# Report on Efficient LLMs via Switchable and Dynamic Quantization

Harvey Li[a,*]

[a]*Georgia Institute of Technology, Atlanta, USA*

## 1. Implementation Details

### *1.1. GPT2 for SQuAD*

Before quantization, I added a question answering head to the GPT2 model, following the implementation in Huggingface's transformers library [1,2]. The question answering head consists of one linear layer that maps the hidden states of the GPT2 model to the start and end positions of the answer span in the input text.

The modified model is then fine-tuned on the SQuAD v1 dataset [3] for 1 epoch, using the AdamW optimizer with a learning rate of $2 \times 10^{-5}$ and a batch size of 4. The fine-tuning process achieves an Exact Match score of 64.0 and an F1 score of 73.5 on the validation set.

All following quantization and training steps are based on this fine-tuned model. If you are having trouble reproducing the results, please use the provided pre-trained model at https://www.dropbox.com/scl/fi/7uerce730v8lcsprpe7dt/gpt2-squad-finetuned.zip.

### *1.2. LLM-QAT*

Every aspect of the original LLM-QAT paper is implemented [4], except the Data-free Distillation, which was replaced by the Cascade Distillation in InstantNet according to the instructions [5].

LLM-QAT explores different quantization methods, including asymmetric, symmetric, MinMax and outlier clipping methods, and concluded with the choice of MinMax Symmetric quantization. Therefore, for sim-

*Corresponding author. E-mail address: harvey-l@gatech.edu

*September 15, 2025*

plicity, I only implemented the MinMax Symmetric quantization method, formulated as follows:

$$\mathbf{X}_q = \alpha \left\lfloor \frac{\mathbf{X}}{\alpha} \right\rceil \qquad \alpha = \frac{\max(|\mathbf{X}|)}{2^{b-1} - 1} \tag{1}$$

Where $b$ is the bit-width, $X$ is the tensor to be quantized, and $X_q$ is the quantized tensor.

I also tried to play with different gradient estimators, but the Straight-Through Estimator (STE) remains the best choice.

### *1.3. Cascade Distillation Training (CDT)*

Cascade Distillation Training (CDT) is implemented following the instructions in InstantNet [5]. CDT allows all bit-width configurations to be trained simultaneously, by balancing the hard loss and distillation loss from higher bit-width configurations. The original loss function is formulated as follows:

$$L = L_{\text{hard}}(Q_i(x), \hat{y}) + \beta \sum_{j=i+1}^{N-1} L_{\text{distill}}\big(Q_i(x), Q_j(x)\big) \tag{2}$$

Note that the teacher model $Q_j$ is not updated when training the student model $Q_i$ . The implementation drops the gradient of the teacher model to avoid unnecessary computation.

The loss function choice in the original paper is Cross Entropy Loss for the hard loss and Mean Squared Error for the distillation loss [5]. However, I replaced the distillation loss with Kullback-Leibler Divergence (KLDivLoss) to better align with the original knowledge distillation paper [6]. The hard loss remains as Cross Entropy Loss.

$\beta = 1$ , following the distill_weight in the original implementation at https://github.com/GATECH-EIC/InstantNet/blob/main/config_train.py

In my implementation, I maintained one instance of full precision model, and dynamically switched the quantization configurations during

Table 1: Configurations of Different Quantization Policies

| **Policy** | **weight** | **activation** | **bias** | **LoRA weights** |
|---|---|---|---|---|
| Outlier Adaptive | Adaptive (4-8 bits) | 8 bits | Adaptive (4-8 bits) | 32 bits (if active) |
| Aggressive LoRA | Adaptive (4-6 bits) | 8 bits | Adaptive (4-6 bits) | 32 bits |
| Depth Adaptive | Adaptive (5-8 bits) | Adaptive (6-8 bits) | Adaptive (5-8 bits) | |
| Conservative LoRA | 6-8 bits | 6-8 bits | 32 bits | 6-8 bits |
| Uniform LoRA 8 | 8 bits | 8 bits | 32 bits | 8 bits |
| Uniform LoRA 6 | 6 bits | 6 bits | 32 bits | 6 bits |
| Uniform LoRA 4 | 4 bits | 4 bits | 32 bits | 4 bits |

training. This is more memory efficient than maintaining multiple instances of the model, and allows for parameter sharing among different configurations.

