

A Formulation of Layer-wise Maximum Precision Selection

To select the precision of each layer under a memory budget constraint, we follow the approach inspired by [8], [9], and [13], using a second-order Taylor expansion of the loss function to formulate an integer programming problem. Equation 4 presents the second-order Taylor expansion of the model’s loss function, where g and H denote the gradient and the Hessian for W , respectively.

$$\mathcal{L}(W_Q) \approx \mathcal{L}(W) - g^T(W - W_Q) + \frac{1}{2}(W - W_Q)^T H(W - W_Q) \quad (4)$$

Assuming the model has converged, the first-order term can be ignored, allowing the second-order term to approximate the loss difference caused by quantization. Furthermore, by neglecting cross-weight interactions, only the diagonal elements of the Hessian (H) remain relevant. As a result, the loss difference can be expressed as shown in Equation 5.

$$\mathcal{L}(W_Q) - \mathcal{L}(W) \approx \frac{1}{2}(W - W_Q)^T \text{diag}(H)(W - W_Q) = \frac{1}{2} \sum H_{k,k}((W - W_Q)_k)^2 \quad (5)$$

When the weights of each layer W_i are quantized to a b -bit weight $W_{i,b}$, the objective function to minimize can be expressed as follows. Here N denotes the number of layers, B is the set of available precisions, M_i is the memory usage of the i th layer, and b_{target} is the target precision.

$$\begin{aligned} & \underset{c_{i,b}}{\text{argmin}} \sum_i^N \sum_b^B c_{i,b} \cdot \sum_k H_{k,k}((W - W_Q)_k)^2 \\ & \text{s.t. } c_{i,b} \in \{0, 1\}, \sum_b^B c_{i,b} = 1, \sum_i^N \sum_b^B c_{i,b} \cdot b \cdot M_i \leq b_{\text{target}} \cdot \sum_i^N M_i \end{aligned} \quad (6)$$

To keep computational costs reasonable, we approximate H using the Fisher information matrix F , whose diagonal elements can be calculated by accumulating the squared gradients over the calibration set. After solving Equation 6 via integer programming, $c_{i,b}$ indicates whether b -bit weight was selected for layer i . We then set the maximum precision for each layer accordingly.

B Evaluation Details

B.1 Setup and Methodology

Perplexity Datasets. For both the WikiText2 and C4 datasets, samples are concatenated and divided into chunks of size 2048. The perplexity evaluation follows the same procedure as Any-Precision LLM [1]. We interpret the perplexity evaluation as a teacher-forced decoding process, computing the loss at each decoding step to determine the overall perplexity.

Downstream Tasks. GSM8K is evaluated in a 5-shot setting using the LM-evaluation-harness framework [38], with `exact_match` as the evaluation metric. MBPP is evaluated in a 3-shot setting using instruction-embedded prompts. The reported metric for MBPP is `pass@1`. The user instruction and assistant response prefix strings are adopted from [39]. An example MBPP prompt is shown below, with [BEGIN] and [DONE] tokens used to delimit the model’s response.

```
1 You are an expert Python programmer, and here is your task:
2 Write a function to find the shared elements from the given two lists.
3
4 Your code should pass these tests:
5 assert set(similar_elements((3, 4, 5, 6), (5, 7, 4, 10))) == set((4, 5))
6 assert set(similar_elements((1, 2, 3, 4), (5, 4, 3, 7))) == set((3, 4))
7 assert set(similar_elements((11, 12, 14, 13), (17, 15, 14, 13))) == set((13,
14))
8 [BEGIN]
```

Figure 7: Example MPPP Prompt

BBH is evaluated in a 3-shot setting using the LM-evaluation-harness framework, with Chain-of-Thought (CoT) reasoning enabled. The reported metric is `exact_match`. MATH is evaluated using the MATH-500 [40] variant, also with 3-shot prompting. The evaluation metric for MATH is `math_verify`.

Fine-tuning. A subset of the C4 train dataset (512 tokens \times 1000 samples) is used for threshold fine-tuning, with the number of epochs and learning rate each set to 5 and 0.01, respectively. Fine-tuning is performed using the AdamW optimizer. The hyperparameter α is set to 1 for all target precisions, except when the target precision is 3.25, where α is set to 10 to better align with the target precision. The fine-tuning framework is built on top of the Any-Precision LLM [1] codebase, enabling the use of quantized weights for efficient fine-tuning.

Latency Measurements. To ensure full GPU saturation during the decoding phase, each model is compiled using PyTorch’s `torch.compile` function. This compiles a CUDA Graph of the model, eliminating kernel launch overhead and improving GPU utilization efficiency. Latency is measured by generating 100 tokens across 10 repetitions, with the average effective bitwidth set equal to the target precision. This bitwidth is fixed by setting each layer’s threshold to either infinity or negative infinity.

