DynaQuant: RL-Based Dynamic Quantization for Large Language Models

Oleg Roshka [1] **Ilia Badanin** [2]

Abstract

Large Language Models (LLMs) often suffer from excessive memory footprints and slow inference, creating barriers to deployment on commodity devices or in high-traffic scenarios. Quantization—using lower numerical precision—is a straightforward way to compress model size, but applying a *uniform* precision (e.g., 8-bit) is suboptimal: some layers tolerate lower precision better than others. We propose **DynaQuant**, a Reinforcement Learning (RL) framework that *dynamically* selects per-layer quantization schemes (e.g., 4-bit `nf4`, `fp4`, `int8`, or `fp16`) during short, iterative fine-tuning. Specifically, an RL agent observes each layer's statistics, memory usage, and partial performance signals to choose how to quantize that layer. We employ a multi-term reward that balances perplexity, KL divergence (vs. a reference model), attention entropy changes, and memory savings. Experiments on GPT-2 over BoolQ and PIQA show that DynaQuant finds *mixed-precision* assignments that outperform uniform 4-bit or 16-bit baselines in perplexity and accuracy, while still yielding significant memory savings.

1. Introduction

Recent progress in Large Language Models (LLMs) has driven state-of-the-art results in various NLP tasks. However, the rapid growth in parameter counts (hundreds of millions to billions of parameters) poses significant memory and runtime challenges, especially for resource-constrained deployments or high-throughput applications. *Quantization* is a practical approach to reduce model size by lowering numerical precision (e.g., from float16 to 8-bit), enabling smaller memory footprints and often faster inference (Dettmers et al., 2022; Frantar et al., 2022).

Yet, standard uniform quantization assigns the same bit-width to all layers, which can be suboptimal. Some layers—particularly in attention or feed-forward blocks—are more sensitive to precision loss, while others can be safely compressed to 4-bit or even 2-bit with minimal accuracy degradation (Dong et al., 2019; Dettmers et al., 2023). This motivates *mixed-precision quantization*, where each layer's precision is chosen adaptively.

We present **DynaQuant**, a reinforcement learning (RL) framework that *sequentially* determines each layer's quantization scheme. Specifically:

1. We treat each layer as a step in an RL episode.
2. The RL agent selects one action from $\{\texttt{nf4}, \texttt{fp4}, \texttt{int8}, \texttt{fp16}\}$ for that layer.
3. After quantizing, we perform a short fine-tuning step on that partially quantized model to mitigate accuracy loss.
4. We compute a *reward* that balances perplexity (vs. a reference model), KL divergence, attention entropy changes, and memory savings.

By the end of one *episode* (quantizing all layers in sequence), we have a fully quantized model with a *mixed-precision* assignment across layers.

Empirically, we demonstrate that **DynaQuant** discovers policies that *improve* perplexity and often accuracy relative to uniform 4-bit or 16-bit baselines, with memory usage that falls between those extremes. Our code is built on GPT-2 as a testbed and extends readily to other LLMs.

2. Related Work

LLM Quantization. Numerous works compress large models with low-precision formats: int8 (Dettmers et al., 2022), 4-bit normal float (nf4) (Frantar et al., 2022), and others (Malinovskii et al., 2024). Typically, a *uniform* scheme is used. Our approach is complementary, as we select distinct bit-widths across layers via RL.

[1]Department of Computer Science, Stanford University, Stanford, CA, USA [2]École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. Correspondence to: Oleg Roshka <oros@stanford.edu>, Ilia Badanin <ilia.badanin@epfl.ch>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Mixed-Precision and NAS. Prior research in Neural Architecture Search (NAS) explores per-layer bit allocations (Dong et al., 2019; Wang et al., 2020), but typically focuses on smaller networks and does not fully account for the *layer-by-layer dynamic impact* of quantization. We adapt these ideas for large Transformer-based LLMs by introducing short, iterative fine-tuning for each layer alongside multi-objective RL signals (e.g., perplexity, KL, memory, attention). Crucially, we fine-tune *both* the quantized model *and* an *episodic reference model* **after every layer is quantized**, using the same training data for a fixed number of steps. This allows us to precisely capture how quantizing one layer affects subsequent performance—something, to our knowledge, not explored by existing mixed-precision quantization methods for large LLMs.

