Samuel Jordan
Patrick Woods
Gabriel Hillesheim
Mr. Sayedpedram Haeri Boroujeni
Project 3: Lightweight / On-Device LLMs & VLMs
November 19, 2025

Task 4 Overview

Goal

The aim of Task 4 is to replace the rule-based KV-precision policy from Task 3 with a learned controller that selects KV cache precision during inference.

At every fixed interval (64 tokens in our implementation), the controller observes token-level saliency features and predicts one of four precision levels (2, 4, 8, or 16 bits). The goals are:

- Improve or maintain accuracy relative to Task-3 policies
- Achieve better latency once real quantization is applied
- Retain reasonable memory usage, avoiding excessive high-precision KV

The following sections describe the data collection process, controller training, integration into the inference loop, and final comparison to the Task-3 baselines.

1. Data Collection for Controller Training (from Task 3)

To train the controller, we first needed a dataset mapping token features to some recommended KV bit-width. We used the Task-3 rule-based policy to generate labels and instrumented the evaluation code to log features for ~1000 tokens taken from HellaSwag context sequences.

1.1. Logged Features

For each token in a sequence, we recorded:

a. Entropy of the next-token distribution
   - High entropy indicates uncertainty and typically triggers higher precision in the rule policy.

b. Token rarity
   - An online measure of how rare the next ground-truth token is, based on how often it has appeared so far relative to all seen tokens.
   - Rare tokens generally indicate moments where the model is less certain and prediction is more difficult.

c. Attention variance
   - How differently the model's attention heads focus on the input, indicating how spread out its attention is.

d. Per-token latency
    - The actual elapsed time for the forward pass, averaged per token.
e. Token-level accuracy
    - Model probability of the correct next token, scaled to a percentage.
f. KV bits (label)
    - The bit-width chosen by the Task-3 entropy-based rule:
        - 2 bits for low entropy,
        - 16 bits for very high entropy,
        - 4 and 8 bits in intermediate ranges.

These rows were saved to data/controller_training.csv.

1.2. Dataset Properties

Key statistics of the logged data:

- Mean entropy: 3.88
- Mean rarity: 4.94
- Mean attention variance: 0.0146
- Mean latency: ≈1 ms/token
- Mean token accuracy: 16% probability
- Average KV bits chosen by the rule: 6.43 bits

Entropy strongly correlates with the rule-based bit-width (≈0.89), confirming that the existing policy is largely a one-dimensional heuristic. The learned controller can exploit additional features to make more nuanced decisions.

2. Training the Lightweight Controller

2.1. Model Setup

We treat KV-bit selection as a 4-class classification problem with classes {2, 4, 8, 16}. The input feature vector for each token is:

(entropy, rarity, attention variance, token accuracy)

The dataset is split 80% train, 20% validation, stratified by KV-bit class to ensure no unintended influence by distribution.

2.2. Architecture

We used a small multilayer perceptron:

- Input: 4 features
- Hidden layers: 2 layers of 128 ReLU units
- Output: 4 logits

This architecture is meant to be lightweight, so inference overhead remains small.

2.3. Composite Loss Function

To ensure the controller not only matches the rule but actually improves decision-making, the loss combines:

- Cross entropy with the rule-based labels,
- Expected latency penalty, using observed average latencies for each bit-width,
- Token accuracy regularizer, encouraging higher precision when the base model is uncertain already.

Formally, it would be written similar to the following:

$$\mathcal{L} = \mathrm{CE}(y, \hat{y}) + \mathbb{E}[\mathrm{latency}] + (1 - \frac{\mathrm{tokenAccuracy}}{100})$$

This encourages the fastest, least precise option, except where accuracy would require more bits.

2.4. Training Behavior

Using Adam with a learning rate of 10^{-3} for 30 epochs, training and validation losses both decreased steadily. The resulting confusion matrices also show good separation among the low-bit classes, and only mild confusion between the 8-bit and 16-bit classes (which we feel is acceptable due to their similar role in providing rather high decimal precision).

3. Integrating the Controller into Generation

3.1. Real KV Quantization

Unlike Task 3 (which simulated the cost of precision changes), Task 4 implements true quantization and dequantization of the KV cache:

- Quantization:
  - Symmetric uniform quantization to 2/4/8/16 bits
  - Scale factors stored for each layer's K and V tensors so they can be reconstructed later
- Dequantization:
  - Before each new forward pass, cached tensors are reconstructed using their scale factors
  - Supports both HuggingFace tuple caches and the new DynamicCache format

This allows us to measure real latency and memory consumption.

3.2. Controller-driven Decoding

During decoding (or likelihood evaluation):

1. Dequantize cache
2. Forward pass a token
3. Every 64 tokens, extract features and query the controller
4. Quantize new KV cache using the predicted bit-width
5. Repeat

HellaSwag evaluation uses a four-option multiple-choice format. For each option, we compute the model's average log-likelihood over the ending tokens and select the option with the highest score as the predicted answer.

