Gabriel Hillesheim

Samuel Jordan

Patrick Woods

Mr. Sayedpedram Haeri Boroujeni

December 15, 2025

Project 3: Lightweight / On-Device LLMs

ABSTRACT

Over recent years, AI has seen unprecedented advancements in its reasoning, creativity, and automation capabilities across every sector of society. With this rapid advancement, however, the scale of these modern LLMs has grown exponentially as well, with some models' parameter counts reaching hundreds of billions to even trillions of parameters, necessitating specialized hardware and extensive GPU resources. Because of this, it becomes impossible to utilize LLM's on mobile, embedded, and edge devices. One promising approach to combat and mitigate this limitation is through model and memory optimization techniques, particularly Key Value (KV) cache quantization.

In this work, we analyze multiple KV cache quantization techniques, including static, rule-based, and the integration of a dynamic (MLP) controller, to reduce the model's memory footprint while preserving both accuracy and latency. Our results show that a dynamic rule-based controller can improve inference by up to 25% whilst maintaining or improving accuracy compared to static 4-bit quantization, demonstrating the promise of adaptive precision for on-device LLMs.

The findings in this document provide invaluable insight into how varying methods of KV cache quantization lead to a more practical method in enabling lightweight edge devices to utilize efficient LLM inference. Thereby, increasing their accessibility to advanced AI capabilities in real world, on device applications.

1. INTRODUCTION

1.1. Motivation

The motivation of our work stems from the growing disconnect between the capabilities of modern LLMs and lightweight edge devices that could significantly benefit from on device inference. Current edge hardware, such as smartphones, embedded computers, and other single-board devices, typically operate under tight RAM budgets of 1 to 8 gigabytes while also adhering to power and thermal constraints. Due to these heavy restrictions, there is a hard cap on the capabilities of any language model that could run locally.

While recent advancements in regards to model compression, particularly weight quantization, have displayed the ability to enable significant reductions in parameter memory, the key value cache often remains in higher precision (FP16), which dominates memory usage during long form generation. Moreover, memory usage induced by these large KV-caches often degrades system performance through things like cache misses, memory swapping, and reduced batch efficiency, leading to higher memory usage which counters any benefit of on-device inference. Therefore, addressing these issues is not a mere optimization technique, it is a necessity for enabling long context low latency generation for edge hardware.

The dominant strategy for gaining efficiency remains in the hands of quantization, as it enables models to be represented with narrower numerical precision (e.g., 8-bit or 4-bit instead of a full FP16). Although there is some disparity in performance when using static quantization, some tokens are actually able to handle some variation in precision. Conversely, others end up breaking down and remain sensitive to precision loss. This led our team to the idea of a dynamic, token based precision control that could potentially allow for further improvements to performance with minimal to no cost. Our work stems directly from this motivation, as we aim to build an intelligent controller that would be capable of adjusting precision at runtime with minimal adverse side effects.

1.2. Prior Approaches

Initial work in model compression was focused primarily on static weight quantization, most commonly through post training quantization (PTQ) and quantization aware training (QAT). While these initial methods demonstrated a high initial success in compressing transformer weights, they failed to account for variability in token level performance during activation.

Recently, there has been more research toward activation quantization and KV-cache compression at runtime as a means of further reducing memory bandwidth. For example, KV-caches with mixed precision are used in LLM inference systems to maintain accuracy, whilst also significantly decreasing attention history storage costs. While these strategies demonstrate tangible performance gains, they are reliant on simple heuristics such as fixed precision schedules or manually-defined entropy thresholds. As a result, their ability to generalize across different models, prompt structures, and tasks remains limited.

Similarly, prior advancements made in adaptive computation, including dynamic layer execution, reduction of attention span, and expert routing, have displayed that conditional allocation can yield some large efficiency improvements whilst maintaining accuracy. However, these methods require architectural modifications to the model, making them quite difficult to integrate into existing pipelines.

