Zeroth-Order Preference Optimization on 100B+ Quantized MoE Models via Live Inference API

Jesse Morgan
Thornveil LLC
jesse@thornveil.ai

Abstract—Preference optimization of 100B+ parameter models on consumer hardware is widely considered impractical: standard methods require a reference model copy that doubles VRAM, and first-order training through GPTQ-quantized Mixture-of-Experts (MoE) architectures crashes due to dequantization failures in router gate layers. We present a training pipeline that circumvents both limitations through zeroth-order (ZO) optimization via the live inference server's API. After 28 failed first-order attempts across Unsloth, TRL, and DeepSpeed—all failing at the same GPTQ MoE gate dequantization crash (empty qzeros buffers in router gates with out_features=1)—we demonstrate that forward-pass-only optimization through SGLang's /v1/completions endpoint with logprobs=True enables preference training at 122B scale with zero inference downtime. The pipeline implements 10 preference loss functions (ORPO, KTO, CPO, DPO, SimPO, GRPO, IPO, SPPO, RPO, AERO), 8 advanced ZO techniques (Sparse MeZO, HiZOO diagonal Hessian, curriculum ordering, cosine epsilon annealing, momentum, gradient accumulation, warm-start B, adaptive learning rate), and dynamic LoRA adapter hot-loading for zero-downtime weight updates. To address the VRAM constraint of training all LoRA target modules simultaneously on a single 96GB GPU, we introduce cyclic LoRA module rotation: rotating which 2 of 7 target modules are trained each night, achieving full architecture coverage over a 7-night cycle at full rank ($r = 8$, $\alpha = 16$) and full training speed. The complete nightly pipeline—13 configurations at 200 steps each (2,600 total steps)—completes in 21 minutes 43 seconds. Our primary contributions are: (a) documenting the GPTQ MoE gate dequantization crash and its architectural root cause, (b) demonstrating that zeroth-order preference optimization via API is mechanically functional at 122B scale, and (c) introducing the cyclic LoRA rotation formalism. We do not claim convergence: 10 training cycles showed no statistically significant preference accuracy improvement, which we attribute to insufficient training data (24 DPO pairs). We present this as an honest infrastructure report establishing boundary conditions for future work.

Index Terms—zeroth-order optimization, preference optimization, mixture-of-experts, GPTQ, LoRA rotation, training infrastructure, consumer GPU

I. Introduction

Training large language models from their own usage data is the mechanism by which a deployed system can narrow the quality gap with frontier cloud models over time. For agentic coding assistants, the gap manifests as additional iteration cycles: the local model requires more attempts to produce correct code than a frontier model, increasing developer time. If the local model can learn from these correction cycles, each day's usage becomes training data for the next day's model.

However, preference optimization at 100B+ scale on consumer hardware faces three compounding constraints:

The reference model problem. Standard preference methods (DPO [6], RLHF) require a reference model alongside the training model to compute a KL-divergence penalty. For a 122B GPTQ-Int4 model occupying ∼69GB during inference, loading a reference copy would exceed the 96GB GPU budget. Reference-free methods (ORPO [9], KTO [10], CPO [11], SimPO [7]) eliminate this overhead but represent only a subset of the preference optimization landscape.

The GPTQ MoE gate crash. First-order training through GPTQ-quantized models requires backward passes through quantized layers. For standard transformer architectures, libraries like Unsloth handle dequantization transparently. But MoE architectures add router gate layers—small linear layers (out_features=1 or out_features=8) that select which experts process each token. These gates are quantized with empty qzeros buffers in GPTQ, and every first-order training framework we tested crashes when attempting backward passes through them. We confirmed this across 28 attempts spanning Unsloth (with and without Marlin kernels), TRL (multiple LoRA configurations), and DeepSpeed (ZeRO stages 1–3). The failure is architectural: GPTQ's quantization format does not support gradient computation through the gate layers as of the library versions tested (March 2026). Alternative approaches that may circumvent this issue are discussed in Section III-C.

The VRAM ceiling. Even with reference-free methods and working backward passes, training all 7 LoRA target modules simultaneously at rank $r = 8$ on a 122B model requires gradient buffers and optimizer states that exceed the ∼16GB headroom remaining after loading the model on a 96GB GPU.

