DATA-FREE CONTINUAL MODEL MERGING: ARRIVAL ORDER, NOT MISSING DATA, IS THE BOTTLENECK

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual model merging folds a stream of task experts into one deployed model, one at a time, with neither joint retraining nor the data any expert was trained on, and without revisiting one already absorbed. The merge step is denied what one-shot merging takes for granted: the data and the whole pool. However, the gap is read as a data problem, and the remedy is not free: calibration needs labelled samples at every arrival, which raises the merge cost, and data saturates within a few samples per task. In addition, arrival order goes unpriced, though taking the stream in order costs a data-free rule up to 18.9 points. To address those issues, we hold pool, backbone and evaluator fixed, price each constraint on its own, and propose SCOPE (Sign-Conflict, Order-scaled, Parameter-class Estimation), a merge-time-data-free operator of three scalars. Moreover, the constraints interact rather than add: the order penalty is large for a data-free rule and nearly vanishes for a calibrated one. On eight CLIP ViT-B/32 experts over thirty arrival orders, SCOPE improves on the machine-designed operator of our earlier search by a small but consistent margin. Two results bound the claim: parity on longer streams needs the damping scaled by stream length, and automated discovery is blocked by the fitness signal. These results argue for measuring the order constraint on its own terms, since a data-free number need not transfer from a pool to a stream.

1 INTRODUCTION

Model merging combines task-specific checkpoints that share a pre-trained initialization into a single set of weights without retraining, which yields one multi-task model at the cost of a merge rather than of a retraining run. The published record, however, is written for pools held in hand: task arithmetic and the sparsified, sign-aligned and rescaled variants of it (Ilharco et al., 2023; Yadav et al., 2023; Yu et al., 2024), methods that weight parameters by statistics computed from data (Jin et al., 2023; Matena & Raffel, 2022), and recent methods that act on the geometry of the task matrices (Gargiulo et al., 2025; Marczak et al., 2025; Jang et al., 2024) all need the experts at once. The setting we study is instead a stream of CLIP ViT-B/32 experts (Radford et al., 2021) merged in arrival order under a fixed evaluator (Tang et al., 2024), revisiting nothing already absorbed.

Continual model merging is the paradigm for that need: experts arrive in an order the algorithm does not choose, and each is folded into a running model before the next arrives, which is how independently produced checkpoints reach a deployed system. The setting is described through two constraints that are then discussed apart (C1, C2; Fig. 1): the merge step may not revisit an expert already absorbed, which removes the global view one-shot aggregation depends on, and it may not consult data, since the data of earlier experts is gone and the data of the incoming one is unavailable at merge time. To tackle this, we hold pool, backbone, evaluator and orders fixed, price each constraint on its own, and build an operator that respects both.

1.1 CHALLENGES

Two problems make the constraint that matters hard to see.

Indirect evidence for the data story. The one-shot data-free frontier sits near 0.80 to 0.83 while continual methods report close to 0.70, so data looks like what is missing. Yet every comparison supporting that reading changes the operator with the regime, confounding the two. With pool,

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.

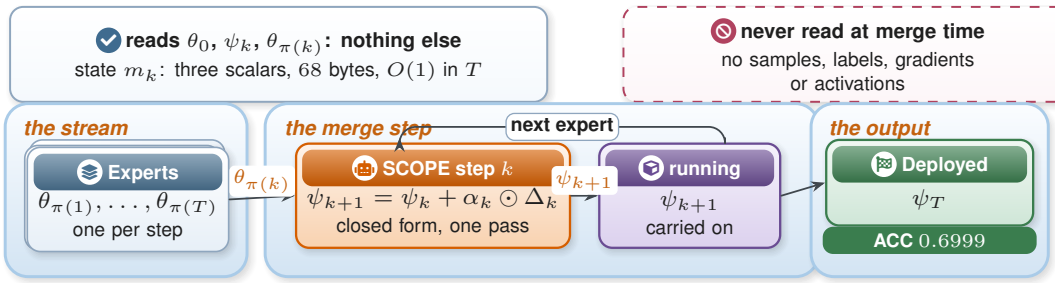

Figure 1: The continual merge step: one expert per step, in an order the algorithm does not choose, reading the base model, the running model and the incoming expert and nothing else, and carrying three scalars. The accuracy in the output box is a mean over the thirty orders 100–129. Colour marks the role of a box, not the protocol class of a method.