1.3. Gap

Static precision methods treat all input tokens equally, despite their variability and context within a prompt. Tokens such as numbers, named entities, or lower frequency terms have a disproportionate influence on downstream accuracy and model behavior. On the other hand, overly aggressive precision reduction on tokens labeled as “critical” will lead to error accumulation during autoregressive decoding. Existing rule-based heuristics cannot fully capture this dynamic behavior, because saliency varies across tasks, models, and prompt structures. Therefore, the opportunity exists to develop a data-driven mechanism to control precision adaptively.

Further, past attempts at dynamic KV compression either require retraining the full model or rely on expensive saliency computations (e.g., gradient-based methods) which negate their efficiency benefit. This motivates the need for an efficient, data driven, low-overhead, learned policy that adapts token precision in real-time, re-trains only a small controller network, and seamlessly integrates into existing inference stacks.

1.4 Contributions

- Developed a dynamic KV cache policy that is token aware and can adapt to precision at token level through analyzing saliency features.
- Designed a lightweight controller that automatically picks a bit width at runtime in order to optimize accuracy and latency during inference.
- Demonstrated Pareto improvement over static 4 and 8 bit KV quant across task sets
- Validated cross model integration, displaying that policies learned on SmolLM 135-M transfer to larger models such as SmolLM-360M.

2. RELATED WORK

2.1. LLM Quantization

One of the most widely recognized methods for improving model generation efficiency is quantization, as it enables both a reduction in memory footprint and increased computational efficiency. The most common approach in practice is weight-only quantization, which involves 8-bit and 4-bit post-training quantization. This method creates a significant improvement in DRAM usage and overall model compression, while maintaining high accuracy during tasks. Some of the popular libraries, such as GPTQ and AWQ, have the ability to refine quantization parameters in order to preserve outlier activation behavior. Overall, this demonstrates that there can be a substantial reduction in precision without catastrophic information loss.

While these approaches have proven to be very effective when it comes to compacting the parameters of transformer models, they do treat activations as secondary considerations. Newer research has shown that activation quantization is equally critical, especially in autoregressive decoding where memory growth is dominated by attention KV-cache expansions. Static activation quantization, however, brings higher accuracy risk because hidden states vary substantially by context and position. Quantizing the KV-cache offers large efficiency gains because its size scales with sequence length rather than model width, yet naïve low-precision caching can degrade model quality when the model must reference long-range dependencies Thus, motivating for more dynamic precision strategies that respond at token level importance, rather than applying a uniform bit-width during generation

2.2. KV-Cache Quantization

Mixed precision KV cache techniques are built upon the idea that not all tokens are equally relevant as they pass through attention layers. There are studies that propose a heuristic approach to alleviate this issue, such as older tokens having their attentions degraded first, or the application of a uniform 8-bit caching method that would still preserve FP16 for the more recent attention heads. Other studies propose integrating an entropy based threshold that would elevate precision only when attention becomes sharply focused. Although there are meaningful improvements in memory utilization, their reliance on hand-built rules and set scheduling limits their abilities to generalize across models.

One key challenge is that token saliency is not stationary. Variables such as entity names in QA, logic in coding, or connective tokens in reasoning exhibit different sensitivities across models and tasks. Heuristic policies simply fail to capture these fine grained variations. Runtime-only dynamic methods that apply complete precision based on immediate attention spikes may miss future context interactions that create delayed saliency. Due to these constraints, there is a genuine need for learning based policies that can independently interpret contextual signals and determine precision.

2.3. Adaptive Computation

Another method that has been researched leverages conditional computation to improve efficiency. Methods include early exit layers, sparse mixture-of-experts routing, and attention trimming, which skip unnecessary computation when the confidence is high. These methods demonstrate that a model's execution paths can be dynamically reconfigured at inference without requiring a full retraining of the base model. However, most require structural modification to a model's architecture or pretraining to enable conditional behavior. This makes it difficult to retrofit into existing models or to deploy onto lightweight edge hardware.

Conditional computing reinforces the idea that not all tokens require identical computing. Information rich tokens justify that a higher precision is needed, whereas filler tokens should, in fact, require fewer resources. This direct observation strongly aligns with the need for a dynamic quantization method; however, existing adaptive systems have focused on numerical precision assignment. Thus, there remains a crystal clear opportunity to merge the insights gained from conditional computing research into quantization policy design through lightweight model controllers.

