Dynamic Semantic Adaptation for Efficient Large Language Model Inference

1st Guanghui Han
School of Computer Science
Qufu Normal University
Rizhao, China
hgh11282024@163.com

2nd Yidan Yan*
College of Biological Sciences
China Agricultural University
Beijing, China
yyd@cau.edu.cn

3rd Lei Wei
College of Biological Sciences
China Agricultural University
Beijing, China
leiwei@cau.edu.cn

4th Da Gao
School of Computer Science
Qufu Normal University
Rizhao, China
15763763003@163.com

5th Shougang Feng
Shandong Branch of China Mobile Group Co., Ltd.
Binzhou, China
18766678338@139.com

***Abstract*—Large language models (LLMs) have demonstrated remarkable capabilities across a wide range of reasoning and generation tasks, but their inference-stage inefficiencies remain a significant bottleneck for scalable deployment, particularly in latency-sensitive or resource-constrained environments. To solve this, this paper introduces a unified and adaptive inference optimization framework. It addresses three critical limitations in existing methods. These limitations include inter-layer semantic redundancy, static attention computation, and rigid precision scheduling. First, we propose a cross-layer context reuse mechanism with dynamic drift correction, enabling conditional bypass of redundant Transformer layers while preserving semantic fidelity. Second, we introduce a predictive attention path pruning strategy guided by token-level contextual complexity, allowing the model to selectively allocate attention computation based on entropy and dependency metrics. Third, we propose a self-evolving precision scheduler that adjusts bit-width based on token-level confidence. An online feedback mechanism further refines the precision policy during inference. Experiments on AlpacaEval, HotpotQA, and Vicuna QA demonstrate that our method achieves up to 2.43× decoding speedup and 41.7% memory savings, while maintaining comparable output quality to state-of-the-art baselines including Flash-Decoding, Radix Attention, and DEFT-Flatten. These results highlight the potential of fine-grained semantic adaptivity and runtime feedback as core principles for efficient and intelligent LLM inference.**

***Index Terms*—Adaptive Inference, Inference Acceleration, LLM Inference Optimization, Semantic-Aware Sparsity.**

I. Introduction

The rapid deployment of large language models (LLMs) [1]–[3] have marked a significant leap in general-purpose reasoning and language understanding capabilities. However, their inference-time computational demands remain a critical bottleneck, particularly in latency-sensitive, memory-constrained, or multi-tenant environments [4]–[6]. These challenges are exacerbated in edge deployment settings, where limitations in compute, energy, and storage restrict the feasibility of running large-scale models.

A range of techniques have been proposed to address inference inefficiencies, including model quantization [7]–[9], low-rank decomposition [10], key-value (KV) caching [4], structured pruning [11], [12], and static sparse attention mechanisms [13], [14]. While effective in controlled scenarios, these methods typically rely on fixed design heuristics or coarse-grained compression strategies that do not adapt to the dynamic semantics of the input or the varying complexity of downstream tasks. Consequently, they often exhibit suboptimal performance in multi-turn, long-context, or multi-task workloads where information density and reasoning depth fluctuate substantially across tokens and layers.

We argue that the key to unlocking efficient LLM inference lies in embracing dynamic, input-conditioned control strategies that exploit semantic redundancy, predict input complexity, and adapt precision allocation at runtime. To this end, we propose a unified inference optimization framework composed of three tightly coupled innovations. First, we design a cross-layer context reuse mechanism that detects inter-layer semantic similarity and bypasses redundant computation via lightweight feature transfer and semantic drift correction. Second, we introduce a predictive attention pruning strategy that estimates contextual complexity from token-level features and dynamically adjusts attention sparsity patterns to retain high-complexity regions. Third, we implement a self-evolutionary precision scheduling system that adjusts token-level compute precision based on output uncertainty and a feedback loop, enabling fine-grained energy–accuracy tradeoffs.

Together, these components form an adaptive inference pipeline that reduces latency and memory cost while preserving output quality. The framework is modular and hardware-agnostic, and does not require retraining the base LLM weights; only lightweight auxiliary predictors are calibrated using model-derived signals such as entropy or attention statistics with a small set of samples. Empirical evaluations across text generation and reasoning bench-

marks—including AlpacaEval, HotpotQA, Vicuna QA, and Tree-of-Thoughts—show that our method improves throughput and memory efficiency while remaining closely aligned with full-precision outputs.

Our key contributions are summarized as follows:

- We propose a cross-layer context reuse module that eliminates semantic redundancy via inter-layer similarity detection and drift-aware recomputation.
- We introduce a predictive attention pruning mechanism that conditions attention paths on token-level complexity and supports progressive path repair.
- We develop a self-evolutionary precision scheduling system that adjusts compute bitwidths online using uncertainty estimation and feedback-driven updates.
- We demonstrate consistent speedups (up to $2.43\times$), memory savings (up to 41.7%), and minimal quality degradation (< 0.4 ROUGE drop) across diverse LLM workloads.

II. Related Work

A. Static Inference Optimization

Static inference optimization for LLMs typically relies on quantization, pruning, and low-rank approximation to reduce memory and computation. Post-training quantization methods (e.g., SmoothQuant, GPTQ) improve fidelity via activation scaling and calibration [7], [15], while structured pruning and sparsification reduce FLOPs by removing weights or attention components [11], [16]. Low-rank decomposition similarly compresses projection layers under a fixed global structure.