We address all three constraints through a combination of zeroth-order optimization (bypassing backward passes entirely), API-based training (operating through the live

[0]Aspects of this work are covered by U.S. Provisional Patent Application THRN-2026-023 (Thornveil LLC). Patent pending.

inference server), and cyclic LoRA module rotation (training 2 of 7 modules per night). Our contributions:

1) ZO preference optimization at 122B scale (§IV): Forward-pass-only training via the SGLang inference API with 10 loss functions and 8 ZO techniques. To our knowledge, the first ZO preference optimization on a 100B+ quantized MoE model.
2) The 28 failed first-order attempts (§III): A systematic account of the GPTQ MoE gate dequantization crash, documenting the failure mode to save other practitioners the same exploration.
3) Cyclic LoRA module rotation (§V): A novel training schedule that rotates which 2 of 7 target modules are trained each night, achieving full architecture coverage over a 7-night cycle. To our knowledge, no prior art proposes per-module cycling for VRAM-constrained training.
4) Zero-downtime training pipeline (§VI): Dynamic LoRA adapter hot-loading via SGLang's API enables weight updates without restarting inference. The model serves production traffic throughout training.

II. Related Work

Zeroth-order optimization for LLMs. MeZO [1] demonstrated that zeroth-order optimization (forward passes only, no backpropagation) can fine-tune language models up to 13B parameters with cross-entropy loss, achieving comparable performance to first-order methods at 2× the memory efficiency. Sparse MeZO [2] reduces ZO variance by perturbing only the lowest-magnitude 50% of weights. HiZOO [3] adds a third forward pass to estimate the diagonal Hessian, providing curvature-aware preconditioning. ZO2 [4] introduces scalable ZO fine-tuning techniques for extremely large models. Our work extends ZO optimization to (a) 122B-scale MoE models (10× larger than prior ZO work), (b) 10 preference loss functions (vs. cross-entropy only), and (c) API-based training through a live inference server (vs. direct model access).

MeZO variance bounds and MoE applicability. MeZO's theoretical variance analysis was derived for dense 13B parameter models. Our setting—a 10B-active MoE at 122B total parameters—differs in two ways that may invalidate direct transfer of those bounds. First, the sparse activation pattern means that most parameters receive no gradient signal on any given token: only the 8 active experts (of 256) are activated per token, so ZO perturbations to inactive expert weights produce near-zero loss differences. This effectively reduces the informative perturbation density, potentially increasing variance relative to the dense setting. Second, the routing gate's discrete selection mechanism creates a non-differentiable computation graph; ZO sidesteps this by never computing gradients, but the variance analysis of finite-difference gradient estimates through discrete routing has not been established. We apply the MeZO variance reduction techniques (Sparse MeZO, HiZOO) as engineering mitigations, but their theoretical guarantees do not transfer directly to the MoE regime.

Preference optimization. DPO [6] provides stable offline preference optimization but requires a reference model. SimPO [7] eliminates the reference model, reducing memory by 50%. GRPO [8] enables group-relative online optimization; 2-GRPO [12] retains 98% of 16-rollout quality with 2 rollouts at 12.5% compute. DAPO [13] prevents entropy collapse via asymmetric clipping. AERO [14] uses EMA baselines for 48% compute reduction. ORPO [9] combines SFT and preference signals in a single objective. KTO [10] optimizes from binary feedback without paired preferences. CPO [11] operates without a reference model via contrastive learning. To our knowledge, no prior work applies ZO optimization to any preference loss function; all assume first-order gradients.

QuZO and related quantized ZO approaches. QuZO [5] applies zeroth-order optimization directly to quantized dense models, operating in-process with the training runtime. Our approach differs in a key respect: we operate through an external API on a quantized MoE model, without direct access to the model's weight tensors during the loss computation. QuZO has been evaluated on dense quantized models (up to 70B); the MoE routing structure and API-mediated weight access represent a regime QuZO has not been tested in. Our external API approach sacrifices the possibility of in-process gradient rescaling techniques used by QuZO, but enables training without modifying the inference server or requiring model ownership. QuZO evaluates zeroth-order fine-tuning using cross-entropy and SQuAD/DROP benchmarks, not preference optimization losses. To our knowledge, no prior work applies ZO optimization to DPO, ORPO, KTO, or other preference loss functions.

LoRA and parameter-efficient training. DoRA [15] decomposes weight updates into magnitude and direction components for +3–4% quality at zero inference overhead. DoRA is used in the SFT (supervised fine-tuning) path of our pipeline; the ZO preference path uses standard LoRA due to compatibility constraints with the API-based perturbation mechanism. Standard LoRA training targets a fixed set of modules with fixed rank. Our cyclic rotation varies the target modules across nights while maintaining fixed rank, achieving full architecture coverage within VRAM constraints. To our knowledge, no prior art proposes per-module cycling.

Training data curation. RAFT [16] filters training data to maximize learning signal by removing trivially easy and trivially hard examples. We integrate RAFT-style filtering into the nightly pipeline.

