DynaQuant: RL-Based Dynamic Quantization for Large Language Models

Oleg Roshka [1] **Ilia Badanin** [2 3]

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 (**??**).

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 (**??**). 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. While our primary experiments use GPT-2 as a testbed, the code also supports other LLMs such as Qwen and Phi-2, and extends readily to additional architectures.

2. Related Work

LLM Quantization. Numerous works compress large models with low-precision formats: int8 (**?**), 4-bit normal float (nf4) (**?**), and others (Malinovskii et al., 2024). Typically, a *uniform* scheme is used. Our approach differs, 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 [3] Invited Collaborator. 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. Copyright 2018 by the author(s).

Mixed-Precision and NAS. Prior research in Neural Architecture Search (NAS) explores per-layer bit allocations (?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 (?) or prune channels (?). 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.

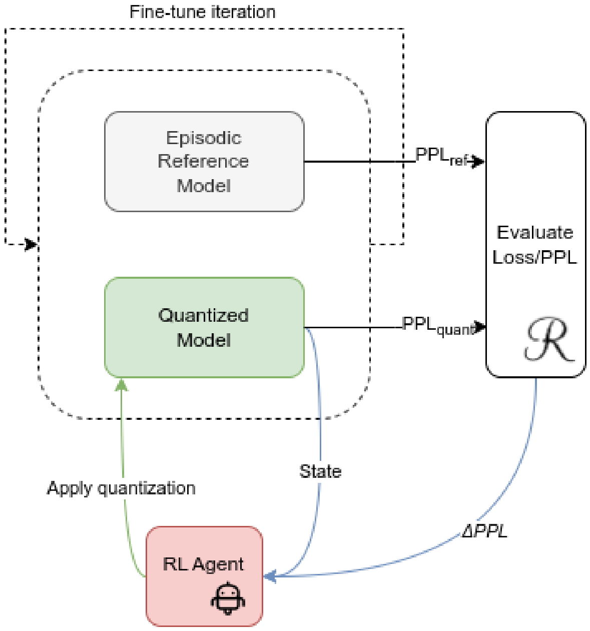

Figure 1. RL-based dynamic quantization loop. The agent selects a quantization type for the current layer, we fine-tune the quantized and reference models, then compute performance signals and memory usage to form a reward.

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, we also incorporate memory usage in the state (by including the quantization bits of the current and previous layers), and the *reward* further reflects how many bits are saved by quantizing the current layer, weighted by that layer's fraction of total parameters.

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 ($\mathrm{PPL}_{\text{ref}}$) and the quantized model ($\mathrm{PPL}_{\text{quant}}$):

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

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:

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

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 $\text{layer\_size\_ratio}_i$:

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

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 $(\mathrm{PPL}_{\text{quant}}, \mathrm{PPL}_{\text{ref}}, \mathrm{KL}, E_{\text{quant}}, E_{\text{ref}})$, 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 $\mathcal{M}_{ref}$ (baseline), RL policy $\pi_\theta$, reward weights
    $(w_{perf}, w_{KL}, w_{entropy}, w_{memory})$
$\mathcal{M}_{quant} \leftarrow$ clone of $\mathcal{M}_{ref}$
$s_0 \leftarrow$ InitState(); $i \leftarrow 0$; done $\leftarrow$ False
rollout $\leftarrow$ []
while not done do
    $a_i \sim \pi_\theta(a_i|s_i)$
    QuantizeLayer($\mathcal{M}_{quant}, i, a_i$)
    FineTune($\mathcal{M}_{ref}$, dataBatch, epochs)
    FineTune($\mathcal{M}_{quant}$, dataBatch, epochs)
    $(r_i, \text{info}_i) \leftarrow$ ComputeReward($\mathcal{M}_{ref}, \mathcal{M}_{quant}, i$)
    $s_{i+1} \leftarrow$ NextState($\mathcal{M}_{quant}, i+1$)
    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
ComputeAdvantages(rollout)
UpdatePolicyPPO($\pi_\theta$, rollout)
```

Algorithm **??** 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. We load the final model checkpoint (either reference or quantized) into GPT-2 and apply our quantization utilities as needed. For evaluation, we calculate validation perplexity on the dataset using token-level cross-entropy. To assess multiple-choice accuracy, we score each candidate answer and select the one with the lowest average cross-entropy. Finally, we measure inference performance by recording throughput and peak memory usage across repeated forward passes using synthetic input 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

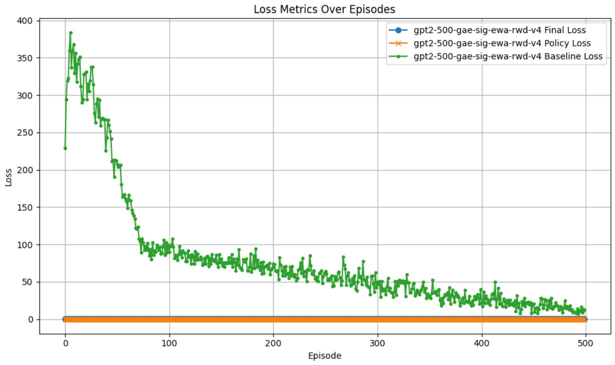

Figure 2. Three separate loss values over 500 training episodes (GPT2 small): **(blue)** the final validation loss ("final_loss") of the quantized model after each episode, **(orange)** the policy loss (the PPO objective), **(green)** the baseline loss (the value function's MSE).

Figure **??** tracks three distinct metrics logged each episode:

- **Final Validation Loss (Blue).** After each full episode (i.e., once all layers have been quantized), we compute the quantized model's validation loss (cross-entropy). In our runs, this loss remains close to zero or very small on the chosen scale, indicating the model's performance does not degrade severely despite aggressive mixed-precision.
- **Policy Loss (Orange).** This is the PPO objective we optimize when updating the quantization policy. It typically fluctuates early on as the agent explores different quantization schemes, then settles near zero, suggesting the policy has converged to stable decisions.
- **Baseline Loss (Green).** This is the MSE of the "base-

line network" used in PPO to estimate state values. We see it starts high, briefly spikes, and then gradually decreases, reflecting the baseline's learning to predict returns more accurately over time.

In summary, the orange policy loss and blue final loss both end near zero, while the green baseline loss steadily declines from large initial values. This pattern indicates that *(1)* the quantization policy stabilizes, *(2)* the value-function baseline converges, and *(3)* the final quantized model maintains strong performance (low validation loss).