Despite their effectiveness, these techniques are fundamentally *static* and apply uniform policies across inputs, making them less responsive to token-level semantic variability at runtime. In contrast, our framework adapts computation online based on semantic redundancy, predicted complexity, and uncertainty.

B. Sparse and Adaptive Attention

Self-attention dominates inference cost at long sequence lengths, motivating sparse attention variants such as Longformer, BigBird, and Performer that reduce quadratic complexity using predefined sparsity patterns or kernel approximations [17]–[19]. Hardware-aware implementations like FlashAttention-2 further improve latency via memory-efficient attention kernels [20].

However, most prior approaches are driven by token position, fixed layouts, or global heuristics rather than input semantics, which limits their ability to distinguish low-utility tokens from reasoning-critical regions during generation. Our method introduces complexity-guided attention pruning with lightweight prediction and a recovery mechanism to maintain robustness under dynamic sparsity.

C. Precision Scheduling and Confidence Estimation

Mixed-precision inference is widely used to improve efficiency with minimal accuracy loss, typically through fixed operator- or layer-level policies [21], [22]. Separately, confidence and uncertainty estimation has been studied for detecting errors and guiding decoding, and has been incorporated into inference procedures such as Radix Attention [23].

Nevertheless, prior work rarely connects uncertainty estimation to *fine-grained* precision allocation in a unified, runtime-adaptive manner. We bridge this gap with self-evolutionary precision scheduling that adjusts token/block-level precision using uncertainty signals and feedback to balance efficiency and output fidelity.

III. Method

Figure 1 summarizes the adaptive inference pipeline executed at each decoding step. The framework performs adaptive inference through lightweight control signals while keeping a single, unified execution path for all tokens. At each layer l, we first evaluate semantic similarity between the current layer input $h^{(l-1)}$ and a cached representation from the previous execution. If the similarity exceeds a threshold τ, we skip the full attention/FFN computation for the entire layer and generate an approximate output via a gated mapping. To avoid error accumulation, reuse is constrained by a bounded consecutive-skip budget and can be overridden when drift or low-confidence signals are detected.

When a layer is executed, we compute a complexity score c_i for every token to configure an adaptive attention mask, assigning denser attention to high-complexity tokens and more aggressive sparsity to low-complexity tokens. In parallel, we estimate token-level uncertainty and select numerical precision at token/block granularity, grouping tokens into blocks to reduce kernel switching overhead. Importantly, tokens are never routed into separate branches; all tokens are processed at every layer, and adaptation only changes how computation is carried out (skip/mask/precision), not whether a token is processed.

Finally, the three mechanisms share minimal runtime state (e.g., reuse counters and confidence/error signals) and can conservatively fall back to denser attention, higher precision, or forced recomputation when instability is detected, ensuring robust output quality under dynamic execution.

Different from combining independent heuristics, our three controls are coupled through shared runtime state (reuse counters, complexity/uncertainty scores, and divergence signals), enabling coordinated fallback: when instability is detected, the system jointly upgrades precision, relaxes sparsity, and forces recomputation to bound accumulated error.

A. Cross-Layer Context Reuse and Semantic Redundancy Elimination

Hidden representations in deep Transformers often exhibit strong similarity across adjacent layers during autoregressive decoding, creating redundant per-layer computation [2], [24], [25]. To exploit this redundancy, we introduce a cross-layer reuse mechanism that decides, at the *entry* of each layer l, whether to skip the full attention/FFN computation.

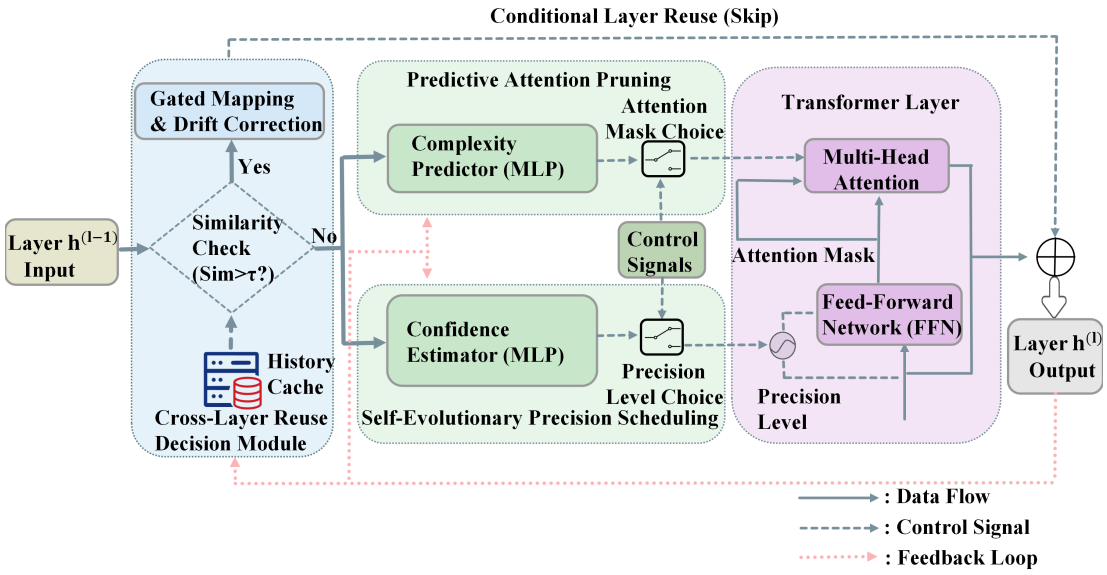

Fig. 1. Overview of the proposed adaptive inference pipeline for one decoding step. All tokens follow a single execution path without routing. At each layer, semantic similarity may trigger cross-layer reuse; otherwise, complexity-guided attention pruning and uncertainty-guided precision scheduling adapt the computation. Solid arrows denote hidden-state flow, and dashed arrows denote control signals.