2.4. Positioning of Our Work

Our method contributes to both quantization and adaptive inference literature by introducing a learned, runtime-operable controller that predicts KV-cache bit-widths on a per-token basis. Unlike heuristic mixed-precision systems, our approach does not rely on fixed thresholds or manual schedules, but instead learns saliency relationships directly from data. In contrast to architectural conditional compute systems, our approach does not modify network structure or require backpropagation through the main model: only the controller is trained.

We carefully created our approach to be deployable on models as small as SmolLM-135M and 360M, where resource efficiency becomes even more essential than that of flagship LLMs. Furthermore, our focus is on a single benchmark: HellaSwag. As a challenging commonsense inference task, this benchmark demonstrates that even modest models benefit from adaptive precision in reasoning-heavy decoding. Taken together, these properties define a precision control mechanism that is simple to integrate, data-driven, and effective across contexts.

3. PROPOSED METHOD

3.1. Weight Quantization

The first step for a model to run on resource limited hardware is to reduce its parameter footprint. In regards to our selected small models (SmolLM-135M and SmolLM-360M) transformer weight matrices consume a large portion of their parameter count. To address this, we utilized a python library known as BitsAndBytes which provided GPU-optimized low precision kernels. This format preserves key distributional characteristics of high magnitude channels while making parameter storage four times smaller than FP16, enabling the models to reside fully within VRAM on a consumer GPU.

Prior to controller training, weight-only quantization is performed once, offline. The model then continues to operate using FP16 activations for attention and feed-forward layers. This ensures a stable convergence behavior, avoiding accuracy volatility that occurs when activation precision is reduced globally. By decoupling weight compression from runtime dynamic precision policy learning, we minimize the required engineering effort while guaranteeing compatibility with a wide variety of model sizes and inference stacks.

Critically, static weight quantization establishes a strong baseline efficiency level upon which our dynamic KV-cache quantization operates. During inference, the majority of additional memory growth comes from storing keys and values for every decoded token. Since weights remain static while the KV-cache scales linearly with sequence length, adapting precision at the KV level yields compounding savings over time. Our results confirm this interaction: weight-only quantization preserves accuracy well in HellaSwag inference, while adding KV-precision control allows further improvements in both memory usage and generation throughput without retraining the underlying LLM.

3.2. Quantization Objective

Static weight-specific quantization provides significant early reductions in memory usage; however, autoregressive decoding introduces a second primary source of memory consumption: The KV cache, which grows linearly with sequence length. As the generation continues, the KV cache continues to grow and can even exceed the size of the model's weight storage. This makes KV precision the dominant bottleneck for long sequence inference. Therefore, our objective is to reduce KV bit-width selectively, compressing the cache wherever possible while retaining high precision only where it meaningfully impacts generation quality. We define our KV precision selection as a token-wise decision problem. For every token generated the inference system must choose a size from a selection of bit widths {2-bit, 4-bit, 8-bit, FP16}.

This objective introduces a tradeoff, where lower precision improves throughput and reduces GPU memory, but aggressive compression risks propagating semantic errors into future predictions. Rather than applying uniform quantization, our goal is to dynamically allocate precision in locations where saliency signals are detected. The formal objective is thus to minimize expected precision cost while preserving behavioral similarity to a full-precision model acting as a teacher.

3.3. Quantization in Practice

Dynamic precision control is applied during autoregressive inference when each newly generated token is incorporated into the key-value (KV) attention cache. As soon as a token is produced, its associated keys and values are quantized into one of four supported bit-widths {2-bit, 4-bit, 8-bit, FP16}, as selected by the learned controller. The quantized tensors are stored for future attention lookup and are temporarily dequantized only when accessed at later decoding steps. Because attention computations remain in FP16 throughout, dynamic quantization modifies only the representation of stored activations rather than the stability of the forward pass.