RL for Model Optimization. Prior works have employed reinforcement learning to discover neural architectures (Zoph & Le, 2016) or prune channels (He et al., 2018). We build on these techniques by designing a *custom RL environment* tailored to quantizing large Transformer models *layer-by-layer*. In our setup, an agent selects from multiple bit-width formats at each step, and the environment provides a reward based on changes in perplexity, KL divergence, attention entropy, and memory usage. This allows the policy to optimize a *dynamic, multi-term objective* for LLM quantization, rather than applying a static, uniform scheme.

3. Methodology

3.1. RL Environment: Dynamic Layer Quantization

We maintain two models: a reference model $\mathcal{M}_{\text{ref}}$ (e.g., GPT-2 in FP16) and a copy $\mathcal{M}_{\text{quant}}$ for quantization. Each **episode** processes all N layers in sequence, where each layer corresponds to an RL step:

1. *State* s_i: incorporates diverse features of the current model and layer. Specifically, we extract:
   - *Layer statistics:* mean and standard deviation of weights, gradient norms, and attention entropy for the layer being quantized.
   - *Global signals:* the normalized layer index (i/N), current model perplexity, and perplexity deltas (how quantizing previous layers changed perplexity).
   - *Previous layer's quantization choice:* encoded as a numeric ratio (e.g., bit-width divided by 16).
   - *Exponential moving averages (EWAs):* we maintain running EWAs of key reward components (performance, KL, entropy, memory), smoothing the training signal over steps.
2. *Action* a_i: selecting the quantization format from $\{\texttt{nf4}, \texttt{fp4}, \texttt{int8}, \texttt{fp16}\}$ for layer i.
3. *Transition*: we quantize layer i in $\mathcal{M}_{\text{quant}}$, then apply *short iterative fine-tuning* on both $\mathcal{M}_{\text{ref}}$ and $\mathcal{M}_{\text{quant}}$, using the *identical* mini-batch for a fixed number of steps. This ensures the agent observes the immediate impact of quantizing that layer.
4. *Reward* r_i: once both models are briefly fine-tuned, we measure the resulting perplexities, KL divergence, attention entropy, and memory savings. These are combined (with EWAs) into a multi-term scalar reward. Notably, although we do not include direct "memory usage" in the state vector, the *reward* reflects how many bits are saved by quantizing the current layer, weighted by that layer's fraction of total parameters.

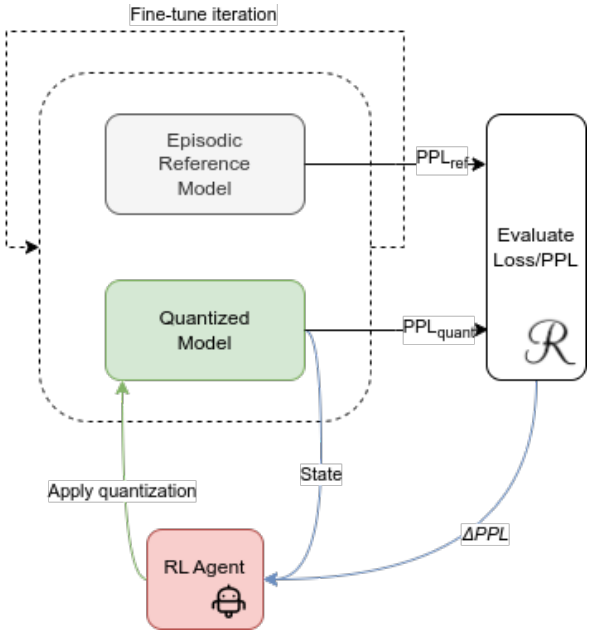


Figure 1. RL-based dynamic quantization loop. The agent selects a quantization type for the current layer, we fine-tune the quantized and reference models, then compute performance signals and memory usage to form a reward.