In this paper, **semantic redundancy** refers to behavioral equivalence during decoding: when adjacent-layer representations induce near-identical predictive behavior (e.g., similar hidden representations and negligible output divergence), executing the full layer becomes redundant. We operationalize this notion using projected hidden-state cosine similarity as an efficient proxy.

Given the layer input $h^{(l-1)}$ and a cached representation $\hat{h}^{(l-1)}$ from the previous execution, we compute a cosine similarity in a shared projected space:

$$\hat{\text{Sim}}^{(l)} = \cos(W_p h^{(l-1)}, W_p \hat{h}^{(l-1)}), \tag{1}$$

where $W_p \in \mathbb{R}^{d \times d/2}$ is shared across layers. If $\hat{\text{Sim}}^{(l)} > \tau$ (default τ=0.85), we bypass layer l and generate an approximate output using a gated linear mapping $f(\cdot)$:

$$\tilde{h}^{(l)} = f(\hat{h}^{(l-1)}). \tag{2}$$

When a layer is skipped via cross-layer reuse, we do not generate new key–value (K/V) pairs for the current token. Instead, the decoder reuses the most recent valid K/V cache produced by the last full execution of the same layer. This design preserves the structural and semantic consistency of the KV cache across decoding steps, while avoiding the need to synthesize approximate K/V representations. Importantly, this design does not treat the current token as a previous token. Queries for the current token are still computed normally, while the approximation is confined to the hidden representation produced by the skipped layer. As a result, the primary source of error arises from hidden-state approximation rather than structural corruption of the KV cache.

To prevent error accumulation during repeated reuse of the K/V cache, we apply two safeguard mechanisms. First, we bound consecutive skips with a reuse budget $L_{\max}$ (default $L_{\max}$=3), after which the layer is forcibly recomputed. Second, we employ a lightweight drift detector $g(\cdot)$ that monitors the consistency between reused and cached states and triggers a small corrective update when drift exceeds ϵ (default ϵ=0.15):

$$\begin{aligned} \text{drift}_{\text{score}} &= g\big([\tilde{h}^{(l)}; \hat{h}^{(l-1)}]\big), \\ \tilde{h}^{(l)} &\leftarrow \tilde{h}^{(l)} + \Delta(\tilde{h}^{(l)}) \quad \text{if drift}_{\text{score}} > \epsilon. \end{aligned} \tag{3}$$

In all cases, reuse is applied at *layer granularity* (i.e., all tokens share the same skip decision), keeping the execution path simple and avoiding token routing.

B. Predictive Attention Path Pruning Based on Input Complexity

Self-attention dominates decoding cost at long context lengths. We therefore prune attention paths using a learned, token-level complexity estimate rather than a fixed sparsity pattern. For each layer that is executed, a lightweight predictor maps hidden states $h \in \mathbb{R}^{n \times d}$ to normalized complexity scores $c_i \in [0, 1]$ for all tokens:

$$c_i = \mathcal{C}(h_i). \tag{4}$$

The predictor is trained offline while keeping the base LLM frozen. Supervision is obtained from model-derived pseudo-labels computed from logits and attention statistics, capturing token uncertainty and long-range dependency strength (e.g., entropy-based uncertainty, context-sensitivity, and attention

span). This avoids task-specific labels and yields a reusable calibration procedure across backbones.

Complexity-guided masking. During inference, c_i controls a three-tier attention policy:

$$\text{AttnPolicy}(i) = \begin{cases} \text{full}, & c_i > \tau_{\text{high}}, \\ \text{window}, & \tau_{\text{low}} \le c_i \le \tau_{\text{high}}, \\ \text{sparse}, & c_i < \tau_{\text{low}}, \end{cases} \quad (5)$$

where window attention attends to the most recent w positions, and sparse attention retains a fixed budget of k keys per query. In our implementation, sparse keys are selected deterministically by a recency-biased top-k score that optionally incorporates a running attention prior from earlier layers, ensuring reproducible execution.

To mitigate occasional underestimation of complexity, we maintain a smoothed error signal for each token whenever a reference output is available (triggered by the precision scheduler). If the error exceeds a threshold, we relax sparsity by adding back attention connections for the affected tokens in subsequent steps:

$$e_i^{(t)} \leftarrow 0.9\, e_i^{(t-1)} + 0.1\, \|\hat{y}_i - y_i^{\text{ref}}\|_2, \quad \text{refresh if } e_i^{(t)} > \epsilon. \quad (6)$$

This progressive path refresh provides conservative fallback behavior, preserving fidelity under dynamic sparsity.

