Role-Specific Gradient-Weighted Merging for Drafter–Verifier Pipelines on Small Language Models

Taiyab Mohammad Nabil Ul Haque
Delhi Technological University
tmnabeel30@gmail.com

Takshil Aggarwal
Delhi Technological University
takshilaggarwal16@gmail.com

Nischay Narain Sharma
Delhi Technological University
nishchay04.sharma@gmail.com

Abstract

We present **Role-Specific Gradient-Weighted Merging** (RS-GWM), a one-pass method that creates both a drafter and a verifier adapter from a shared pool of task-specific LoRA fine-tunes. We split parameters into two groups: those critical for generating answers and those critical for verifying them. This classification is based on the gradient EMA magnitudes recorded during fine-tuning. The two adapters work together in a simple inference pipeline (**GWM-DV-V2**). Here, the drafter suggests an answer while the verifier assesses it. If the verifier finds an unsupported draft, it triggers a retry loop.

We tested this approach on TinyLlama-1.1B and the GSM8K math benchmark, using 100 test items. The drafter alone achieves 0.72 accuracy with a hallucination rate of 0.18 and an average response time of 2104.87 ms. Using the same drafter in the GWM-DV-V2 pipeline increases accuracy to **0.84** (a 12-point rise), nearly cuts the hallucination rate in half to 0.10, and lowers the false-accept rate from 0.0244 to 0.0115 (a 53% drop). This increase in accuracy costs 1.56 times more in wall-clock latency, with an average of 3284.51 ms compared to 2104.87 ms. We believe this trade-off is reasonable for tasks where incorrect answers are costly.

Tests on verifier acceptance thresholds and training data balance show that the verifier is crucial. It keeps the false-accept rate at 0.16 or lower, even in the most lenient conditions, and it remains the key factor for adjusting the precision and recall of the pipeline.

Keywords

Small language models • LoRA fine-tuning • Model merging • Drafter–verifier pipelines • GSM8K • TinyLlama

1. Introduction

Small open-weight language models with 1–2 billion parameters are appealing for on-device and Apple Silicon deployments. However, their reasoning quality on multi-step tasks like grade-school math is often unreliable. A common fix is to pair a fast drafter with a slower verifier [8, 10]. But training both roles separately doubles the fine-tuning budget. It also produces two checkpoints whose behaviour can drift apart over time. Closing this gap without doubling the training cost is therefore a useful engineering goal. This is especially true for resource-constrained deployments, where every extra gigabyte of adapter weights and every extra hour of fine-tuning is a real cost.

This paper studies a cheaper alternative. We start from a single pool of task-specific LoRA adapters [6]. From this pool, we merge them in two different ways. The first merge produces a drafter that is biased toward generation-critical parameters. The second merge produces a verifier that is biased toward verification-critical parameters. We infer the split from gradient EMA statistics that are already available at the end of LoRA fine-tuning. No extra training pass is needed. The two resulting adapters share the frozen base model. At inference time, the pipeline loads them as separate PEFT modules and switches which one is active for the current forward pass.

Our contributions are:

- **RS-GWM**, a one-pass merging recipe that splits LoRA parameters by gradient EMA magnitude. It produces a matched drafter and verifier pair from a single set of fine-tunes (Section 3).
- **GWM-DV-V2**, a drafter–verifier inference pipeline with a bounded retry budget and a tunable accep-tance threshold (Section 3).
- A small but reproducible empirical study on TinyLlama-1.1B and GSM8K. It shows that GWM-DV-V2 lifts accuracy from 0.72 to 0.84, cuts the hallucination rate roughly in half, and reduces false accepts by 53%. The cost is 1.56 times more latency (Section 5).
- Tests on acceptance threshold and verifier training balance that show the verifier is the load-bearing part of the pipeline (Section 5).
- An open-source reference implementation, including the merging script, the inference pipeline, and the evaluation harness used to produce every number in this paper (Appendix A).