At the end of all N layers, the environment signals the episode is complete and the PPO agent updates its policy using the collected trajectory. This cycle repeats until the agent converges on an effective per-layer quantization policy.

3.2. Reward Design

Our reward function for layer i, denoted r_i, is a weighted sum of four terms capturing performance, divergence, attention preservation, and memory savings:

(1) Performance (Perplexity) Reward. We compare the perplexities of the reference model (PPL_{ref}) and the quantized model ($\text{PPL}_{\text{quant}}$):

$$r_{\text{perf}} = \Big(\text{PPL}_{\text{ref}} - \text{PPL}_{\text{quant}}\Big) \times w_{\text{perf}}. \tag{1}$$

If quantization reduces perplexity below that of the reference, r_{perf} is positive; otherwise, it is typically negative.

(2) KL Divergence Penalty. To penalize large deviations in predictive distributions, we compute:

$$r_{\text{KL}} = -\,w_{\text{KL}} \times \text{KL}\big(p_{\text{quant}} \,\|\, p_{\text{ref}}\big), \tag{2}$$

where p_{quant} and p_{ref} are the output distributions (softmax of the logits) from the quantized and reference models, respectively. A higher KL divergence incurs a larger negative reward.

(3) Attention Entropy Preservation. Retaining rich attention patterns can be crucial for model quality. We quantify this by comparing the attention entropies of layer i in both models:

$$r_{\text{entropy}} = \big(E_{\text{quant}} - E_{\text{ref}}\big) \times w_{\text{entropy}}, \tag{3}$$

where E_{quant} and E_{ref} denote the mean attention entropy in the current layer for the quantized and reference models, respectively.

(4) Memory Savings. Finally, we reward bit savings relative to a 16-bit baseline. For each parameter in layer i, the agent saves $16 - \text{bits}(a_i)$ bits if action a_i is chosen. This is weighted by the fraction of total parameters in layer i, denoted $\text{layer\_size\_ratio}_i$:

$$r_{\text{mem}} = \Big(\frac{16 - \text{bits}(a_i)}{16}\Big) \times \text{layer\_size\_ratio}_i \times w_{\text{memory}}. \tag{4}$$

Here we define

$$\text{layer\_size\_ratio}_i = \frac{\text{numParams}(i)}{\text{numParams}(\text{model})},$$

so that layers with more parameters yield proportionally higher savings.

The final reward for layer i combines all terms:

$$r_i = r_{\text{perf}} + r_{\text{KL}} + r_{\text{entropy}} + r_{\text{mem}}. \tag{5}$$

By adjusting the weights $w_{\text{perf}}, w_{\text{KL}}, w_{\text{entropy}}$, and w_{memory}, practitioners can prioritize different trade-offs between model fidelity and compression.

3.3. Policy Learning via PPO

We employ **Proximal Policy Optimization (PPO)** (Schulman et al., 2017) to update a small MLP policy π_θ that maps states to discrete actions (quantization types). At each new *episode*, we:

1. *Reset* the environment: copy the reference model to reinitialize $\mathcal{M}_{\text{quant}}$, set layer index to 0.
2. For each layer $i = 0 \ldots N-1$:
   - Agent picks $a_i \sim \pi_\theta(\cdot|s_i)$.
   - We apply quantization type a_i to layer i, perform short fine-tuning, measure $(\text{PPL}_{\text{quant}}, \text{PPL}_{\text{ref}}, \text{KL}, E_{\text{quant}}, E_{\text{ref}})$, compute reward r_i.
   - Next state s_{i+1} updated with new stats, layer index, etc.
3. We collect (s_i, a_i, r_i) for all i, compute advantages (e.g., GAE), and run a few epochs of PPO updates on π_θ.