### *1.4. Cyclic Precision Training (CPT)*

## 2. Quantization Policies

7 different quantization configurations are explored. The configurations are summarized in Table 1. The following sections describe the heuristics of each dynamic configuration.

### *2.1. Outlier Adaptive*

This policy uses kurtosis to measure the outlier level of each layer, and assigns higher bit-widths to layers with higher outlier levels. The intuition is that outliers in weights can significantly harm the precision of quantization by increasing the scaling factor $\alpha$ , thus using higher bit-widths for layers with more outliers can help mitigate this issue. [4]

Layers are first assigned with a base bit-width according to their position in the network:

- Question Answering Head: Not quantized (32 bits)
- Attention Input Projections (QKV fused): 6 bits weights, 8 bits activations
- Attention Output Projections: 5 bits weights, 8 bits activations
- MLP Input Projections: 4 bits weights, 8 bits activations
- MLP Output Projections: 5 bits weights, 8 bits activations

Biases are quantized to the same bit-width as their corresponding weights. Then, layers are categorized into three levels of outlier-ness based on their kurtosis values:

1. Low outlier level (kurtosis $< 6$): No change
2. Medium outlier level ($6 \leq$ kurtosis $< 10$): Increase weight bit-width by 1
3. High outlier level (kurtosis $\geq 10$): Increase weight bit-width by 2

32-bit LoRA is applied as refinement for sensitive layers when their bit-width is less than 6 bits, and for attention layers if they have excessive outliers (kurtosis $\geq 15$).

### *2.2. Depth Adaptive*

In depth adaptive policy, early ($\leq 20\%$ ) and late ($\geq 80\%$ ) layers have +1 bit-width for both weights and activations, while middle layers have the base bit-width. The intuition is that early layers are responsible for extracting low-level features, while late layers are responsible for high-level features, both of which are sensitive for model performance. Middle layers, on the other hand, can afford to be quantized more aggressively.

Question Answering Head is not quantized (32 bits). Biases are quantized to the same bit-width as their corresponding weights. LoRA is not applied in this policy.

### *2.3. Aggressive LoRA*

Similar to Outlier Adaptive, but with lower base bit-widths and lower LoRA ranks.

Table 2: Performance of Different Quantization Policies with CDT

| QAT Policy | Size (MB) | Exact | F1 |
|---|---|---|---|
| *Full Precision (after QAT)* | *324.32* | *0.5943* | *0.7002* |
| Outlier Adaptive | 55.24 | 0.5685 | 0.6754 |
| Aggressive LoRA | 50.53 | 0.5448 | 0.6535 |
| Depth Adaptive | 65.04 | 0.5316 | 0.6434 |
| Conservative LoRA | 70.35 | 0.5078 | 0.6262 |
| Uniform LoRA (8-bit) | 81.32 | **0.5809** | **0.6867** |
| Uniform LoRA (6-bit) | 61.07 | 0.3486 | 0.4719 |
| Uniform LoRA (4-bit) | **40.82** | 0.0009 | 0.0562 |

### *2.4. Conservative LoRA*

Conservative LoRA policy manually assigns higher bit-widths (7 or 8 bits) to sensitive layers, and 6 bits to the rest. All layers except the head have quantized LoRA with mild ranks.

## 3. Results with CDT

This section answers the questions for Step 1-4.

- [Step 4] What is the task accuracy achieved after applying various quantization bit-width configurations to the SQuAD dataset?

The detailed results are summarized in Table 2. The size is calculated based on the bit-width configurations and the shape of parameters in each layer. The Exact Match and F1 scores are evaluated on the SQuAD v1 validation set.

- [Step 4] How did you determine the optimal quantization bit-width configurations? Have you gleaned any insights from your observations that could guide future work to further enhance performance?