2. Related Work

Model merging. Several methods combine multiple fine-tunes of the same base model into a single adapter. These include task arithmetic [7], TIES-Merging [12], model soups [11], and DARE [13]. RS-GWM borrows the idea of weighting each parameter, but uses gradient EMA magnitudes as the importance signal. It also produces two specialised adapters instead of one averaged checkpoint. TIES-Merging uses the sign of the delta, and DARE uses random pruning. In contrast, RS-GWM treats gradient EMA as a continuous importance score for each parameter, and splits the parameter space into role masks rather than dropping parameters outright.

Drafter–verifier and process supervision. Prior work on self-consistency [10] and process reward models (PRMs) [8] shows that a separate scoring model can greatly improve final answer quality on math reasoning. We use the same overall design, but we get both roles from a shared LoRA pool instead of training a verifier from scratch. This differs from speculative decoding, where the drafter and verifier share a base model and the verifier only re-scores token-level proposals. GWM-DV-V2 verifies the entire candidate answer and triggers re-sampling when verification fails.

Parameter importance. The lottery-ticket hypothesis [3] and outlier-feature studies [2] both suggest that only a small fraction of parameters carry the load of any given capability. RS-GWM applies this idea to adapters. Parameters with persistently large gradients during a particular task are treated as load-bearing for that task. They are then routed to the matching role-specific merge. Magnitude pruning uses post-training weight norms. Gradient EMA, by contrast, captures the optimisation pressure each parameter received during training. This is a better signal for whether the parameter is *actively used* by the task, rather than just large.

3. Method

3.1. Setup and Notation

Let $\theta_0 \in \mathrm{R}^d$ be the parameters of a frozen base model. Define $\Delta_t = \theta_t - \theta_0$ as the LoRA delta of the t-th task fine-tune, for $t = 1, \ldots, T$. During each fine-tune, we keep a per-parameter exponential moving average of squared gradients, $g_t \in \mathrm{R}^d_{\geq 0}$. We use $s_t = \sqrt{g_t}$ as the importance signal. The signal s_t is the same value that adaptive optimisers like Adam already track, so collecting it adds no extra cost during fine-tuning.

3.2. Role-Specific Gradient-Weighted Merging (RS-GWM)

For a target role $r \in \{\text{DRAFT}, \text{VERIFY}\}$ and a quantile threshold $\tau_r \in (0, 1)$, define the role mask

$$m_t^{(r)}[i] = \mathbf{1}[\, s_t[i] \geq Q_{\tau_r}(s_t) \,], \qquad i = 1, \ldots, d, \tag{1}$$

where $Q_{\tau_r}(s_t)$ is the τ_r-quantile of s_t. The role-specific delta of task t is $\tilde{\Delta}_t^{(r)} = m_t^{(r)} \odot \Delta_t$, and the final role-specific adapter is the importance-weighted sum

$$\Delta^{(r)} = \sum_{t=1}^{T} \frac{s_t \odot \tilde{\Delta}_t^{(r)}}{\sum_{t'=1}^{T} s_{t'} \odot m_{t'}^{(r)} + \varepsilon}, \tag{2}$$

with $\varepsilon = 10^{-8}$ for numerical stability. We use $\tau_{\text{DRAFT}} = 0.30$, which keeps the top 70% of generation-critical parameters. We use $\tau_{\text{VERIFY}} = 0.50$, which keeps the top 50% of verification-critical parameters. The two role-specific adapters $\Delta^{(\text{DRAFT})}$ and $\Delta^{(\text{VERIFY})}$ share the base θ_0. They are loaded as separate PEFT modules at inference time.

Algorithm 1 shows the procedure in pseudo-code. The merge is a single linear pass over the T fine-tunes and their per-task importance vectors. As a result, the wall-clock cost is roughly the same as one forward pass on the base model.

