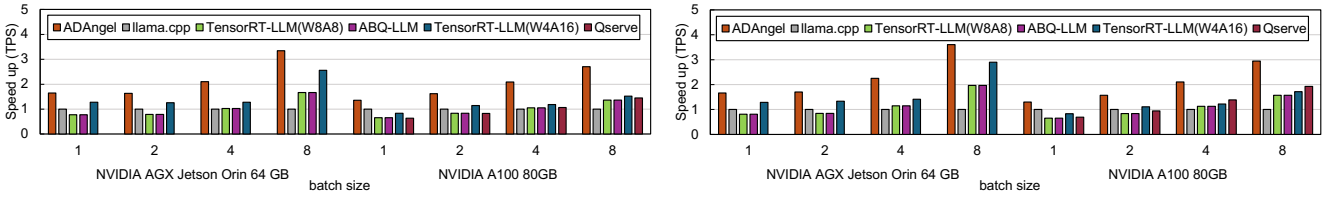
Although ADAngel is primarily designed for resource-constrained edge scenarios, such as intelligent cockpits and robotics, the increasing adoption of arbitrary-precision quanti-



(a) Llama3-8B, batch size = 1

(b) Llama3-8B, batch size = 4

Figure 10: **Prefill Stage Performance Speedup Across Hardware Platforms**. All results are normalized to the llama.cpp baseline on each respective platform (i.e., llama.cpp = 1.0×). ADAngel consistently outperforms both baselines.



(a) Llama3-8B, prompt length = 64

(b) Llama3-8B, prompt length = 512

Figure 11: **Decode Stage Performance Speedup Across Hardware Platforms**. All results are normalized to the llama.cpp baseline on each respective platform. ADAngel maintains its performance leadership on both the Orin and the A100 across all tested batch sizes.

zation in cloud inference highlights its potential contribution to server-grade GPU efficiency. Figures 10 and 11 demonstrate this platform scalability by extending our evaluation to the NVIDIA A100 GPU (80GB). While QServe could not be executed on the Jetson AGX Orin due to a lack of support, we successfully deployed it on the A100 to enable a direct comparison. QServe employs strategies such as progressive group quantization tailored for high-throughput optimization; while the associated overheads of these strategies are effectively hidden by massive parallelism in large-batch scenarios, they remain exposed in small-batch regimes. In contrast, ADAngel's adaptive approach successfully identifies performance bottlenecks in these specific scenarios and applies optimization, achieving speedups of 2.12× in TTFT and 1.72× in TPS relative to QServe. Furthermore, thanks to its hardware-aware adaptive mechanism, ADAngel maintains its performance advantage over TensorRT-LLM, llama.cpp, and ABQ-LLM on the A100 platform.

6.4 Ablation Study

To evaluate the specific contributions of our fine-grained dispatching mechanism and diverse strategy portfolio, we constructed three distinct ablation baselines:

(1) ADAngel-*: To disentangle the contribution of our core adaptive methodology from the benefits of low-level kernel optimizations, we evaluated systems using a single static strategy, denoted as ADAngel-*, where (*) represents the specific fixed strategy used (e.g., ADAngel-Padding). This serves as a control group to isolate the gains attributed solely to dynamic scheduling.

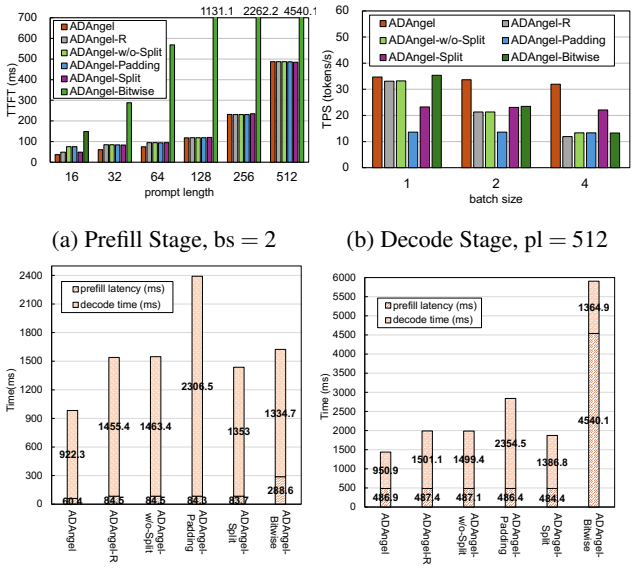
(2) ADAngel-R: To assess the necessity of our fine-grained dispatcher, we compared ADAngel against a strong rule-based baseline. This variant employs a static heuristic based solely on M: mapping tasks with $M \in [1,8]$ to Bitwise, $M \in (8,64]$ to Split, and $M > 64$ to Padding.

(3) ADAngel-w/o-Split: To evaluate the criticality of our diverse strategy portfolio, we implemented an ablated variant that excludes the Split strategy from the candidate pool.