4.4. Total Reward Curve

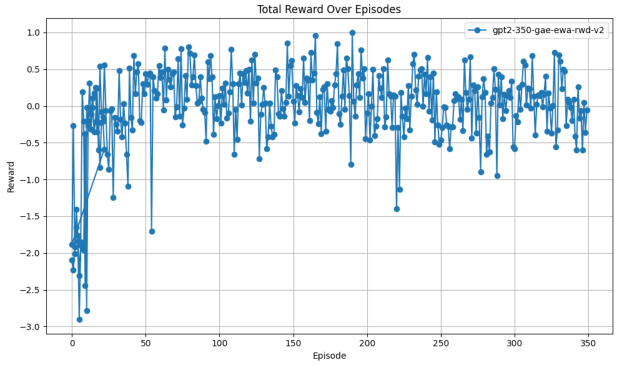

Figure 3. Example of total reward vs. episode (GPT2 small). The agent steadily improves as it refines its per-layer quantization decisions.

Figure **??** 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.

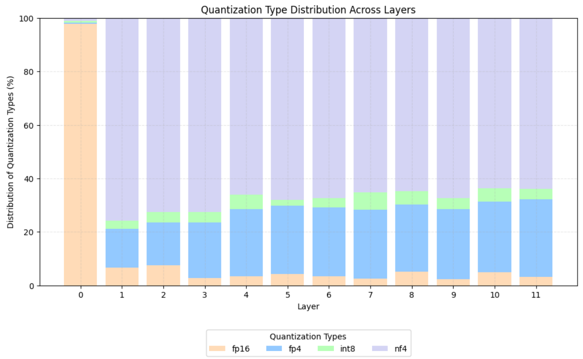

Figure 4. Quantization type distribution across layers (GPT2 small).

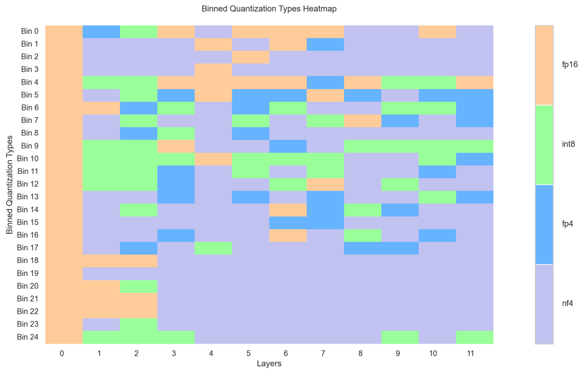

Figure 5. Heatmap showing quantization type usage by layer across binned episodes (GPT2 small). The RL policy typically picks lower bits for many layers, using higher precision on sensitive blocks.

4.5. Layer-Wise Quantization Distribution

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

- **Distribution Bar Chart (Figure ??).** 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 ??).** 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.

5. Results

Notation. Throughout the tables below, each RL-based quantization variant is labeled as "*Mix A*," "*Mix B*," etc., indicating a distinct learned configuration of layer-wise bit allocations.

5.1. Results on BoolQ (GPT-2 Small)

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

Table 1. BoolQ validation results for GPT-2 Small. The first two rows show uniform quantization baselines (FP16 and NF4). The remaining rows demonstrate our learned DynaQuant mixed-precision schemes that adaptively quantize different layers.

Observations for GPT-2 Small on BoolQ. From Table **??**, uniform NF4 beats FP16 in perplexity (873.65 vs. 920.56) with notably lower memory usage (300.95 MB vs. 422.69 MB). Our RL approach (e.g., DynaQ Mix B) lowers perplexity further *and* boosts accuracy significantly (53.85% vs. NF4's 52.02%), though at a slight memory overhead (383.15 MB vs. NF4's 300.95 MB).

5.2. Results on PIQA (GPT-2 Small)

Observations for GPT-2 Small on PIQA. Table **??** shows a similar trend. Uniform NF4 again outperforms FP16 in perplexity (2265.88 vs. 2531.21) with lower memory usage. *Mix C* yields the highest accuracy (61.53%) but also the highest memory usage among these methods (464.29 MB). *Mix D* maintains a perplexity of 2099.65 and slightly lower accuracy, at a memory cost well below FP16.

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. PIQA validation results for GPT-2 Small. The first two rows show uniform quantization baselines (FP16 and NF4). The remaining rows demonstrate our learned DynaQuant mixed-precision schemes.

5.3. Results on BoolQ (GPT-2 Medium)

Method	PPL↓	Acc(%)↑	Mem(MB)↓
FP16 (baseline)	1009.31	54.22	956.26
NF4 (uniform)	**953.16**	54.53	513.04
DynaQ Mix A	1094.60	**54.80**	694.23
DynaQ Mix B	1091.37	54.71	622.10
DynaQ Mix C	978.64	54.71	**532.71**

Table 3. BoolQ validation results for GPT-2 Medium. The first two rows show uniform quantization baselines (FP16 and NF4). The remaining rows demonstrate our learned DynaQuant mixed-precision schemes.

Observations for GPT-2 Medium on BoolQ. As in the smaller model, uniform NF4 beats FP16 in perplexity (953.16 vs. 1009.31) at about half the memory footprint (513 MB vs. 956 MB). Several mixes (e.g., Mix A) slightly boost accuracy to 54.80% but can raise memory usage. Notably, Mix C dips below the uniform NF4 perplexity (978.64 vs. 953.16) but remains comparable in accuracy.