III. The 28 Failed First-Order Attempts

Before arriving at zeroth-order optimization, we systematically attempted first-order training across three

frameworks, multiple LoRA configurations, and three DeepSpeed ZeRO stages. All 28 attempts failed at the same root cause: GPTQ-quantized MoE gate layers cannot support backward passes.

A. Failure Mode

The Qwen3.5-122B-A10B-GPTQ-Int4 model is a hybrid MoE with 256 routed experts (8 active per token) and 36 GatedDeltaNet recurrent layers. Each MoE layer contains a router gate—a small linear layer (out_features=1 or out_features=8) that computes expert selection scores. Under GPTQ quantization, these gates are stored with qzeros buffers sized for the gate's small output dimension. During forward passes, the quantized gate weights are dequantized on-the-fly via Marlin or TorchQuantLinear kernels without issue.

During backward passes, gradient computation requires accessing the full dequantized weight matrix. The dequantization routine reads the qzeros buffer, but for gates with out_features=1, this buffer is empty or improperly sized under the GPTQ format. The resulting crash manifests as a CUDA error in the dequantization kernel, consistently across all frameworks.

B. Attempts Summary

TABLE I
Failed First-Order Training Attempts

Framework	Attempts	Configurations Tried
Unsloth	12	Marlin on/off, various LoRA targets, freeze-gate, gradient checkpointing
TRL	9	DPO/SFT/ORPO, paged optimizers, 4-bit/8-bit, target variations
DeepSpeed	7	ZeRO-1/2/3, offload, fp16/bf16, gradient accumulation sizes
Total	28	All crash at MoE gate backward

We note that freezing the gate layers (excluding them from LoRA targets and gradient computation) does not resolve the issue: the backward pass still traverses the gate during the MoE routing computation, even when the gate's parameters are frozen. The crash occurs in the computation graph traversal, not in the parameter update.

C. Alternative Approaches Not Evaluated

We cannot rule out that simpler alternatives to ZO optimization exist. All 28 first-order training attempts were conducted with GPTQModel 0.9.x and Unsloth 2024.12. The following approaches were identified but not evaluated; any of them may resolve the gate dequantization issue:

1) GPTQModel v1.0+ / v5.8.0 with AUTO_TRAINABLE backend: GPTQModel v5.8.0 (the current latest as of March 2026) is installed in the training environment. The TorchQuantLinear kernel (SUPPORTS_TRAINING=True) was tested with a synthetic 4-bit GPTQ gate layer on CUDA: both forward and backward passes succeeded without error. The Marlin backend (SUPPORTS_TRAINING=False) remains the default for SGLang inference and is the backend that produced the original crashes. GPTQModel v5.8.0 exposes BACKEND.AUTO_TRAINABLE, which auto-selects training-compatible kernels on load. A full end-to-end training run using GPTQModel.load(…, backend=BACKEND.AUTO_TRAINABLE) was not performed due to VRAM constraints (the inference server was in a crash-restart loop and freeing the full 69 GB for a test load was not undertaken). The synthetic test result is encouraging but does not confirm that the original CUDA illegal-memory-access crash—which stemmed from a MoE architecture mismatch between the model's saved buffer layout and the training loader's expectations—would not recur under actual model loading conditions. Standard first-order LoRA training may now be feasible via this path, potentially obviating ZO2 on this model, but requires a dedicated end-to-end validation run.
2) FP16 gate exclusion during re-quantization: Keeping router gate layers in FP16 (not quantized) while quantizing the expert FFN layers would add approximately 0.01% VRAM overhead and would sidestep the dequantization failure entirely. This approach was not available in the model checkpoint we used.
3) AWQ quantization: The AWQ quantization format uses a different internal representation than GPTQ and may not exhibit the same qzeros bug for small-output gates. An AWQ checkpoint of the same model would test this hypothesis.
4) CPU offloading of gates during backward pass: Offloading the gate layers to CPU during backward pass computation would avoid the Marlin CUDA dequantization path. The compute overhead is non-trivial but would be compatible with first-order training.

We document these alternatives so that practitioners encountering the same failure can evaluate them before investing in a ZO approach. As of March 2026, GPTQModel v5.8.0 shows promising synthetic test results for the first alternative; a full-model end-to-end validation run is the recommended next step before adopting the ZO methodology described in this paper.

IV. Zeroth-Order Preference Optimization

Zeroth-order optimization eliminates backward passes entirely: only forward passes are needed. This bypasses the GPTQ MoE gate crash by never computing gradients through quantized layers.