ADAngel maintains a consistent performance advantage across all ablation settings. In extreme regimes where the optimal strategy is static, such as single-batch decoding or long-sequence prefilling, ADAngel correctly identifies the optimal kernel with negligible dispatch overhead (total $<$ 3 ms).

Specifically, ADAngel-R achieves gains over static Padding by selecting Split at pl $= 16$ during prefill and Bitwise during decode; however, due to ignoring the dimensional heterogeneity introduced by N and K, it underperforms the full ADAngel system in decode scenarios (bs $= 2 \sim 4$), highlighting the critical need for fine-grained kernel selection that considers the full (M,N,K) for every GEMM.

Furthermore, ADAngel-w/o-Split, operating on a restricted strategy set, defaults to Padding behavior in prefill to avoid the latency explosion of the Bitwise strategy, yet sacrifices the significant speedup opportunities at pl $= 16 \sim 64$ provided by



(a) Prefill Stage, bs = 2 (b) Decode Stage, pl = 512

(c) Total Time, bs = 2, pl = 64 (d) Total Time, bs = 2, pl = 512

Figure 12: **Performance breakdown and ablation study of ADAngel compared with different variants**. (a) Time to First Token (TTFT) across varying prompt lengths with batch size bs = 2. Note that ADAngel-Bitwise suffers from high latency in longer sequences. (b) Generation throughput (TPS) across different batch sizes with pl = 512. (c) & (d) Breakdown of total end-to-end latency (prefill latency vs. decode time) for short sequences (pl = 64) and long sequences (pl = 512) at bs = 2, with 32 tokens generated. ADAngel achieves the lowest total latency by balancing prefill and decode efficiency.

Split. Similarly, in the decode phase (bs = $2 \sim 4$), although it outperforms static Padding, it still lags behind ADAngel due to the absence of the Split strategy, which uniquely balances parallelism starvation against the Shared Memory Wall.

Since the benefits of ADAngel manifest across different phases, we conducted an end-to-end evaluation simulating typical intelligent cockpit scenarios (instruction parsing and short QA: generating 32 tokens with bs = 2 and pl = 64/512). The measured total latency demonstrates that ADAngel outperforms all ablation baselines across all scenarios, leveraging its effective fine-grained adaptive mechanism and a rich set of computational strategies.

7 Related Work

Post-Training Quantization (PTQ) is a key technique for compressing LLMs. Seminal methods like GPTQ [12] use second-order information for error compensation, a direction refined by works like HQQ [1]. To handle outliers, AWQ [16] protects salient weights via scaling, while other works preprocess weights via smoothing (MagR [39]) or orthogonal transformations (e.g., QuIP [2]), learn the error with adapters (e.g., QLoRA [10]), or dynamically correct errors at runtime (DecDEC [23]). To further enable integer-only hardware, another line of work performs weight-activation quantization. SmoothQuant [33] pioneered this by migrating activation difficulty to the weights, paving the way for advanced methods that reorder activations (RPTQ [37]) or reshape distributions with learnable transformations (FlatQuant [28]). While these methods produce accurate quantized models, ADAngel addresses the critical downstream challenge of executing the resulting complex GEMM workloads with maximum, workload-adaptive efficiency.

Kernel-Level Acceleration for mixed-precision GEMM is the domain most directly related to our work. Prior art in this domain has primarily explored two major directions: software adaptation for general-purpose hardware, and the design of direct hardware accelerators. The majority of work focuses on software adaptation, which aims to optimally map these complex operations onto existing hardware. One prominent approach is to develop highly-specialized, static kernels for specific precision pairings, as exemplified by Marlin [13], which provides a state-of-the-art W4A16 GEMM kernel for NVIDIA GPUs. A different software approach, seen in frameworks like llama.cpp [11, 17], performs on-the-fly data transformation; its kernels employ techniques like online padding and native integer vectorization, where low-precision weights are unpacked and converted to a compatible format just-in-time for computation. Alternative software methods reframe the problem entirely, either leveraging bit-logic operations as in ABQ-LLM [38], or replacing multipliers with memory lookups as seen in LUT-based methods like T-MAC [31]. In contrast, the direct hardware design approach proposes new accelerator architectures. For instance, MixPE [40] designs specialized multiplier-less processing units using shift-and-add operations, while BitFusion [27] introduces a methodology to dynamically fuse bit-level processing elements.

8 Conclusion

In this paper, we identified and characterized the fundamental computation heterogeneity in arbitrary-precision LLM inference, revealing the inherent limitations of static, workload-unaware approaches. To address this, we proposed ADAngel, a framework that leverages our DPR computation model and an exhaustive offline analysis to construct a specialized, workload-adaptive computation core for any given LLM. Our extensive evaluation demonstrates that the ADAngel-specialized engine significantly outperforms state-of-the-art inference frameworks, establishing that principled, workload-adaptive specialization is an essential paradigm for unlocking the full performance potential of low-precision hardware in edge LLM inference.

References

[1] Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models. *Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob*, 2023.

[2] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 4396–4429. Curran Associates, Inc., 2023.

[3] Huancheng Chen and Haris Vikalo. Mixed-precision quantization for federated learning on resource-constrained heterogeneous devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6138–6148, June 2024.

