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

**OSDI '26**

Yao Liu, Wenjie Wang, Yifei Feng, Bo Peng, Jianguo Yao, Haibing Guan
Shanghai Jiao Tong University

# LLM on Edge & Arbitrary-Precision Quantization

Large language models are pushing into **edge** and data-center deployment, but memory and latency are tight. Arbitrary-precision quantization (APQ), e.g., W4A8, W3A8, W4A4, uses heterogeneous bit-widths for weights and activations to shrink models and speed up inference.

**APQ turns the dominant GEMM operator into mixed-precision GEMM (mpGEMM):**

Weight bits: {2, 3, 4, 5}

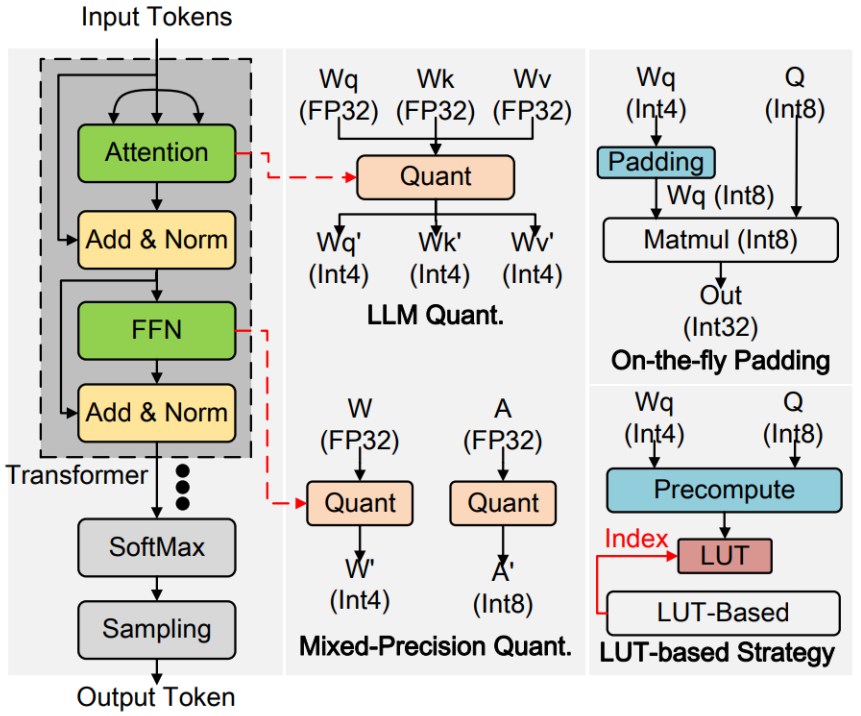
Activation bits: {4, 8}

GEMM shape (M, N, K) varies across LLM inference phases

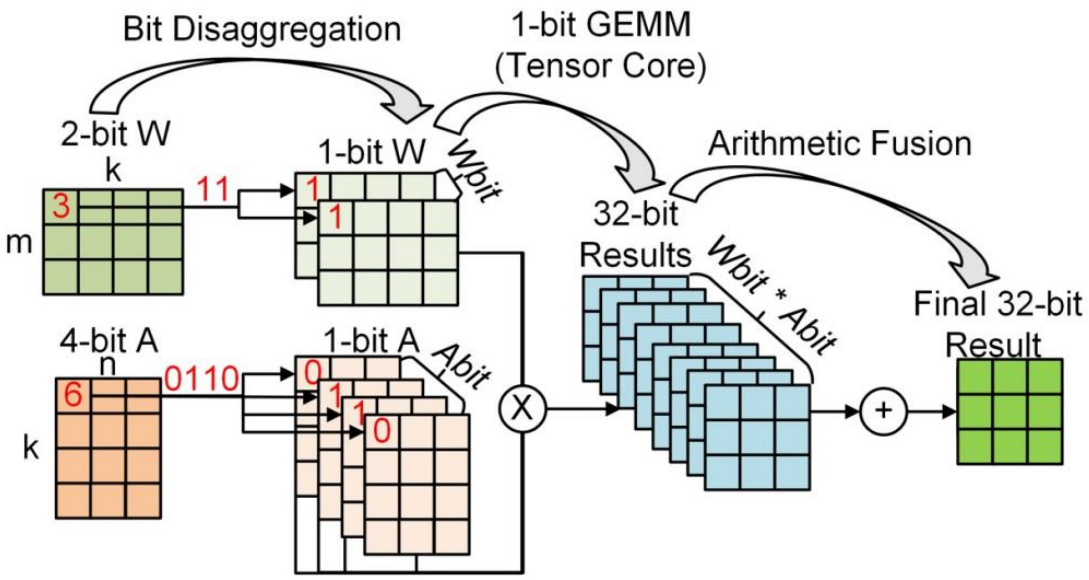
Existing strategy:

- Padding (promote to 8-bit)
- Bit-disaggregation.