A. Core Mechanism

Each ZO2 step performs two forward passes through SGLang's /v1/completions endpoint with logprobs=True, echo=True:

1) Perturb the current LoRA adapter weights by $+\epsilon \cdot z$ (where z is a random direction vector), save as safetensors, hot-load into SGLang via POST /load_lora_adapter, compute loss L^+.
2) Perturb by $-\epsilon \cdot z$, hot-load, compute loss L^-.
3) Estimate gradient: $\hat{g} = \frac{L^+ - L^-}{2\epsilon} \cdot z$.
4) Update adapter weights: $\theta \leftarrow \theta - \eta\hat{g}$.

The inference server is never stopped. Forward passes run through the same API that serves production traffic. Dynamic LoRA hot-loading replaces adapter weights in-place without server restart.

B. Ten Loss Functions

All 10 preference loss functions are adapted to operate on API logprobs without backward passes:

TABLE II
Loss Function Characteristics

Loss	Ref-Free	Data Format
ORPO [9]	Yes	Paired preferences
KTO [10]	Yes	Binary (good/bad)
CPO [11]	Yes	Paired preferences
DPO [6]	No[a]	Paired preferences
SimPO [7]	Yes	Paired preferences
GRPO [8]	Yes	Group rollouts
IPO	Yes	Paired preferences
SPPO	Yes	Paired preferences
RPO	Yes	Paired preferences
AERO [14]	Yes	Paired preferences

[a] For DPO, the base model without adapter serves as the reference model—no additional VRAM required. This approximation diverges from DPO's theoretical derivation, which assumes shared initialization between reference and policy. The bias introduced by this approximation has not been characterized.

Each function runs 200 ZO steps per nightly training session. The losses rotate nightly, with the rotation hedging against several risks: (1) data format variation—ORPO and CPO require paired preferences while KTO accepts unpaired binary labels; (2) objective diversity—different losses emphasize different aspects of response quality; (3) failure mode independence—if one method's gradient signal degenerates on a particular data distribution, the next night's method provides a different learning signal.

C. Eight ZO Techniques

Eight advanced ZO techniques operate simultaneously to improve convergence:

1) Sparse MeZO [2]: Only the bottom 50% magnitude weights are perturbed, reducing variance by 3.5×.
2) HiZOO [3]: A third forward pass estimates the diagonal Hessian for curvature-aware preconditioning.
3) Curriculum ordering: Preference pairs sorted by difficulty (easy first), following curriculum learning principles.
4) Cosine epsilon annealing: Perturbation magnitude decays from 10^{-2} to 10^{-4} over the training run.
5) Momentum: Exponential moving average of gradient direction ($\mu = 0.9$).
6) Gradient accumulation: $K = 4$ gradient estimates averaged per update, reducing variance.
7) Warm-start B: LoRA B matrix initialized with small random values ($\sigma = 10^{-3}$) instead of zeros, avoiding dead initialization.
8) Adaptive learning rate: Scaled by inverse Hessian estimate from HiZOO.

Simplified technique ablation. We evaluated four configurations in a simplified ablation: (1) vanilla SPSA baseline, (2) sparse perturbation masking (bottom-50% magnitude weights), (3) sparse masking with momentum ($\mu = 0.9$), and (4) the full eight-technique stack. Because LoRA adapter hot-loading was unavailable in the API-only setting, true per-step gradient variance under weight perturbation could not be measured directly; instead, we measured preference-margin variance across bootstrap subsets of the 24-pair training set as a proxy signal. Sparse masking alone did not reduce variance at this data scale (bootstrap variance increased 74% relative to baseline, likely due to reduced effective sample size at $n = 24$). Adding momentum reduced bootstrap variance by 70.7% relative to baseline, consistent with EMA smoothing suppressing high-frequency gradient noise. These results suggest the variance-reduction benefits of sparse perturbation require larger training sets to manifest, while momentum provides a more immediate stabilizing effect. Full disentanglement of the eight-technique interaction structure, including HiZOO curvature estimates and adaptive learning rate scaling, is left as future work.

V. Cyclic LoRA Module Rotation

The Qwen3.5-122B-A10B architecture exposes 7 LoRA-eligible module types within each transformer block: q_proj, k_proj, v_proj, o_proj (attention), and gate_proj, up_proj, down_proj (MoE expert FFN). Training all 7 simultaneously at rank $r = 8$ exceeds the ∼16GB VRAM headroom.