3.4. Algorithmic Pseudocode

Algorithm 1 DynaQuant (One PPO Iteration)

Require: Model $\mathcal{M}_{\text{ref}}$ (baseline), RL policy π_θ, reward weights $(w_{\text{perf}}, w_{\text{KL}}, w_{\text{entropy}}, w_{\text{memory}})$

$\mathcal{M}_{\text{quant}} \leftarrow$ clone of $\mathcal{M}_{\text{ref}}$
$s_0 \leftarrow \text{INITSTATE}()$; $i \leftarrow 0$; done $\leftarrow$ **False**
`rollout` $\leftarrow []$
while not done **do**
  $a_i \sim \pi_\theta(a_i|s_i)$
  $\text{QUANTIZELAYER}(\mathcal{M}_{\text{quant}}, i, a_i)$
  $\text{FINETUNE}(\mathcal{M}_{\text{ref}}, \text{dataBatch}, \text{epochs})$
  $\text{FINETUNE}(\mathcal{M}_{\text{quant}}, \text{dataBatch}, \text{epochs})$
  $(r_i, \text{info}_i) \leftarrow \text{COMPUTEREWARD}(\mathcal{M}_{\text{ref}}, \mathcal{M}_{\text{quant}}, i)$
  $s_{i+1} \leftarrow \text{NEXTSTATE}(\mathcal{M}_{\text{quant}}, i+1)$
  `rollout` $\leftarrow$ `rollout` $\cup \{(s_i, a_i, r_i)\}$
  $i \leftarrow i + 1$
  if $i \geq N$ **then**
    done $\leftarrow$ **True**
  end if
end while
$\text{COMPUTEADVANTAGES}(\texttt{rollout})$
$\text{UPDATEPOLICYPPO}(\pi_\theta, \texttt{rollout})$

Algorithm 1 shows a single training iteration (episode). We typically repeat many episodes, reinitializing $\mathcal{M}_{\text{quant}}$ and $\mathcal{M}_{\text{ref}}$ each time.

4. Experiments

4.1. Setup and Datasets

Reference Model. We begin with GPT-2 (12-layer) as our base model, fine-tuning it on CommonsenseQA using standard procedures (e.g., AdamW optimizer for several

epochs) to produce an FP16 *reference model*. This serves as the foundation for all subsequent quantization.

Quantization Approach. All experiments—including both baselines and our RL-driven policy—use the same in-house quantization utilities. These support 4-bit (nf4/fp4) and 8-bit (int8) formats, as well as FP16/FP32 copy operations. This ensures consistent layer-wise transforms for both training and evaluation, so that any performance difference stems purely from how bits are allocated per layer, rather than from mismatched quantization methods.

Tasks. We evaluate on two downstream benchmarks:

- **BoolQ**: binary (yes/no) reading comprehension,
- **PIQA**: a multiple-choice physical reasoning dataset.

Both are assessed on publicly available validation sets, where we measure:

1. *Validation perplexity*, computed as the exponentiated mean cross-entropy over either the full or chunked sequences,
2. *Multiple-choice accuracy*, where each choice is scored via negative cross-entropy and the highest-likelihood answer is selected,
3. *Peak GPU memory usage* (MB), obtained by monitoring allocated memory before and after a forward pass,
4. *Inference throughput* (tokens/s), measured by timing multiple forward passes over synthetic token batches.

4.2. Baselines and Evaluation Methodology

Uniform Precision Baselines. **Uniform FP16** directly uses the fine-tuned reference model. For **Uniform NF4**, the same reference model is converted to 4-bit across every linear layer in one uniform pass. In both cases, the underlying weights, hyperparameters, and training data remain identical, differing only in their final precision.

RL-Based Mixed Precision. Our proposed **DynaQuant** uses a reinforcement learning policy to assign different bit-widths on a per-layer basis. After policy training, the layer-wise quantization scheme is finalized and applied to the reference model. The rest of the architecture and training data remain unchanged, ensuring a fair comparison with uniform baselines.

Evaluation Procedure. We evaluate all models—FP16, NF4, and RL-based mixed precision—through the same pipeline:

1. *Load* the final checkpoint (reference or quantized) into GPT-2, applying our quantization utilities where needed.
2. *Compute* validation perplexity on the relevant dataset via token-level cross-entropy.
3. *Measure* multiple-choice accuracy by scoring each candidate answer; the correct one is identified by the lowest average cross-entropy.
4. *Record* inference throughput and peak memory usage with repeated forward passes on synthetic batches.