Each works only in a **narrow regime**, leaving a hardware-software gap for arbitrary bit-widths.



# Bit-Disaggregation GEMM and GEMV




W2A4 GEMM bit-disaggregation workflow

Bit-disaggregation is a mathematical approach for directly executing GEMM and GEMV with asymmetric bit-widths.

**Idea**: Decompose weight W and activation A into **1-bit bitplanes**.

**Compute**: Perform W-bit × A-bit times 1-bit GEMM/GEMV operations on Tensor Cores.

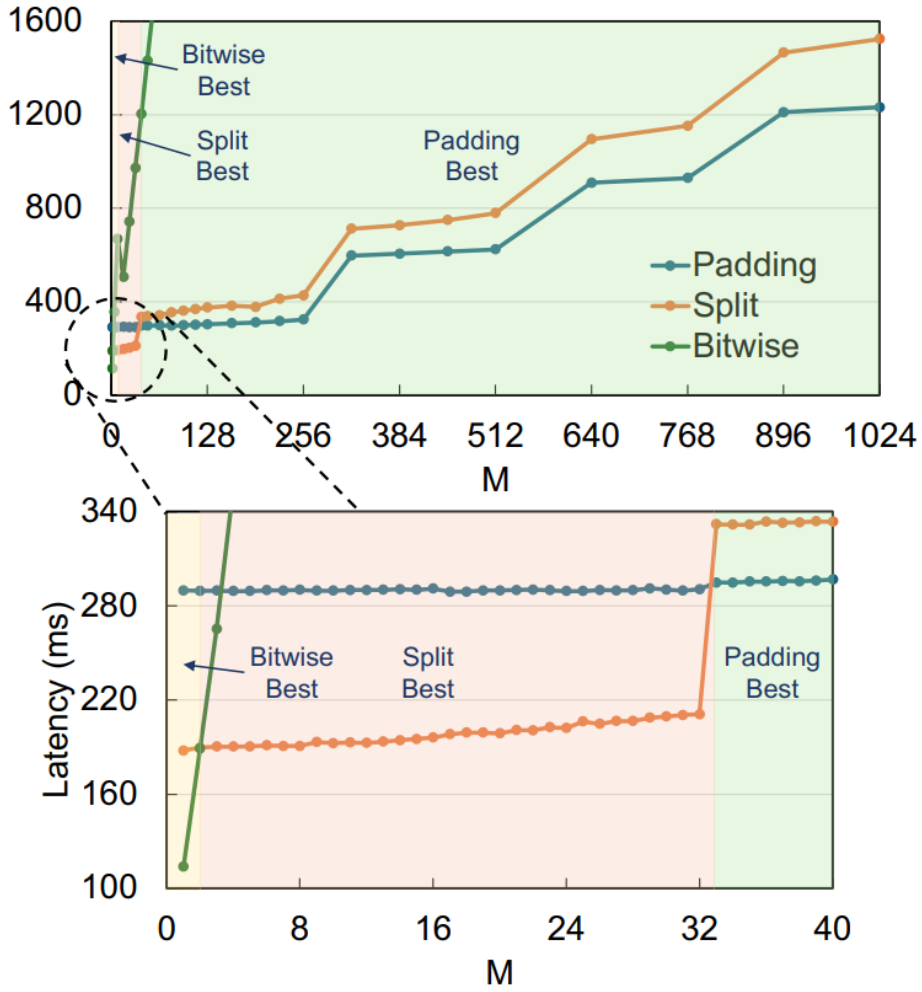
**Reconstruct**: Shift and accumulate partial 32-bit results into final output.

**Cost**: Num of GEMM/GEMV operations grows by WBit × ABit; Extra combination overhead.

$$Result = \sum (2^0 W^{W_Bit-1} + \cdots + 2^{W_Bit-1} W^0) \times (2^0 A^{A_Bit-1} + \cdots + 2^{W_Bit-1} W^0)$$

OBSERVATION

# No One-Size-Fits-All Strategy




The fastest kernel changes with workload shape (W4A8, N=4096, K=6144)

**GEMM shape:** $[M, K] \times [K, N] = [M, N]$

**The winner changes with M**

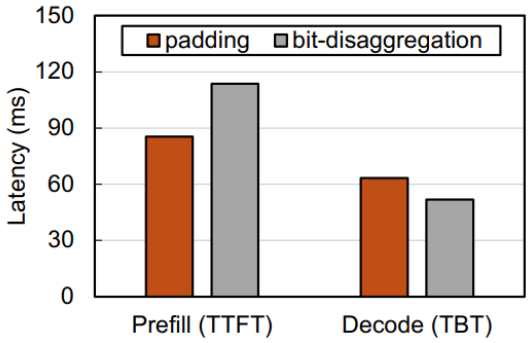
- Bitwise wins at very small M
- Split wins in the middle regime
- Padding wins when compute dominates