In fact, the optimal quantization bit-width configuration is determined by the trade-off between model size and performance. To visualize this trade-off, Table 2 is replotted in a Pareto Front graph, as shown in Figure 1.

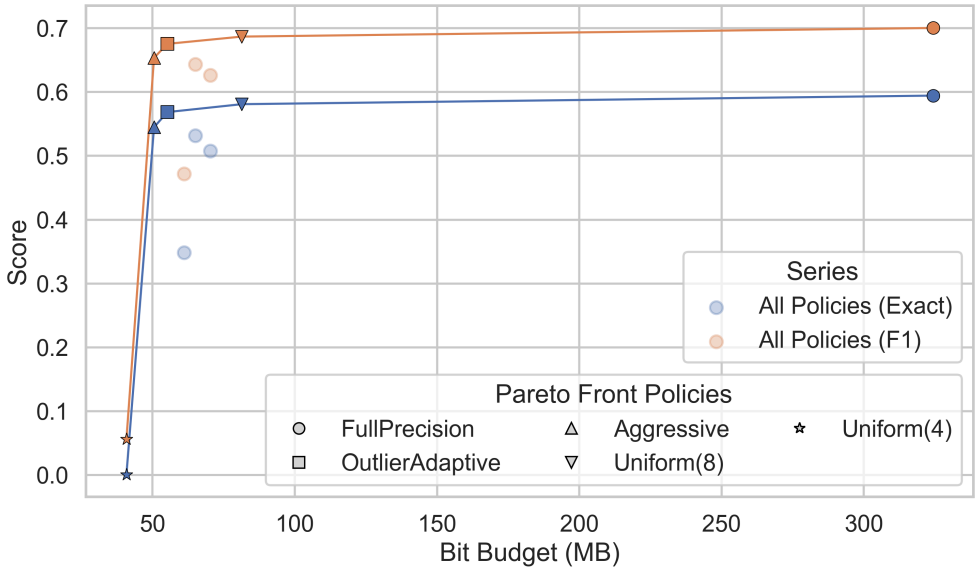


Figure 1: Pareto Fronts for Policies with CDT

One can see that three policies stand out in the Pareto Front: Outlier Adaptive, Uniform LoRA (8-bit), and Aggressive LoRA. These three policies offer the best trade-off between size and performance, and are therefore considered optimal.

An important observation is that Outlier Adaptive policy, which uses the least amount of bits, outperforms the believed best Conservative LoRA policy by a large margin (0.57 v.s. 0.51 exact scores). This suggests that more bits do not necessarily lead to better performance, and that intelligent allocation of bits can yield better results.

Table 3: Comparison between CDT and Independently Fine-tuned Quantization Policies

| **QAT Policy** | **Exact** | **F1** | **Exact w/CDT** | **F1 w/CDT** |
|---|---|---|---|---|
| Outlier Adaptive | **0.5752** | **0.6800** | 0.5685 | 0.6754 |
| Aggressive LoRA | 0.5448 | 0.6529 | **0.5448** | **0.6535** |
| Depth Adaptive | 0.5223 | 0.6355 | **0.5316** | **0.6434** |
| Conservative LoRA | 0.4422 | 0.5520 | **0.5078** | **0.6262** |

I also experimented with training the same quantization policies independently without Cascade Distillation Training (CDT) and using only the hard loss. The training hyperparameters remain the same, and the results are summarized in Table 3. Note that this is slightly different from the purpose of InstantNet, which uses one set of parameters to support multiple quantization configurations. Here, I trained separate models for each configuration, which, based on intuition, should yield better performance.

One can see that CDT consistently outperforms all individually trained policies, with the most significant improvement seen in the Conservative LoRA policy (0.51 v.s. 0.44 exact scores). This suggests that CDT is an effective training strategy for both **switchable and non-switchable quantization**.

- [Step 4] A motivation behind switchable quantization is to support diverse layer-wise quantization configurations simultaneously, accommodating different resource allocation needs. Could you suggest additional training objectives that could more effectively facilitate the mechanism for switching quantization bit-widths?

