No Gradients, No Problem?

A Comparative Study of Zeroth-Order Optimizers for Memory-Efficient LLM Fine-Tuning

Maria Pilligua, Nil Biescas, Chengheng Li Chen
CS-439 Optimization for Machine Learning, EPFL

***Abstract*—Fine-tuning large language models with backpropagation is memory hungry because it stores activations and gradients for every parameter. Zeroth-order (ZO) optimizers avoid this by estimating the gradient from forward passes alone, and many ZO variants have appeared recently, but each is reported on a different model, dataset and budget, which makes them hard to compare. We put ten ZO optimizers and a first-order AdamW baseline on a single harness and fine-tune Qwen3.5-0.8B, and the smaller Qwen2.5-0.5B, on two SuperGLUE tasks under the same protocol, measuring accuracy, time and memory. Sparse-MeZO is the strongest ZO method on the harder task, although every ZO method stays several points below AdamW. We also exploit a property unique to ZO, that it can optimize a non-differentiable objective directly, to train on accuracy instead of cross-entropy, and find that the cross-entropy surrogate remains the better training signal for every method but one. The code for this study is available at https://github.com/mpilligua/NoGradients-NoProblem.**

I. Introduction

Fine-tuning is how pre-trained language models are adapted to a task, but doing it with a standard first-order optimizer such as Adam is expensive in memory: backpropagation stores forward-pass activations and a gradient for every parameter, plus the optimizer state. For hundreds of millions of parameters this dominates the budget.

Zeroth-order (ZO) optimization offers a cheaper alternative. By estimating the gradient from the difference of two loss values at perturbed weights, a ZO step needs only forward passes and stores no activations, so its memory footprint is close to inference. The idea was made practical for language models by MeZO [1], and since then a steady stream of follow-up methods has tried to close the accuracy gap to first-order training by reducing the variance of the estimate or by spending the perturbation budget more cleverly. The difficulty is that almost every paper evaluates its method on a different model, dataset, and training budget, often against its own re-run of the MeZO baseline, which makes it hard to tell which method is actually better.

This work is a controlled comparison. We implement ten ZO optimizers and a first-order AdamW baseline behind a single training harness, so that only the optimizer changes between runs, and evaluate them on the same two SuperGLUE tasks and the same two base models. Our contributions are: (i) a shared, open implementation of ten ZO methods that run from one configuration file; (ii) a head-to-head accuracy, time and memory comparison on identical settings, including how the choice of base model changes the ranking; and (iii) a study of a property only ZO has, training on non-differentiable accuracy, which rarely helps in practice.

II. Models and Methods

A. Zeroth-order optimization

All the methods we compare build on top of MeZO [1], and share the same core idea of estimating gradients using only forward passes, without backpropagation. They all rely on a finite-difference estimator to approximate the gradient. Let θ be the model parameters and $L(\theta)$ the training loss. At each step we sample a random direction z with the same shape as θ, with entries drawn from a standard Gaussian, and evaluate the loss at two perturbed points. Their difference gives a finite-difference estimate of the directional derivative along z,

$$\hat{g} \;=\; \frac{L(\theta+\epsilon z) - L(\theta-\epsilon z)}{2\,\epsilon}\, z, \tag{1}$$

where ϵ is a small perturbation scale. The parameters are then updated as $\theta \leftarrow \theta - \eta\,\hat{g}$ with learning rate η. Only two forward passes are needed and no activations are stored, so the memory cost stays close to inference. The price is that $\hat{g}$ is a high variance estimate of the true gradient, and that variance grows with the number of parameters. Reducing this variance, or spending it more wisely, is what separates the methods below.

B. Methods compared

We compare ten zeroth-order optimizers and a first-order AdamW baseline [2]. Rather than describe them one by one, we group them by what they change about the estimator in Equation (1).

Sparsity and quantization. These methods shrink the cost of the perturbation itself. Sparse-MeZO [3] perturbs only a small fraction of the weights at each step, chosen as the entries with the smallest magnitude, so the random direction z is masked to a sparse support. QuZO [4] keeps the full perturbation but runs the forward passes in low precision (8-bit), which lowers memory at the cost of a noisier loss.

Low-rank and subspace. Instead of perturbing the full parameter space, these methods restrict z to a low-dimensional subspace, which lowers the effective dimension the variance depends on. LOZO [5] builds the perturbation from a low-rank product $UV^\top$ and refreshes the subspace only every few steps. SubZero [6] uses per-layer orthogonal projections so that the variance scales with the subspace rank rather than the full parameter count.