C. Self-Evolutionary Precision Scheduling

Static mixed-precision policies apply a uniform format across tokens and layers, which can waste compute on confident regions while harming fidelity on uncertain tokens. We therefore schedule precision dynamically using lightweight confidence and uncertainty signals.

For each token i, a small dual-headed estimator produces confidence p_i and an uncertainty proxy u_i from the hidden state h_i (optionally refined with logit entropy when available). We also include a simple ambiguity term a_i computed from hidden-state changes across steps. These signals are combined into a scheduling score

$$s_i = 0.4\, u_i + 0.3\, a_i + 0.3\, (1 - p_i), \quad (7)$$

where larger s_i indicates higher risk of numerical error.

We map s_i to one of four precision levels using fixed thresholds

$$\text{Prec}(i) = \begin{cases} \text{FP32}, & s_i > \gamma_3 \text{ (or feedback requests upgrade)}, \\ \text{FP16}, & \gamma_2 < s_i \le \gamma_3, \\ \text{INT8}, & \gamma_1 < s_i \le \gamma_2, \\ \text{INT4}, & s_i \le \gamma_1, \end{cases} \quad (8)$$

and implement switching at *token-block* granularity by grouping tokens with the same selected precision to reduce kernel switching overhead. When per-token variance is high, we fall back to a layer-level majority precision to keep dispatch stable. Unless specified otherwise, we use γ_1=0.3, γ_2=0.5, and γ_3=0.7 in all experiments.

To preserve output quality, we occasionally compute a sparse FP32 reference under periodic or instability triggers and measure divergence between quantized and reference outputs. If the divergence increases beyond a target range, the scheduler conservatively upgrades precision and updates the thresholds with a momentum-style rule, enabling the policy to adapt online without retraining the base LLM.

Algorithm 1 Per-layer adaptive inference at decoding step t

```
Compute uncertainty/confidence and select precision blocks
Compute similarity Ŝim^(l) between h^(l−1) and cached ĥ^(l−1)
if Ŝim^(l) > τ and r_l < L_max then
    h^(l) ← f(ĥ^(l−1))                          ▷ reuse & skip layer
    Optionally apply drift correction if enabled and drift > ε
    r_l ← r_l + 1
else
    r_l ← 0
    Compute complexity scores {c_i} and build attention mask M
    Run attention/FFN under selected precision to obtain h^(l)
    if reference check is triggered then
        Compute FP32 reference and update feedback/refresh signals
    end if
end if
```

D. Synergy and Execution Order

The three modules are coordinated through shared runtime signals to balance efficiency and fidelity. Cross-layer reuse reduces redundant layer execution; within executed layers, complexity-guided pruning reduces attention cost; and precision scheduling reduces arithmetic and memory bandwidth. A small set of shared state variables (reuse counter, complexity/uncertainty scores, and occasional FP32 reference checks) enables conservative fallback behavior: when confidence is low or divergence increases, the system upgrades precision, relaxes sparsity (progressive refresh), and/or forces recomputation to contain accumulated error. Algorithm 1 summarizes the per-layer control flow.

IV. Experimental Setup

A. Tasks and Benchmarks

We evaluate both efficiency and output fidelity on instruction following, multi-hop reasoning, and structured reasoning. For instruction following, we use AlpacaEval [26] and Vicuna QA [27]. For reasoning, we use HotpotQA [28] and Tree-of-Thoughts [29]. We additionally report results on ASQA, CNN/DailyMail, and HumanEval when studying generalization.

Unless stated otherwise, we use the official evaluation split for each benchmark and fix random seeds for any sampling. Metrics follow common practice: EM/F1 for QA-style tasks, ROUGE-L for generation/summarization, and LLM-as-judge

scores for open-ended generation. For LLM-as-judge evaluation, we follow a deterministic and bias-reduced protocol. We use GPT-4-Turbo (gpt-4-1106-preview) as the judge with temperature 0.0, max tokens 512, and top-p 1.0. For each prompt–response pair, the two candidate outputs are scored twice with swapped order to mitigate positional bias, and the final score is obtained by averaging the two judgments. This protocol is applied consistently across AlpacaEval and Vicuna QA.

B. Baselines

We compare against strong recent inference-acceleration baselines: Flash-Decoding (FlashAttention-2 decoding kernel) [20], TreeAttn-Medusa [30], Radix Attention [23], and DEFT-Flatten [31]. We also include static quantization and fixed sparse attention (e.g., BigBird [18]) as reference points.

For fairness, all methods share the same backbone, prompts, decoding parameters (Top-k=50, temperature=0.7), and KV-cache policy when applicable, and are run under the same software/hardware stack described below. Unless stated otherwise, experiments run on A100-80GB with PyTorch 2.1 and CUDA 12.1. We measure per-step end-to-end decoding latency using CUDA events with explicit synchronization, reporting the median of 100 runs after 10 warmup iterations. Batch size is 1 and maximum sequence length is 2048 (up to 8192 for long-context experiments).

C. Implementation and Reproducibility

No retraining of the base model. The base LLM weights are kept frozen. Only lightweight auxiliary modules (similarity/gating, drift detector, complexity predictor, and confidence estimator) are trained or calibrated offline.

