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 deploying them on commodity devices or serving 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. Concretely, 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 a variety of NLP tasks. However, the rapid growth in parameter counts (hundreds of millions to billions of parameters) poses nontrivial 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. Concretely:

1. We treat each layer as a step in an RL episode.
2. The RL agent picks one action from $\{\texttt{nf4}, \texttt{fp4}, \texttt{int8}, \texttt{fp16}\}$ for that layer.
3. After quantizing, we do 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 show that **DynaQuant** discovers policies that *improve* perplexity and often accuracy relative to uniform 4-bit or 16-bit baselines, with memory usage that sits between those extremes. Our code is built on GPT-2 as a testbed and extends easily to other LLMs.

2. Related Work

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

Mixed-Precision and NAS. Prior research in Neural Architecture Search (NAS) explores per-layer bit allocations (**??**) but often focuses on smaller networks. We adapt these ideas for large Transformer-based LLMs, adding layer-by-layer *short fine-tuning* and multi-objective RL signals (e.g. perplexity, KL, memory, attention).

[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.badadnin@epfl.ch>.

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

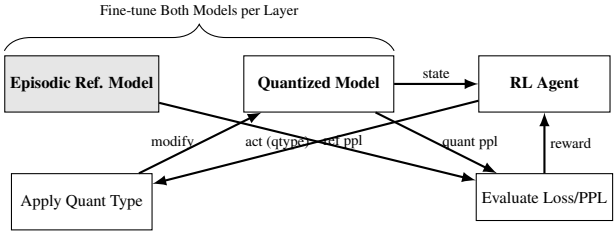

Figure 1. RL-based dynamic quantization loop. The agent picks a quant type for the current layer, we fine-tune the quantized and reference models, then compute performance signals and memory usage to form a reward.

RL for Model Optimization. RL has been employed to search network architectures (**?**) or prune channels (**?**), but is less explored for dynamic LLM quantization. Our environment extends prior RL-based compression setups to handle short iterative fine-tuning, and an episodic reference model that closely mirrors the quantized model's training steps.

3. Methodology

3.1. RL Environment: Dynamic Layer Quantization

We assume a reference model $\mathcal{M}_{\text{ref}}$ (e.g. GPT-2 in FP16) and a copy $\mathcal{M}_{\text{quant}}$ that we will quantize layer-by-layer. Each **episode** processes all N layers in sequence, with each layer representing one RL step:

1. *State* s_i: includes layer statistics (weight mean/std, gradient norms, attention entropy), the current layer index i, the model's current perplexity, memory usage so far, and the previous layer's quantization choice.
2. *Action* a_i: one of $\{\texttt{nf4}, \texttt{fp4}, \texttt{int8}, \texttt{fp16}\}$.
3. *Transition*: We quantize layer i in $\mathcal{M}_{\text{quant}}$, then briefly fine-tune both $\mathcal{M}_{\text{ref}}$ and $\mathcal{M}_{\text{quant}}$ on the same minibatch.
4. *Reward* r_i: measures performance vs. memory trade-offs (details below).

3.2. Reward Design

From the **final** version of our code, the *reward* r_i for layer i is a weighted sum of four terms:

(1) Performance (Perplexity) Reward. We measure the perplexity difference between reference (PPL_{ref}) and quantized ($\text{PPL}_{\text{quant}}$) models:

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

If the quantized model has lower perplexity (compared to reference), r_{perf} is positive. Often, $\text{PPL}_{\text{quant}}$ is higher, giving a negative contribution.

(2) KL Divergence Penalty. We compute $\text{KL}(p_{\text{quant}} \,\|\, p_{\text{ref}})$ on a validation batch. A large KL means the quantized distribution diverges from reference:

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

(3) Attention Entropy Preservation. We compare the attention entropies in layer i: E_{quant} vs. E_{ref}. Retaining diverse attention patterns is often crucial:

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

(4) Memory Savings. We measure how many bits are saved in layer i relative to an FP16 baseline. If a_i is $\texttt{nf4}$, we save $16 - 4 = 12$ bits per parameter, scaled by the fraction of total params in layer i:

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

Hence:

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

3.3. Policy Learning via PPO

We employ **Proximal Policy Optimization (PPO)** (**?**) 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, do 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 **??** shows a single training iteration (episode). We typically repeat many episodes, reinitializing $\mathcal{M}_{\text{quant}}$ and $\mathcal{M}_{\text{ref}}$ each time.

