PMOPD: TASK ORDERING, CYCLING, AND PARAMETER-UPDATE SUBSPACE PROTECTION IN MULTI-TEACHER ON-POLICY DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-teacher on-policy distillation (MOPD) has emerged as a popular post-training paradigm for integrating specialized capabilities in frontier language models. Existing OPD research has primarily focused on optimizing single-task distillation through objective design, distillation scope, and teacher signal construction, whereas MOPD must aggregate multiple capabilities in shared parameters and address the resulting capability seesaw, in which improving one domain suppresses capabilities acquired from another. Inspired by the distinctive update geometry of OPD, we find that parameter updates from different tasks rapidly concentrate in their respective low-dimensional subspaces during MOPD, providing a direct geometric basis for identifying and controlling cross-task interference. We therefore propose PMOPD (Projection-based Multi-Teacher On-Policy Distillation), which constructs subspace memories from the cumulative parameter displacements of different tasks and projects both gradients and optimizer updates to remove components that interfere with protected task directions. We further develop a lightweight conflict probe to characterize task interactions and guide task ordering, together with a cycling strategy that balances subspace estimation and timely task revisitation. Experiments on representative Code, Reason, and Math tasks show that PMOPD improves every evaluated capability over MOPD, raising the average score across the three tasks by 2.54 points on Qwen2.5-7B and 2.09 points on Llama-3.1-8B. These consistent gains establish geometry-aware optimization as an effective and transferable approach to balanced multi-teacher distillation.

1 INTRODUCTION

Multi-teacher on-policy distillation (MOPD) is increasingly used as a post-training paradigm for integrating specialized capabilities into a single language model (Ma et al., 2026; DeepSeek-AI, 2026; Xiaomi LLM-Core Team, 2026; Kimi Team, 2026). Starting from a shared base model, separate teachers can be optimized for domains such as mathematics, reasoning, and code, after which their behaviors are distilled into one deployable student. Compared with maintaining an ensemble of specialists, MOPD avoids inference-time routing and multi-model serving costs while retaining on-policy supervision: teachers provide token-level targets on trajectories sampled from the student itself (Agarwal et al., 2024; Ma et al., 2026).

Most existing OPD studies, however, optimize distillation in single-task settings through objective design, distillation scope, and teacher signal construction (Li et al., 2026; Gao et al., 2026). MOPD introduces a distinct optimization challenge: multiple teachers act on the same student parameters, and their objectives need not be compatible. Consequently, improving one capability can suppress another, producing a capability seesaw (Ma et al., 2026; Gao et al., 2026). Standard task mixing reduces the interval between domains but neither identifies conflicting update directions nor protects useful changes already induced by earlier teachers. MOPD therefore requires the student to balance stability and plasticity by preserving acquired capabilities while retaining enough freedom to absorb new ones.

We approach this problem through the update geometry of OPD. Instead of treating each noisy mini-batch gradient independently, we examine the cumulative parameter displacement over a task