Setup. We use a fixed calibration set of 2,048 C4 (val) sequences and 512 ShareGPT sequences (truncated to 512 tokens). Auxiliary modules are trained with AdamW (lr 1×10^{-4}, wd 0.01, batch size 64) for 3 epochs with early stopping, using pseudo-labels derived from entropy and attention signals (Section III-B).

Latency and overhead. Reported latency includes all online overheads (predictor inference, mask generation, drift correction, precision scheduling, and kernel dispatch), excluding offline quantization. Auxiliary modules are lightweight two-layer MLPs ($d \to d/4 \to 1$), introducing less than 1.2 ms per decoding step on A100-80GB and accounting for under 0.5% of model parameters.

V. Results and Analysis

A. Inference Efficiency

Latency. We measure per-step end-to-end decoding latency under the unified protocol (Section IV-C). Table I shows that our method is consistently fastest across few-shot prompting (varying tree width b) and multi-step reasoning workloads. Relative to Radix Attention, we reduce latency by up to 16.8% (Sorting: 104.79→87.12 ms) and remain faster than DEFT-Flatten in all settings. The gains increase with larger b, where redundancy and sparsity are more exploitable.

TABLE I

Decoding latency comparison (ms) across few-shot prompting and multi-step reasoning tasks. Tree width b indicates the number of parallel branches in few-shot scenarios, while task names represent different reasoning workloads. Lower values indicate better performance. **Fixed parameters:** tree depth $t = 64$ for few-shot tasks; sequence length 1024 tokens; batch size 1; LLaMA-2-7B backbone; A100-80GB GPU; CUDA 12.1; PyTorch 2.1. Latency averaged over 100 runs after 10 warmup iterations.All values report the median end-to-end decoding latency per decoding step (100 runs, 10 warmup discarded).

Method	b = 20	b = 30	b = 50	Sorting	Document	Keyword	Set
Flash-Decoding	78.96	131.09	191.09	429.65	241.20	32.75	51.76
TreeAttn-Medusa	52.58	103.90	144.07	380.87	236.86	33.52	50.10
Radix Attention	12.37	14.08	16.54	104.79	69.61	11.25	17.03
DEFT-Flatten	9.98	10.99	12.48	94.67	66.95	10.90	16.10
Ours	8.32	9.34	11.01	87.12	63.40	10.03	15.12

TABLE II

Speculative decoding latency (ms) with varying tree depths. Tree depth t represents the number of tokens in candidate sequences. ⋆ indicates out-of-memory failure ($> 80GB$). **Fixed parameters:** tree width $b = 20$; prefix length 512 tokens; batch size 1; LLaMA-2-7B backbone; A100-80GB GPU; CUDA 12.1; PyTorch 2.1. Latency averaged over 50 runs after 10 warmup iterations.

Method	t = 32	t = 64	t = 128	t = 256
Flash-Decoding	574.50	1128.45	⋆	⋆
TreeAttn-Medusa	263.40	483.35	924.97	1881.51
Radix Attention	54.66	69.75	108.56	188.66
DEFT-Flatten	42.23	46.60	56.96	84.27
Ours	38.61	43.07	51.02	77.45

Speculative decoding scalability. Table II evaluates deeper candidate trees ($t \in \{32, 64, 128, 256\}$). Our method maintains stable latency and avoids OOM where applicable; at $t = 256$ we stay below 80 ms/step (77.45 ms), improving over DEFT-Flatten by 8.1% and Radix by 59.0%.

TABLE III

Speedup analysis relative to Radix Attention baseline. The table shows decoding speedup, attention-only speedup, and theoretical upper bound (assuming zero attention cost) across all experimental configurations.

Metric	Decoding Speedup	Attention Speedup	Upper Bound (No Attention)
b=20	1.49×	1.78×	2.51×
b=30	1.51×	1.83×	2.51×
b=50	1.50×	1.88×	2.51×
Sorting	1.20×	1.39×	1.96×
Document	1.10×	1.24×	1.82×
Keyword	1.12×	1.31×	1.70×
Set	1.13×	1.34×	1.76×
t=32	1.42×	2.01×	2.89×
t=64	1.62×	2.29×	3.34×
t=128	2.13×	2.98×	4.36×
t=256	2.43×	3.21×	4.89×

Speedup and bottleneck shift. Table III reports speedup over Radix Attention. We achieve up to 2.43× end-to-end speedup (speculative $t = 256$) and up to 3.21× attention-only speedup, indicating that attention is no longer the dominant bottleneck under our adaptive pipeline.

Measurement details. Reported latency is the median end-to-end time per decoding step, measured with CUDA events and explicit `torch.cuda.synchronize()` before/after each step. It includes all online overheads introduced by our pipeline (predictor forward, mask generation, drift correction when triggered, precision scheduling, (de)quantization, and kernel dispatch), while excluding one-time offline quantization/calibration. We discard the first 10 warmup iterations and report the median over 100 runs (50 on edge).

TABLE IV
ABLATION STUDY ON LATENCY (MS). EACH ROW SHOWS PERFORMANCE WHEN ONE COMPONENT IS DISABLED. THE FULL MODEL INTEGRATES ALL THREE MODULES.