  In addition to the hard loss and distillation loss used in CDT, one could also consider adding a regularization term that encourages the model to learn similar representations across different bit-width configurations. This could be achieved by minimizing the distance between the hidden states of the model under different quantization configurations, such as using Mean Squared Error or Cosine Similarity as the distance metric.

  I would also suggest introducing Parametric Noise Injection (PNI) [7] during training, which adds learnable noise to the weights and activations. Quantization noise resembles random noise [8], and PNI can help the model learn to be robust against such noise.

## 4. Results with CPT

I fine-tuned the same quantization policies using Cyclic Precision Training (CPT) [9] for 1000 steps, with a learning rate of $2 \times 10^{-5}$ and a batch size of 4. Each CPT cycle consists of 100 steps, with the bit-widths increasing from the minimum to the maximum in one cycle.

Table 4: Comparison of Results with CPT and without CPT

| QAT Policy | Exact | F1 | Exact w/CPT | F1 w/CPT |
|---|---|---|---|---|
| Outlier Adaptive | 0.5229 | 0.6264 | **0.5567** | **0.6644** |
| Aggressive LoRA | 0.4529 | 0.5573 | **0.5268** | **0.6362** |
| Depth Adaptive | 0.3712 | 0.4787 | **0.4754** | **0.5921** |
| Conservative LoRA | 0.3977 | 0.5135 | **0.4706** | **0.5890** |
| Uniform LoRA (8-bit) | **0.6009** | **0.6960** | 0.5726 | 0.6787 |
| Uniform LoRA (6-bit) | 0.1307 | 0.2074 | **0.2683** | **0.3775** |
| Uniform LoRA (4-bit) | **0.0015** | 0.0555 | 0.0012 | **0.0586** |

Table 5: Comparison of Results with CPT and without CPT. Uniform policies without LoRA.

| QAT Policy | Exact | F1 | Exact w/CPT | F1 w/CPT |
|---|---|---|---|---|
| Uniform (8-bit) | **0.6009** | **0.6960** | 0.5816 | 0.6852 |
| Uniform (7-bit) | | | 0.5469 | 0.6534 |
| Uniform (6-bit) | 0.1307 | 0.2074 | **0.3408** | **0.4599** |
| Uniform (5-bit) | | | 0.0121 | 0.0671 |
| Uniform (4-bit) | **0.0015** | **0.0555** | 0.0009 | 0.0498 |

As a comparison, I also fine-tuned the same policies without CPT for 1000 steps, with the same hyperparameters. Note that for fair comparison, the without CPT models are forced to share the parameters, as appropriate for the CPT setting. The results are summarized in Table 4.

The original CPT paper only explored uniform quantizations without LoRA and different per-layer configurations. To respect the original setting, I also experimented with uniform quantization without LoRA, with results summarized in Table 5.

- [Step 5] Does this phenomenon align with the observations in CPT (ICLR'21)? If not, what could be the potential reasons?

  The short answer is **yes**. CPT consistently improves the performance of all quantization policies, with the most significant improve-

ment seen in the Aggressive LoRA policy (0.53 v.s. 0.45 exact scores). This aligns with the observations in the original CPT paper [9], which also reported consistent improvements in CNNs.

However, in terms of switchable quantization, the improvement is less significant compared to the InstantNet paper [5]. But this could result from a confounding factor. Moth my implementation and the official InstantNet implementation involve accumulating gradients over multiple bit-width configurations before updating the parameters in a single step. This is an optimization technique to prevent duplicate GPU VRAM usage. Nevertheless, this also means that the effective batch size is larger than that in the original CPT paper, which could result in unfair comparison with fixed 1000 steps.

## 5. Impact on Robustness

### *5.1. HotFlip Attack*

HotFlip attack is a white-box gradient-based adversarial attack targeting Text Classification models [10]. The attack uses the gradient of the loss function with respect to the input text to identify the most influential characters, and flips them to maximize the loss. I found this method to be also effective for question answering tasks, with a similar implementation.