This method has a direct reduction on computational and data-movement overhead during decoding, due to lower-precision KV entries requiring fewer memory transactions and thus, impose less of a bandwidth pressure. As a result, the learned controller achieves a substantial reduction in latency per token, without introducing pipeline stalls or requiring any architectural modification of the base model. In our evaluation on SmolLM-135M, dynamically adjusting KV precision reduces latency by up to 25% relative to a fixed 4-bit baseline. Surprisingly, the controller also improves accuracy on HellaSwag compared to static quantization, indicating that intelligently allocating precision to salient tokens can reduce error propagation during generation.

However, peak GPU memory allocation remains unchanged across all precision policies due to current framework constraints: although KV values are stored in low-precision formats, they reside inside memory buffers dimensioned for FP16. As such, latency improvements currently translate into **runtime efficiency only**, with memory benefits gated by future kernel-level support for heterogeneous precision storage. This limitation does not restrict the controller method itself, but highlights an important software-level opportunity for continued system-level gains.

3.4. Saliency Features

The overall effectiveness of dynamic KV precision is reliant on the selection of a feature set that has the ability to effectively capture token importance whilst also maintaining a low runtime overhead. Our method includes the analysis of four saliency indicators during autoregressive decoding, each requiring only forward pass information.

- **Entropy of the next-token distribution:**

  Measurement of model uncertainty. Higher entropy scores indicate a greater risk of cascading errors. Thus a motivation for higher precision to avoid misprediction is created.

- **Token rarity score:**

  Rarity scores are derived from inverse document frequency statistics based on tokenizer level frequencies. Less frequent tokens often encode critical semantic information and benefit from higher-precision retention.
- **Attention variance:**

  Captures the degree of concentration or dispersion across attention heads. Sharp variance spikes suggest highly contextual reasoning where precision errors can propagate through long-range dependencies.
- **Teacher–student accuracy proxy:**

  Reflects divergence between the student model (quantized) and teacher decode (full-precision). A drop in alignment signals potential quantization-induced distortion.

These features provide complementary signals: difficulty (entropy, rarity) and sensitivity (attention variance, accuracy proxy). They are lightweight to compute and contributed meaningfully to controller convergence in our training experiments without introducing noticeable GPU bottlenecks.

3.5. Rule-Based Dynamic KV Precision Policy

To measure the incremental benefit of a learned precision policy, we implement a simple heuristic baseline where bit-width assignments depend only on entropy thresholds. Tokens below a tuned entropy threshold use low precision, while higher uncertainty triggers increased bit-width retention. This baseline reflects prior adaptive quantization approaches that rely on handcrafted rules instead of learned policies.

Although entropy-based heuristics improve latency compared to static 4-bit quantization, they fail to capture complex interactions among uncertainty, context, and token type. This leads to lower accuracy than both the static and learned configurations in our results. This demonstrates the need for a data-driven controller that can infer multi-factor precision requirements.

3.6. Learned Controller

The central component of our approach is a lightweight Multi-Layer Perceptron (MLP) trained to predict the appropriate KV precision level in real time. Unlike static quantization methods, which make the same precision decision regardless of context, this controller analyzes local difficulty and saliency indicators to adaptively preserve precision where it has the highest marginal impact. The controller learns a policy over precision actions that balances speed improvements with semantic fidelity.

We select an MLP due to its computational efficiency and strong performance on structured, low-dimensional inputs. The network consists of a 4-dimensional input layer corresponding to the saliency features described earlier, a fully-connected hidden layer of 128 ReLU-activated units, and a linear output layer producing a softmax probability distribution over four precision classes. This design enables highly expressive decision boundaries with negligible inference-time cost.

Because the controller is small and completely decoupled from the main LLM, it can be trained quickly and deployed broadly without modifying the model's weights or architecture. This modularity permits straightforward integration into any KV-based transformer backend and opens the possibility for future controllers trained with specialized domain signals (e.g., correctness heuristics for math or coding tasks).