5.4. Results on PIQA (GPT-2 Medium)

Method	PPL↓	Acc(%)↑	Mem(MB)↓
FP16 (baseline)	5791.41	**64.58**	956.26
NF4 (uniform)	4932.96	64.53	513.04
DynaQ Mix A	5399.04	64.42	694.23
DynaQ Mix B	5493.03	63.71	622.10
DynaQ Mix C	**4894.23**	64.36	**532.71**

Table 4. PIQA validation results for GPT-2 Medium. The first two rows show uniform quantization baselines (FP16 and NF4). The remaining rows demonstrate our learned DynaQuant mixed-precision schemes.

Refer to Appendix **??** for the specific mixed quantization schemas that were evaluated.

Observations for GPT-2 Medium on PIQA. Table **??** shows uniform NF4 significantly reduces perplexity from 5791.41 to 4932.96 (vs. FP16) while nearly matching FP16's accuracy (64.53% vs. 64.58%). Among the RL-based mixes, *Mix C* achieves the lowest perplexity (4894.23) at a moderate memory cost (532.71 MB), albeit with a slight dip in accuracy (64.36%). Overall, the same pattern emerges: selectively using higher precision for a subset of layers can yield attractive trade-offs compared to purely uniform quantization.

5.5. Evaluation Analysis

As shown in Tables **??** and **??** (GPT-2 Small) and Tables **??** and **??** (GPT-2 Medium), uniform NF4 is a strong baseline—often outperforming or matching FP16 in perplexity while reducing memory usage by up to 50%. With our RL-based *DynaQuant*, we see further perplexity gains or slight accuracy boosts by mixing bit-width formats. On the smaller GPT-2 model, *Mix B* or *Mix C* often yield the best accuracy, while *Mix D* provides a compromise in memory overhead. For the larger GPT-2 Medium, certain mixes (e.g., *Mix C*) maintain or improve perplexity with minimal memory overhead compared to NF4 alone. Our results in Appendix **??** illustrate that many final learned configurations keep the initial and final Transformer layers in higher precision (FP16), while aggressively compressing middle layers with 4-bit parameters.

Note on Memory Usage. Although one might expect a strictly lower overall memory footprint from mixed precision, we occasionally observe peak memory usage exceeding the purely FP16 baseline. This is due in part to overhead from kernel implementations that temporarily cast INT8 to FP32 in certain GPU libraries,[1] as well as extra tensors stored during fine-tuning. In practice, these overheads do not necessarily reflect parameter storage size but rather transitory usage during forward/backward passes.

6. Discussion

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

- **Uniform NF4** is already strong, typically offering better perplexity than FP16 on these tasks, plus ∼30–50% 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, so speed can degrade.

6.1. Limitations

- **Compute Overhead**: Each RL episode fine-tunes *all* layers, so total training cost is non-trivial.
- **Scalability**: We tested GPT-2 (124M) and GPT-2 Medium (345M). Larger LLMs (1.5B–13B) might require more careful scheduling or partial-layer grouping.
- **Reward Calibration**: Weighting memory vs. perplexity vs. KL is subjective; tuning these hyperparameters carefully is essential.