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

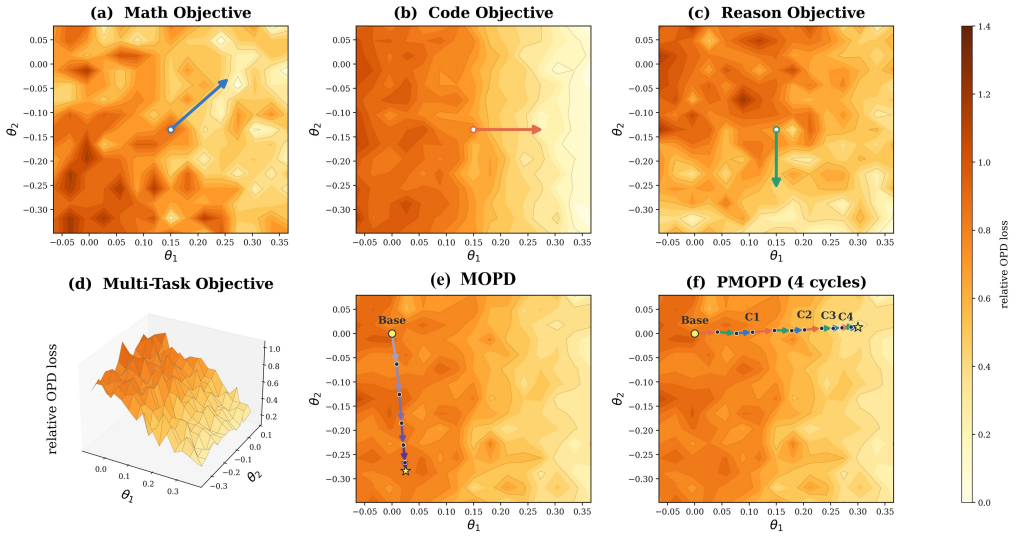


Figure 1: Visualization of task-preferred update directions and the optimization trajectories of MOPD and PMOPD. The axes θ_1 and θ_2 define a rank-2 PCA plane derived from actual checkpoint displacements. (a-c) show the normalized Math, Code, and Reason OPD losses. The arrows indicate their preferred next-update directions from a shared reference checkpoint. (d) shows the equally weighted multi-task relative OPD loss, while (e-f) overlay the actual MOPD and four-cycle PMOPD trajectories on this joint landscape. Lighter colors indicate lower relative loss and darker colors indicate higher relative loss. The task-specific arrows differ substantially at the same checkpoint, and MOPD does not consistently descend on the joint landscape. By correcting updates against protected task directions, PMOPD follows a more stable trajectory toward a shared low-loss region.

block. Its dominant singular subspace stabilizes early, consistent with recent analyses of OPD update geometry (Shen et al., 2026). It also exhibits high consistency across independent shards of the same task and low overlap across tasks. This suggests that a compact subspace can summarize the main directions used by a teacher and that the overlapping components of later updates provide a tractable target for interference control without globally freezing the parameter space.

Based on this observation, we propose PMOPD (Projection-based Multi-Teacher On-Policy Distillation), which constructs per-matrix subspace memories from the cumulative parameter displacements of completed task blocks. When learning a subsequent task, it projects both the gradient and the preconditioned optimizer update to remove components that interfere with protected task directions. The second projection is necessary because the element-wise adaptive preconditioning used by optimizers such as Adafactor (Shazeer & Stern, 2018) can rotate an already projected gradient back toward the protected subspaces. Memories are rebuilt in each cycle, allowing the protected geometry to track the current trajectory without accumulating an unbounded archive.

Figure 1 provides a geometric view of this interference. At the same reference checkpoint, Math, Code, and Reason favor markedly different update directions, so an update that benefits one objective need not reduce the others. Correspondingly, the measured MOPD trajectory fluctuates on the joint loss landscape, whereas PMOPD more consistently enters a shared low-loss region by correcting updates against previously identified task-update subspaces.

MOPD also depends on temporal organization. Because pairwise interference is directional, we measure both directions and aggregate them with a lightweight conflict probe to rank tasks before training. We cycle through teachers to balance timely revisitation against reliable block-displacement estimates. In our setting, the probe selects Code→Reason→Math, and four cycles perform best.

Across Qwen2.5-7B and Llama-3.1-8B, PMOPD raises the average score across the three tasks over MOPD by 2.54 and 2.09 points, respectively, while improving every evaluated task in both model families. This uniform advantage shows that PMOPD strengthens joint capability integration instead of redistributing performance among domains.

Contributions. Our contributions are threefold:

- We characterize MOPD interference geometrically, showing that cumulative OPD updates form early-stabilizing, low-dimensional subspaces that are consistent within tasks and exhibit low cross-task similarity.
- We introduce PMOPD, which constructs subspace memories from realized task displacements and protects them by projecting both gradients and adaptive-optimizer updates.
- We develop a lightweight conflict probe and cycling strategy for organizing task updates, and demonstrate consistent improvements across two model families and three capability domains.

2 Related Work

2.1 Knowledge distillation and on-policy supervision

Knowledge distillation transfers teacher behavior through distribution matching, generated sequences, or policy supervision (Hinton et al., 2015; Kim & Rush, 2016; Rusu et al., 2015). OPD instead queries teachers on student trajectories and supplies token-level targets for visited prefixes, reducing the mismatch between training and inference states (Ross et al., 2011; Agarwal et al., 2024). Recent systems use multiple teachers to integrate domain specialists (Ma et al., 2026; DeepSeek-AI, 2026; Xiaomi LLM-Core Team, 2026; Kimi Team, 2026), while concurrent work studies OPD objectives and practical interference (Li et al., 2026; Gao et al., 2026). We study this interference through shared-parameter update geometry.

2.2 Multi-task optimization and capability integration

Conflicting multi-task gradients motivate projection, common-descent, and magnitude-balancing methods (Yu et al., 2020; Sener & Koltun, 2018; Chen et al., 2018). PCGrad removes pairwise conflicting components, multi-objective optimization seeks common descent directions, and GradNorm balances task scales. These methods operate on instantaneous gradients. In contrast, PMOPD stores realized OPD block displacements, including clipping and adaptive-optimization effects, to constrain later updates.

Prediction-time ensembles retain specialist models (Breiman, 1996; Lakshminarayanan et al., 2017; Huang et al., 2017), whereas model soups and task arithmetic combine independently trained parameters (Wortsman et al., 2022; Ilharco et al., 2023). Such parameter-space combinations can conceal conflicts until merging and do not expose the combined model to its generated states. PMOPD instead integrates capabilities under on-policy supervision and directly constrains the optimization path.