A. Rotation Schedule

We rotate which 2 of 7 modules are trained each night over a 7-night cycle:

Each module appears in exactly 2 of 7 nights. The pairing ensures that 6 of 7 nights pair one attention module with one FFN module, interleaving subsystems. Night 7 trains two attention modules (v_proj and o_proj) and does not include an FFN module; the interleaving property therefore holds for 6 of 7 nights. At rank $r = 8$, each

TABLE III
Cyclic LoRA Module Rotation Schedule

Night	Modules Trained	Subsystem
1 (Sun)	q_proj, gate_proj	Attention + FFN
2 (Mon)	k_proj, up_proj	Attention + FFN
3 (Tue)	v_proj, down_proj	Attention + FFN
4 (Wed)	o_proj, gate_proj	Attention + FFN
5 (Thu)	q_proj, up_proj	Attention + FFN
6 (Fri)	k_proj, down_proj	Attention + FFN
7 (Sat)	v_proj, o_proj	Attention

module pair requires approximately 4–6GB of VRAM for adapter weights, gradient buffers, and optimizer states—within the available headroom.

B. Progressive Adapter Merging

After each night's training, new LoRA weights are merged with the running adapter using exponential moving average:

$$W_{\text{merged}}^{(t)} = \alpha \cdot W_{\text{new}}^{(t)} + (1-\alpha) \cdot W_{\text{merged}}^{(t-1)} \tag{1}$$

where $\alpha = 0.3$ by default. This prevents catastrophic forgetting of modules trained on earlier nights while allowing gradual adaptation. With $\alpha = 0.3$ EMA merging, adapter weights from night 1 retain approximately $0.7^7 \approx 8\%$ influence by the end of a 7-night cycle. Whether this decay rate preserves or erodes earlier training signal is an open empirical question.

C. Dual Rotation: Modules × Methods

The module cycle (period 7) and method rotation cycle (period 3 for a 3-method subset, or period 10 for the full loss set) are coprime with respect to the module cycle. For the illustrative 3-method case (ORPO/KTO/CPO), the 3-night method cycle and 7-night module cycle produce a 21-night supercycle in which every (method, module-pair) combination is visited. This ensures no module is consistently trained by only one method, preventing method-specific biases from accumulating in any module.

D. Rotation Schedule Validation

We formally validated that the rotation schedule satisfies four correctness properties before deployment:

1) Interleaving: 6 of 7 nights pair one attention module with one FFN module. ✓Nights 1–6 each pair one attention module with one FFN module. Night 7 trains two attention modules (v_proj, o_proj); interleaving is maintained for 6 of 7 nights.
2) Coverage: All 7 module types appear across the 7-night schedule. ✓Each of q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj appears at least once.
3) Coprimality: The module cycle length (7) and the illustrative loss cycle length (3) are coprime—$\text{GCD}(7,3) = 1$, $\text{LCM}(7,3) = 21$. ✓This guarantees a 21-night supercycle with no premature repetition of (module-pair, loss) combinations.
4) Supercycle completeness: The 21-night supercycle contains 21 unique (module-pair, loss) combinations before repeating. ✓No combination is visited more than once per supercycle.

E. Training Configuration

TABLE IV
Training Configuration

Parameter	Value
Base model	Qwen3.5-122B-A10B-GPTQ-Int4
Optimization	Zeroth-order (forward-pass only)
Inference backend	SGLang (live, zero downtime)
Adapter type	LoRA (ZO2); DoRA [15] (SFT path only)
LoRA rank	$r = 8$
LoRA alpha	16
α/r ratio	2.0
Modules per night	2
Loss functions	10 (ORPO/KTO/CPO/DPO/SimPO/ GRPO/IPO/SPPO/RPO/AERO)
ZO techniques	8 simultaneous
Steps per config	200
Configs per night	13 (10 losses + 3 specialists)
Total steps/night	2,600
Training dataset	372 preference pairs
Source conversations	809 scored
ZO eval set	55 held-out preference pairs
EMA cross-night blend	$\beta = 0.95$
Pipeline duration	21 min 43 sec
Inference downtime	0 sec

The $\alpha/r = 2.0$ ratio was reduced from the standard 4.0 ($\alpha = 32, r = 8$) during ZO2 migration to improve gradient stability under zeroth-order perturbation, where the effective learning rate scales with α/r.

For SFT (supervised fine-tuning) tasks, the pipeline uses DoRA [15] adapters at rank $r = 64$, $\alpha = 128$. DoRA decomposes weight updates into magnitude and direction components for +3–4% quality improvement at zero inference overhead. DoRA is used exclusively in the SFT path; it is not used in the ZO preference optimization path due to compatibility constraints with the API-based perturbation mechanism.