Model Variant	b = 20	b = 30	b = 50	Sorting	Document	Keyword	Set
Full Model (Ours)	8.32	9.34	11.01	87.12	63.40	10.03	15.12
Without Cross-Layer Reuse	11.42	13.05	16.88	121.33	88.97	13.72	20.44
Without Predictive Pruning	9.75	10.92	13.81	102.56	74.90	11.87	18.67
Without Precision Scheduling	8.91	10.34	12.77	93.48	67.51	10.94	16.70
Radix Attention (Baseline)	12.37	14.08	16.54	104.79	69.61	11.25	17.03

TABLE V
ABLATION STUDY ON MEMORY AND THROUGHPUT. THE TABLE SHOWS PEAK MEMORY USAGE, THROUGHPUT, AND RELATIVE CHANGES COMPARED TO THE FULL MODEL.

Metric	Full Model (Ours)	Without Cross-Layer Reuse	Without Predictive Pruning	Without Precision Scheduling	Radix Attention
Memory Peak (GB)	18.2	23.5	25.8	19.9	21.4
Throughput (tok/s)	328.5	241.7	218.3	302.4	254.1
Memory Δ vs Full	0%	+29.1%	+41.7%	+9.3%	+17.6%
Throughput Δ vs Full	0%	-26.4%	-33.5%	-7.9%	-22.6%

TABLE VI
ABLATION STUDY ON OUTPUT QUALITY. METRICS INCLUDE EXACT MATCH (EM) ON ASQA, ROUGE-L ON HOTPOTQA, AND AVERAGED LLM-AS-JUDGE SCORES. HIGHER VALUES INDICATE BETTER QUALITY.

Model Variant	EM (ASQA)	ROUGE-L (HotpotQA)	LLM Score	Δ LLM Score vs Full
Full Model (Ours)	48.2	42.1	6.87	0%
Without Cross-Layer Reuse	45.6	40.9	6.51	-5.2%
Without Predictive Pruning	46.1	41.4	6.47	-5.8%
Without Precision Scheduling	47.5	41.8	6.72	-2.1%
Full-Precision Baseline	47.8	42.0	6.75	-1.7%

B. Ablation Studies

Tables IV–VI ablate each module under the same evaluation settings. **Efficiency:** removing cross-layer reuse causes the largest latency increase (Table IV), confirming reuse as the primary contributor to compute reduction. **Memory/throughput:** removing predictive pruning yields the largest memory increase and throughput drop (Table V), consistent with its role in reducing active attention paths. **Precision scheduling** provides smaller but consistent gains across tasks.

Quality: Table VI shows that disabling reuse or pruning yields the largest quality drop, while adaptive precision has the smallest impact. Notably, the full framework matches or slightly exceeds the full-precision baseline, indicating that our approximations are effectively controlled.

These ablations indicate that the three modules contribute in complementary ways. Reuse dominates compute reduction, pruning primarily reduces memory and bandwidth, and precision scheduling provides fine-grained control. Together, they form a coupled system rather than independent heuristics.

TABLE VII
TRADE-OFF BETWEEN REUSE RATIO AND OUTPUT QUALITY FOR CROSS-LAYER CONTEXT REUSE. HIGHER τ VALUES REQUIRE STRICTER SIMILARITY FOR REUSE, RESULTING IN LOWER REUSE RATIOS BUT BETTER QUALITY PRESERVATION. THE DEFAULT $\tau = 0.85$ ACHIEVES A FAVORABLE BALANCE.

Threshold τ	Reuse Ratio	Latency (ms)	ROUGE-L	LLM Score	Quality Retention
0.70 (aggressive)	67.3%	6.21	39.2	6.31	93.3%
0.75	58.4%	6.89	40.1	6.48	95.4%
0.80	49.1%	7.52	41.3	6.69	98.3%
0.85 (default)	41.2%	8.32	42.1	6.87	99.5%
0.90	28.7%	9.78	42.0	6.84	99.8%
0.95 (conservative)	12.4%	11.24	42.0	6.82	100.0%
No reuse ($\tau = 1.0$)	0.0%	12.37	42.0	6.75	100.0%

TABLE VIII
ENERGY-ACCURACY TRADE-OFF FOR PRECISION SCHEDULING THRESHOLDS. ENERGY IS MEASURED IN JOULES PER 1000 TOKENS; QUALITY IS ROUGE-L ON HOTPOTQA. THE DEFAULT THRESHOLDS (0.7/0.5/0.3) ACHIEVE NEAR-OPTIMAL ENERGY EFFICIENCY WITH MINIMAL QUALITY LOSS.

Threshold Setting	FP32%	FP16%	INT8%	INT4%	Energy (J/1k tok)	ROUGE-L
All FP32	100%	0%	0%	0%	48.7	42.0
All FP16	0%	100%	0%	0%	24.3	42.0
All INT8	0%	0%	100%	0%	12.2	40.8
All INT4	0%	0%	0%	100%	6.1	37.2
Conservative (0.8/0.6/0.4)	18%	52%	24%	6%	22.1	42.0
Default (0.7/0.5/0.3)	12%	48%	32%	8%	18.6	42.1
Aggressive (0.6/0.4/0.2)	8%	38%	38%	16%	14.2	41.4
Very aggressive (0.5/0.3/0.15)	5%	28%	42%	25%	10.8	39.6

TABLE IX
GENERALIZATION ANALYSIS ACROSS TASKS, SEQUENCE LENGTHS, AND MODEL SIZES. SPEEDUP IS RELATIVE TO THE CORRESPONDING BASELINE (FLASH-DECODING FOR THE SAME CONFIGURATION). QUALITY RETENTION IS MEASURED AS THE PERCENTAGE OF BASELINE ROUGE-L SCORE PRESERVED.