Curvature. HiZOO [7] keeps a diagonal estimate of the Hessian and uses it to precondition the update, so that the step is rescaled per coordinate. It pays one extra forward pass per step to maintain this estimate and stays purely forward-only.

Direction and variance reduction. The remaining methods keep the full perturbation but change how the direction is

sampled or how the estimate is aggregated. ConMeZO [8] restricts the sampled direction to a cone around a running momentum estimate, so search concentrates where the gradient is likely to point. FZOO [9] averages several one-sided probes per step with Rademacher perturbations and adapts the step size to the spread of the measured losses. DiZO [10] rescales the update per layer so that its magnitude matches what a first-order step would produce. PseuZO [11] estimates the Jacobian of the model output and applies an exponential moving average to reduce the variance of the estimate. We also report plain MeZO [1] as the zeroth-order reference point.

C. Experimental setup

We run every method on the same harness so that the only thing that changes between runs is the optimizer. Our main model is Qwen3.5-0.8B, and we also run the smaller Qwen2.5-0.5B [12] to see how the choice of base model affects the comparison. We fine-tune on two tasks from SuperGLUE [13] that were chosen to cover the two different answer formats the benchmark uses: MultiRC, a binary yes or no judgment about whether a candidate answer to a question is correct given a passage, and COPA, a two-way multiple choice between a plausible cause or effect. We use 1000 training examples per task following the few-shot protocol of MeZO. Training uses prompt-based language modeling, and at evaluation we score each candidate answer by its length-normalized log-likelihood and pick the highest, which is the standard way to evaluate these tasks without a separate classification head.

We do not fix a compute budget. Instead we run each method until it stops improving on the validation set and report the best validation accuracy reached, since the methods converge at very different rates and a fixed step count would favor whichever method happens to take larger steps. All runs use a single H100 GPU. Alongside accuracy we record the average wall-clock time per optimizer step and the peak GPU memory, so that the comparison covers cost as well as quality.

III. Results

A. Accuracy

Table I reports accuracy for every method, on both tasks and both base models, under two metrics: logit-level accuracy, which ranks the candidate answers by their length-normalized log-likelihood, and token-level accuracy, the fraction of answer tokens predicted correctly.

On MultiRC the two metrics agree closely, since the answer is a single Yes or No token, and the methods spread out enough to be ranked. On Qwen2.5 the best ZO methods cluster near 0.71 (ConMeZO 0.713, SubZero 0.712, DiZO 0.711, MeZO 0.709), and on Qwen3.5 Sparse-MeZO is clearly the strongest at 0.783, with LOZO second at 0.729. Every ZO method stays below the first-order AdamW baseline, which reaches 0.737 on Qwen2.5 and 0.832 on Qwen3.5; the gap is about five points for the best ZO method and larger for the rest.

COPA behaves very differently. Its validation set has only 100 examples and the base model already answers it fairly well with no training, so logit-level accuracy sits around 0.70 for every method and barely moves during training; in several runs it even drifts down toward chance. The token-level numbers expose this: while logit accuracy reads about 0.70, token accuracy on COPA is only around 0.42, which means the model is not actually generating the correct answer and the ranking metric mostly reflects the model's prior. We therefore treat COPA as saturated and draw our conclusions from MultiRC.

TABLE I
BEST VALIDATION ACCURACY ON MULTIRC AND COPA UNDER QWEN2.5-0.5B (Q2.5) AND QWEN3.5-0.8B (Q3.5). EACH ENTRY IS THE MAXIMUM OVER EVALUATIONS OF THAT METRIC. **GREEN** = BEST ZO METHOD PER COLUMN. GRAY ITALIC ROWS ARE REFERENCES..

	MultiRC				COPA			
	Q2.5		Q3.5		Q2.5		Q3.5	
Method	lgt	tok	lgt	tok	lgt	tok	lgt	tok
AdamW	*0.737*	*0.737*	*0.832*	*0.832*	*0.680*	*0.541*	*0.710*	*0.515*
Zero-shot	*0.581*	*0.580*	*0.461*	*0.253*	*0.690*	*0.449*	*0.700*	*0.419*
SparseMeZO	0.689	0.689	**0.783**	**0.783**	**0.730**	0.506	0.710	0.483
HiZOO	0.703	0.703	0.611	0.609	0.690	0.509	0.690	0.425
QuZO	0.668	0.668	0.692	0.692	0.690	0.414	0.710	0.414
LOZO	0.674	0.674	0.729	0.730	0.670	0.420	0.680	0.427
SubZero	0.712	0.712	0.667	0.668	0.680	0.506	0.680	0.455
PseuZO	0.695	0.695	0.656	0.656	0.690	0.501	**0.720**	**0.503**
MeZO	0.709	0.709	0.593	0.592	0.690	**0.523**	0.680	0.413
ConMeZO	**0.713**	**0.713**	0.670	0.671	0.690	0.503	0.660	0.416
FZOO	0.706	0.706	0.468	0.454	0.690	0.472	0.710	0.446
DiZO	0.711	0.711	0.599	0.602	0.690	0.506	0.680	0.405