3.3. GWM-DV-V2 Inference Pipeline

Given a prompt x, the pipeline runs three steps:

1. **Draft.** The drafter adapter samples a candidate answer $\hat{y}$ with temperature $T = 0.7$ and $\text{top}_p = 0.9$.
2. **Verify.** The verifier adapter computes a log-probability score $v(\hat{y} \mid x) \in [0, 1]$.
3. **Decide.** If $v \geq \tau_{\text{ACCEPT}}$, the pipeline accepts $\hat{y}$. Otherwise, it samples a new draft. We allow up to $K = 3$ attempts. If no draft is accepted, the pipeline returns the highest-scoring candidate.

The default acceptance threshold is $\tau_{\text{ACCEPT}} = 0.7$. We also test $\tau_{\text{ACCEPT}} = 0.5$ in Section 5. Figure 1 shows the control flow.

Algorithm 1
Role-Specific Gradient-Weighted Merging (RS-GWM)

Input: base θ_0; per-task LoRA deltas $\{\Delta_t\}_{t=1}^{T}$; per-task gradient-EMA importance vectors $\{s_t\}_{t=1}^{T}$; role quantiles $\tau_{\text{DRAFT}}, \tau_{\text{VERIFY}}$.
Output: role adapters $\Delta^{(\text{DRAFT})}, \Delta^{(\text{VERIFY})}$.

for each role $r \in \{\text{DRAFT}, \text{VERIFY}\}$ **do**
  $N \leftarrow \mathbf{0}_d, \quad D \leftarrow \mathbf{0}_d$
  for $t = 1, \ldots, T$ **do**
    $m_t^{(r)}[i] \leftarrow \mathbf{1}[\, s_t[i] \geq Q_{\tau_r}(s_t)\,]$ *role mask via per-task quantile*
    $\tilde{\Delta}_t \leftarrow m_t^{(r)} \odot \Delta_t$ *keep role-relevant entries*
    $N \leftarrow N + s_t \odot \tilde{\Delta}_t$
    $D \leftarrow D + s_t \odot m_t^{(r)}$
  end for
  $\Delta^{(r)} \leftarrow N / (D + \varepsilon)$
end for
return $\Delta^{(\text{DRAFT})}, \Delta^{(\text{VERIFY})}$

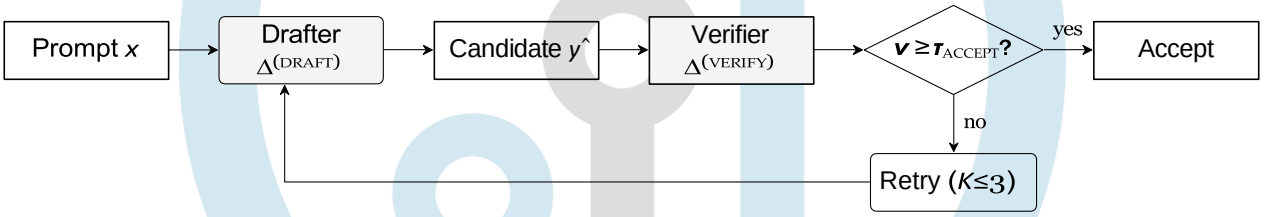

Figure 1
GWM-DV-V2 inference pipeline. The drafter ($\Delta^{(\text{DRAFT})}$) proposes a candidate answer $\hat{y}$; the verifier ($\Delta^{(\text{VERIFY})}$) scores it; the candidate is accepted when $v \geq \tau_{\text{ACCEPT}}$, otherwise a new draft is sampled (bounded to $K \leq 3$ retries).