2.3 Continual learning and low-dimensional update geometry

Continual-learning methods preserve prior tasks through parameter penalties, episodic constraints, or activation-derived subspace projection (Kirkpatrick et al., 2017; Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2019; Saha et al., 2021). GPM extracts protected bases from activations. In contrast, PMOPD derives per-matrix bases from realized OPD block displacements, applies them to gradients and optimizer updates, and rebuilds memory each cycle.

Neural-network adaptation often lies in low-dimensional spaces, as exploited by intrinsic-dimension methods and LoRA (Aghajanyan et al., 2021; Hu et al., 2022). Cumulative OPD updates similarly exhibit early subspace locking (Shen et al., 2026). We further show strong same-task alignment and low cross-task overlap, making the overlapping directions a tractable target for interference control.

3 Problem Formulation and Geometric Motivation

3.1 Multi-teacher on-policy distillation

Let $\mathcal{T} = \{\text{Math}, \text{Reason}, \text{Code}\}$ denote the task set, D_t the prompt distribution for task t, π_θ the shared student, and π_T^t the corresponding specialist teacher. For $x \sim D_t$, the student samples a response $y \sim \pi_\theta(\cdot \mid x)$ and the teacher is evaluated on the same prefixes, following the standard

on-policy distillation setup (Agarwal et al., 2024; Ma et al., 2026). A multi-task run minimizes

$$\min_{\theta} \ \mathbb{E}_{t\sim q,\, x\sim D_t,\, y\sim\pi_\theta} \left[\mathcal{L}^t_{\mathrm{OPD}}(\theta; x, y)\right], \tag{1}$$

where q is induced by the task schedule.

3.2 Task-update subspaces

For a two-dimensional weight matrix W, define the accumulated block update and its singular value decomposition as

$$\Delta W^t = W^{\mathrm{after}} - W^{\mathrm{before}}, \qquad \Delta W^t = U^t \Sigma^t (V^t)^\top. \tag{2}$$

We retain the top-K right-singular vectors $V_{t,K}$. For two blocks a and b, we quantify subspace overlap by

$$S_K(a, b) = \frac{1}{K} \left\| V_K(a)^\top V_K(b) \right\|_F^2. \tag{3}$$

Following prior analysis of OPD update geometry (Shen et al., 2026), we set $K = 16$ to retain the dominant update directions in a compact basis with low memory and projection costs. With this setting, the subspaces extracted from cumulative displacements after 20% of training already attain an average similarity of 0.62 to their corresponding final subspaces across Math, Reason, and Code. This result shows that the dominant update subspace for each task emerges early and remains stable throughout OPD, enabling it to serve as a compact geometric representation of task-specific parameter updates.

We next compare the final task-update subspaces across domains. As shown in Table 1, the similarities among the top-16 update subspaces for Math, Code, and Reason range from 0.133 to 0.151, demonstrating that their parameter updates occupy substantially different subspaces. This strong cross-task separation provides the geometric basis for compactly protecting task-specific update directions, while the overlapping components identify the directions in which later task updates interfere with protected capabilities.

Table 1: Similarities among the top-16 OPD update subspaces induced by different training domains. The low cross-domain similarities support selective subspace protection.

	Math	Code	Reason
Math	1.000	0.151	0.142
Code	0.151	1.000	0.133
Reason	0.142	0.133	1.000

4 Method

4.1 Overview

Our method, PMOPD, performs multi-teacher OPD in ordered task blocks. Within each cycle, the first task is learned without a protection constraint and its realized parameter displacement is compressed into compact per-matrix memories. Later tasks are trained after removing the components of both their gradients and optimizer updates that lie in the accumulated memory. The memory is rebuilt from an empty state in every cycle, so it represents the update geometry induced by the current data shards rather than an ever-growing archive across cycles. Figure 2 illustrates the Code→Reason→Math progression within cycle N. The upper row follows the shared student through the three task stages, while the lower row shows how completed block displacements construct and expand the subspace memory used to project subsequent gradients. The core procedure contains three components: reverse-KL on-policy distillation, task-update subspace extraction, and dual projection of gradients and Adafactor updates. We select the task order and cycle count with lightweight diagnostics analyzed in Section 7.