By standardizing the quantization routines, data preprocessing, and evaluation scripts across all settings, we ensure that any observed differences in performance or memory usage reflect genuine trade-offs arising from the chosen precision formats.

4.3. Loss Metrics

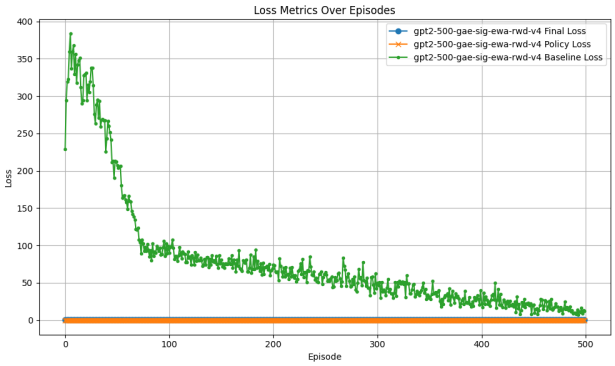


Figure 2. The final validation loss and the policy loss vs. episode. The quantized model's validation loss remains close to that of the reference model, while the RL policy loss stabilizes near zero, indicating convergence.

Figure 2 plots the quantized model's validation loss alongside the RL policy loss over the course of training.

- **Validation Loss (Green Curve).** This represents the *quantized* model's average cross-entropy on a held-out set after each full episode (i.e., once all layers have been quantized). Despite progressive quantization, the validation loss gradually decreases and remains close to that of our FP16 reference, indicating that the agent preserves the model's accuracy even with aggressive per-layer quantization.

- **RL Policy Loss (Orange Curve).** This curve measures the PPO objective that updates the quantization policy. Early in training, it fluctuates considerably as the policy explores different bit-width assignments. Over time, it stabilizes near zero, suggesting that the policy's updates become relatively minor, having found a reasonably good strategy.
- **Baseline Loss (Blue Curve).** For reference, we also show the baseline model's loss. As expected, it plateaus once the baseline finishes its own fine-tuning, providing a performance anchor.

Overall, these trends confirm that our RL approach does *not* severely degrade model quality, while the policy itself converges.

4.4. Total Reward Curve

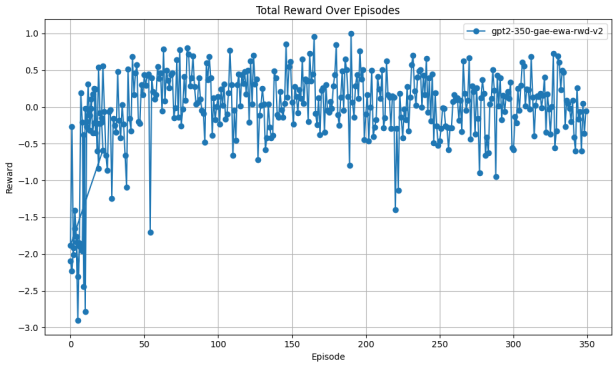


Figure 3. Example of total reward vs. episode. The agent steadily improves as it refines its per-layer quantization decisions.

Figure 3 illustrates how the *total reward* evolves with each training episode. Recall that our reward encompasses perplexity differences (versus the reference model), KL divergence, attention entropy, and memory savings.

- **Initial Negative Values.** Early episodes yield negative rewards, as random or naive quantization decisions often deteriorate performance substantially.
- **Steady Improvement.** Within a few dozen episodes, the agent discovers beneficial bit assignments that yield moderate memory savings with minimal perplexity increase, driving the reward into positive territory.
- **Late Stabilization.** Beyond 150–200 episodes, most runs hover around slightly above zero reward, reflecting a stable trade-off between memory gains and model fidelity.

This steady rise confirms that the RL agent effectively learns how to balance precision requirements with compression targets.