backbone, evaluator and orders fixed, one rule applied to its own one-shot and sequential variants moves accuracy from 0.7311 to 0.6217, while a rule calibrated on eight labelled samples per task pays 0.08 for the same constraint: the price of streaming is set by the rule, not by the stream.

An unpriced cell. Because the constraints are discussed separately, the cell where both bind has no measured price. An operator built for that cell must carry a state that does not grow with the stream, read no samples, and still track a running merged model; the published rules we could place in it (§3.3) solve an inner optimisation per arrival or ignore the stream altogether.

Our New Observation. The two constraints have to be measured jointly rather than priced apart: arrival order, not data, separates 0.70 from 0.83, and what a sample is worth depends on the rule that consumes it (§3.2).

1.2 Contributions

We price the two constraints against each other, build an operator that satisfies both, and test where it stops working. The contributions are three.

- **Joint constraint pricing.** The two constraints interact rather than add: the order penalty is 10.9 points for a rule that reads no data and 0.08 for one calibrated on eight samples per task, while data is cheaper and saturates, the first eight samples per task being worth 10.2 of the 12.9 points that 512 give (§3).
- **A three-scalar operator.** SCOPE (Sign-Conflict, Order-scaled, Parameter-class Estimation) is a merge-time data-free sequential operator, a closed-form per-tensor update of three scalars and 68 bytes of state (§4); it gives $+0.0015$ over the machine-designed operator our earlier search produced ($t = 3.876$, $22/30$ orders won), and a seven-row leave-one-out ablation shows which components do the work (§5.1).
- **Numerical verification.** At twenty experts that operator and SCOPE fall under an unmerged base model, which we trace to a damping schedule indexed by absolute step; scaling it by stream length is a bit-exact no-op at eight experts and recovers 10.7 points at twenty, restoring parity rather than an advantage (§5.5). Automated discovery is blocked by the fitness signal rather than by the search procedure (§5.4).

2 Setting: Two Constraints and Three Data Budgets

The continual merge step. Let θ_0 be the pre-trained weights and $\theta_1, \ldots, \theta_T$ the T experts fine-tuned from it, one per task. An arrival order is a permutation π; the k-th expert to arrive is $\theta_{\pi(k)}$, and ψ_k is the running model at the start of step k, with $\psi_1 = \theta_0$. A merging operator M advances the stream,

$$(\psi_{k+1},\, m_{k+1}) \;=\; M\big(\psi_k,\; \theta_{\pi(k)},\; \theta_0,\; m_k;\; k,\; T\big), \tag{1}$$

where m_k is the state carried between steps, $\Delta_k = \theta_{\pi(k)} - \theta_0$ the incoming task vector and $u_k = \psi_k - \theta_0$ the accumulated direction, zero at $k = 1$.

What the step may read. Two constraints apply to M; they are easy to conflate, so we state them apart.

Arrival order (C1). The operator receives one expert per step and processes the stream once, in the order given. It may not revisit or replay $\theta_{\pi(j)}$ for $j < k$, removing the global view one-shot aggregation needs. The horizon T is known and enters the schedule of §4; withheld, it falls back to its unscaled form, the collapse row of Table 5.