B.2 Static Precision Assignment Baseline.

LLM-MQ formulates the precision assignment problem as follows:

$$\begin{aligned} & \underset{c_{i,b}}{\operatorname{argmin}} \sum_i^N \sum_b^B c_{i,b} \cdot |g_i^T (W_i - W_{i,b})| \\ & \text{s.t. } c_{i,b} \in \{0, 1\}, \sum_b^B c_{i,b} = 1, \sum_i^N \sum_b^B c_{i,b} \cdot b \cdot M_i \leq b_{\text{target}} \cdot \sum_i^N M_i \end{aligned} \quad (7)$$

where N denotes the number of layers, B is the set of available precisions, W_i is the original weight of layer i , $W_{i,b}$ is the b -bit quantized weight of layer i , g_i is the gradient of W_i , M_i is the memory usage of layer i , and b_{target} is the target precision. However, since the original constraint only enforces an upper bound on memory usage, the scheme often fails to find a feasible allocation for higher precision targets (e.g., 4.5 bits) due to the similarity of high precision weights. To address this, we introduce a lower bound into the constraint formulation:

$$\sum_i^N \sum_b^B c_{i,b} \cdot b \cdot M_i \geq b_{\text{targetmin}} \cdot \sum_i^N M_i \quad (8)$$

$b_{\text{targetmin}}$ is gradually increased from zero to b_{target} in increments of 0.01, until the average memory usage falls within 0.005 bits of the target precision.

HAWQ-V2 uses the second-order information to calculate the sensitivity of a layer:

$$\Omega_i = \overline{\operatorname{Tr}(H)} \|W - W_Q\|_2^2 \quad (9)$$

Since computing the exact Hessian matrix is impractical for an LLM, we approximate it using the Fisher information matrix, following the approach in [13]. Once the sensitivities are computed, we formulate an integer programming problem to identify the optimal adaptation set, as done in LLM-MQ.

B.3 Fine-tuning Results

Fine-tuning Cost. Both Llama-3-8B and Phi-3-Medium are fine-tuned on a single RTX 3090 GPU with 24GB of VRAM. Llama-3-8B completes fine-tuning in approximately one hour, utilizing 14GB of VRAM, while Phi-3-Medium takes about two hours with 21GB of VRAM usage. On an A100 80GB GPU, Llama-3-8B requires around 30 minutes, and Phi-3-Medium takes approximately one hour to complete fine-tuning.

Distribution of Average Precision. We present examples of fine-tuned average precisions for target precisions of 3.5 and 4.0 bits under a 5-bit memory budget. Figures 8 and 9 show the results for Llama-3-8B, while Figure 10 and 11 correspond to Phi-3-Medium. These results demonstrate that the average precisions are distributed across the full range of available values, rather than being concentrated at the lower or upper extremes.

Relative Error Estimation Method Selection. Table 7 presents the number of layers assigned to each error estimation method for every h and l pair. For Llama-3-8B, nearly half of the layers use linear regression for error estimation, which introduces negligible overhead and significantly reduces GPU memory and latency overhead compared to relying solely on random projections based on the Johnson-Lindenstrauss (JL) Lemma (see Section 6.2). In the case of Phi-3-Medium, approximately two-thirds of the layers are estimated using linear regression.