**No single static strategy is optimal across the workload spectrum.**

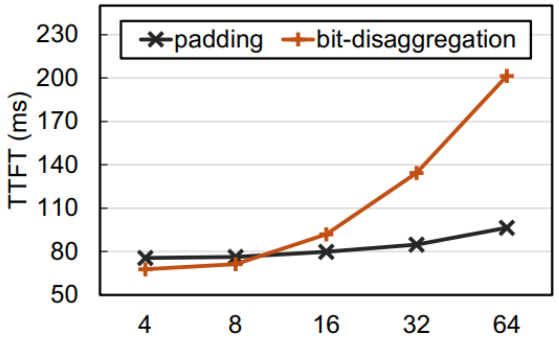
OBSERVATION

# Computational heterogeneity

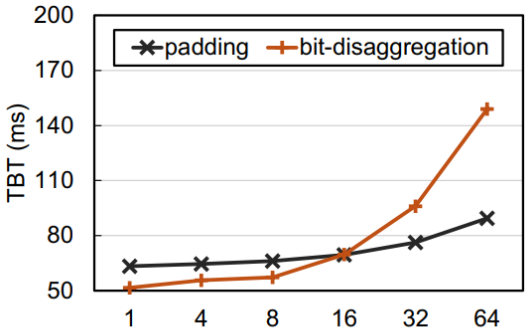
- Between Prefill And Decode
- Intra-Prefill
- Intra-Decode
- Quantization Bit-Width



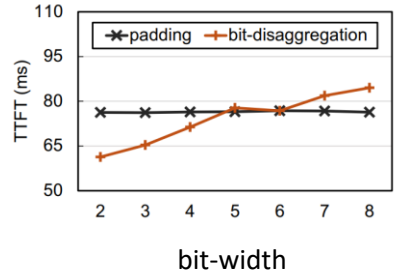

(a) Prefill and Decode



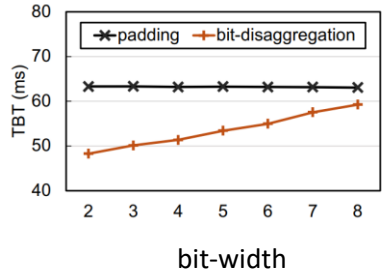

(b) Prompt Length




(c) Decode Batch Size




(a) Prefill




(b) Decode

1 **Phase**

Prefill is compute-heavy; decode is memory-bound.

2 **Prompt length**

Arithmetic intensity rises as sequence length grows.

3 **Batch size**

Decode moves toward GEMM-like behavior with batching.

4 **Bit-width**

The cost balance shifts across W2A8 to W5A8.

# KEY IDEA

## The Gap

Static kernels cannot match APQ workload heterogeneity.

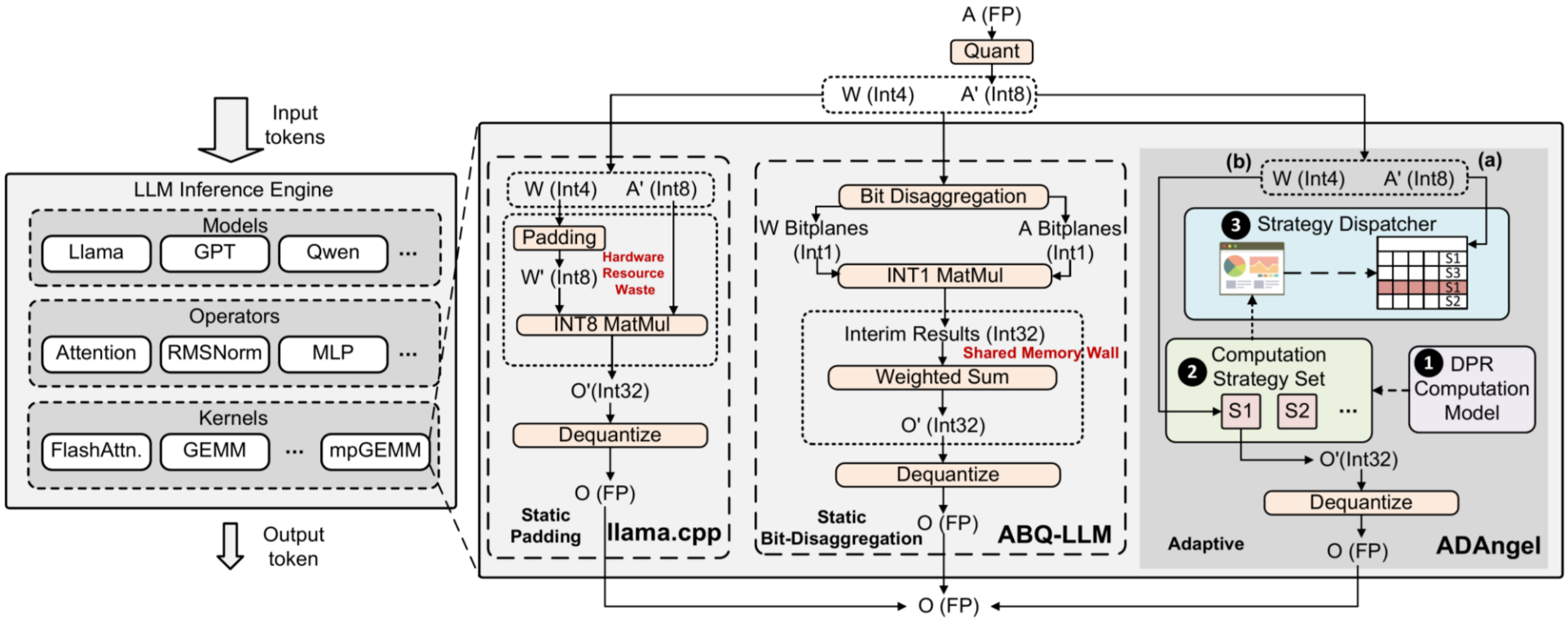
- One kernel cannot cover all (M, N, K, bit-width) combinations
- Padding and Bitwise each work only in a narrow regime