The HotFlip process begins by performing a forward and backward pass on the original input to compute the gradient of the loss $L$ with respect to the input word embeddings. This yields $g_i = \nabla_{e_i} L$ for each embedding $e_i$ at position $i$ in the input sequence.

This gradient indicates the direction in the embedding space that would most increase the loss. We approximate the change in loss resulting from replacing the original token's embedding with a candidate embedding $e'_v$ from the vocabulary's embedding matrix. This is estimated using a first-order Taylor approximation:

$$Z(v) = g_i^T (e'_v - e_i) \tag{3}$$

This score $Z(v)$ is computed for all tokens in the vocabulary, and the token with the highest score is selected as the replacement for the token at position $i$ . This process is repeated for every non-special token that is

Table 6: Absolute performance drops of CDT models.

| **Policy** | **Exact** | **F1** | **Exact Drop** | **F1 Drop** |
|---|---|---|---|---|
| Full Precision | 0.4270 | 0.5015 | 0.2129 | 0.2334 |
| Outlier Adaptive | 0.4060 | 0.4896 | 0.1625 | 0.1857 |
| Aggresive LoRA | 0.4040 | 0.4828 | 0.1408 | 0.1707 |
| Depth Adaptive | 0.3740 | 0.4588 | 0.1576 | 0.1846 |
| Conservative LoRA | 0.3820 | 0.4600 | 0.1258 | 0.1662 |
| Uniform LoRA (8-bit) | 0.4110 | 0.4847 | 0.1789 | 0.2013 |
| Uniform LoRA (6-bit) | 0.2930 | 0.3685 | 0.0556 | 0.1034 |
| Uniform LoRA (4-bit) | 0.0010 | 0.0413 | −0.0001 | 0.0149 |

not a part of the question or answer span. The flip that causes the highest increase in loss is selected.

A randomly selected subset of SQuAD v1 validation set with size 1000 was applied HotFlip attack, and both quantized and non-quantized models were evaluated on the perturbed dataset. The results are summarized in Table 6.

The evaluated quantized models are trained with CDT, as described in the previous sections. The Exact Drop and F1 Drop columns indicate the absolute performance drop after the HotFlip attack.

*5.2. Results*

- [Step 6] Does this phenomenon align with the observations in Double-Win Quant (ICML'21)? If not, what could be the potential reasons?

  Yes, in terms of accuracy drop under attack. All quantized models exhibit smaller drops in both Exact Match and F1 scores compared to the full precision model, as shown in Table 6 and Figure 2. This aligns with the observations in Double-Win Quant (ICML'21) [8], which also reported improved robustness against adversarial attacks for quantized models compared to their full precision counterparts.

  I believe this originates from the fact that quantization introduces noise to the model parameters and activations. Noise injection has

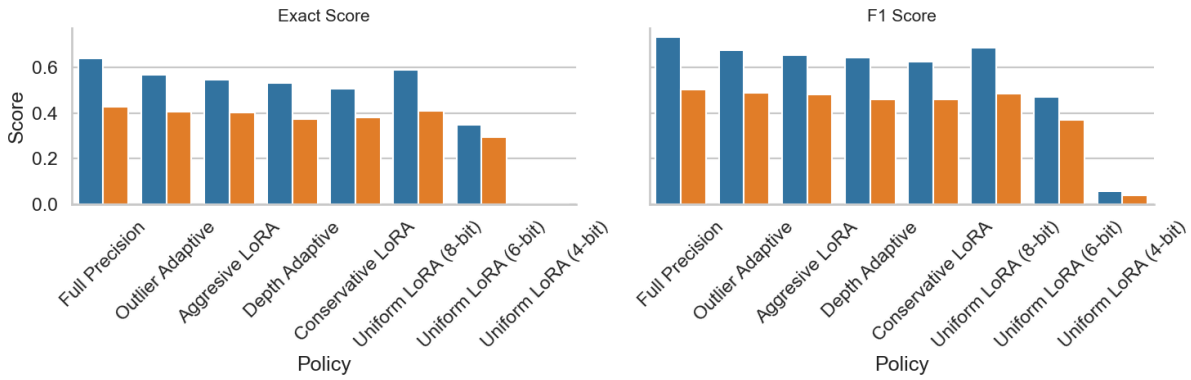


Figure 2: Absolute performances before and after HotFlip Attack

been shown to improve model robustness against adversarial attacks [7]. Therefore, quantized models are inherently more robust to small perturbations in the input, such as those introduced by adversarial attacks.