4.2 Reverse-KL on-policy distillation

For a prompt x from task t, the student first samples a response $y \sim \pi_\theta(\cdot \mid x)$. The task teacher is then evaluated on the same student-generated prefixes $h_i = (x, y_{<i})$. The main experiments minimize

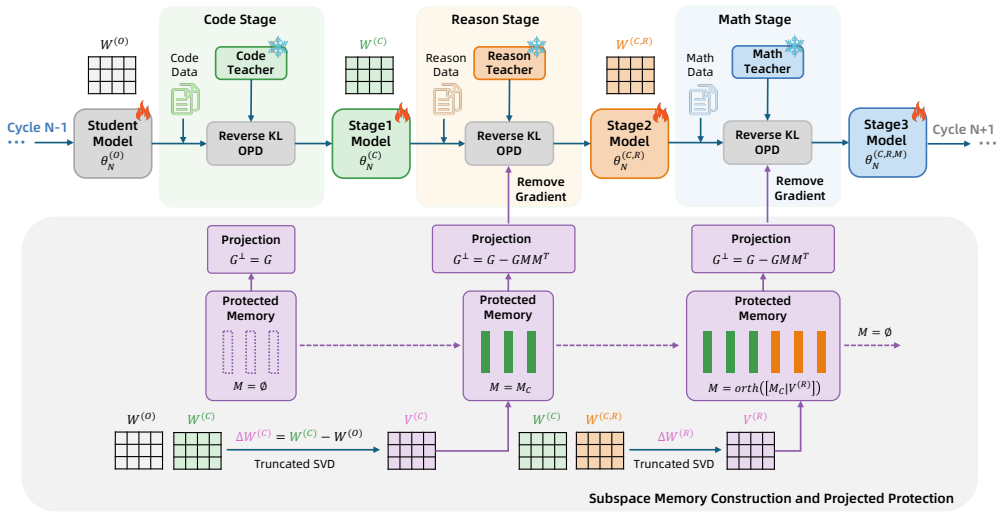


Figure 2: Overview of PMOPD in cycle N. The upper row shows sequential OPD across the Code, Reason, and Math stages, and the lower row shows subspace-memory construction and projected protection. Code is trained with an empty memory. Its block displacement constructs the first protected basis, which is expanded after Reason and used to constrain the Math stage. The final student initializes cycle $N{+}1$, where the memory is rebuilt from new task blocks.

reverse KL,

$$\mathcal{L}_{\mathrm{RKL}}^{t} = \frac{1}{|y|}\sum_{i=1}^{|y|} \mathrm{KL}\big(\pi_\theta(\cdot \mid h_i) \,\|\, \pi_T^t(\cdot \mid h_i)\big)\,, \tag{4}$$

where

$$\mathrm{KL}(\pi_\theta \| \pi_T^t) = \sum_{v\in\mathcal{V}} \pi_\theta(v \mid h_i) \log \frac{\pi_\theta(v \mid h_i)}{\pi_T^t(v \mid h_i)}. \tag{5}$$

Thus, every teacher supervises states actually visited by the shared student, while the task identity determines which specialized teacher supplies the target distribution.

4.3 Extracting Task-Update Subspaces

PMOPD performs subspace extraction and projection independently for every two-dimensional trainable parameter matrix, rather than constructing a single unified subspace for the entire model. To keep the notation concise, we describe the operation for one arbitrary matrix and omit its matrix index throughout this section.

Consider a trainable parameter $W \in \mathbb{R}^{m\times n}$. At the beginning and end of a task block, we record W^{before} and W^{after}, and form the realized displacement

$$\Delta W^t = W^{\mathrm{after}} - W^{\mathrm{before}}. \tag{6}$$

Unlike an individual mini-batch gradient, ΔW^t includes the cumulative effect of the task loss, gradient clipping, and the optimizer over the entire block. We approximate it with a randomized truncated SVD,

$$\Delta W^t \approx U_K^t \Sigma_K^t (V_K^t)^\top, \qquad K = 16. \tag{7}$$

The columns of $V_K^t \in \mathbb{R}^{n\times K}$ are the dominant input-side directions of the task-induced parameter change: $\Delta W^t v_i = \sigma_i u_i$. We use right rather than left singular vectors because the protection operation acts on the right side of the gradient and optimizer-update matrices.

4.4 Orthogonal Task-Subspace Memory

Let M denote the subspace memory associated with the current parameter matrix. After a remembered task block, its new directions are concatenated with the existing basis and orthogonalized by a reduced

QR decomposition,

$$A = [M \mid V_K^t] = QR, \qquad M \leftarrow Q[:, |\operatorname{diag}(R)| > \epsilon], \tag{8}$$

where ϵ is a small redundancy threshold. QR removes numerically redundant directions and ensures $M^\top M = I$. Consequently, $P = MM^\top$ is the orthogonal projector onto the protected input-side subspace.