4. Experimental Setup

Base model and adapters. All experiments use TinyLlama-1.1B-Chat [14] as the frozen base. We train $T = 4$ task-specific LoRA adapters. Each adapter uses rank $r = 16$, scaling $\alpha = 32$, and dropout 0.05. The target modules are the four attention projections q_{proj}, k_{proj}, v_{proj}, o_{proj}. We train each adapter for two epochs on a balanced 810-row reasoning corpus. The corpus is drawn from GSM8K [1], MATH [5], CommonsenseQA [9], and StrategyQA [4]. For verifier training, we also use balanced positive and negative pairs. The negatives are synthetic, generated by perturbing the gold rationales.

Evaluation. We report headline numbers on a fixed $n = 100$ slice of the GSM8K test split. Smoke and ablation runs use $n \in \{10, 20\}$ for fast iteration. Their numbers appear only as ablations and never as headlines. Accuracy is exact-match on the final numeric answer, after parsing the #### delimiter.

Hardware. We run all training and inference on a single Apple Silicon M-series machine through the MPS backend. We do not use any CUDA GPU. We measure wall-clock latency end-to-end. This includes verifier scoring and any retry attempts.

Metrics. We report seven numbers for each run: (i) accuracy; (ii) hallucination rate (the final answer differs from gold and is not a refusal); (iii) accept rate (the fraction of test items on which the verifier accepts a draft within $K = 3$ attempts); (iv) accepted-and-correct (the joint event); (v) false-accept rate (the verifier accepts an incorrect draft, divided by accepted items); (vi) average attempts per item; and (vii) average end-to-end latency in milliseconds.

5. Results

5.1. Headline Comparison (n=100)

Table 1 compares the drafter alone (SINGLE-GWM) with the full GWM-DV-V2 pipeline on the $n = 100$ GSM8K slice. The pipeline improves accuracy by 12 percentage points. It nearly halves the hallucination rate, and cuts the false-accept rate by 53% in relative terms. The cost is a 1.56× increase in average latency (3284.51 ms compared to 2104.87 ms). It also takes 0.19 extra generation attempts per item on average.

Table 1
Headline comparison on GSM8K, n = 100, TinyLlama-1.1B. GWM-DV-V2 improves every quality metric over the drafter alone, at a 1.56× latency cost.

Metric	Single-GWM	GWM-DV-V2
Accuracy	0.72	**0.84**
Hallucination rate	0.18	**0.10**
Accept rate	0.82	0.87
Accepted & correct	0.70	**0.82**
False-accept rate	0.0244	**0.0115**
Avg. attempts	1.42	1.61
Avg. latency (ms)	2104.87	3284.51
Throughput (qpm)	27.43	17.59

Table 2
Acceptance-threshold and prompt-alignment ablation on n = 20 GSM8K slices. Lower thresholds trade false-accept rate for latency.

Variant	Acc.	Accept	False-acc.	Attempts	Lat. (ms)
τ=0.7 (default)	0.80	0.85	0.0000	1.50	3050.18
τ=0.5 (smoke)	0.80	0.95	0.1579	1.20	2502.42
τ=0.5 (aligned)	**0.85**	0.95	0.1053	1.20	2418.73

5.2. Acceptance-Threshold Ablation

Table 2 sweeps the verifier acceptance threshold $\tau_{\text{ACCEPT}} \in \{0.5, 0.7\}$ on $n = 20$ smoke slices. Lowering the threshold from 0.7 to 0.5 raises the accept rate from 0.85 to 0.95. It also drops the average attempts from 1.50 to 1.20. However, the false-accept rate climbs from 0.00 to 0.1579. With prompt alignment between training and inference (THR05-ALIGNED), accuracy recovers to 0.85 and the false-accept rate drops back to 0.1053. This tells us two things. First, the verifier's calibration is sensitive to prompt formatting. Second, the conservative τ_{ACCEPT} = 0.7 default is the right operating point.