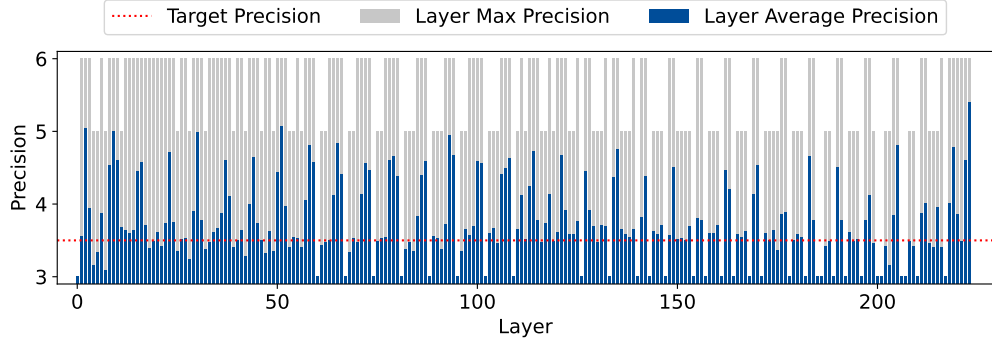


Figure 8: Average precisions for Llama-3-8B with target precision of 3.5 bits

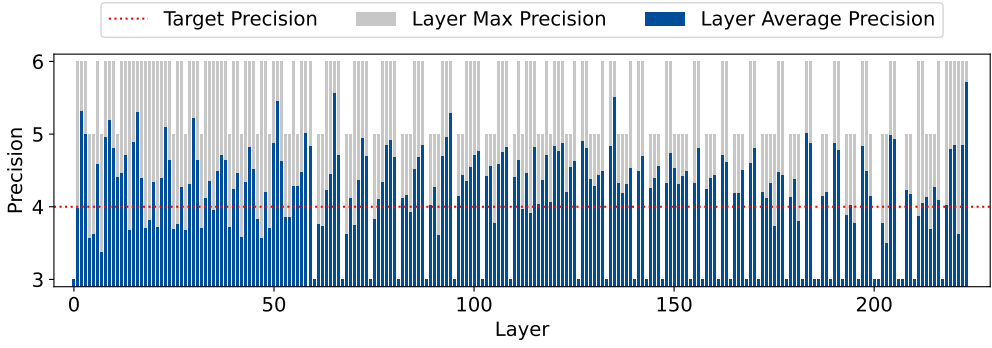


Figure 9: Average precisions for Llama-3-8B with target precision of 4.0 bits

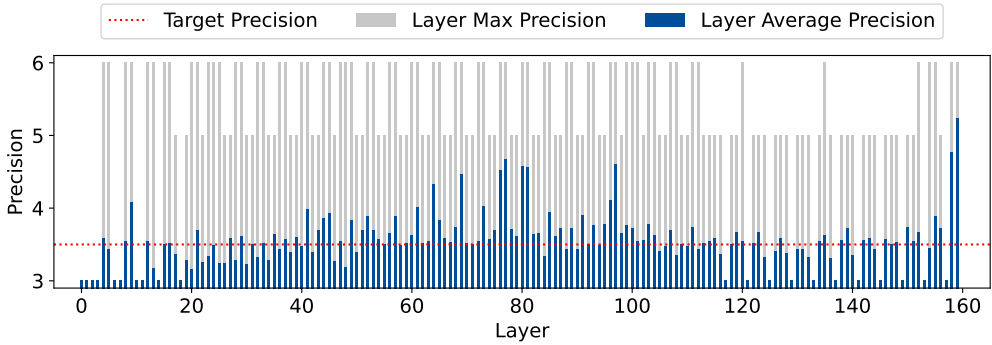


Figure 10: Average precisions for Phi-3-Medium with target precision of 3.5 bits

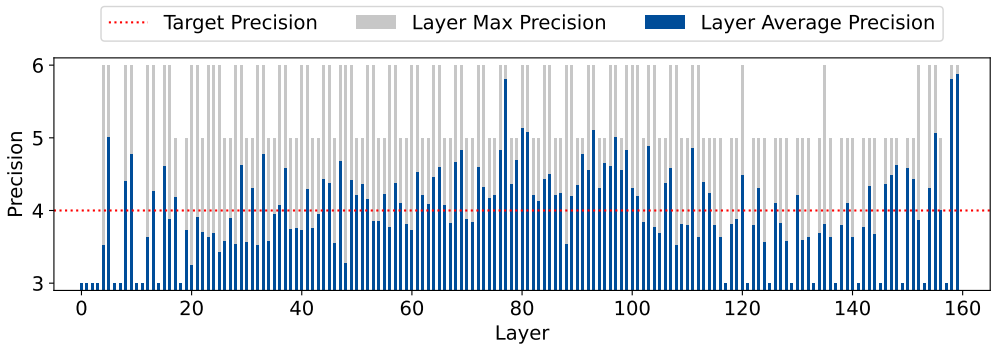


Figure 11: Average precisions for Phi-3-Medium with target precision of 4.0 bits

Table 7: Number of layers for each relative error estimation method

l, h	3, 4		4, 5		5, 6	
Model	Linear	JL	Linear	JL	Linear	JL
Llama-3-8B	107	117	113	111	111	113
Phi-3-Medium	115	45	110	50	108	52

GPU Memory Overhead The combined GPU memory overhead is measured by summing the additional capacity required for relative error estimators that support runtime model adaptation across multiple target precisions (3.25, 3.5, 3.75, 4.0, 4.25, 4.5, 4.75). The results are presented in Table 8. The overhead is computed relative to a baseline model constrained by a 5-bit memory capacity budget, consistent with the main evaluation.