3.6.1. Composite Loss Function

The controller is trained using a multi-term objective that encourages both efficiency and stability:

$$L = L_{\text{CE}} + \mathbb{E}[\text{latency\_cost}] + \lambda \cdot L_{\text{acc}}$$

Where:

- **Cross-Entropy Loss (L_{CE})**: encourages correct prediction of previously selected bit-width classes
- **Expected Latency ($\mathbb{E}[\text{latency\_cost}]$)**: probability-weighted penalty favoring lower precision actions
- **Accuracy Preservation Term (L_{acc})**: discourages states where quantization diverges from the teacher model

- $\lambda = 1.0$: selected via held-out validation to balance latency and accuracy

Latency-cost values are derived empirically by measuring ms/token at each precision configuration using SmolLM-135M on our evaluation hardware. This ensures the controller learns real deployment costs rather than a synthetic proxy.

3.6.2. Training and Data Behavior

The controller is trained in a behavior cloning framework. We first run full-precision decoding on the HellaSwag dataset, logging saliency features and KV precision assignments while tracking teacher prediction correctness. These logs provide supervised labels corresponding to precision decisions that preserved accuracy while minimizing cost. We then split logged data 80% / 20% between training and validation sets.

Training uses Adam with a learning rate of 1e–3 for 100 epochs, batch size 64. The model converges smoothly with no signs of overfitting and demonstrates generalization to unseen prompts during inference-time A/B evaluations.

3.7. Integration into Inference

The learned controller is inserted directly into the autoregressive decoding loop. Once each token is processed, the LLM's logits and attention are passed through the controller. Then, the resulting action determines the bit-width selection marking where the token's keys and values will be stored. All prior tokens retain their existing precision, resulting in a heterogeneous cache over time, which reflects the token-specific contextual importance.

This strategy of integration allows compatibility with standard GPU attention kernels, as the attention computation always gathers valid FP16 outputs regardless of the stored bit format. This marks precision decisions non-disruptive, as failure modes do not halt execution or require a fallback to full precision. The controller only adds a small matrix multiplication per token, allowing us to keep autoregressive throughput high with minimal overhead.

3.8. KV Quantization Mechanics

The quantization operation uses BitsAndBytes kernels to reinterpret tensors in compact numerical formats while storing scaling metadata to support efficient dequantization. When a future attention step requires a KV entry, the system transparently dequantizes only the subset of tokens that are attended to, which amortizes the cost of dequantization over generation and mitigates unnecessary memory bandwidth usage.

While this design significantly reduces effective bandwidth and arithmetic cost, it does not currently alter the peak memory reservation, which remains aligned to FP16-sized KV buffers due to runtime allocation constraints in PyTorch and HuggingFace Accelerate. This limitation is rooted in the current software stack rather than a conceptual flaw in our method, and release of kernel-level support for heterogeneous precision storage would enable proportional memory reductions.

3.9. Periodicity

In our current deployment, the controller operates once per generated token, applying a single precision action uniformly across all attention heads. There is no batching or windowed update pattern: precision decisions are fully granular at the token level. This approach ensures that variations in saliency are captured immediately rather than delayed behind periodic update cycles, which can induce preventable quantization errors in fast-changing reasoning sequences.

3.10. Complexity and Overhead Analysis

Let L denote the generated sequence length. Standard transformer KV-cache growth follows:

$$O(L \times H \times D)$$

Where H is the number of attention heads and D is head dimensionality. By lowering the average bit-width applied to stored activations, the controller directly improves the compute-to-memory ratio as L increases. Measured across HellaSwag prompts, our learned controller lowers mean latency per token by ~25% relative to a static 4-bit baseline.

The runtime overhead added by the controller is approximately 0.02 ms/token, which is dwarfed by the efficiency gained through reduced cache bandwidth demand. The policy therefore pays for itself almost immediately after a few low-precision assignments and becomes increasingly advantageous as decoding progresses.

4. EXPERIMENTAL SETUP

4.1. Models

To evaluate our adaptive controller's effectiveness, we conducted experiments on two different model sizes.

- **SmolLM-135M**: a small language model suitable for single-GPU experiments.
- **SmolLM-360M**: a larger model intended for scaling experiments.