## ADAngel

1. Generate a strategy portfolio under DPR (Decomposition-Partial Product-Reconstruction) Computation Model
2. Profile the finite workload space offline
3. Dispatch the oracle kernel at runtime

**ADAngel： Map every mpGEMM task to fastest computation strategy.**

# ADAngel Architecture




**Target Platform**

Jetson AGX Orin (**mainly**) / NVIDIA A100

**LLM Inference Engine**

FasterTransformer

## ADAngel Components

- DPR Computation Model
- Computation Strategy Set
- Strategy Dispatcher

DESIGN

# DPR Computation Model

$W_x \times A_y$

Mixed-precision GEMM

**1 Decomposition**

**Bit Partitioning**

Split W and A into lower-bit components.

**2 Partial Products**

**Native GEMM Execution**

Map components to INT1, INT4, or INT8.

**3 Reconstruction**

**Result Reconstruction**

Shift and accumulate all partial products.

Logical Bit-Partition $P$

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

Turn A to sum of logical bit-partition on $P_A$

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

Compute &Reconstruction

$$Y = AW = \left(\sum_{i=1}^{m} A_i \cdot 2^{shift(i)}\right)\left(\sum_{j=1}^{n} W_j \cdot 2^{shift(j)}\right)$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{n} A_i W_j \cdot 2^{shift(i)+shift(j)}$$

# DPR Framework

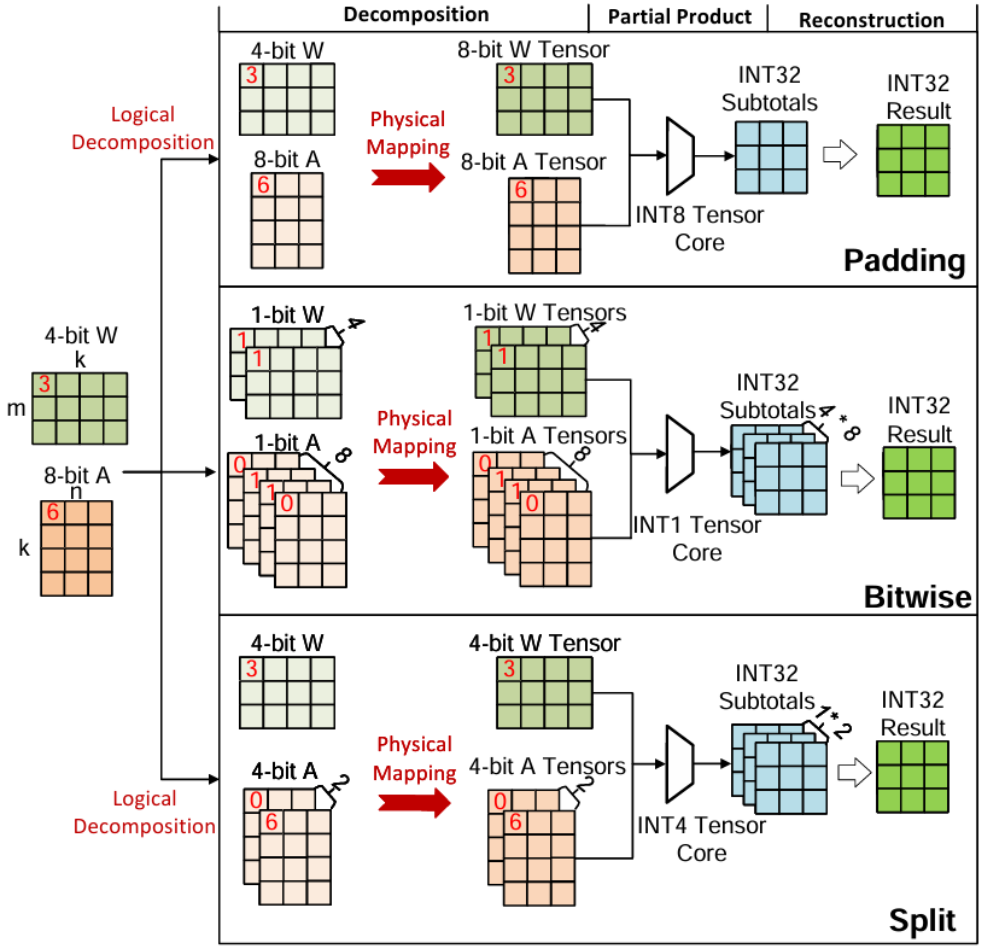
We prune the design space by minimizing the promotion overhead, which is achieved by aligning the decomposed bit-widths with the **native Tensor Core** precisions.