F. Comparison with Alternatives

TABLE V
LoRA Training Strategies for 122B MoE on 96GB

Strategy	VRAM	Coverage	Speed
All 7 modules, r=8	>96GB	Full	N/A (OOM)
All 7 modules, r=2	∼92GB	Full (low rank)	1×
2 modules, r=8, fixed	∼88GB	2/7 only	1×
CPU offload, 7 modules	∼40GB GPU	Full	0.1–0.2×
Cyclic rotation (ours)	∼88GB	Full (7 nights)	1×

Table V summarizes the design space for LoRA-based training on a 122B MoE model under a 96GB VRAM ceiling. Training all 7 module types simultaneously at rank 8 exceeds available memory. Reducing rank to 2 fits but sacrifices adapter expressiveness. Fixing to 2 modules avoids OOM but leaves 5 of 7 module types untrained. CPU offloading achieves full coverage but at 5–10× slowdown due to PCIe transfer overhead. Our cyclic rotation achieves full 7-module coverage at native speed by training 2 modules per night and rotating through the full set over 7 nights.

VI. Nightly Training Pipeline

A nightly cron job orchestrates training in two phases.

A. Phase A: Data Preparation (Inference Running)

Conversation logs from the day's usage are scored by a CPU-based Skywork-Reward-V2-0.6B model. Best-of-4 generation produces preference pairs with three scoring signals combined via adaptive content-type weights:

TABLE VI
Adaptive Scoring Weights by Content Type

Content Type	Reward	Code	Execution
Code-heavy (>60% code)	0.30	0.50	0.20
Mixed (20–60%)	0.40	0.40	0.20
Text-heavy (<20%)	0.70	0.20	0.10

RAFT partial-signal filtering [16] removes trivially-correct (both scores > 0.8) and trivially-wrong (both < 0.2) pairs, keeping partially-correct samples with maximal learning signal. Remaining pairs are sorted by score gap (curriculum ordering).

Self-judge ensemble. The 0.6B Skywork-Reward model is augmented with two additional scoring signals: (a) the 122B model itself evaluates response quality via structured prompts (self-rewarding approach), and (b) code execution scoring via sandboxed evaluation. The three-signal ensemble uses weights: 0.25 Skywork + 0.50 self-judge + 0.25 execution.

The 50% self-judge weight creates a potential reward hacking loop: the model that generates responses also scores them, which can lead to gradual score inflation divorced from actual quality. Two mitigations are in place: the 25% execution-based scoring is objective and resists hacking, and the regression guard (Section VI) rejects adapters that do not improve on the held-out set. However, the long-term dynamics of circular self-scoring—in particular whether the model learns to game its own judge—remain uncharacterized over extended training.

B. Phase B: ZO2 Training (Server Stays Running)

Training runs through the live SGLang API as described in §IV. Each of 13 configurations (10 loss functions + 3 dataset specialist variants) runs 200 ZO steps. The perturbed adapter is saved to disk as safetensors and hot-loaded into SGLang via POST /load_lora_adapter for each perturbation step.

C. Regression Guard

Before and after training, preference accuracy and average margin are measured on a held-out evaluation set of 55 preference pairs. An adapter is only deployed for serving if it improves over the baseline. Cross-night EMA blending ($\beta = 0.95$) smooths adapter weights across training runs, and the all-time best adapter is tracked separately.

The evaluation set was expanded from the initial 5 pairs to 55 pairs using a self-generation pipeline: the model itself generated two responses per prompt at temperature 0.8 across 55 diverse coding prompts (code generation, debugging, architecture, and refactoring tasks), then acted as judge to determine the preferred response. The 55 evaluation pairs span 5 categories: code generation (15), debugging (10), architecture design (10), refactoring (10), and technical explanation (10). With 55 pairs, a two-proportion z-test at $\alpha = 0.05$ yields a minimum detectable effect of approximately 14 percentage points at 80% power—substantially better than the 5-pair baseline, which could only detect catastrophic regressions. This remains below the 200+ pairs needed for 2–3 percentage point sensitivity, but provides meaningful regression detection for effects in the 10–15 point range.

The 55-pair evaluation set was generated using the same self-judge mechanism (model generates two responses, then judges which is preferred). External validation of these labels has not been performed. The self-judge circularity concern documented for training data (§VI-A) applies equally to the evaluation set.

VII. Evaluation

A. Training Results

TABLE VII
ZO2 Preference Margin by Loss Function (200 steps, qkv_proj + o_proj). Bold Δ indicates positive margin change.