## 6. Conclusion and Future Work

In conclusion, this report presents my implementation and evaluation of switchable and dynamic quantization techniques for large language models, specifically GPT2 fine-tuned on the SQuAD v1 dataset. The implementation includes LLM-QAT for quantization-aware training, Cascade Distillation Training (CDT) for simultaneous training of multiple quantization configurations, and Cyclic Precision Training (CPT) for comparison. Finally, the impact of quantization on model robustness against HotFlip adversarial attacks was investigated.

A few relevant directions for future improvements on switchable and dynamic quantization are suggested below:

1. Borrowing the idea of Parametric Noise Injection (PNI) [7], one could introduce learnable noise akin to quantization noise to the weights and activations during training. More over, PNI allowed the "noise level" to be learned, which could shed light on the optimal bit-width configuration for each layer.

2. Adding a regularization term that encourages the model to learn similar representations across different bit-width configurations. This

could be achieved by minimizing the distance between the hidden states of the model under different quantization configurations. This would help models to prepare for the unknown inference-time quantization configuration.

3. Explore ideas from recent quantization methods, such as the GPU-Adaptive Non-Uniform Quantization (GANQ) [11]. They could also be applied with Quantization Aware Training and switchable quantization.

## References


[1] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T.L. Scao, S. Gugger, M. Drame, Q. Lhoest, A.M. Rush, HuggingFace's Transformers: State-of-the-art Natural Language Processing, (2020). https://arxiv.org/abs/1910.03771.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language Models are Unsupervised Multitask Learners, (2019).

[3] P. Rajpurkar, J. Zhang, K. Lopyrev, P. Liang, SQuAD: 100,000+ Questions for Machine Comprehension of Text, in: J. Su, K. Duh, X. Carreras (Eds.), Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Austin, Texas, 2016: pp. 2383–2392. https://doi.org/10.18653/v1/D16-1264.

[4] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, V. Chandra, LLM-QAT: Data-Free Quantization Aware Training for Large Language Models, (2023). https://arxiv.org/abs/2305.17888.

[5] Y. Fu, Z. Yu, Y. Zhang, Y. Jiang, C. Li, Y. Liang, M. Jiang, Z. Wang, Y.C. Lin, InstantNet: Automated Generation and Deployment of Instantaneously Switchable-Precision Networks, (2025). https://arxiv.org/abs/2104.10853.

[6] G. Hinton, O. Vinyals, J. Dean, Distilling the Knowledge in a Neural Network, (2015). https://arxiv.org/abs/1503.02531.

[7] A.S. Rakin, Z. He, D. Fan, Parametric Noise Injection: Trainable Randomness to Improve Deep Neural Network Robustness against Adversarial Attack, (2018). https://arxiv.org/abs/1811.09310.

[8] Y. Fu, Q. Yu, M. Li, V. Chandra, Y. Lin, Double-Win Quant: Aggressively Winning Robustness of Quantized Deep Neural Networks via Random Precision Training and Inference, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, PMLR, 2021: pp. 3492–3504. https://proceedings.mlr.press/v139/fu21c.html.

[9] Y. Fu, H. Guo, M. Li, X. Yang, Y. Ding, V. Chandra, Y.C. Lin, CPT: Efficient Deep Neural Network Training via Cyclic Precision, (2025). https://arxiv.org/abs/2101.09868.

[10] J. Ebrahimi, A. Rao, D. Lowd, D. Dou, HotFlip: White-Box Adversarial Examples for Text Classification, (2018). https://arxiv.org/abs/1712.06751.

[11] P. Zhao, X. Yuan, GANQ: GPU-Adaptive Non-Uniform Quantization for Large Language Models, (2025). https://arxiv.org/abs/2501.12956.