4.5 Gradient and optimizer-update projection

For an unprojected gradient $G \in \mathbb{R}^{m\times n}$, we decompose

$$G_\parallel = GMM^\top, \qquad G_\perp = G - GMM^\top. \tag{9}$$

The backward gradient is replaced by $G_\perp$ before global gradient-norm clipping. Since $M^\top M = I$, the retained gradient satisfies

$$G_\perp M = (G - GMM^\top)M = 0. \tag{10}$$

Therefore, at the level of this linear map, the projected update does not directly change the transformation along inputs in $\operatorname{span}(M)$.

For a parameter matrix, we measure the fraction of its gradient aligned with the protected memory as

$$r_g = \frac{\|GMM^\top\|_F}{\|G\|_F}. \tag{11}$$

The global diagnostic reported in our experiments accumulates the squared numerator and denominator norms over all projected parameter matrices before taking their ratio.

Because Adafactor's adaptive preconditioning can change the direction of an already projected gradient, we additionally project the preconditioned optimizer update:

$$U_\perp = U - UMM^\top, \qquad W \leftarrow W - U_\perp, \tag{12}$$

where U is the raw Adafactor update after preconditioning.

5 Experimental Setup

5.1 Models, tasks, and teachers

We evaluate two independently developed base-model families: Qwen2.5-7B (Qwen Team, 2024) and Llama-3.1-8B (Grattafiori et al., 2024). For each family, the Math, Reason, and Code teachers begin from the same base checkpoint and are specialized for their respective domains. Specifically, each teacher is obtained by reinforcement learning (RL) on its domain data, with the training-set sizes reported in the RL column of Table 2. This shared initialization makes the integration setting controlled: teacher differences arise from domain-specific RL rather than unrelated pretraining histories.

The primary Qwen configuration is summarized in Table 2. Math uses the orca_math split of NuminaMath-CoT (Li et al., 2024), Reason uses CommonsenseQA (Talmor et al., 2019), and Code uses APPS (Hendrycks et al., 2021). Each task contributes 600 OPD prompts. Evaluation uses the test sets and counts listed in Table 2. Under the selected four-cycle schedule, each task block contains 150 prompts.

Table 2: Data configuration for the main experiments.

Task	RL	OPD	Test
Math	2700	600	450
Reason	4870	600	1221
Code	2400	600	300

5.2 Baselines and evaluation

We compare against: (i) the shared base model; (ii) each single-domain teacher; (iii) parameter merging, which averages compatible specialist parameters; (iv) MOPD, the reported multi-teacher OPD baseline that mixes examples from different tasks within each mini-batch (Ma et al., 2026); (v) BB-MOPD, which performs between-batch mixing by interleaving task-homogeneous mini-batches

Table 3: Results across the Qwen2.5-7B and Llama-3.1-8B families. Each group reports task-specific scores and their unweighted average. Bold values mark the best within each model family and column.

	Qwen2.5-7B				Llama-3.1-8B			
Model / method	Math	Reason	Code	Avg.	Math	Reason	Code	Avg.
Student Model	51.78	50.20	49.67	50.55	11.33	58.56	27.33	32.41
Math Teacher	66.89	56.84	49.67	57.80	18.22	57.08	27.33	34.21
Reason Teacher	58.89	70.35	50.00	59.75	8.89	68.55	24.33	33.92
Code Teacher	64.22	60.69	58.00	60.97	11.33	56.43	**42.00**	36.59
Parameter Merge	61.56	60.44	53.67	58.56	12.22	63.64	30.00	35.29
MOPD	65.56	71.74	56.00	64.43	18.22	68.63	30.00	38.95
BB-MOPD	65.33	70.68	55.67	63.89	15.33	64.21	29.33	36.29
Open-MOPD	66.44	70.52	53.67	63.54	14.89	65.52	25.00	35.14
PMOPD	**67.78**	**73.46**	**59.67**	**66.97**	**21.33**	**69.45**	32.33	**41.04**

while updating the same shared student; and (vi) Open-MOPD (Gao et al., 2026), which we reproduce on our model families and task datasets using its proposed capability-balancing method. Details and variants of our MOPD implementation are deferred to Appendix A.3. All OPD comparisons in the main tables use the reverse-KL objective and the same total number of task prompts. We report the task-specific percentage score produced by the same evaluation pipeline for every method and their unweighted average across the three tasks. PMOPD uses a Code→Reason→Math schedule with four cycles and a batch size of 16. The reported PMOPD results are averaged over three independent runs with different random seeds.

6 Main Results

6.1 Qwen2.5-7B

Table 3 gives the primary Qwen2.5-7B result. PMOPD achieves the best score in every task column and raises the average score across the three tasks to 66.97, outperforming MOPD by 2.54 points and BB-MOPD by 3.08 points. Relative to MOPD, it gains 2.22 points on Math, 1.72 on Reason, and 3.67 on Code. These across-the-board gains demonstrate stronger integration of all three capabilities in a single student.