Dimension	Configuration	Speedup	Quality Retention	Notes
Task Type	Text Generation (AlpacaEval)	2.21×	99.1%	Instruction following
	Multi-hop QA (HotpotQA)	1.89×	98.7%	Long-range reasoning
	Summarization (CNN/DM)	2.08×	98.4%	Document-level
	Code Generation (HumanEval)	1.76×	97.9%	Structured output
Sequence Length	512 tokens	1.82×	99.2%	Short context
	2048 tokens	2.15×	98.8%	Medium context
	4096 tokens	2.31×	98.5%	Long context
	8192 tokens	2.43×	98.1%	Extended context
Model Size	LLaMA-2-7B	2.21×	99.0%	Primary evaluation
	Vicuna-13B	2.18×	98.8%	Larger scale
	Mistral-7B	2.25×	99.1%	Different architecture
	LLaMA-2-70B	2.09×	98.6%	Very large scale

TABLE X
MULTI-SEED QUALITY EVALUATION WITH CONFIDENCE INTERVALS. RESULTS ARE REPORTED AS MEAN±STD ACROSS 5 SEEDS, WITH 95% CI COMPUTED VIA BOOTSTRAP. THE FULL-PRECISION BASELINE SERVES AS REFERENCE. OVERLAPPING CIS INDICATE NO STATISTICALLY SIGNIFICANT DIFFERENCE.

Dataset	Method	Score (mean±std)	95% CI	p-value vs. Baseline
AlpacaEval	Full-Precision Baseline	6.75±0.08	[6.68, 6.82]	–
	Ours	6.87±0.11	[6.77, 6.97]	0.142
HotpotQA	Full-Precision Baseline	42.0±0.3	[41.7, 42.3]	–
	Ours	42.1±0.4	[41.7, 42.5]	0.687
Vicuna QA	Full-Precision Baseline	7.21±0.09	[7.12, 7.30]	–
	Ours	7.18±0.12	[7.06, 7.30]	0.721
ASQA (EM)	Full-Precision Baseline	47.8±0.5	[47.3, 48.3]	–
	Ours	48.2±0.6	[47.6, 48.8]	0.298

TABLE XI
EDGE DEVICE EVALUATION RESULTS. LATENCY (SECONDS) AND PEAK MEMORY (GB) FOR INT4-QUANTIZED VICUNA-7B. SPEEDUP IS RELATIVE TO INT4 BASELINE WITHOUT OUR ADAPTIVE MODULES. RPI5 USES CPU-ONLY INFERENCE WITH LLAMA.CPP BACKEND; JETSON USES TENSORRT-LLM WITH OUR CUSTOM PLUGINS.

Device	Method	256 tokens		512 tokens		Speedup	Memory
		Latency	tok/s	Latency	tok/s		
Jetson Orin NX	INT4 Baseline	2.34s	109.4	5.87s	87.2	1.00×	11.2 GB
	+ Attention Pruning	1.42s	180.3	3.28s	156.1	1.65×	8.7 GB
	+ Ours	0.95s	269.5	2.18s	234.9	2.47×	7.4 GB
Raspberry Pi 5	INT4 Baseline	18.4s	13.9	42.6s	12.0	1.00×	3.2 GB
	+ Attention Pruning	12.7s	20.2	28.9s	17.7	1.45×	2.8 GB
	+ Ours	10.1s	25.3	22.4s	22.9	1.83×	2.5 GB

TABLE XII
QUALITY EVALUATION ON EDGE DEVICES. LLM-AS-JUDGE SCORES (1–10) ON VICUNA QA BENCHMARK WITH INT4-QUANTIZED VICUNA-7B. QUALITY RETENTION IS RELATIVE TO THE FULL-PRECISION A100 BASELINE.

Configuration	LLM Score	Quality Retention	Notes
A100 FP16 Baseline	7.21	100.0%	Reference
A100 + Ours (Adaptive)	7.18	99.6%	Full framework
Jetson INT4 Baseline	6.84	94.9%	Quantization loss
Jetson + Ours	6.79	94.2%	Minimal additional loss
RPi5 INT4 Baseline	6.78	94.0%	CPU inference
RPi5 + Ours	6.71	93.1%	Acceptable degradation

C. Robustness and Trade-offs

We analyze key trade-offs and robustness across thresholds, tasks, sequence lengths, and model scales. The reuse threshold sweep (Table VII) shows a clear Pareto frontier; we use $\tau = 0.85$ as a default, achieving high reuse with negligible quality loss. Precision thresholds exhibit a smooth energy–accuracy Pareto curve, and the default setting achieves near-optimal efficiency with minimal quality change (Table VIII). Together, these analyses suggest that the framework is robust to moderate variations of key hyperparameters, rather than relying on fragile tuning.

Generalization results (Table IX) remain stable across task types, longer contexts, and larger backbones. Multi-seed evaluations (Table X) show overlapping confidence intervals with the full-precision baseline, indicating no statistically significant quality degradation.