Table 8: GPU memory overhead of DP-LLM

Model	Llama-3-8B	Phi-3-Medium
Quantized Model Capacity	6.7GB	9.4GB
Average Estimator Capacity Per Target Precision	0.08GB	0.07GB
Total Estimator Capacity	0.16GB	0.12GB
Overhead	2.4%	1.3%

C Additional Evaluations

C.1 Evaluations on Different Memory Budgets

We evaluate DP-LLM under varying memory budgets, specifically at 6-bit and 4-bit configurations. Since the baseline Any-Precision model supports precisions ranging from 3 to 6 bits, all layers are permitted to utilize any available precision under the 6-bit memory budget. Table 9 presents the evaluation results for the 6-bit budget, while Table 10 shows the results for the 4-bit budget.

Table 9: Evaluation results under 6-bit memory budget

Dataset		WikiText2 (\downarrow)					C4 (\downarrow)				
		Target Precision (Bits)					Target Precision (Bits)				
Models	Method	3.5	4.0	4.5	5.0	5.5	3.5	4.0	4.5	5.0	5.5
L3-8B	LLM-MQ	7.27	7.07	6.63	6.65	6.32	11.48	11.21	10.40	10.50	9.81
	HAWQ-V2	7.21	6.83	6.56	6.32	6.20	11.41	10.76	10.28	9.81	9.60
	DP-LLM	7.02	6.59	6.39	6.25	6.19	11.06	10.28	9.91	9.67	9.60
P3-M	LLM-MQ	5.14	4.85	4.77	4.53	4.39	9.47	9.26	9.22	9.07	8.97
	HAWQ-V2	5.06	4.83	4.57	4.40	4.36	9.38	9.21	9.05	8.95	8.93
	DP-LLM	4.82	4.60	4.50	4.42	4.36	9.17	9.04	8.97	8.93	8.92

C.2 Evaluations on Models with Different Parameter Scales

We further evaluate DP-LLM on the Qwen2.5-3B and Qwen2.5-32B models to examine how model size impacts its performance. Table 11 presents the evaluation results under a 5-bit memory budget. The results show that DP-LLM consistently achieves superior performance across different model sizes.

C.3 Ablation study for selecting h and l

DP-LLM selects $h = \lceil p \rceil$ and $l = \lfloor p \rfloor$ as the high and low precisions, respectively, to construct an average precision p . While multiple combinations of h and l are possible, we empirically find that choosing values close to the target precision yields the best model performance. Table 12 shows the fine-tuning results under various h and l combinations when targeting a 4.5-bit precision. To isolate the effects of these choices, all layers are constrained to use the same h and l pair during fine-tuning. The results clearly show that selecting neighboring precisions to the target value is most effective for constructing an average precision.

Table 10: Evaluation results under 4-bit memory budget

Dataset		WikiText2 (\downarrow)			C4 (\downarrow)		
		Target Precision (Bits)			Target Precision (Bits)		
Models	Method	3.25	3.50	3.75	3.25	3.50	3.75
L3-8B	LLM-MQ	7.62	7.39	7.21	12.01	11.66	11.42
	HAWQ-V2	7.47	7.21	7.01	11.77	11.41	11.09
	DP-LLM	7.38	7.08	6.91	11.64	11.19	10.92
P3-M	LLM-MQ	5.26	5.18	5.06	9.57	9.52	9.45
	HAWQ-V2	5.18	5.06	4.94	9.45	9.38	9.28
	DP-LLM	5.06	4.95	4.88	9.34	9.26	9.21

Table 11: Evaluation results for models with different parameter scales

Dataset		WikiText2 (\downarrow)							C4 (\downarrow)						
		Target Precision (Bits)							Target Precision (Bits)						
Models	Method	3.25	3.50	3.75	4.00	4.25	4.50	4.75	3.25	3.50	3.75	4.00	4.25	4.50	4.75
Q2.5-3B	LLM-MQ	9.83	9.58	9.34	9.29	9.07	9.07	8.89	16.07	15.72	15.40	15.36	15.06	15.06	14.83
	HAWQ-V2	9.34	9.00	8.77	8.59	8.42	8.30	8.21	15.50	15.01	14.67	14.40	14.20	14.02	13.92
	DP-LLM	9.13	8.83	8.58	8.43	8.33	8.27	8.21	15.17	14.73	14.42	14.25	14.05	13.95	13.88
Q2.5-32B	LLM-MQ	5.76	5.66	5.56	5.51	5.48	5.43	5.30	10.92	10.83	10.76	10.73	10.70	10.67	10.59
	HAWQ-V2	5.68	5.51	5.40	5.30	5.23	5.17	5.13	10.85	10.71	10.62	10.55	10.50	10.46	10.43
	DP-LLM	5.57	5.40	5.28	5.21	5.17	5.12	5.10	10.79	10.64	10.54	10.51	10.48	10.45	10.43