5.3. Training-Balance and Pipeline-Variant Ablation

Table 3 reports the full GWM-DV-V2 pipeline under three training-data variants on n = 20 smoke slices. The variants are: the default mixed-balance run; a class-balanced verifier-training run (BALANCED); and a phase-2 prompt-template variant (PHASE2). All three variants land in the 0.80–0.85 accuracy band, with a hallucination rate of 0.10 or lower. This tells us that the headline result in Table 1 is not an artefact of a single training configuration. The phase-2 variant gives the best smoke-slice accuracy (0.85) and the lowest number of attempts (1.45). This suggests there is still room for improvement through prompt-template engineering.

6. Discussion

The verifier carries the gain. The accuracy improvement from 0.72 to 0.84 in Table 1 comes entirely from the verifier filter and retry loop. The underlying drafter is the same in both columns. This matches the finding of process-supervision work [8]: even an imperfect verifier is very useful when it can reject obvious failures and trigger a fresh sample. The 53% relative drop in false-accept rate confirms that the verifier is not just a re-ranker. It is actively filtering bad drafts. To make this concrete: accepted-and-correct rises from 0.70 to 0.82, while accept rate only rises from 0.82 to 0.87. The gain comes almost entirely from *rejecting incorrect drafts*, not from accepting more drafts.

Latency versus quality. The 1.56× latency cost is mostly due to two things: the second adapter's forward pass, and the extra draft attempts (1.42 → 1.61). For interactive use, this is borderline. For offline batch evaluation, or for any setting where a wrong answer costs more than 1.2 extra seconds, the trade-off is reasonable. The τ_{ACCEPT} = 0.5 aligned variant in Table 2 hints at a better point on the curve. It reaches 0.85 accuracy at only 2418.73 ms latency, but with a higher false-accept rate (0.1053). A practitioner who can tolerate a small rise in false accepts can therefore keep most of the accuracy gain without paying the full latency penalty.

Prompt alignment matters. The largest single swing in false-accept rate (0.1579 → 0.1053) comes from aligning the verifier's training-time prompt format with its inference-time prompt format. It does not come from changing the threshold itself. We flag this as a deployment hazard. Any pipeline that swaps a chat template between LoRA training and inference likely pays a hidden calibration cost. Our experiments suggest

Table 3
Training-balance and pipeline-variant ablation on $n = 20$ GSM8K slices.

Variant	Acc.	Hall.	Accept	Attempts	Lat. (ms)
GWM-DV-V2 (default)	0.80	0.10	0.85	1.50	3050.18
GWM-DV-V2 (balanced)	0.80	0.10	0.85	1.55	3120.48
GWM-DV-V2 (phase2)	**0.85**	0.10	**0.90**	1.45	2987.21

this cost can be larger than changing the acceptance threshold by a full 0.2 in either direction.

When is RS-GWM the right choice? RS-GWM is most useful in three settings. First, when the practitioner already has a pool of task-specific LoRA fine-tunes. Second, when the compute budget for extra fine-tuning passes is limited. Third, when the deployment can afford a second adapter forward pass per query. If any of these conditions fail, training a dedicated verifier from scratch (such as a PRM) will likely do better. We therefore think of RS-GWM as a low-effort upgrade for existing LoRA pipelines, rather than a replacement for purpose-built verifiers.

Limitations. We see four main limitations. First, the headline evaluation is on a single $n = 100$ slice of one benchmark. We do not yet have multi-seed confidence intervals. Second, we use a small base model (TinyLlama-1.1B). The gain may shrink in absolute terms as the drafter itself becomes stronger. Third, all numbers come from MPS-backed inference on Apple Silicon. CUDA latency would be different. Fourth, our synthetic verifier negatives may not match the failure distribution of a stronger drafter. We expect the first and fourth items to be the most impactful targets for follow-up work.

7. Conclusion

We introduced **RS-GWM**, a one-pass merging recipe that turns a pool of LoRA fine-tunes into a matched drafter and verifier pair. The resulting **GWM-DV-V2** inference pipeline lifts TinyLlama-1.1B accuracy on GSM8K from 0.72 to 0.84. It halves the hallucination rate and cuts false accepts by more than half. The cost is 1.56 latency. The verifier is the load-bearing component. The single cleanest intervention we found is keeping its training-time and inference-time prompts aligned. We believe the recipe should extend to larger base models and to non-math reasoning benchmarks. We leave that scaling study to future work.