The base model changes the ranking in a telling way. Most methods reach a higher optimum on the smaller Qwen2.5 and lose accuracy on the larger Qwen3.5. The two clear exceptions are Sparse-MeZO (0.689 to 0.783) and LOZO (0.674 to 0.729), the two methods that confine the perturbation to a sparse or low-rank subspace; they are the only ZO methods, besides AdamW, that exploit the larger model. This fits the intuition that the ZO estimate gets noisier as the parameter count grows, so methods that reduce the effective dimension of the perturbation cope better. The trend is suggestive rather than conclusive, since SubZero is also a low-rank method yet does not improve on Qwen3.5.

B. Time and memory

Table II reports the cost of one optimizer step and the GPU memory, measured on MultiRC with Qwen3.5 at batch size 1 so that the first-order and zeroth-order methods are directly comparable. The bare fp32 backbone occupies 1.46 GB, a lower bound for any optimizer.

The memory advantage of zeroth-order training is clear. AdamW keeps 5.73 GB resident between steps, because it stores a gradient and two moment buffers per parameter, and peaks at 9.69 GB during the backward pass. The lean ZO methods (QuZO, MeZO, LOZO, SubZero, DiZO, FZOO) sit essentially at the model-weights lower bound, around 1.5 GB resident, since they keep no optimizer state at all. The methods that do keep state cost more: Sparse-MeZO and HiZOO hold a mask or a diagonal Hessian (about 2.9 GB), and ConMeZO keeps momentum buffers (about 4.3 GB), but all of them stay well below AdamW.

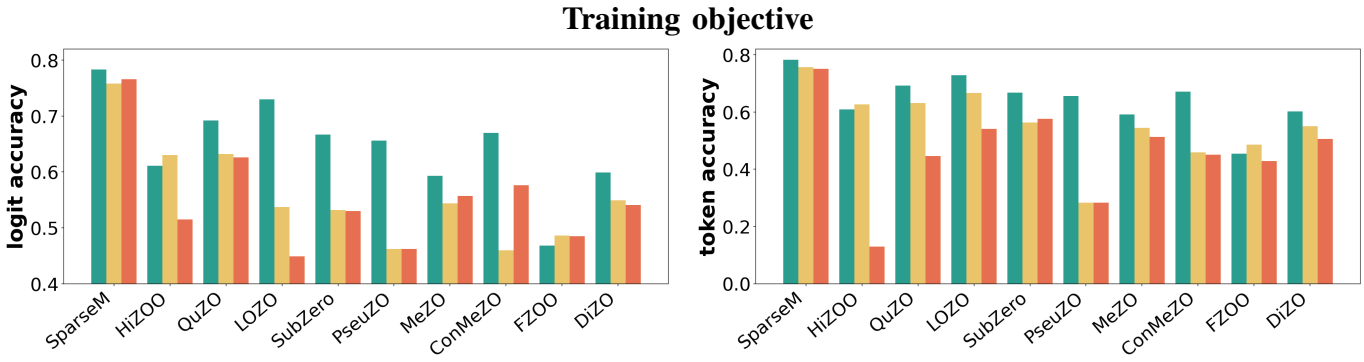

Fig. 1. Result of training each method on three objectives on MultiRC Qwen3.5: **cross-entropy**, **token-accuracy**, and **logit-accuracy**. Measured by logit accuracy (left) and token accuracy (right). CE wins almost everywhere; the gap is far larger for token accuracy.

TABLE II
EFFICIENCY OF EACH OPTIMIZER ON QWEN3.5-0.8B, MULTIRC (H100, BATCH SIZE 1; MEAN OVER 10 TIMED STEPS, 1 WARMUP STEP EXCLUDED): TIME PER OPTIMIZER STEP (S), GPU MEMORY REPORTED AS PEAK (RESIDENT AVERAGE) IN GB, AND THE NUMBER OF TRAINING STEPS UNTIL THE BEST LOGIT-ACCURACY EVAL. GRAY ITALIC ROWS ARE REFERENCES.