4.5. Layer-Wise Quantization Distribution

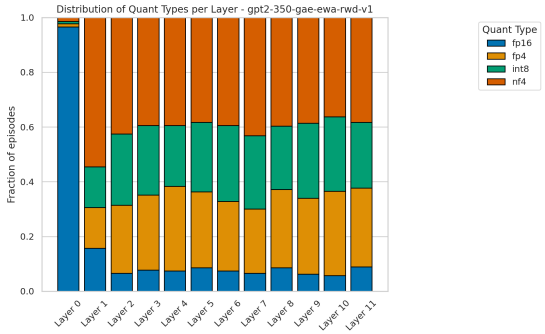


Figure 4. Quantization type distribution across layers

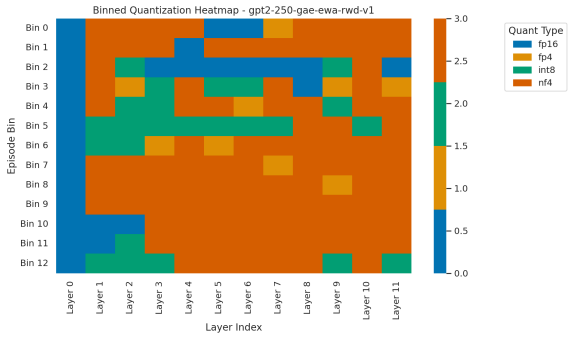


Figure 5. Heatmap showing quantization type usage by layer across binned episodes. The RL policy typically picks lower bits for many layers, using higher precision on sensitive blocks.

We next examine how often each layer is assigned a particular bit-width once the policy converges. Figures 4 and 5 visualize the final distribution of quantization formats across 12 Transformer layers:

- **Distribution Bar Chart (Figure 4).** Each column corresponds to a layer, and each color segment in the column shows the fraction of episodes for which that layer was assigned `fp16`, `fp4`, `int8`, or `nf4`. We observe that some layers predominantly end up in 4-bit or 8-bit formats, while others remain in `fp16`. This *layer-specific* pattern suggests certain layers are more sensitive to precision reduction.
- **Binned Heatmap (Figure 5).** Episodes are grouped into bins on the vertical axis, and layers are shown horizontally. The color indicates the chosen quantization type. Early in training, many layers fluctuate. As training progresses (moving down the vertical axis), the agent "locks in" stable choices, with orange (`nf4`) and green (`int8`) dominating most layers, while a few sensitive blocks remain in `fp16`.

Both plots confirm that the RL policy *does not* rely on a uniformly lower bit-width. Instead, it actively tailors the format per layer, highlighting the benefit of dynamic quantization.

Interpretation. Altogether, these results indicate that different transformer layers *vary* in their robustness to low-bit quantization. The agent learns to compress most layers (often with 4-bit or 8-bit) while retaining higher precision where needed. As a result, overall memory usage is significantly reduced, with only a marginal hit to validation loss compared to a purely FP16 baseline.

4.6. Results on BoolQ

Table 1. **BoolQ** (Validation). The first two rows are uniform baselines. The remaining rows are learned DynaQuant *mixed* schemes.

Method	PPL	Acc(%)	Mem(MB)
FP16 (baseline)	920.56	51.93	422.69
NF4 (uniform)	873.65	52.02	300.95
DynaQ Mix A	777.13	52.08	464.29
DynaQ Mix B	773.78	**53.85**	383.15

From Table 1, uniform NF4 beats FP16 in perplexity with notably lower memory usage. Our RL approach (*Mix B*) lowers perplexity further *and* boosts accuracy significantly, though at a slight memory overhead (383 MB vs. NF4's 300 MB).

4.7. Results on PIQA

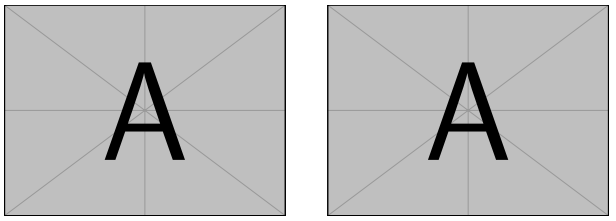
Table 2. **PIQA** (Validation). The first two rows are uniform baselines. The remaining are RL-based mixed.