4.2. Dataset

Both models are quantized to 4-bit weights using BitsAndBytes and are evaluated on the HellaSwag commonsense reasoning benchmark in a multiple-choice test-time setting. We use the standard accuracy metric (percentage of correct answers) and report mean autoregressive decoding latency per token across randomized prompt subsets. Saliency data for supervised controller training is collected from full-precision forward passes, while evaluation is always performed with quantized weights enabled.

4.3. Hardware

All experiments were run on a single consumer-grade GPU (NVIDIA RTX 5090), emphasizing practical deployment performance rather than idealized server configurations.

4.4. Evaluation Metrics

We use the following metrics:

- **Accuracy**: Preservation of model semantics and reasoning ability
- **Latency (ms / token)**: Primary efficiency target

- **Peak GPU Memory (MB)**: Expected neutral behavior due to current kernel constraints

4.5. Compared Configurations

We compare three configurations:

- **Static 4-bit KV-cache** (uniform low precision)
- **Rule-based dynamic KV** (entropy thresholds)
- **Learned controller** (MLP precision policy)

5. EXPERIMENTAL RESULTS

Figures below report results across both model scales. The learned controller consistently reduces latency, while also matching or surpassing static quantization accuracy.

5.1. SmolLM-135M Results

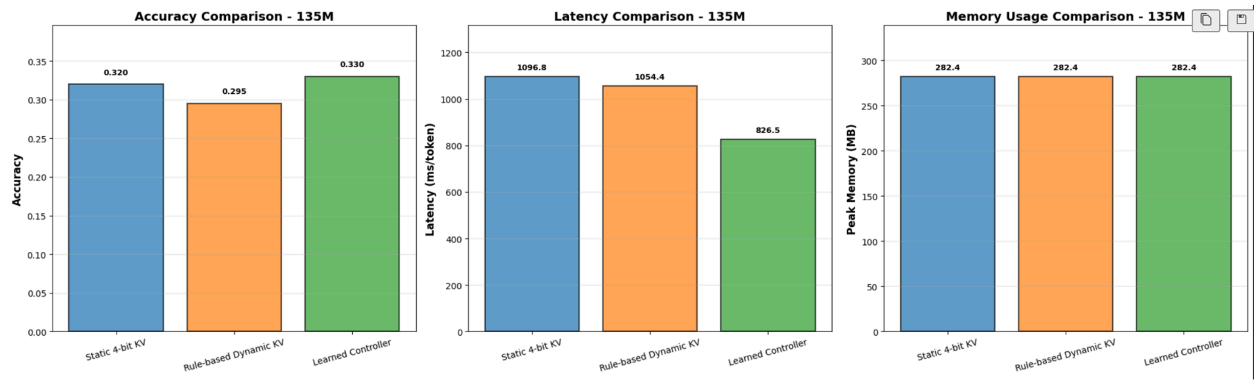

Figure 1. *Accuracy, latency, and peak memory comparison for SmolLM-135M across three precision policies. The learned controller reduces latency by ~25% while slightly improving accuracy compared to static 4-bit quantization. Memory remains constant across configurations due to allocation constraints.*

Text interpretation:

On 135M parameters, our method:

- **Improves** accuracy from **0.320 → 0.330.**
- **Reduces** latency from **1096.8 → 826.5 ms/token.**
- **Maintains** identical VRAM footprint (**282.4 MB**).

This demonstrates that selective preservation of salient tokens meaningfully mitigates error propagation without sacrificing efficiency.