6.2 Llama-3.1-8B

The Llama-3.1-8B columns in Table 3 establish transfer across model families. PMOPD achieves the best integrated-model average of 41.04, outperforming MOPD by 2.09 points and BB-MOPD by 4.75 points. Relative to MOPD, it improves Math by 3.11 points, Reason by 0.82 points, and Code by 2.33 points. Improvements on every task confirm that the ordering and projection rule generalize beyond the Qwen configuration and strengthen balanced capability integration across model families.

6.3 Projection ablation

We conduct a controlled ablation to separate the effects of the two projection stages from the shared training schedule. All variants use the same Code→Reason→Math order, four cycles, task data, and optimization budget. As shown in Table 4, gradient projection raises the average from 63.85 to 66.22, with particularly clear gains on Reason and Code, showing that removing components aligned with protected task directions substantially mitigates cross-task interference. Projecting the preconditioned optimizer update further improves the average to 66.97 and yields the strongest Math and Code results. Overall, the complete PMOPD pipeline outperforms the

Table 4: Projection ablation on Qwen2.5-7B under the same Code→Reason→Math order and four-cycle schedule. The variants isolate the contributions of gradient projection and optimizer-update projection.

Variant	Math	Reason	Code	Avg.
No projection	67.33	68.55	55.67	63.85
+ Gradient projection	67.11	**73.87**	57.67	66.22
+ Optimizer-update projection	**67.78**	73.46	**59.67**	**66.97**

matched no-projection variant by 3.12 points, supporting the complementary roles of gradient-space protection and optimizer-update correction.

7 Analysis

We derive lightweight geometric diagnostics for selecting the two principal scheduling choices of PMOPD: task order and the number of cycles. Exhaustive search over either choice becomes increasingly costly as the number of tasks or the training scale grows. We therefore connect aggregate evaluation performance to lightweight geometric diagnostics and use them to provide practical, low-cost guidance for selecting an order and cycle count before a full MOPD run.

7.1 Selecting task order by average undirected conflict

The three tasks can be arranged in six possible orders. We evaluate all six under the same one-cycle training and data budget, with the results reported in Table 5. Their averages span 2.97 points, demonstrating that task order materially affects multi-task integration. Code→Reason→Math (C→R→M) achieves the highest average of 65.81. To explain why this order is preferable and to avoid exhaustive enumeration in larger task sets, we introduce an average undirected conflict score.

Table 5: One-cycle performance of all six task orders. C, R, and M denote Code, Reason, and Math, respectively.

Order	Math	Reason	Code	Avg.
C→M→R	65.78	74.04	55.67	65.16
M→R→C	69.11	70.68	55.33	65.04
C→R→M	70.22	73.87	53.33	**65.81**
M→C→R	66.22	69.70	54.67	63.53
R→C→M	65.11	72.32	56.67	64.70
R→M→C	64.22	72.97	51.33	62.84

For a memory M_a extracted from task a and a gradient G_b measured on task b, we first define the directional conflict

$$r_{a\to b} = \frac{\|G_b M_a M_a^\top\|_F}{\|G_b\|_F}. \tag{13}$$

Because this quantity is asymmetric, we symmetrize each task pair and average its conflicts with the remaining tasks:

$$c(a,b) = \tfrac{1}{2}\left(r_{a\to b} + r_{b\to a}\right), \qquad c(a) = \frac{1}{|\mathcal{T}|-1}\sum_{b\neq a} c(a,b). \tag{14}$$

The resulting task-level average undirected conflict scores are 22.34% for Code, 28.50% for Reason, and 29.38% for Math. Sorting them from low to high gives Code→Reason→Math, exactly matching the strongest order in the exhaustive ablation.

This ordering is consistent with the mechanism of projected protection. Once a task has been written to memory, subsequent gradients must discard components aligned with its protected subspace. Placing lower-conflict tasks earlier keeps the accumulated memory less broadly conflicting with later optimization, thereby limiting unnecessary removal while still suppressing components that overlap with protected directions.

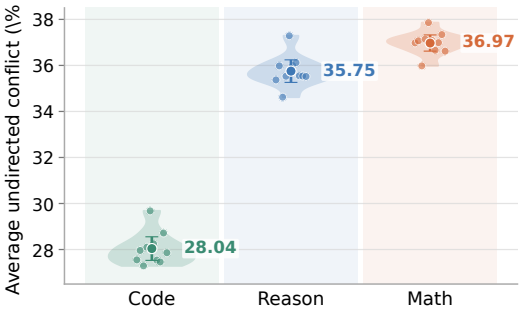


Figure 3: Ten-fold probe estimates of task-level average undirected conflict using 60 examples per fold. Shaded regions distinguish the three tasks. Circles show individual folds, diamonds and adjacent values denote fold means, and error bars indicate 95% confidence intervals. All folds recover Code < Reason < Math.