Merge-time data-free (C2). The operator may not run forward passes, compute gradients or collect activation statistics, and its state m_k must be of size $O(1)$ in the stream length; per-task second-moment or Fisher estimates violate this even without labels, because they grow linearly in T. SCOPE satisfies both. We price operators whose merge step is one pass of closed-form arithmetic, so a method that solves an inner optimisation per arrival leaves the class even when it reads no data (§3.3).

Three data budgets. *T0, merge-time data-free*: nothing in Eq. 1 touches samples, labels, gradients or activations; TA, TIES, running average and SCOPE are T0. *T1, per-merge calibration*: the merge step consumes a batch of data every time it runs; ODE-M uses about 1024 samples per step and RegMean and Fisher merging 256 labelled samples per task. *T2, one-off design cost*: a labelled signal is used once, offline, to choose an operator that is then frozen. The design of SCOPE is T2, so it cannot call itself data-free on the strength of its design step; the comparison below is between methods that are all T0 at merge time.

Protocol and statistics. The eight-expert pool (TA8) holds the CLIP ViT-B/32 experts of FusionBench (Tang et al., 2024) on SUN397, Stanford Cars, RESISC45, EuroSAT, SVHN, GTSRB, MNIST and DTD, and pools of 14 and 20 experts come from the same collection (§5.5). Accuracy is the macro average of the pool's zero-shot evaluator over the T tasks and their held-out splits. Main results use orders 100–109 and paired tests 100–129, none of them from the design blocks of §5.4; two methods on one order see the same sequence, so we compare per-order accuracies with a paired one-sample t-test.

3 What each constraint costs

Both constraints are priced on one pool, one backbone and one evaluator, so an accuracy difference can be charged to a constraint rather than to a change of regime.

3.1 Arrival order costs a data-free rule 10.9 to 18.9 points

We take one rule, TIES at density 0.2 and threshold 20, and run it in two regimes over the same eight experts, evaluator and ten held-out orders: at $\lambda = 0.4$ one-shot, the best point of the λ grid we swept there, and at $\lambda = 0.3$ sequentially, the released default of the two sequential implementations in the FusionBench CMM candidate library, sign gating and pairwise merging, unchanged. One-shot aggregation reaches 0.7311, and 0.7190 at the matched $\lambda = 0.3$; applied sequentially, sign gating reaches 0.6217 ± 0.0110 and the pairwise variant 0.5420 ± 0.0200. The arrival-order constraint thus costs 10.9 and 18.9 points against the best one-shot setting, and 9.7 and 17.7 against a matched one, to a rule whose hyper-parameters were chosen for accuracy rather than for sequential stability (Fig. 2, left).

The argument does not need the same rule: task singular vectors reach 0.8307 and isotropic merging 0.8039 on this pool, both data-free but acting on all task matrices at once and so failing C1. With the rule held fixed, the global view costs nine to nineteen points; tuning λ sequentially recovers 1.5.

3.2 Data is cheaper, and it saturates

RegMean merging with eight labelled samples per task reaches 0.8018, and with 32, 128, 256 and 512 it reaches 0.8139, 0.8250, 0.8281 and 0.8285. Measured from the sequential data-free level of 0.6999, the first eight samples deliver $+10.2$ points of the 12.9 that 512 provide, the remaining 504 add 2.7, and the curve is flat past 128 (Fig. 2, right).

The value of data depends on what consumes it: RegMean and Fisher merging both see 256 labelled samples per task in our runs, and reach 0.8281 and 0.7034, so "merging needs data" is about the method, not the data. The RegMean curve relaxes C1 and C2 at once, which bounds the data effect without isolating it; the arm that isolates it keeps arrival order and reads 256 labelled samples per



Figure 2: The two constraints priced under one pool and evaluator. Left: one rule, two arrival regimes, no data in either; bars are means over orders 100–109 with standard-deviation whiskers and the dashed line is SCOPE. Right: label budget of a data-consuming method (RegMean), flat past 128 samples per task.