[4] Jack Choquette. Nvidia hopper h100 gpu: Scaling performance. *IEEE Micro*, 43(3):9–17, May 2023.

[5] Jack Choquette and Wish Gandhi. NVIDIA A100 GPU: Performance & Innovation for GPU Computing . In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–43, Los Alamitos, CA, USA, August 2020. IEEE Computer Society.

[6] NVIDIA Corporation. Fastertransformer. https://github.com/NVIDIA/FasterTransformer, 2022. Accessed: 2025-05-21.

[7] NVIDIA Corporation. Jetpack 6.1 release notes. https://docs.nvidia.com/jetson/archives/jetpack-archived/jetpack-61/release-notes/index.html, 2024. Accessed: 2025-03-18.

[8] NVIDIA Corporation. Cutlass 4.1.0. https://github.com/NVIDIA/cutlass, 2025. Accessed: 2025-07-18.

[9] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025.

[10] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023.

[11] Boyuan Feng, Yuke Wang, Tong Geng, Ang Li, and Yufei Ding. Apnn-tc: accelerating arbitrary precision neural networks on ampere gpu tensor cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'21, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[13] Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language

models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '25, page 239–251, New York, NY, USA, 2025. Association for Computing Machinery.

[14] Ziyi Guan, Hantao Huang, Yupeng Su, Hong Huang, Ngai Wong, and Hao Yu. Aptq: Attention-aware post-training mixed-precision quantization for large language models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, New York, NY, USA, 2024. Association for Computing Machinery.

[15] Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. Billm: Pushing the limit of post-training quantization for llms, 2024.

[16] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 87–100, 2024.

[17] The llama.cpp team. llama.cpp: Llm inference in c/c++. https://github.com/ggml-org/llama.cpp, 2025. Accessed: 2025-06-18.

[18] Saeed Maleki. Look-up mai gemm: Increasing ai gemms performance by nearly 2.5x via msgemm, 2023.

[19] NVIDIA. Nvidia jetson nano, 2025.

[20] NVIDIA. Nvidia jetson orin, 2025.

[21] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner,

Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[22] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models, 2024.

[23] Yeonhong Park, Jake Hyun, Hojoon Kim, and Jae W Lee. Decdec: A systems approach to advancing low-bit llm quantization. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 803–819, 2025.

[24] Inc. Qualcomm Technologies. Qualcomm® sa8155p product brief. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/qul7413_sa8155_productbrief_r4.pdf, 2019. Accessed: 2025-03-18.

[25] Mariam Rakka, Mohammed E. Fouda, Pramod Khargonekar, and Fadi Kurdahi. A review of state-of-the-art mixed-precision neural network frameworks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):7793–7812, 2024.

[26] Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea J. Goldsmith, and Mert Pilanci. Compressing large language models using low rank and low precision decomposition. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 88981–89018. Curran Associates, Inc., 2024.

[27] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 764–775, 2018.

[28] Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, Wulong Liu, and Jun Yao. Flatquant: Flatness matters for llm quantization, 2025.

[29] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[30] Changyuan Wang, Ziwei Wang, Xiuwei Xu, Yansong Tang, Jie Zhou, and Jiwen Lu. Q-vlm: Post-training quantization for large vision-language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 114553–114573. Curran Associates, Inc., 2024.

[31] Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys'25, page 278–292, New York, NY, USA, 2025. Association for Computing Machinery.

[32] Junyi Wu, Haoxuan Wang, Yuzhang Shang, Mubarak Shah, and Yan Yan. Ptq4dit: Post-training quantization for diffusion transformers. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62732–62755. Curran Associates, Inc., 2024.

[33] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR, 23–29 Jul 2023.

[34] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[35] Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. Exploring post-training quantization in

llms from comprehensive study to low rank compensation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19377–19385, 2024.

[36] Lu Yin, Ajay Jaiswal, Shiwei Liu, Souvik Kundu, and Zhangyang Wang. Junk dna hypothesis: Pruning small pre-trained weights irreversibly and monotonically impairs "difficult" downstream tasks in llms, 2025.

[37] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. Rptq: Reorder-based post-training quantization for large language models, 2023.

[38] Chao Zeng, Songwei Liu, Yusheng Xie, Hong Liu, Xiaojian Wang, Miao Wei, Shu Yang, Fangmin Chen, and Xing Mei. Abq-llm: Arbitrary-bit quantized inference acceleration for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22299–22307, 2025.

[39] Aozhong Zhang, Naigang Wang, Yanxia Deng, Xin Li, Zi Yang, and Penghang Yin. Magr: Weight magnitude reduction for enhancing post-training quantization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 85109–85130. Curran Associates, Inc., 2024.

[40] Yu Zhang, Mingzi Wang, Lancheng Zou, Wulong Liu, Hui-Ling Zhen, Mingxuan Yuan, and Bei Yu. Mixpe: Quantization and hardware co-design for efficient llm inference, 2024.

[41] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 196–209, 2024.