7. 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, both for GPT-2 Small and GPT-2 Medium, 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. This suggests that a fine-grained, per-layer approach to quantization can offer a more favorable balance than traditional, strictly uniform settings.

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.

8. Author Contributions

Oleg Roshka proposed the main idea and developed the methodology for this project. He also implemented the core RL code base for dynamic quantization. **Ilia Badanin** joined in the final stages, contributed essential experiment visualizations, and helped run large-scale GPT-2 experiments, which was instrumental in finalizing the project on time. Both authors collaborated on writing the manuscript and interpreting the results.

9. Code Availability

To foster reproducibility and further research, we have open-sourced our implementation of DynaQuant at the following repository:

`https://github.com/olegroshka/rl-dynamic-quant`

This repository contains the core RL code for per-layer dynamic quantization, training scripts, and instructions to replicate our experiments.

[1] For example, in some PyTorch versions or GPU driver stacks, INT8 GEMM kernels still allocate intermediate FP32 buffers, leading to short-lived (but increased) peak memory usage compared to uniform FP16.

References

Choi, J., Wang, Z., Venkataramani, S., Najafirad, P., Shenoy, V., and Keutzer, K. PACT: Parameterized Clipping Activation for Quantized Neural Networks, 2018.

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

Egiazarian, V., Panferov, A., Kuznedelev, D., Frantar, E., Babenko, A., and Alistarh, D. Extreme compression of large language models via additive quantization, 2024.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, 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.

Sun, Z., Saab, K., Tepper, M., Dettmers, T., Basu, S., Gonzalez, J. D. M., Shen, S., Rush, A. M., Duckworth, D., Lewis, M., et al. Qlora: Efficient finetuning of quantized llms, 2023.

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.

Zoph, B. and Le, Q. V. Neural Architecture Search with Reinforcement Learning, 2017.

A. Mixed-Precision Schemas for Each Experiment

Below we list all per-layer quantization assignments discovered by **DynaQuant** for each mix in the main text. Each bracketed list shows the formats from layer 0 up to layer $N - 1$.

- **GPT-2 Small on BoolQ:**
  - **Mix A:** `[fp16, int8, fp16, nf4, fp16, int8, fp16, int8, fp4, nf4, int8, fp4]`
  - **Mix B:** `[fp16, nf4, nf4, fp16, nf4, nf4, nf4, int8, int8, nf4, nf4, fp4]`
- **GPT-2 Small on PIQA:**
  - **Mix C:** `[fp16, int8, fp16, nf4, fp16, int8, fp16, int8, fp4, nf4, int8, fp4]`
  - **Mix D:** `[fp16, nf4, nf4, fp16, nf4, nf4, nf4, int8, int8, nf4, nf4, fp4]`
- **GPT-2 Medium on BoolQ:**
  - **Mix A:** `[fp4, nf4, nf4, int8, nf4, nf4, fp16, fp4, fp4, nf4, fp4, nf4, nf4, nf4, nf4, nf4, fp4, int8, nf4, nf4, fp4, nf4, fp4, int8]`
  - **Mix B:** `[fp4, fp4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, nf4, fp4, nf4, nf4, int8, nf4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, fp4, int8]`
  - **Mix C:** `[fp4, nf4, fp4, nf4, nf4, nf4, nf4, nf4, nf4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, nf4, nf4, nf4, fp16]`
- **GPT-2 Medium on PIQA:**
  - **Mix A:** `[fp4, nf4, nf4, int8, nf4, nf4, fp16, fp4, fp4, nf4, fp4, nf4, nf4, nf4, nf4, nf4, fp4, int8, nf4, nf4, fp4, nf4, fp4, int8]`