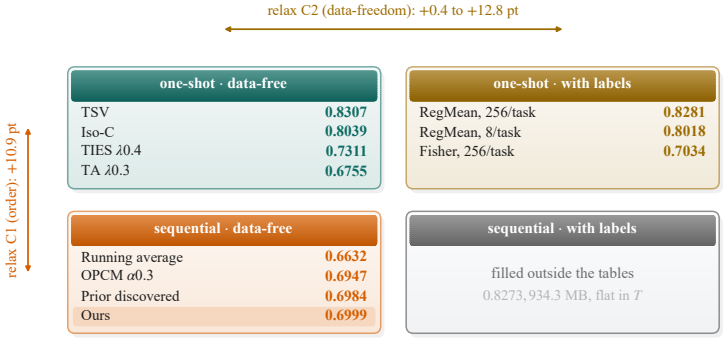

Figure 3: The two constraints as a map, with the methods of Table 6 in the cell their protocol assigns them. In the lower-left cell SCOPE, the prior operator and OPCM are means over the thirty orders 100–129 and the running average over 100–109; every other entry is a ten-order mean. Reading up the left column isolates the arrival-order constraint at a data-free rule, $+10.9$ points; reading across the top row isolates data-freedom, $+0.4$ points for Fisher and $+12.8$ for RegMean at the same labelled budget. The fourth cell is filled outside this map (§A.10).

task, reaching 0.8273 against 0.8281 one-shot. The order penalty is therefore conditional on the rule having data: 10.9 points for a rule that reads none, 0.08 for this calibrated one. C1 and C2 interact rather than add.

3.3 Where the class boundary sits

DOP (Yang et al., 2025) also reads one expert per step and touches no data; under this pool, evaluator and orders (100–109; alignment in §A.7) it reaches 0.779756 and wins all ten orders, but its merge solves a per-leaf-module MGDA problem with 200 gradient steps at $\approx 1.9\times$ our merge cost, so it leaves the closed-form class we compare. An element-wise maximum of the task vectors (Marczak et al., 2024) is the mirror case: it carries no running merged model, so its output does not depend on arrival order at all, which puts it outside the class the tables compare (§A.14). Neither result moves the price set of §3.1–§3.2. Fig. 3 puts the four protocol classes on one map: the gap between roughly 0.70 and 0.83 is crossed by relaxing either constraint, but not by relaxing neither.

4 SCOPE: A merge-time data-free operator of three scalars

SCOPE (Sign-Conflict, Order-scaled, Parameter-class Estimation) answers the two constraints of §2: it reads the base model, the running merged model and the incoming expert, carries a state of three scalars (ρ, n and k, 68 bytes with the serialized container), and returns a closed-form update. Fig. 4 shows the two parts of the design: the per-tensor update family of Eq. 2, inside which every operator we compare lives, and the four factors of Eq. 3 that the one-off search of §5.4 selected within it. Three of the four factors give the operator its name, $c_k[j]$, d_k and a_j; the fourth, norm stabilisation $s_k[j]$, completes the update, and §5.2 prices all four.