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

4. Experiments

4.1. Setup and Datasets

Model. We use GPT-2 (12-layer) as a proof-of-concept. The *reference model* is GPT-2 in FP16, fully fine-tuned on CommonsenseQA. For RL episodes, we then quantize layer-by-layer on the same or similar data.

Tasks. We evaluate on **BoolQ** (yes/no QA) and **PIQA** (multiple-choice physical reasoning). We measure:

- *Validation perplexity*
- *Multiple-choice accuracy*
- *Peak GPU memory usage* (MB)
- *Inference throughput* (tokens/s)

Baselines. We compare **Uniform FP16** and **Uniform NF4** with learned RL-based *mixed* assignments discovered after PPO training.

4.2. Quantization Distributions and Reward Curves

Layer-Wise Heatmap. Over many episodes, the RL policy converges to mostly 4-bit or 8-bit on many layers, with a few layers assigned to `fp16`. Figure **??** (schematic) shows how different layers end up being assigned specific bit-widths more frequently.

4.3. Results on BoolQ

From Table **??**, uniform NF4 beats FP16 in perplexity with notably lower memory usage. Our RL approach (*Mix B*) lowers perplexity further *and* boosts accuracy significantly,

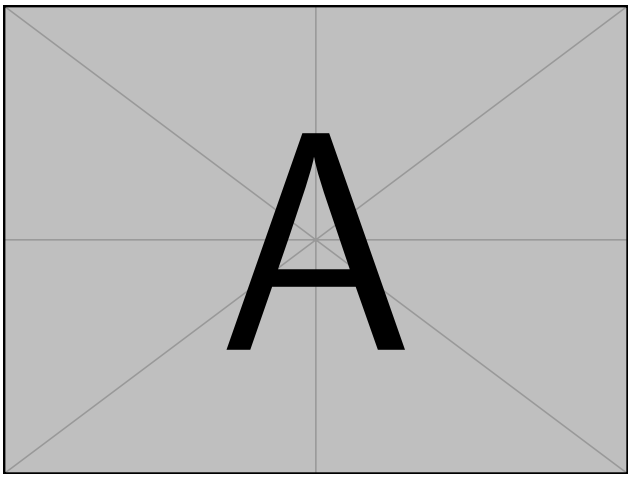

Figure 2. Example of total reward vs. episode. The agent steadily improves as it refines its per-layer quant decisions.

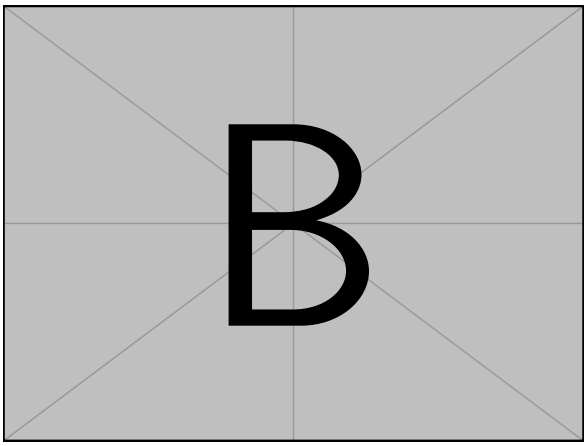

Figure 3. Schematic heatmap showing quant type usage by layer across episodes. The RL policy typically picks lower bits for many layers, using higher precision on sensitive blocks.

though at a slight memory overhead (383 MB vs. NF4's 300 MB). Inference throughput also decreases somewhat, presumably due to mixing kernels.

4.4. Results on PIQA

Table **??** 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.

5. Discussion

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

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

Method	PPL	Acc(%)	Mem(MB)	Speed
FP16 (baseline)	920.56	51.93	422.69	48.8k
NF4 (uniform)	873.65	52.02	300.95	42.8k
DynaQ Mix A	777.13	52.08	464.29	39.3k
DynaQ Mix B	773.78	**53.85**	383.15	40.4k

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

Method	PPL	Acc(%)	Mem(MB)	Speed
FP16	2531.21	60.72	422.69	48.2k
NF4	2265.88	60.77	300.95	42.0k
DynaQ Mix C	2143.76	**61.53**	464.29	39.4k
DynaQ Mix D	2099.65	61.26	383.15	39.5k

- **Uniform NF4** is already a strong baseline, typically offering better perplexity than FP16 in these tasks, plus $\sim$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.