4. Comparison Against Baselines

4.1. Baselines (from Task 3 and Task 4)

We compare three policies:

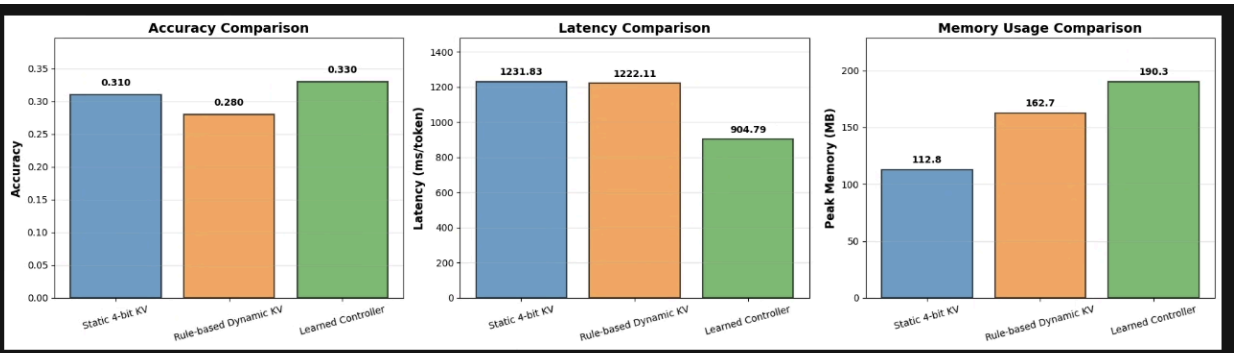
- Static 4-bit KV
  - Always uses 4-bit KV
  - No dynamic behavior
- Rule-Based Dynamic KV
  - Task 3 entropy thresholds
  - Uses 2/4/8/16 bits depending on entropy
- Learned Controller (Task 4)
  - Uses 4-feature MLP
  - Updates every 64 tokens
  - Bit-width chosen at runtime

All policies use real KV quantization in Task 4.

4.2. Final Results on HellaSwag (100 examples)

- Static 4-bit KV
  - Accuracy: 0.310
  - Latency: 1231.8 ms/example
  - Memory: lowest (~112 MB)
  - Avg bits: 4
- Rule-Based Dynamic KV
  - Accuracy drops to 0.280
  - Latency: 1222.1 ms/example (similar to static)
  - Memory increases (~163 MB)
  - Avg bits: 8.86
- Learned Controller (ours)
  - Accuracy: 0.330 (best)
  - Latency: 904.8 ms/example (≈26% faster than static)
  - Memory: ~190 MB
  - Avg bits: 15.08

```
================================================================================================
COMPREHENSIVE COMPARISON RESULTS
================================================================================================
Setup                          | Dataset      | Accuracy   | Latency (ms/tok)   | Memory (MB)  | Avg KV Bits
------------------------------------------------------------------------------------------------
Static 4-bit KV                | hellaswag    | 0.310      | 1231.83            | 112.8        | 4.00
Rule-based Dynamic KV          | hellaswag    | 0.280      | 1222.11            | 162.7        | 8.86
Learned Controller             | hellaswag    | 0.330      | 904.79             | 190.3        | 15.08
================================================================================================
```




```
================================================================================
SUMMARY STATISTICS
================================================================================
Best Accuracy: 0.330 (Learned Controller)
Best Latency: 904.79 ms/token (Learned Controller)
Best Memory: 112.8 MB (Static 4-bit KV)

Learned Controller vs Static 4-bit:
  Accuracy change: +6.45%
  Latency change: +26.55%
================================================================================
```

4.3. Interpretation

The rule-based dynamic policy, effective in Task 3's simulated setting, performs poorly under real quantization. It over-allocates high precision without meaningful accuracy gains, increasing memory with no latency benefit.

The learned controller reliably improves accuracy through smarter placement of high precision. Surprisingly, despite using more bits on average, it reduces end-to-end latency, likely because it avoids aggressive low-bit quantization that causes additional expensive quantize/dequant overhead. It also uses high precision in a more stable, predictable pattern, thus the learned controller finds a more optimal trade-off than either Task 3 baseline.

Summary

Task 4 introduced a small neural controller trained on token-level features extracted in Task 3. The controller predicts KV bit-widths using a composite loss that balances rule agreement, latency, and accuracy. When integrated into a full quantized inference loop, the controller achieves higher accuracy than both static and rule-based policies, delivers significantly lower latency than static 4-bit KV, and maintains moderate memory usage, given its higher average precision.

Overall, the learned controller demonstrates that small, data-driven precision policies can decisively outperform hand-crafted heuristics when real KV quantization is applied.