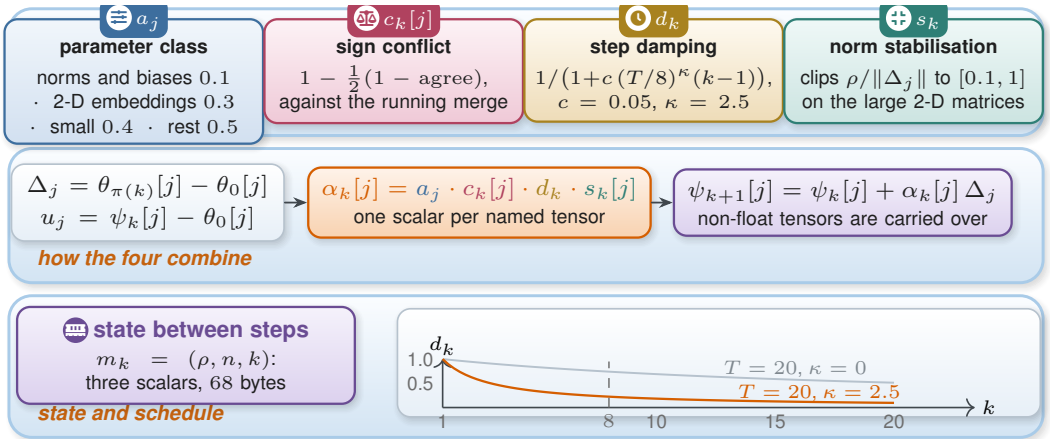

Figure 4: One step of SCOPE: four per-tensor factors combine into one scalar per named tensor, and the panel is the schedule of §5.5. Each factor's leave-one-out value at eight tasks is in Table 3.

4.1 The design space

An operator that satisfies C1 and C2 can only use what a step makes available, and within that envelope we restrict attention to updates of the form

$$\psi_{k+1} = \psi_k + \alpha_k \odot \Delta_k, \tag{2}$$

where Δ_k is the incoming task vector and α_k is a collection of *per tensor* scalars, one number for each named entry of the state dict. Per-element masks, low-rank factors and per-task statistics are excluded by C2, so a per-tensor scalar is the most expressive object that still fits in $O(1)$ state. Equation 2 contains task arithmetic (α constant), running average ($\alpha = 1/k$ with ψ_k tracked as a mean) and scalar-recency updates, which is why §5.1 compares one family rather than unrelated algorithms.

4.2 The operator

For each named tensor j, write $\Delta_j = \theta_{\pi(k)}[j] - \theta_0[j]$ for the incoming task vector and $u_j = \psi_k[j] - \theta_0[j]$ for the direction accumulated so far. SCOPE sets

$$\alpha_k[j] = a_j \cdot c_k[j] \cdot d_k \cdot s_k[j], \tag{3}$$

with the four factors of Fig. 4; tensors that do not hold floating point values are carried over from ψ_k unchanged.

Parameter-class base coefficients a_j. Each tensor is assigned a coefficient by parameter class: normalisation weights and biases get $a_j = 0.1$, two-dimensional embedding matrices 0.3, tensors with fewer than 5000 elements 0.4, and every remaining tensor 0.5. The classification uses the parameter name and shape alone, so no task-specific information enters. Parameter class is the one place where a data-free operator can still encode structure, and removing the class split costs 1.54 points (Table 3).

Sign-conflict factor $c_k[j]$. Let agree be the fraction of elements where $\text{sign}(u_{j,i}) \cdot \text{sign}(\Delta_{j,i}) > 0$, on which the incoming update points the same way as the accumulated direction, and set $c_k[j] = 1 - \frac{1}{2}(1 - \text{agree})$; at $k = 1$ the accumulated direction is zero, so the first step is at half size. The factor plays the role that sign election plays in TIES (Yadav et al., 2023), but the statistic is computed against the running merge rather than against all task vectors, which C1 forbids reading again. It is also the largest single contribution in the ablation, 2.96 points.

Step damping d_k. $d_k = 1/(1+c\,(T/8)^{\kappa}\,(k-1))$, with $c = 0.05$ and $\kappa = 2.5$: later experts perturb the merged model less than earlier ones, and that preference grows with the length of the stream. At $T = 8$ the scaling factor is exactly one, so every eight-task number in this paper is identical for all κ; κ is a parameter of the longer streams only, which set it to 3.0 at fourteen tasks and 2.5 at twenty (§5.5, Table 4). Removing it costs 0.76 points at eight tasks, but the same schedule sets the outcome on the longer pools.

Table 2: Paired comparison over the thirty held-out orders 100–129. Both methods see the same experts in the same sequence, so per-order accuracies are paired: the interval is the 95% confidence interval of the mean per-order difference and t and p come from a one-sample test over orders.