Method	s/step	Mem. GB: peak (avg)
Model weights	–	*1.46*
AdamW	*0.51*	*9.69 (5.73)*
SparseMeZO	0.37	5.61 (2.87)
HiZOO	0.40	5.77 (2.92)
QuZO	0.45	4.78 **(1.46)**
LOZO	0.26	**2.91** (1.48)
SubZero	0.26	3.43 (1.53)
PseuZO	0.27	3.91 (2.00)
MeZO	0.28	4.20 (1.46)
ConMeZO	0.28	7.10 (4.26)
FZOO	1.15	5.60 (1.46)
DiZO	**0.21**	4.25 (1.51)

On speed, most ZO methods take 0.2 to 0.45 seconds per step, comparable to AdamW's 0.51, because a step is just two forward passes with no backward. The exceptions are FZOO (1.15 s) that runs several forward passes per step, and HiZOO that pays an extra forward to maintain its Hessian estimate.

C. Training on accuracy instead of cross-entropy

A property unique to zeroth-order optimization is that it only reads a scalar loss back from the forward pass, so the objective need not be differentiable. Unlike Adam, a ZO method can therefore be trained directly on the evaluation accuracy rather than on the cross-entropy surrogate. We test this on MultiRC with Qwen3.5, training each method on these 2 additional objectives at the token-level accuracy, and logit-level accuracy, and report all the evaluation metrics in Tables III and IV.

The result is consistent: cross-entropy is the best training objective for almost every method (Figure 1). Measured by logit accuracy the only exceptions are HiZOO, which does slightly better trained on token accuracy, and FZOO, whose best logit score comes from training on logit accuracy; every other method is best with cross-entropy. The effect is far stronger on token accuracy, where training on an accuracy objective makes most methods collapse: HiZOO falls to 0.130 and FZOO to 0.256, while only Sparse-MeZO holds up across all three objectives. The reason is that accuracy is piecewise constant, so a small perturbation usually flips no predictions and the two probe losses are equal, which gives a zero gradient estimate for most steps and the model drifts instead of learning. Sparse-MeZO survives because its sparse mask keeps the perturbation small enough to still find informative directions. ZO can optimize the true metric, but in practice the cross-entropy surrogate is the better training signal.

IV. DISCUSSION

The first finding is that Sparse-MeZO is the method we would actually reach for. It is the best ZO method on MultiRC for both base models, it is the only one that clearly improves when we move to the larger Qwen3.5, and it is the only method that survives training on a non-differentiable accuracy objective. The second is more sobering: most of the recent methods do not beat plain MeZO on our setup. On MultiRC with Qwen2.5 the bulk of them, MeZO included, sit in a tight cluster around 0.71, and the more elaborate direction or variance-reduction tricks (ConMeZO, DiZO, FZOO, PseuZO) do not separate themselves from the baseline by a margin that would justify their extra complexity or, in the case of the multi-query methods, their extra cost per step. The two ideas that did pay off, sparsity and low-rank subspaces, both work by reducing the effective dimension of the perturbation, which is consistent with the view that the dominant problem in ZO is the variance of the estimate growing with the parameter count.

V. SUMMARY

We ran a controlled comparison of ten zeroth-order optimizers and a first-order AdamW baseline behind a single training harness, fine-tuning two Qwen models on two SuperGLUE tasks under one shared protocol and measuring accuracy, time and memory. The headline result is that Sparse-MeZO is the strongest and most robust ZO method, while most of the recent variants do not clearly beat plain MeZO on our setup, and every ZO method trades roughly five points of accuracy for about a third of AdamW's memory. We also showed that, although ZO can train directly on non-differentiable accuracy, the cross-entropy surrogate remains the better signal for nearly every method. Taken together, the comparison suggests that the payoff in zeroth-order fine-tuning comes from reducing the dimension of the perturbation rather than from elaborating the update rule.

References

[1] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning language models with just forward passes," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] Y. Liu, Z. Zhu, C. Gong, M. Cheng, C.-J. Hsieh, and Y. You, "Sparse MeZO: Less parameters for better performance in zeroth-order LLM fine-tuning," *arXiv preprint arXiv:2402.15751*, 2024.