**Example partitions for W4A8**

$$W_4 \times A_8$$

| Strategy | Partition | Native Kernel |
|---|---|---|
| **Padding** | $P_A = (8) \quad P_W = (4)$ | INT8 GEMM |
| **Bitwise** | $P_A = (1, \dots, 1) \quad P_W = (1, \dots, 1)$ | BMMA |
| **Split** | $P_A = (4, 4) \quad P_W = (4, 4)$ | INT4 GEMM |

# Strategy Set in ADAngel




## Padding

**1 x CUTLASS INT8 GEMM**

W4 is promoted to W8

Compute-oriented

## Bitwise

**32 x INT1 BMMA**

Many INT32 partial products

Memory-oriented

## Split

**1 x CUTLASS INT4 GEMM**

Less partial products
Low bit weight

Balanced

**Compute-bound** ← Padding | Split | Bitwise → **Memory-bound**

DESIGN

# Split: A Balanced Strategy

### vs. Padding

Both widen operands, but Split uses narrower widths (e.g., 4-bit vs. 8-bit in W2A8), reducing promotion overhead.

### vs. Bitwise

Split uses coarser bit-partitioning, so it has fewer partial products, lower reconstruction overhead, and much smaller intermediate-result footprint.

## W4A8 Example: 16× Footprint Reduction

| | |
|---|---|
| Bitwise: | 32 int32 partial products |
| Split: | 2 int32 partial products |
| Reduction: | 16× smaller intermediate footprint |

Lower shared-memory pressure → More resident CTAs → Better middle-M performance

**Split achieves a middle ground: less promotion overhead than Padding, far lower intermediate footprint than Bitwise.**

# Offline Policy Construction

**DPR**

Construct valid bit partitions

**Strategy Set**

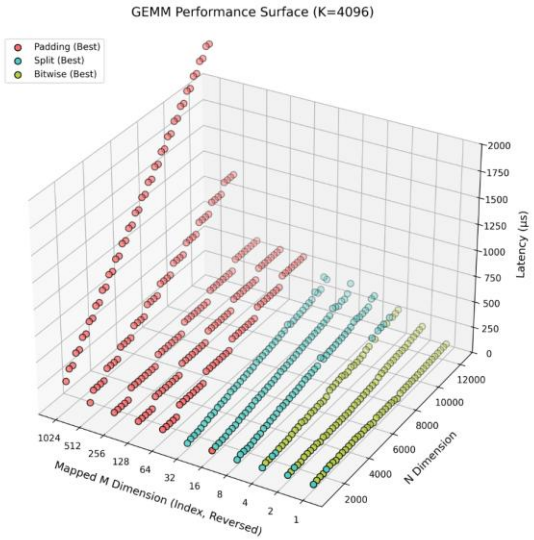
Padding / Split / Bitwise

**Offline Profiling**

Benchmark each (M,N,K,bit-width)

**Oracle PolicyMap**

Workload -> Fastest kernel




One case for profiling

**DPR answers**

What equivalent kernels can we construct?

**Profiling answers**

Which kernel is fastest for this workload and GPU?

For Llama3-8B

- **4** unique (N, K) GEMM types
- **1-8192** profiled M range
- **5.7h** one-time profiling
- About **256 KB** memory footprint

**DPR explores the algorithmic design space. Profiling resolves hardware-dependent performance.**

EXPERIMENTS

# Evaluation setup

## Primary Platform

NVIDIA Jetson AGX Orin 64GB (50W)

JetPack 6.1 / CUDA 12.6

## Validation Platform

NVIDIA A100 80GB

For cross-platform generality check

## Model & Precisions

Model: Llama-3-8B

Quantization: W2A8, W3A8, W4A8, W5A8

## Implementation

- Padding & Split via CUTLASS 3.1
- Bitwise via custom BMMA kernel
- Integrated into FasterTransformer