Comparison	Mean diff.	95% CI	t	Wins	p
SCOPE − MergeOp-M1	$\mathbf{+0.0015}$	$[\mathbf{0.0007}, \mathbf{0.0023}]$	$\mathbf{3.876}$	$\mathbf{22/30}$	$\mathbf{5.6 \times 10^{-4}}$
SCOPE − OPCM	$\mathbf{+0.0052}$	$[\mathbf{0.0021}, \mathbf{0.0084}]$	$\mathbf{3.374}$	$\mathbf{23/30}$	$\mathbf{2.1 \times 10^{-3}}$

Norm stabilisation $s_k[j]$. Applied to the large two-dimensional matrices only. Let ρ be a running estimate of the typical norm of an incoming task vector, initialised to $\|\Delta_j\|$ at the first large matrix and updated as $\rho \leftarrow 0.9\rho + 0.1\|\Delta_j\|$. Then $s_k[j] = \min(1, \max(0.1, \rho/\|\Delta_j\|))$, and if the update is still large relative to the running model, $\|\Delta_j\|/\|\psi_k[j]\| > 0.10$, the factor is further capped to $0.10\,\|\psi_k[j]\|/\|\Delta_j\|$. The ratio is scale-free, but one ρ serves the whole state dict and advances as the tensors are visited, so the value a tensor sees depends on the traversal order; we claim no invariance to it, and §5.3 measures spread over arrival orders instead. Removing the factor costs 0.97 points at eight tasks, while the ratio cap inside it is inert on this pool (Table 3).

Terminal trim. On the last step of the stream, for every tensor with at least two dimensions and at least 256 elements, elements of the accumulated delta u_j whose magnitude falls below the 0.25 quantile are set to zero. This is the magnitude pruning of one-shot merging (Yadav et al., 2023), applied once to the composite delta rather than to each task vector. With the damping schedule it needs the horizon T and is defined only when it is known (§2).

The state carried between steps is $m_k = (\rho, n, k)$, three float64 scalars (68 bytes with the serialized container), independent of the number of tasks and of the number of parameters; it is what keeps the update inside C2 while C1 stops it from revisiting an expert. The constants come from the one-off offline search of §5.4 on the held-out orders 200 and 201, with κ the exception, the one constant we defend on mechanism rather than on score. After that search the operator is frozen: no learned parameters, no calibrated statistics, the same code on every order and task set we report.

5 Experiments

We evaluate on the TA8 pool under the protocol of §2 and extend it to pools of 14 and 20 experts in §5.5. Every number below is official accuracy from the task pool's evaluator, and no result is taken from the internal proxy. Full experimental details and the complete protocol ladder are provided in the appendix.

5.1 Main results

Table 1 compares SCOPE with every same-protocol rule we could reproduce on TA8 over the ten held-out orders. SCOPE is the strongest of these rules, at 0.7003 ± 0.0041, ahead of MergeOp-M1 (the machine-designed operator our earlier automated search produced, §A.13) at 0.6980 and OPCM at 0.6936; every remaining rule is at least 0.7 points behind. The family of our own operator spans 0.6986 to 0.7003 (Table 7), nearly the 0.0023 by which its default beats MergeOp-M1, so the top of the table is closer than the leader's margin. MergeOp-M1 is the closest row and not an unrelated baseline: it shares the closed form, the three scalars of state and the four per-tensor factors of SCOPE, differing only in the damping, the terminal trim, the step counter and the fallback coefficient (§A.13).

Method	ACC	Std
TIES, pairwise	0.5420	0.0200
TIES, sign gating	0.6217	0.0110
Scalar recency update	0.6473	0.0080
Running average	0.6632	< 0.0001
Cumulative sum, trim	0.6925	< 0.0001
OPCM, $\alpha = 0.3$	0.6936	0.0062
MergeOp-M1	0.6980	0.0044
SCOPE (ours)	**0.7003**	**0.0041**

Table 1: Same-protocol comparison on TA8, official accuracy over the ten orders 100–109 of §2. Every entry is merge-time data-free and reads the stream in arrival order; the baselines run from weakest to strongest; the best baseline and our best entry are highlighted, ours is underlined as well, and our operator is the last row, on a tinted background. The other settings of our operator are in Table 7.