[4] J. Zhou, Y. Yang, K. Zhen, Z. Liu, Y. Zhao, E. Banijamali, A. Mouchtaris, N. Wong, and Z. Zhang, "QuZO: Quantized zeroth-order fine-tuning for large language models," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[5] F. Chen *et al.*, "Enhancing zeroth-order fine-tuning for language models with low-rank structures (LOZO)," in *International Conference on Learning Representations (ICLR)*, 2025.

[6] Z. Yu *et al.*, "SubZero: Zeroth-order fine-tuning of LLMs in random subspaces," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

[7] Y. Zhao *et al.*, "Second-order fine-tuning without pain for LLMs: A hessian-informed zeroth-order optimizer (HiZOO)," in *International Conference on Learning Representations (ICLR)*, 2025.

[8] L. D. Behric, L. Zhang, B. Li, and K. K. Thekumparampil, "ConMeZO: Cone-constrained zeroth-order optimization for memory-efficient fine-tuning," *arXiv preprint arXiv:2511.02757*, 2025.

[9] others, "FZOO: Fast zeroth-order optimizer for fine-tuning LLMs with adaptive batched forward passes," *arXiv preprint arXiv:2506.09034*, 2025.

[10] ——, "Harmony in divergence: Towards fast, accurate, and memory-efficient zeroth-order LLM fine-tuning (DiZO)," *arXiv preprint arXiv:2502.03304*, 2025.

[11] Yang *et al.*, "PseuZO: Pseudo-zeroth-order algorithm for training deep neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[12] Qwen Team, "Qwen technical reports and model cards," 2025, https://huggingface.co/Qwen.

[13] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "SuperGLUE: A stickier benchmark for general-purpose language understanding systems," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Appendix

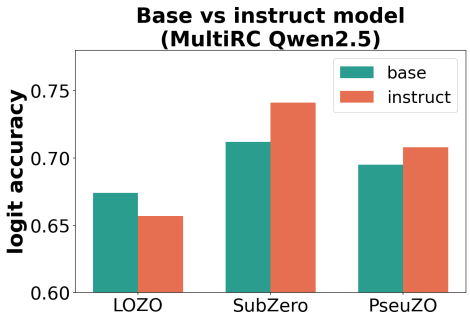

Fig. 2. Ablation study comparing the base and instruction-tuned variants of Qwen2.5 on MultiRC. Instruction tuning has only a minor effect on performance, with accuracies remaining largely unchanged across methods. PseuZO shows a small improvement of 1.3 percentage points, while LOZO slightly decreases and SubZero exhibits a moderate gain.

TABLE III
LOGIT-LEVEL ACCURACY ON MULTIRC (QWEN3.5-0.8B) WHEN TRAINING ON CROSS-ENTROPY, TOKEN-LEVEL ACCURACY, OR LOGIT-LEVEL ACCURACY. **GREEN BOLD** = BEST PER ROW, **YELLOW BOLD** = SECOND BEST PER ROW.

Method	CE	token-acc	logit-acc
SparseMeZO	**0.783**	0.758	**0.766**
HiZOO	**0.611**	**0.630**	0.515
QuZO	**0.692**	**0.632**	0.626
PseuZO	**0.656**	**0.462**	**0.462**
LOZO	**0.730**	**0.537**	0.449
SubZero	**0.667**	**0.532**	0.53
MeZO	**0.593**	0.544	**0.557**
ConMeZO	**0.670**	0.460	**0.576**
FZOO	0.468	**0.486**	**0.485**
DiZO	**0.599**	**0.549**	0.541

TABLE IV
TOKEN-LEVEL ACCURACY ON MULTIRC (QWEN3.5-0.8B) WHEN TRAINING ON CROSS-ENTROPY, TOKEN-LEVEL ACCURACY, OR LOGIT-LEVEL ACCURACY. **GREEN BOLD** = BEST PER ROW, **YELLOW BOLD** = SECOND BEST PER ROW

Method	CE	token-acc	logit-acc
SparseMeZO	**0.783**	**0.757**	0.751
HiZOO	**0.609**	**0.627**	0.130
QuZO	**0.692**	**0.631**	0.446
PseuZO	**0.656**	**0.283**	**0.283**
LOZO	**0.729**	**0.666**	0.541
SubZero	**0.668**	0.563	**0.576**
MeZO	**0.592**	**0.545**	0.513
ConMeZO	**0.671**	**0.459**	0.451
FZOO	**0.454**	**0.486**	0.429
DiZO	**0.602**	**0.550**	0.506