We evaluate whether the conflict ranking can be recovered from a lightweight probe rather than full training data. Using 60 examples per fold, Figure 3 shows the fold-level estimates together with their means and 95% confidence intervals. All ten folds preserve Code < Reason < Math, with mean estimates of 28.04%, 35.75%, and 36.97%, respectively. The probe therefore recovers the task-level ranking from a compact sample and

selects the best observed order before full training, avoiding exhaustive permutation evaluation. The fold-level values and full directional measurements are reported in Appendix A.1.

7.2 Selecting the number of cycles by subspace consistency

Under a fixed per-task data budget, training can be partitioned into different numbers of cycles. We evaluate 1, 2, 4, 5, 8, and 10 cycles while keeping the task order and total number of prompts unchanged. Figure 4(a) shows a non-monotonic trend: the average score across the three tasks increases from 65.81 with one cycle to a maximum of 66.97 with four cycles, then decreases to 65.37 with ten cycles. We next examine why the intermediate schedule performs best.

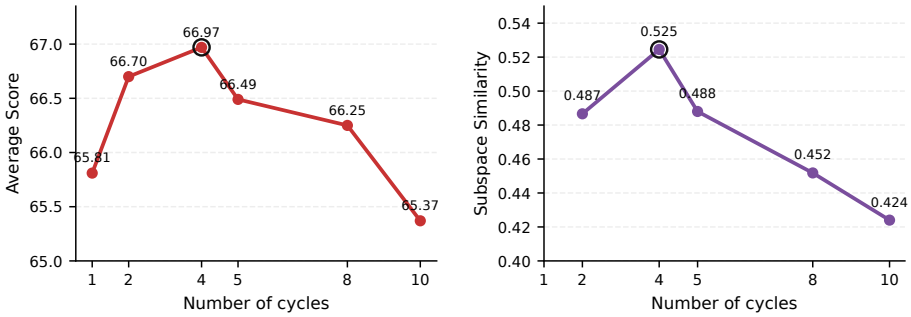


Figure 4: Cycle-count analysis under a fixed per-task data budget. (a) Average score across Math, Reason, and Code for the tested schedules, with four cycles achieving the best result. (b) Mean same-task subspace similarity across cycle revisits, computed using Equation 3. Four cycles also produce the most consistent subspace estimates. Similarity is undefined for the one-cycle schedule because no cross-cycle comparison is available.

We use same-task cross-cycle subspace similarity as a diagnostic of estimation consistency. Specifically, Equation 3 compares the top-K update subspaces estimated for the same task across consecutive cycle visits, and we average the scores across revisits and tasks. A larger value indicates that repeated estimates recover more consistent update directions and therefore provides a proxy for the reliability of the subspace memory. Figure 4(b) follows the same pattern as the average evaluation score: similarity is 0.487 for two cycles, peaks at 0.525 for four cycles, and then decreases to 0.488, 0.452, and 0.424 for 5, 8, and 10 cycles, respectively.

The sweep reveals a two-sided scheduling trade-off. At high cycle counts, shorter task blocks provide less data for each cumulative displacement to stabilize, and more frequent memory reconstruction reduces agreement between successive estimates. At low cycle counts, longer uninterrupted blocks increase trajectory drift between visits to the same task. Four cycles balance reliable subspace estimation with timely task revisitation, producing both the highest consistency and the highest average evaluation score.

The coincidence of the performance and consistency peaks at four cycles supports the intended mechanism of projected protection: PMOPD performs best when its protected subspaces are estimated most consistently. Detailed per-task similarities are provided in Appendix A.2.

8 Conclusion

MOPD interference follows a compact geometric structure: task updates rapidly concentrate in distinct low-dimensional subspaces, and their overlapping components provide a direct target for interference control. PMOPD exploits this structure by constructing memories from realized task displacements and projecting both gradients and adaptive-optimizer updates away from protected directions. Combined with conflict-guided task ordering and cycle selection, this mechanism improves every evaluated capability over MOPD and raises the average score across the three tasks by 2.54 points on Qwen2.5-7B and 2.09 points on Llama-3.1-8B. These results establish parameter-update subspace protection as an effective and transferable strategy for integrating specialized teachers into a single balanced model.