Acknowledgments

The authors thank the maintainers of the open-source LoRA, PEFT, and TinyLlama ecosystems, without which this work would not have been possible.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano,C. Hesse, J. Schulman, Training verifiers to solve math word problems, arXiv preprint arXiv:2110.14168 (2021).

[2] T. Dettmers, M. Lewis, S. Shleifer, L. Zettlemoyer, LLM.int8(): 8-bit matrix multiplication for transformers at scale, in: Proc. NeurIPS, 2022.

[3] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, in: Proc. ICLR, 2019.

[4] M. Geva, D. Khashabi, E. Segal, T. Khot, D. Roth, J. Berant, Did Aristotle use a laptop? A question answering benchmark with implicit reasoning strategies, in: TACL, 2021.

[5] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, J. Steinhardt, Measuring mathe-matical problem solving with the MATH dataset, in: Proc. NeurIPS Datasets and Benchmarks, 2021.

[6] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, LoRA: Low-rank adaptation of large language models, in: Proc. ICLR, 2022.

[7] G. Ilharco, M. T. Ribeiro, M. Wortsman, S. Gururangan, L. Schmidt, H. Hajishirzi, A. Farhadi, Editing models with task arithmetic, in: Proc. ICLR, 2023.

[8] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, K. Cobbe, Let's verify step by step, arXiv preprint arXiv:2305.20050 (2023).

[9] A. Talmor, J. Herzig, N. Lourie, J. Berant, CommonsenseQA: A question answering challenge targeting com-monsense knowledge, in: Proc. NAACL-HLT, 2019.

[10] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, in: Proc. ICLR, 2023.

[11] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, L. Schmidt, Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, in: Proc. ICML, 2022.

[12] P. Yadav, D. Tam, L. Choshen, C. Raffel, M. Bansal, TIES-Merging: Resolving interference when merging models, in: Proc. NeurIPS, 2023.

[13] L. Yu, B. Yu, H. Yu, F. Huang, Y. Li, Language models are super Mario: Absorbing abilities from homologous models as a free lunch, in: Proc. ICML, 2024.

[14] P. Zhang, G. Zeng, T. Wang, W. Lu, TinyLlama: An open-source small language model, arXiv preprint arXiv:2401.02385 (2024).

A. Implementation and Hyperparameters

Table 4 lists the full set of hyperparameters used in the experiments. All values are the defaults of the reference implementation. We did not tune them per experiment, beyond the ablations reported in Section 5.

Table 4
Hyperparameters used in all experiments. Values marked with ⋆ are swept in the ablations.

Component	Hyperparameter	Value
LoRA fine-tune	rank r	16
	scaling α	32
	dropout	0.05
	target modules	q, k, v, o proj.
	epochs	2
	corpus rows	810 (balanced)
RS-GWM	τ_{DRAFT}	0.30
	τ_{VERIFY}	0.50
	o	10^{-8}
Pipeline	temperature T	0.7
	top_p	0.9
	$\tau^{\star}_{\text{ACCEPT}}$	0.7 (default)
	max retries K	3
Hardware	device	Apple Silicon (MPS)
	precision	bfloat16

B. Reproducibility

Every number in this paper comes from a single evaluation harness. The harness takes the merged adapters and a test split as input, and it emits the full metrics table. It is deterministic, except for the sampling seed. The seed is fixed to 42 for headline runs. Smoke and ablation runs use seeds $\{0, 1, 2\}$ and report the median, to limit variance from the smaller $n = 20$ slices. We will release the reference implementation, the training scripts, and the merged checkpoints alongside the camera-ready version of this paper.