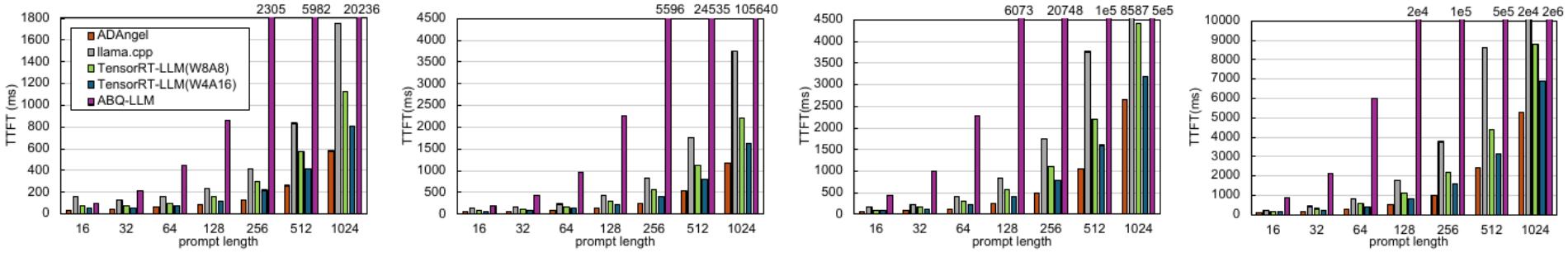
## Baselines

Orin: llama.cpp | TensorRT-LLM | ABQ-LLM

A100: QServe (W4A8KV4) | TensorRT-LLM

EXPERIMENTS

# Prefill & Decode




(a) Llama-3-8B, batch size = 1 (b) Llama-3-8B, batch size = 2 (c) Llama-3-8B, batch size = 4 (d) Llama-3-8B, batch size = 8

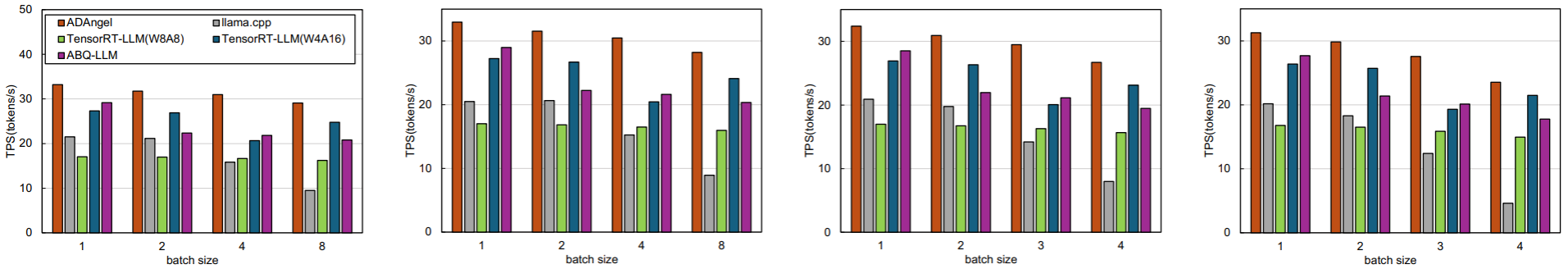
Prefill Stage Performance Comparison

**65.9%**

lower latency than llama.cpp at short prompts

**1.17-2.38x**

speedup over TensorRT-LLM




(a) Llama-3-8B, pl = 128 (b) Llama-3-8B, pl = 256 (c) Llama-3-8B, pl = 512 (d) Llama-3-8B, pl = 1024