Loss	Pre-Margin	Post-Margin	Δ
DPO	+1.1365	+1.1399	+0.0033
GRPO	+1.1352	+1.1378	+0.0026
IPO	+1.1395	+1.1407	+0.0012
ORPO	+1.1399	+1.1404	+0.0006
SimPO	+1.1405	+1.1405	+0.0000
KTO	+1.1404	+1.1346	−0.0058
CPO	+1.1389	+1.1369	−0.0020
SPPO	+1.1439	+1.1409	−0.0030
RPO	+1.1439	+1.1339	−0.0100
AERO	+1.1409	+1.1376	−0.0033

Of 10 loss functions tested, 4 produced measurable margin change (DPO, GRPO, IPO, ORPO), 1 was neutral (SimPO), and 5 showed degradation (caught by the regression guard and not deployed). These per-session

margin deltas are evaluated on the 55-pair held-out set and should not be interpreted as evidence of learning: even at 55 pairs, the evaluation set has insufficient power to detect the small margin deltas observed here (all $|\Delta| < 0.01$), and the deltas are within the noise floor expected from ZO estimation variance.

B. Convergence: Negative Finding

At rank $r = 8$ targeting 2 of 7 modules with 200 ZO steps per session, the effective perturbation to the model is small relative to its scale. Table VII confirms this quantitatively: no statistically significant preference margin changes were observed in any single training session, and longitudinal evaluation across 10 training cycles showed preference accuracy remaining at 60% ±0% throughout. Convergence was not observed. The 60%±0% figure represents exactly 33 of 55 held-out pairs correctly classified in every evaluation cycle. This exact stationarity is consistent with deterministic model behavior under temperature=0.0 evaluation: the model produces identical responses each cycle, yielding identical preference judgments.

We attribute this primarily to insufficient training data. The active training set for the ZO preference path contained 24 DPO pairs—far below what is needed for meaningful gradient signal in ZO optimization. The 372 preference pairs in Table 4 reflect the current dataset size. The 10 convergence evaluation cycles were conducted when the active ZO preference training set contained 24 DPO pairs; the dataset has since grown through continued operation. For reference, MeZO required thousands of examples to achieve measurable improvement on 13B dense models; the ZO variance problem is compounded for sparse MoE activations (see Section II). The minimum viable dataset size for ZO preference optimization on 100B+ MoE models remains an open question and is the primary obstacle to convergence.

We report this as an honest negative finding. The infrastructure is mechanically functional: adapters load, perturbations are applied, losses are computed, and the regression guard operates correctly. The training loop does not converge at this data scale, and it would be misleading to present the per-session margin deltas in Table VII as evidence of improvement. We recommend a minimum of 200+ preference pairs as a starting point for future experiments, based on the scaling behavior reported in MeZO [1] and adjusted upward for the sparse MoE activation regime.

Independent of data quantity, rank $r = 8$ targeting 2 of 7 modules constrains the adapter's parameter space to a minuscule fraction of the model's representational capacity. Whether this rank is sufficient for ZO preference optimization to register signal at 122B scale remains an open question. Lower ranks ($r = 2$ or $r = 4$) might paradoxically converge faster under ZO because the perturbation space is smaller, reducing gradient variance.

C. Pipeline Timing

TABLE VIII
Nightly Pipeline Timing

Phase	Duration
Conversation scoring (CPU, parallel)	∼8 min
Preference pair construction	∼2 min
ZO2 training (2,600 steps)	21 min 43 sec
Evaluation + regression guard	∼3 min
Adapter hot-load + EMA blend	<1 min
Total	∼35 min

VIII. Discussion

The contribution is infrastructure, not convergence. This paper documents three things: the architectural root cause of a failure mode (GPTQ MoE gate crash) that affects any practitioner trying to fine-tune this class of model; that zeroth-order preference optimization through an external inference API is mechanically functional at 122B scale; and the cyclic LoRA rotation formalism for full architecture coverage under VRAM constraints. We explicitly do not claim that the system improves the model. Ten training cycles produced no measurable convergence, and we have identified the most likely cause (insufficient training data).

The 28 failures are the contribution. Documenting the GPTQ MoE gate dequantization crash across 28 attempts saves other practitioners the same exploration. The failure is architectural—not a configuration error—and affects any GPTQ-quantized MoE model with small-output gate layers under the library versions used (GPTQModel 0.9.x, Unsloth 2024.12). GPTQModel v5.8.0 introduces BACKEND.AUTO_TRAINABLE and a training-capable TorchQuantLinear kernel; a synthetic gate backward pass succeeded in that version, suggesting the library constraint may now be resolvable without ZO (see Section III-C). The ZO approach remains the validated path; whether the corrected load path eliminates the crash on the full model requires a dedicated end-to-end validation run that was not performed here.

Cyclic rotation exploits the nightly cadence. VRAM is the binding constraint for single-GPU training, but time is abundant in a nightly pipeline. Trading 7 calendar nights for full architecture coverage at full rank avoids the compromises of rank reduction or CPU offloading. The coprime dual-rotation with loss functions prevents method-module coupling.