  - **Mix B:** `[fp4, fp4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, nf4, fp4, nf4, nf4, int8, nf4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, fp4, int8]`
  - **Mix C:** `[fp4, nf4, fp4, nf4, nf4, nf4, nf4, nf4, nf4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, fp4, nf4, nf4, nf4, nf4, nf4, nf4, fp16]`

B. Additional GPT-2 Medium Visualizations

In this appendix, we provide supplementary plots and analyses for the GPT-2 Medium experiments. These visualizations show layer-wise attention-entropy distributions, the learned quantization formats across episodes, and detailed reward components over the course of training.

B.1. Attention Entropy by Layer

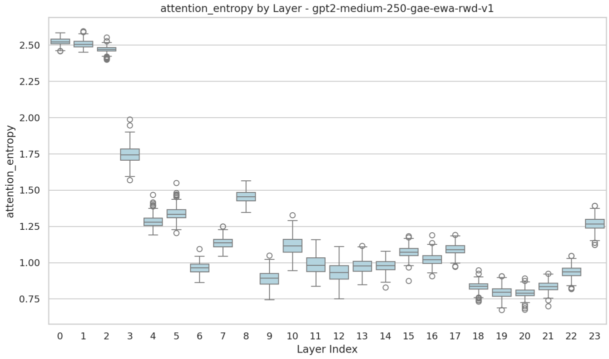

Figure 6. **Attention Entropy Across Layers (GPT-2 Medium).** Each box shows the distribution of average attention entropy values for one layer, collected over multiple episodes. Layers near the start (indices 0–2) exhibit higher entropy, possibly because they are attending to more general tokens, while deeper layers often have lower entropy, reflecting more specialized attention patterns.

B.2. Distribution of Quant Types per Layer

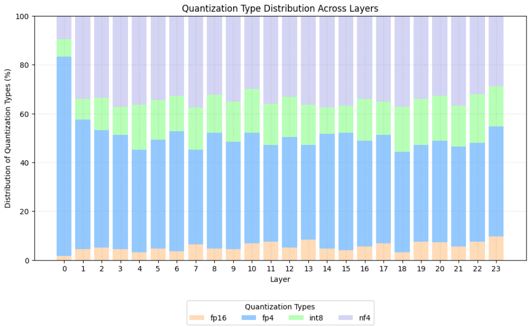

Figure 7. **Quantization Type Distribution.** For each layer index (horizontal axis), we show the fraction of training episodes (vertical) that chose a particular quant format (`fp16`, `fp4`, `int8`, `nf4`). We see that middle layers (e.g. layers 5–15) are more heavily assigned 4-bit or 8-bit formats, whereas the first and last few layers sometimes remain at `fp16`.

B.3. Binned Quantization Heatmap

B.4. Reward Components Over Episodes

As shown in these figures, the RL policy for GPT-2 Medium (in this experiment with aggressive optimisation for memory) typically pushes *more* layers toward low-bit formats (NF4 or FP4), but initially preserves higher precision in early and late layers where it

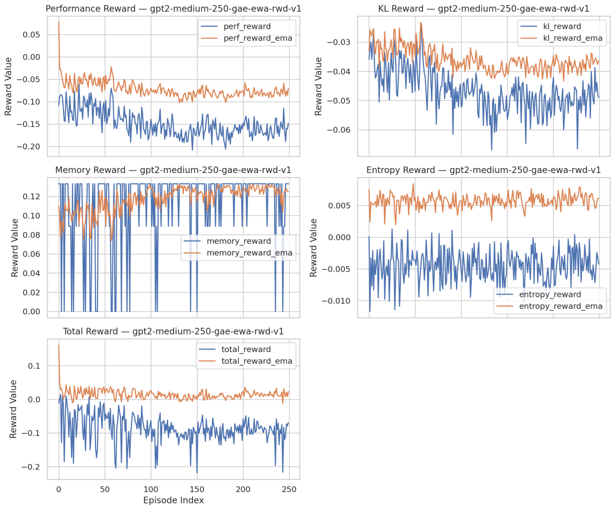

Figure 9. **Per-Episode Reward Components.** We log each reward component (performance, KL, memory, and entropy) as well as the total reward, in both raw (blue) and exponential moving average (orange) form. Observe that memory rewards tend to remain positive (due to bit savings), while performance and KL can fluctuate or even be negative if a newly chosen quant format momentarily degrades perplexity or increases divergence. Overall, the total reward stabilizes around a slightly positive mean, indicating a balanced trade-off across the four objectives.

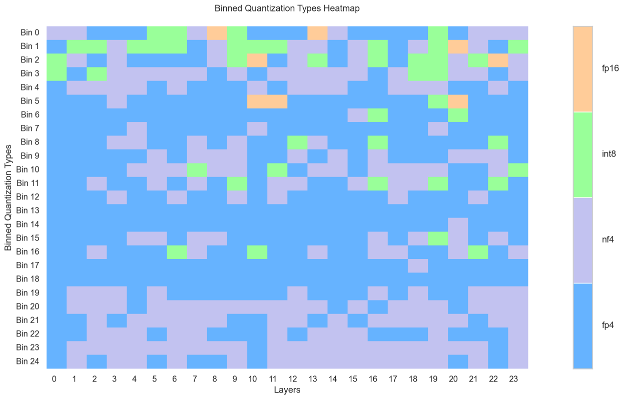

Figure 8. **Quantization Heatmap by Episode Bin.** We group episodes in bins (vertical axis), and layers are shown horizontally. The color indicates which quant type was chosen for that layer within each bin. Early bins (top) show more exploration, while later bins (bottom) converge to a predominantly 4-bit strategy (orange), with occasional `int8` or `fp16` for sensitive layers.

observes larger sensitivity in perplexity or attention changes. The reward breakdown confirms that while performance and KL terms sometimes turn negative after quantizing a "fragile" layer, the net reward remains moderately positive thanks to memory savings and stable attention patterns.