Decode Stage Performance Comparison

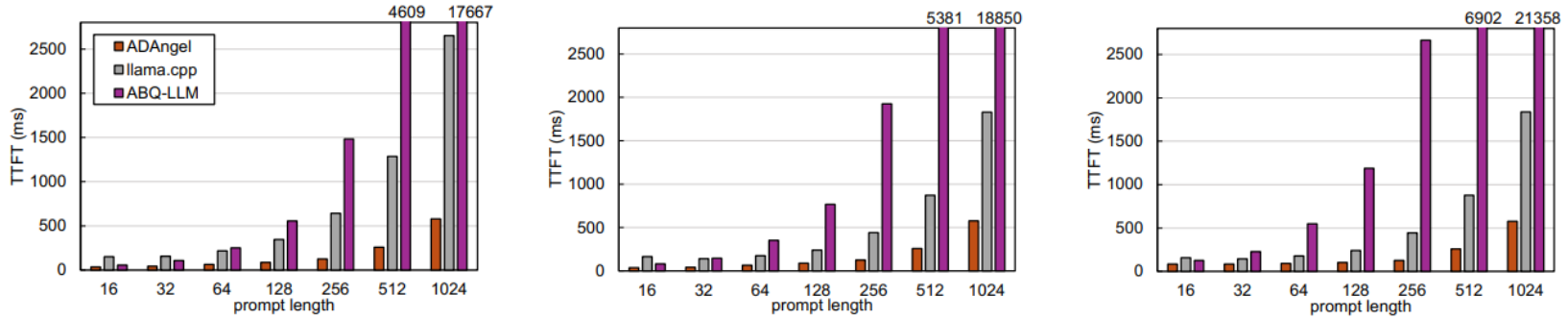
**5.10x**

peak TPS speedup over llama.cpp

**1.95x**

over TensorRT-LLM W8A8 at the M=1 regime

# Across Precision




(a) W2A8, batch size = 1 (b) W3A8, batch size = 1 (c) W5A8, batch size = 1

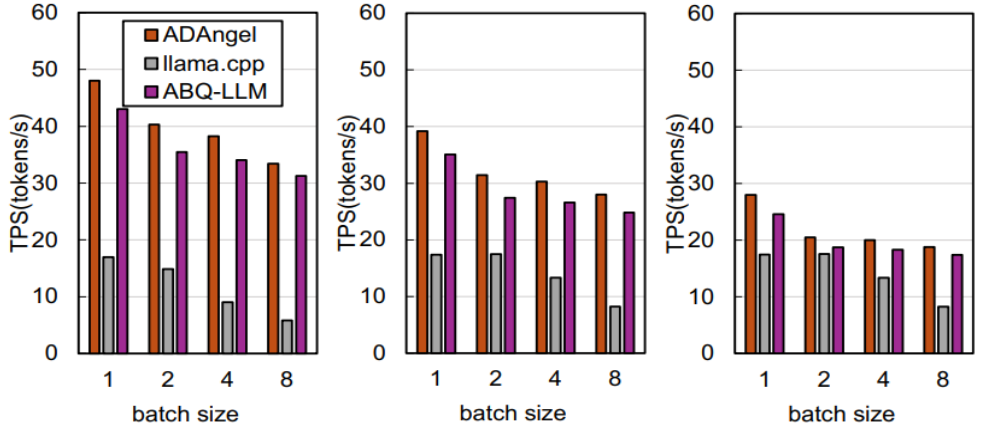
Cross-precision prefill performance

**3.43x**

average TTFT speedup
over llama.cpp

**200x**

peak TTFT speedup
over ABQ-LLM



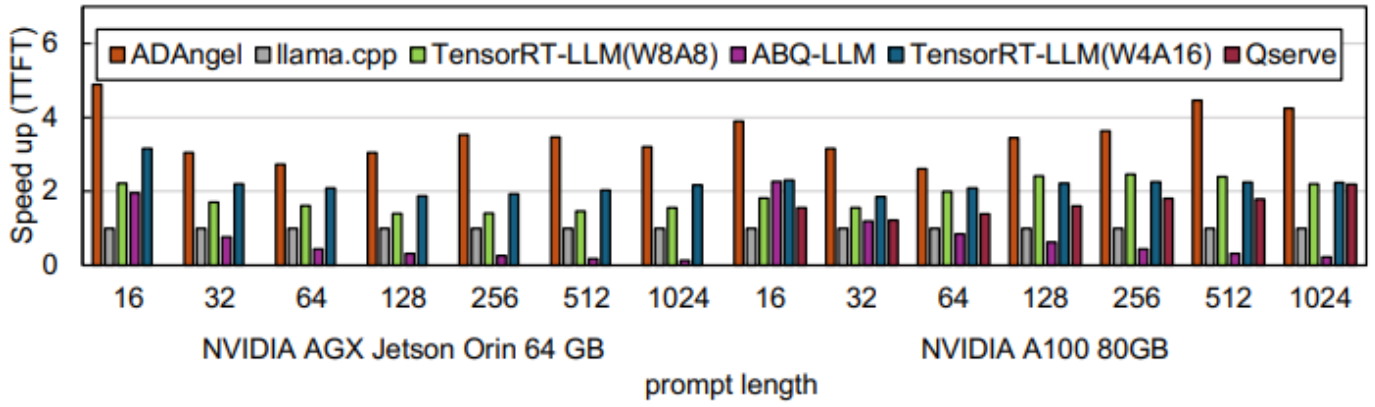

(a) W2A8, pl = 256 (b) W3A8, pl = 256 (c) W5A8, pl = 256

Cross-precision decode performance

**7.35x**

peak TPS speedup
over llama.cpp

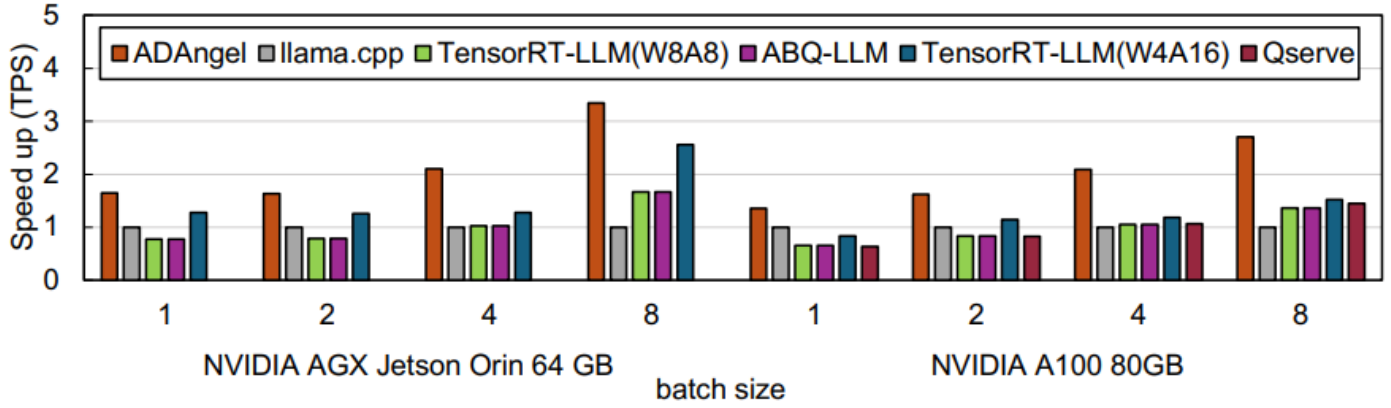
# Across Platform




(a) Llama-3-8B, batch size = 1

**2.12x**

TTFT speedup
over QServe on A100

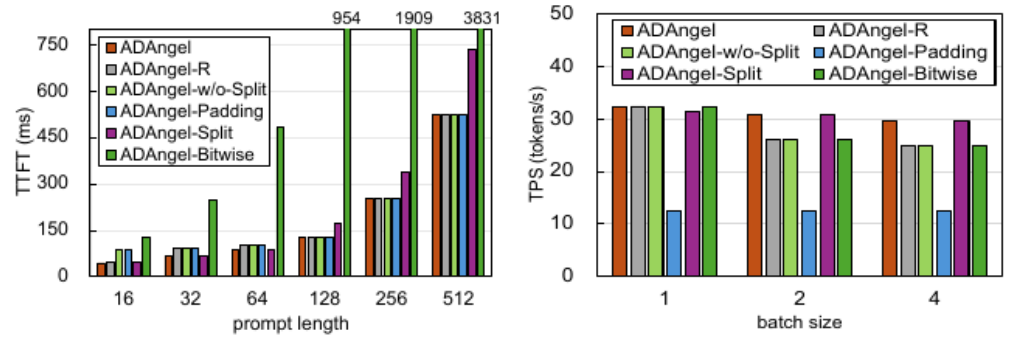



(a) Llama-3-8B, prompt length = 64

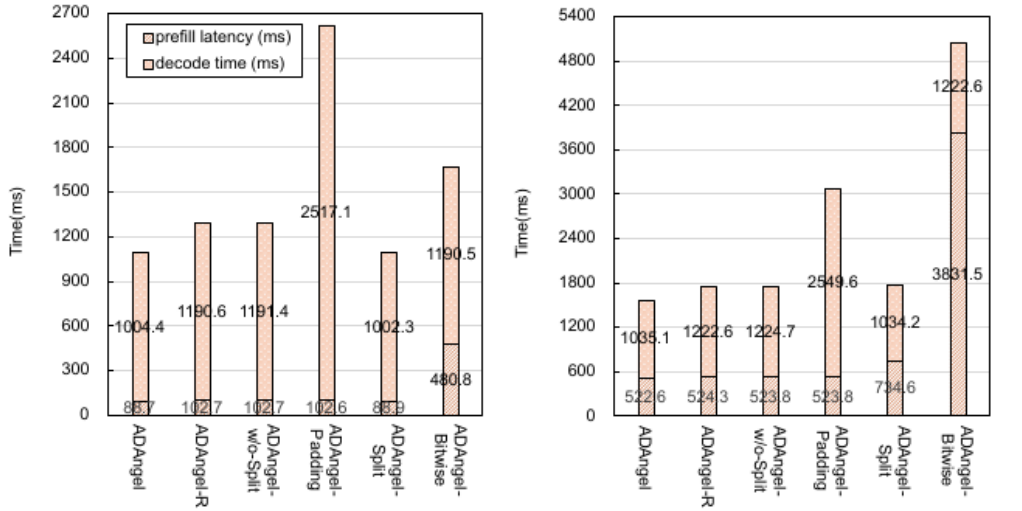
**1.72x**

TPS speedup
over QServe on A100

# Ablation Study




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




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

## Static strategy

Wins only in one regime; fails badly when M shifts.

## Rule-based ADAngel-R

Captures M trends but misses heterogeneity in N and K.

## Without Split

Avoids Bitwise collapse, but gives up the middle-M opportunity.

# Conclusions

1. **Insight: APQ workloads are highly heterogeneous.**
   *M, N, K, bit-width, and inference phase jointly determine the best kernel; no static strategy wins everywhere.*

2. **ADAngel： A framework for creating a workload-adaptive mpGEMM computation core for target LLM.**
   *DPR and offline profiling map every mpGEMM to its fastest strategy.*

3. **Impact: Strong speedups and a principled paradigm.**
   *Up to 5.10x decode throughput over llama.cpp and 1.17-2.38x prefill speedup over TensorRT-LLM.*

# Thanks!

## Q&A