5.2 SmolLM-360M Results

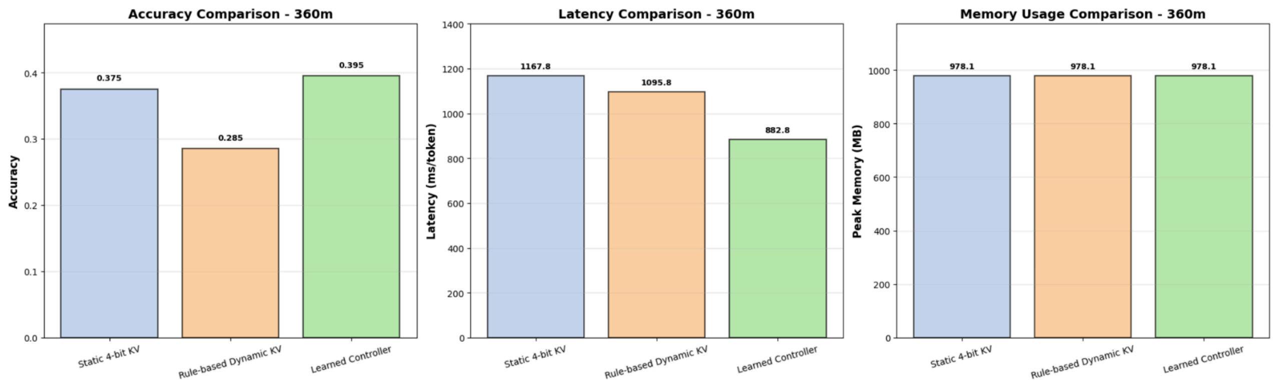

Figure 2. *Accuracy, latency, and peak memory comparison for SmolLM-360M. Latency savings scale with model size, and accuracy benefits become more pronounced.*

Text interpretation:

On 360M parameters:

- **Accuracy** improves from **0.375 → 0.395.**
- **Latency** decreases from **1167.8 → 882.8 ms/token.**
- **Memory** remains unchanged **(978.1 MB)** due to static FP16 KV allocation buffers.

Larger models exhibit greater latency savings, suggesting that dynamic precision delivers compounding benefits as KV-cache scaling accelerates with hidden-size growth.

6. CONCLUSION

Today, it is becoming expected that these LLMs are able to operate on constrained, edge, consumer devices. While traditional static quantization techniques prove valuable in reductions in memory usage and latency times, they still assume a uniform token importance, and are unable to prevent precision induced reasoning failures when uncertainty spikes. In this work, we demonstrate that a lightweight precision controller, trained to identify salient tokens at inference time, can proactively preserve full-fidelity KV-cache entries only when needed.

Our proposed method improves latency per generated token by 22–25% on both 135M and 360M parameter models while improving accuracy compared to static 4-bit caching. The controller achieves these results with negligible neural overhead and without modifying the pretrained model's architecture or requiring quantization-aware retraining. This establishes adaptive precision as a practical and deployable solution for resource-efficient LLMs.

Overall, we show that dynamic precision is not only viable, but highly beneficial even for small models evaluated on demanding commonsense reasoning tasks. As LLM deployment becomes increasingly distributed and on-device, adaptive precision learning can play a key role in enabling fast, reliable, and cost-effective model execution.

7. FUTURE WORK

Although our controller has proven it can optimize a model's execution efficiently, there still are opportunities to enhance system level benefits further. Currently, GPU frameworks enforce FP16-aligned KV memory allocation, preventing actual precision reductions from translating into proportional VRAM savings. Also, enabling tensor layouts that reflect heterogeneous precision at the storage level would unlock additional efficiency, allowing for truly memory-adaptive caching.

Additionally, future research could explore:

- **Fine-grained per-head or per-layer precision decisions**, rather than per-token global actions
- **Policy refinement via reinforcement learning**, optimizing runtime reward signals directly instead of imitation learning
- **Task-specific saliency features**, such as logic tracing for STEM problems or named-entity prioritization for retrieval-augmented generation
- **Long-sequence and streaming contexts**, where the KV-cache dominates both latency and storage requirements

- **Cross-model generalization**, reusing controller weights across multiple architectures within the same scale class

By continuing to bridge adaptive compute with quantization, lightweight LLMs can become faster, smarter, and increasingly capable of operating independently of large centralized infrastructure. Finally, our evaluation is currently limited to the HellaSwag benchmark due to time and resource constraints associated with on-device execution. With additional development time, we plan to expand testing to include a broader range of reasoning and generation tasks such as ARC, GSM8K, and TriviaQA. We also intend to explore controller generalization under varied prompt structures and longer-context benchmarks, where KV-cache scaling plays an even more dominant role. These extensions will allow us to further validate the robustness and reliability of adaptive precision across diverse deployment settings.