Gains are smaller on short-context or highly non-redundant inputs where reuse and pruning opportunities are limited; in these cases the scheduler naturally falls back to conservative decisions (higher precision and lower reuse), and the measured overhead remains bounded by the lightweight predictors (Section IV-C). We additionally cap consecutive reuse ($L_{\max} = 3$) and trigger drift correction/reference checks under instability to prevent error accumulation.

D. Edge Device Deployment

Tables XI–XII evaluate Jetson Orin NX and Raspberry Pi 5 with INT4 Vicuna-7B. Our framework achieves clear latency and memory reductions on both devices (e.g., 2.47× speedup on Jetson for 256 tokens) while introducing only marginal additional quality loss beyond INT4 quantization.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a unified framework for adaptive and efficient inference in large language models, addressing three core challenges: semantic redundancy, static attention computation, and rigid precision control. Our contributions span cross-layer semantic reuse with drift correction, predictive attention pruning conditioned on input complexity, and a self-evolutionary precision scheduler driven by token-level confidence. Through extensive experiments across prompting, reasoning, and speculative decoding scenarios, we demonstrated consistent improvements in latency, throughput, and energy efficiency over recent baselines such as Flash-Decoding, Tree Attention-Medusa, and Radix Attention. Notably, our framework achieves up to 2.43× decoding speedup and 41.7% memory savings while preserving generation quality within 0.4 ROUGE of full-precision outputs. Looking ahead, we plan to integrate our modules into speculative decoding pipelines and explore joint training schemes that co-optimize inference dynamics and model parameters. Moreover, extending our techniques to vision-language models and graph-enhanced Transformers remains an exciting direction for expanding generalizable, token-aware inference strategies.

VII. DATA AND CODE AVAILABILITY

The code implementations supporting the findings of this study are publicly available in the DSA-2026 repository on GitHub at: https://github.com/wnd111/DSA-2026.

REFERENCES

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Red Hook, NY, USA, 2020.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *ArXiv*, vol. abs/2302.13971, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257219404

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[4] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 611–626.

[5] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *CoRR*, vol. abs/1909.08053, 2019. [Online]. Available: http://arxiv.org/abs/1909.08053

[6] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221191193

[7] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: accurate and efficient post-training quantization for large language models," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23, 2023.

[8] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=tcbBPnfwxS

[9] C. Zeng, S. Liu, Y. Xie, H. Liu, X. Wang, M. Wei, S. Yang, F. Chen, and X. Mei, "Abq-llm: Arbitrary-bit quantized inference acceleration for large language models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 21, pp. 22 299–22 307, Apr. 2025. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/34385

[10] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *CoRR*, vol. abs/2106.09685, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[11] M. Sun, Z. Liu, A. Bair, and Z. Kolter, "A simple and effective pruning approach for large language models," in *International Conference on Representation Learning*, B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, Eds., vol. 2024, 2024, pp. 4942–4964.

[12] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, "Squeezellm: Dense-and-sparse quantization," *ArXiv*, vol. abs/2306.07629, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259144954

[13] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. Wei, L. Wang, Z. Xiao, Y. Wang, C. Ruan, M. Zhang, W. Liang, and W. Zeng, "Native sparse attention: Hardware-aligned and natively trainable sparse attention," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 23 078–23 097.

[14] P. Singhania, S. Singh, S. He, S. Feizi, and A. Bhatele, "Loki: Low-rank keys for efficient sparse attention," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37, 2024, pp. 16 692–16 723.

[15] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *ArXiv*, vol. abs/2210.17323, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253237200

[16] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh, "Spqr: A sparse-quantized representation for near-lossless llm weight compression," *ArXiv*, vol. abs/2306.03078, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259076379

[17] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *CoRR*, vol. abs/2004.05150, 2020. [Online]. Available: https://arxiv.org/abs/2004.05150

[18] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, 2020, pp. 17 283–17 297.

[19] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=Ua6zuk0WRH

[20] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," 2023. [Online]. Available: https://arxiv.org/abs/2307.08691

[21] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r1gs9JgRZ

[22] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, and J. Chen, "A review on edge large language models: Design, execution, and applications," *ACM Comput. Surv.*, vol. 57, no. 8, Mar. 2025. [Online]. Available: https://doi.org/10.1145/3719664

[23] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "Sglang: Efficient execution of structured language model programs," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37, 2024, pp. 62 557–62 583.

[24] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, 2022, pp. 24 824–24 837.

[25] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:261697361

[26] Y. Dubois, X. Li, R. Taori, T. Zhang, I. Gulrajani, J. Ba, C. Guestrin, P. Liang, and T. B. Hashimoto, "Alpacafarm: a simulation framework for methods that learn from human feedback," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23, Red Hook, NY, USA, 2023.

[27] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez *et al.*, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," *See https://vicuna. lmsys. org (accessed 14 April 2023)*, vol. 2, no. 3, p. 6, 2023.

[28] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, "HotpotQA: A dataset for diverse, explainable multi-hop question answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2369–2380. [Online]. Available: https://aclanthology.org/D18-1259/

[29] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, 2023, pp. 11 809–11 822.

[30] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple llm inference acceleration framework with multiple decoding heads," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24, 2024.

[31] J. Yao, K. Chen, K. Zhang, J. You, B. Yuan, Z. Wang, and T. Lin, "DeFT: Decoding with flash tree-attention for efficient tree-structured LLM inference," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=2c7pfOqu9k