Method	PPL	Acc(%)	Mem(MB)
FP16	2531.21	60.72	422.69
NF4	2265.88	60.77	300.95
DynaQ Mix C	2143.76	**61.53**	464.29
DynaQ Mix D	2099.65	61.26	383.15

Table 2 shows a similar trend. *Mix C* yields the highest accuracy (61.53%) but also the highest memory usage among the listed approaches. *Mix D* maintains a perplexity of 2099.65 and slightly lower accuracy, at a memory cost well below FP16.

4.8. Evaluation Analysis

The evaluation results (Figure 6) reveal an intriguing pattern. Our RL agent consistently chose higher precision (FP16) for the beginning and ending layers, while aggressively compressing middle layers to NF4/INT8. This matches with the observation of multilingual LMs (Wendler et al., 2024) that middle layers appear to handle abstract concept processing, which seems to require less numerical precision. In contrast, the initial and final layers appear to demand higher precision for accurate computation.



(a) Example quantization configuration for Model A (b) Example quantization configuration for Model B

Figure 6. Final quantization configurations showing the bit-width assignments across layers: high precision (FP16) for input/output layers with compressed intermediate layers (NF4/INT8) for Models A and B.

5. Discussion

Our experiments confirm that *per-layer dynamic quantization* can surpass uniform quantization in perplexity or accuracy:

- **Uniform NF4** is already a strong baseline, typically offering better perplexity than FP16 in these tasks, plus ∼30% memory savings.
- **DynaQuant improves** further, mixing `fp16` or `int8` for certain layers while using 4-bit for others. This yields additional perplexity gains and sometimes a tangible accuracy boost.
- **Speed trade-off**: Mixed precision can reduce throughput by requiring different kernel calls for different layers.

5.1. Limitations

- **Compute Overhead**: Each RL episode fine-tunes *all* layers, so total training cost is non-trivial.
- **Scalability**: We tested GPT-2. Larger LLMs (e.g. 1.5B–13B) might require more careful scheduling or partial-layer grouping to remain efficient.
- **Reward Calibration**: Weighting memory vs. perplexity vs. KL is subjective; tuning these hyperparameters is essential.

6. Conclusion

We have presented **DynaQuant**, an RL-based, layer-by-layer quantization approach that adaptively decides which bit-width format to apply per Transformer layer. Our

multi-term reward function—using perplexity difference, KL penalty, attention entropy preservation, and memory savings—guides the policy to compress the majority of layers aggressively while preserving or even boosting accuracy. On BoolQ and PIQA, DynaQuant's *mixed-precision* solutions outperform uniform quantization in perplexity/accuracy trade-offs, with memory footprints in-between purely 4-bit or purely 16-bit options.

Future Directions. Ongoing and future extensions include:

- **Scaling to bigger LLMs**, e.g. 1.5B–7B parameters, analyzing the trade-off between policy complexity and training overhead.
- **Reward Tuning** for different tasks (e.g. generative chat, summarization).
- **Hardware-level optimizations**: investigating throughput on specialized GPU kernels or accelerators for mixed-precision inference.
- **Integration with quantization-aware fine-tuning frameworks**: combining DynaQuant's layer decisions with advanced data augmentation or knowledge distillation.

References

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale, 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. QLoRA: Efficient Finetuning of Quantized LLMs, 2023.

Dong, Z., Kelly, B., Fomenko, M., Lin, T., Zhang, Z., Wellman, T. J., and Yao, Z. HAWQ: Hessian aware quantization of neural networks with mixed-precision. *arXiv preprint arXiv:1911.03852*, 2019.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers, 2022.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. AMC: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Malinovskii, V., Mazur, D., Ilin, I., Kuznedelev, D., Burlachenko, K., Yi, K., Alistarh, D., and Richtarik, P. Pv-tuning: Beyond straight-through estimation for extreme llm compression, 2024.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., and Han, S. Apq: Joint search for network architecture, pruning and quantization policy, 2020.

Wendler, C., Veselovsky, V., Monea, G., and West, R. Do llams work in english? on the latent language of multilingual transformers, 2024.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.