Table 12: Perplexity under different l and h combinations

Dataset		WikiText2 (\downarrow)	C4 (\downarrow)
Model	l & h	Perplexity	Perplexity
Llama-3-8B	3 & 5	6.55	10.30
	3 & 6	6.78	10.65
	4 & 5	6.36	9.87
	4 & 6	6.43	9.98
Phi-3-Medium	3 & 5	4.49	9.03
	3 & 6	4.56	9.11
	4 & 5	4.45	8.99
	4 & 6	4.48	9.01

C.4 Calibration Set Sensitivity Study

We investigate the impact of the calibration set on model performance. Table 13 presents the evaluation results for Llama-3-8B when the calibration set is switched from the training set of C4 to that of WikiText2. The results indicate that in most cases, DP-LLM performs effective fine-tuning regardless of the choice of calibration set, without signs of overfitting.

Table 13: Evaluation results with different calibration sets

Dataset	WikiText2 (\downarrow)								C4 (\downarrow)							
	Target Precision (Bits)								Target Precision (Bits)							
Calibration Set	3.25	3.50	3.75	4.00	4.25	4.50	4.75		3.25	3.50	3.75	4.00	4.25	4.50	4.75	
WikiText2	7.32	6.98	6.75	6.60	6.50	6.41	6.36		11.59	11.05	10.68	10.33	10.16	10.00	9.91	
C4	7.35	7.00	6.77	6.59	6.49	6.41	6.37		11.57	11.03	10.65	10.25	10.10	9.97	9.89	

D Limitations

DP-LLM is designed to enhance only the decoding phase of on-device LLM inference and does not affect the prefill phase. Additionally, downstream tasks evaluated solely based on log-probabilities are excluded from the experiments, as they do not involve decoding. DP-LLM aims to meet QoS constraints on a best-effort basis; QoS-guaranteed dynamic precision selection is left as future work.

E Decoding Examples

We present examples where DP-LLM successfully generates a response while the static baseline fails. DP-LLM dynamically assigns higher precision to decoding steps where the estimated output difference is significant, potentially preventing errors that could derail the entire decoding process. In the examples, the points of divergence are highlighted in red.

Input Prompt (GSM8K)

Question:

Gretchen has 110 coins. There are 30 more gold coins than silver coins.
How many gold coins does Gretchen have?

Generated Responses

FP16 Original Model

Let x be the number of silver coins Gretchen has. Then the number of gold coins is $x+30$.
The total number of coins is 110, so $x+x+30=110$ $2x+30=110$ $2x=110-30$ $2x=80$ $x=80/2$ $x=40$
So Gretchen has 40 silver coins and $40+30=\llcorner 40+30=70 \gg 70$ gold coins.
70

Static (HAWQ-V2)

Let x be the number of **gold** coins. Then the number of silver coins is $x+30$.
The total number of coins is $x+x+30=\llcorner x+x+30=2x+30 \gg 2x+30$ So $2x+30=110$, and $2x=80$, so
 $x=\llcorner 2x=80$, so $x=40 \gg 40$
40

DP-LLM

Let x be the number of silver coins Gretchen has. Then the number of gold coins is $x+30$.
The total number of coins is 110, so $x+x+30=110$ $2x+30=110$ $2x=80$ $x=40$
So the number of gold coins is $40+30=\llcorner 40+30=70 \gg 70$
70

Input Prompt (BBH)

Question:

The first day of 2019 is a Tuesday, and today is the first Monday of 2019.
What is the date 24 hours later in MM/DD/YYYY?

Options:

- (A) 01/08/1960
- (B) 01/01/2019
- (C) 02/08/2018
- (D) 01/08/2019
- (E) 10/22/2018
- (F) 01/08/2043

873

Generated Responses

FP16 Original Model

Let's think step by step.

The first day of 2019 is a Tuesday, and today is the first Monday of 2019. So today's date is Jan 7, 2019. The date 24 hours later is Jan 8, 2019, that is 01/08/2019. So the answer is (D).

Static (HAWQ-V2)

Let's think step by step.

The first day of 2019 is a Tuesday, and today is the first Monday of 2019. So today's date is Jan 1, 2019. 24 hours later is Jan 2, 2019. So the answer is (B).

DP-LLM

Let's think step by step.

The first day of 2019 is a Tuesday, and today is the first Monday of 2019. So today's date is Jan 7, 2019. 24 hours later is Jan 8, 2019, that is 01/08/2019. So the answer is (D).

874