AI USAGE DISCLOSURE

Generative AI tools were used during manuscript preparation to polish the English writing and to assist with the retrieval and discovery of relevant literature. The authors reviewed and revised all AI-assisted outputs, retained full control over the scientific content and citation selection, and take responsibility for the final manuscript.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7319–7328, 2021. doi: 10.18653/v1/2021.acl-long.568.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *International Conference on Learning Representations*, 2019.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 794–803, 2018.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Huan-ang Gao, Haohan Chi, Yong Yan, Shiyuan Feng, Hanlin Wu, Zheng Jiang, Bingxiang He, Wei-Ying Ma, Ya-Qin Zhang, and Hao Zhou. Open-MOPD: Diagnosing and fixing capability imbalance in multi-teacher on-policy distillation. *arXiv preprint arXiv:2608.19098*, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Advances in Neural Information Processing Systems*, volume 34, pp. 12686–12697, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get m for free. In *International Conference on Learning Representations*, 2017.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327, 2016.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. Hugging Face dataset repository, 2024.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, Jinhao Dong, Zhifang Sui, and Fuli Luo. MOPD: Multi-teacher on-policy distillation for capability integration in LLM post-training. *arXiv preprint arXiv:2606.30406*, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

Andrei A. Rusu, Sergio Gómez Colmenarejo, Çağlar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4596–4604, 2018.

Zhennan Shen, Yanshu Li, Qingyu Yin, Chak Tou Leong, Zhilin Wang, Yanxu Chen, Rongduo Han, Sunbowen Lee, and Yi R. Fung. On the geometry of on-policy distillation. *arXiv preprint arXiv:2606.07082*, 2026.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of NAACL-HLT*, pp. 4149–4158, 2019.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 23965–23998, 2022.

Xiaomi LLM-Core Team. MiMo-V2-Flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5824–5836, 2020.

A Additional Results and Diagnostics

A.1 Conflict estimates and directional measurements

Table 6 reports the task-level average undirected conflict estimated from each of the ten 60-example folds used in Figure 3. Every fold recovers the same ordering, Code $<$ Reason $<$ Math, demonstrating that the ranking is stable across compact probe subsets.

Table 6: Task-level average undirected conflict estimated from each 60-example fold. The final column gives the ascending task order within each fold.

Fold	Code	Reason	Math	Ascending order
1	27.55%	35.37%	36.98%	Code $<$ Reason $<$ Math
2	27.96%	35.98%	37.07%	Code $<$ Reason $<$ Math
3	27.29%	34.62%	35.98%	Code $<$ Reason $<$ Math
4	28.10%	35.53%	37.14%	Code $<$ Reason $<$ Math
5	29.68%	37.29%	37.86%	Code $<$ Reason $<$ Math
6	28.25%	36.00%	37.09%	Code $<$ Reason $<$ Math
7	27.54%	36.12%	36.66%	Code $<$ Reason $<$ Math
8	27.46%	35.55%	36.99%	Code $<$ Reason $<$ Math
9	28.72%	35.54%	37.34%	Code $<$ Reason $<$ Math
10	27.86%	35.52%	36.62%	Code $<$ Reason $<$ Math
Mean	28.04%	35.75%	36.97%	Code $<$ Reason $<$ Math

Table 7 reports the asymmetric measurements used to construct the undirected task scores. Here $a \to b$ means that a memory is first constructed from task a and the removed-gradient ratio is then measured while optimizing task b. The two directions differ for every pair, confirming that task conflict cannot be represented by a single symmetric quantity before aggregation.

Table 7: Directional conflict measured by the removed-gradient ratio.

Direction	Removed gradient
Math→Reason	36.86%
Reason→Math	34.22%
Code→Math	26.56%
Code→Reason	25.06%
Math→Code	19.87%
Reason→Code	17.85%

A.2 Per-task revisit consistency

Table 8: Same-task subspace similarity between cycle revisits.

Cycles	Code	Reason	Math	Mean
2	0.486	0.603	0.371	0.487
4	**0.535**	**0.622**	**0.417**	**0.525**
5	0.472	0.610	0.382	0.488
8	0.433	0.584	0.338	0.452
10	0.411	0.546	0.315	0.424

The four-cycle schedule achieves the highest revisit similarity for every task as well as the highest mean, providing task-wise support for the consistency criterion used to select the cycle count.

A.3 MOPD MIXING VARIANTS

Table 9: Task-mixing variants of MOPD on Qwen2.5-7B.

Variant	Math	Reason	Code	Avg.
WB random	66.67	71.91	55.00	64.52
WB ordered	68.78	72.71	53.00	64.83
BB random	65.33	70.68	55.67	63.89
BB ordered	65.56	69.86	56.67	64.03

We additionally compare several task-mixing implementations. Within-batch (WB) mixing places examples from different tasks in the same mini-batch, whereas between-batch (BB) mixing interleaves task-homogeneous mini-batches. "Random" and "ordered" indicate how tasks or samples are arranged within the corresponding scheme. The strongest mixing variant reaches 64.83, while the complete PMOPD configuration reaches 66.97, preserving a 2.14-point advantage over task mixing alone.