Loss function diversity matters. The variance across loss functions in Table VII (from +0.0033 for DPO to −0.0100 for RPO) demonstrates that the choice of loss function has meaningful impact on single-session behavior, justifying the rotation strategy regardless of convergence. A system committed to a single loss function risks consistent nega-

tive training on data distributions where that function's gradient signal degenerates.

Limitations. (1) Convergence was not achieved: 10 cycles at 24 DPO pairs produced no measurable improvement. The data requirement for ZO preference optimization at this scale is uncharacterized. (2) The ZO gradient estimates are inherently noisier than first-order gradients, and MeZO variance bounds do not transfer directly to the sparse MoE activation regime (see Section II). (3) The 55-pair evaluation set provides limited statistical power: minimum detectable effect is ≈14 percentage points at $\alpha = 0.05$, 80% power. Small improvements (2–5 pp) remain undetectable; full statistical sensitivity requires 200+ pairs. (4) The 8 ZO techniques were not individually ablated; their relative contributions and potential interactions are unknown. (5) The self-judge circular scoring dynamics are uncharacterized over long training horizons. (6) N=1 evaluation: single model, single GPU, single developer. (7) Cyclic module rotation assumes independently-trained adapter components interact productively when merged; potential interference between modules trained on different nights remains theoretically uncharacterized.

IX. Conclusion

We presented a zeroth-order preference optimization pipeline for 100B+ quantized MoE models on consumer hardware, framed as an infrastructure report rather than a convergence result. The paper's contributions are: (a) documenting the GPTQ MoE gate dequantization crash across 28 first-order attempts, establishing its architectural root cause and listing alternative approaches that may resolve it (GPTQModel v5.8.0's AUTO_TRAINABLE backend passed a synthetic gate backward test but was not validated end-to-end on the full model); (b) demonstrating that ZO preference optimization via an external inference API is mechanically functional at 122B scale with 10 loss functions and 8 ZO techniques; and (c) the cyclic LoRA module rotation formalism that achieves full architecture coverage over a 7-night cycle within a single 96GB GPU's VRAM budget.

We explicitly report that convergence was not observed. Ten training cycles at 24 preference pairs (training set; the eval set has since been expanded to 55 held-out pairs) produced preference accuracy of 60% ±0% throughout, with no statistically significant improvement. We attribute this to insufficient training data and identify 200+ preference pairs as a minimum threshold for future experiments. The minimum viable dataset size for ZO preference optimization on sparse MoE models at this scale is an open question that we leave to future work.

The infrastructure—mechanically verified, zero-downtime, completing in 21 minutes 43 seconds—is the deliverable. Whether it produces convergence at scale remains to be demonstrated.

References

[1] S. Malladi et al., "Fine-Tuning Language Models with Just Forward Passes," NeurIPS 2023, arXiv:2305.17333.
[2] Y. Liu et al., "Sparse MeZO: Memory-Efficient Zeroth-Order Optimization via Sparse Perturbation," arXiv:2402.15751, 2024.
[3] Y. Zhao et al., "HiZOO: Hessian-Informed Zeroth-Order Optimization for LLMs," ICLR 2025, arXiv:2402.15173.
[4] Y. Wang et al., "ZO2: Scalable Zeroth-Order Fine-Tuning for Extremely Large Language Models," arXiv:2503.12668, 2025.
[5] "QuZO: Quantized Zeroth-Order Fine-Tuning for Large Language Models," arXiv:2502.12346, 2025.
[6] R. Rafailov et al., "Direct Preference Optimization," NeurIPS 2023.
[7] Y. Meng et al., "SimPO: Simple Preference Optimization with a Reference-Free Reward," NeurIPS 2024.
[8] Z. Shao et al., "DeepSeekMath: Pushing the Limits of Mathematical Reasoning," 2024.
[9] J. Hong et al., "ORPO: Monolithic Preference Optimization without Reference Model," EMNLP 2024.
[10] K. Ethayarajh et al., "KTO: Model Alignment as Prospect Theoretic Optimization," ICML 2024.
[11] H. Xu et al., "Contrastive Preference Optimization," arXiv:2401.08417, 2024.
[12] "2-GRPO: GRPO is Secretly DPO," arXiv:2510.00977, 2025.
[13] "DAPO: An Open-Source LLM Reinforcement Learning System," arXiv:2503.14476, 2025.
[14] "AERO: Softmax-Free Online RLHF," arXiv:2602.14338, 2026.
[15] S. Liu et al., "DoRA: Weight-Decomposed Low-Rank Adaptation," arXiv:2402.09353, 2024.
[16] W. Xiong et al., "A Minimalist Approach to LLM Reasoning: from Rejection Sampling to Reinforce," arXiv:2504.11343, 2025.