Table 3: Leave-one-out ablation of SCOPE on TA8, orders 100–109; each variant deletes one component of Eq. 3 and keeps everything else. The highlighted row is the largest single contribution; the highlighted cell of the tinted last row is the full operator, the reference every variant is measured against, and it reproduces Table 1 to within 2×10^{-5}.

Operator variant	Official ACC	Change
− sign-conflict factor	0.6707	−2.96
− parameter-class coef.	0.6849	−1.54
− norm stabilisation	0.6907	−0.97
− step damping	0.6928	−0.76
− terminal trim	0.7000	−0.04
− norm ratio cap	0.7003	0.00
SCOPE (full)	**0.7003**	–

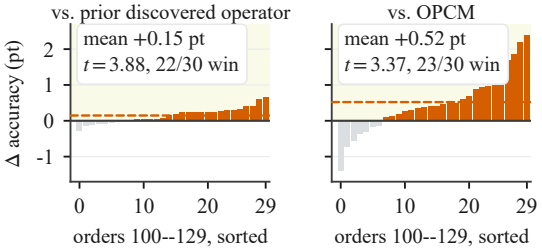

Figure 5: Paired per-order differences over the thirty held-out orders 100–129, sorted within each panel; bars above the axis are orders on which SCOPE wins and the dashed line is the mean difference. Left: against MergeOp-M1; right: against OPCM. Pairing within an order cancels the spread that arrival order induces, which is why a 0.15-point mean difference is resolved.

Widening to thirty orders (100–129) and testing per order keeps the sign (Table 2). The thirty-order means are 0.6999 for SCOPE and 0.6984 for MergeOp-M1, against 0.7003 and 0.6980 over the ten orders of Table 1. The improvement is small in accuracy but consistent in sign across orders, which is what a paired design can resolve, and it comes from the four factors acting together rather than from one (§5.2 prices each). It also buys that accuracy without data and without state that grows with the stream: one closed-form pass on three scalars, where a data-free rule that also takes the stream solves an inner optimisation per arrival at 1.9× our merge step (§3.3).

5.2 Which components do the work

The ordering in Table 3 is not what we expected of the design. The sign-conflict factor, the smallest idea in it, carries the most at 2.96 points, and the four non-inert factors sum to 6.2, so no single one does the work. The two remaining parts of the update can be deleted without moving the eight-task number: the terminal trim costs 0.04 points and the norm ratio cap nothing measurable.

From Table 3 and Fig. 4 we draw two conclusions: (1) the advantage is distributed rather than concentrated, which is why the family of Table 7 is tight and why no component can be dropped to make a cheaper eight-task operator; and (2) two of the six components are inert on this pool. We report the second as a negative result about our design: a component that costs nothing on eight tasks need not cost nothing in general.

5.3 Order robustness

Across orders 100–109 SCOPE scores between 0.6938 and 0.7064, a standard deviation of 0.0041 against 0.0062 for OPCM and 0.0200 for the pairwise sequential TIES variant, so the spread that arrival order induces differs by a factor of five across operators that all read the stream in one pass, and Fig. 5 resolves the two comparisons order by order. An operator that reacts to the running merge state is not automatically more order-sensitive: the sign-conflict factor that makes SCOPE react to the state is also what keeps an adversarial order from reversing the accumulated direction, so it is the rule and not the stream that sets how much the order costs (§1.1).

5.4 Can the operator be found automatically?

The constants of SCOPE came out of an offline search, so whether a language model can find better ones is natural. Warm-started with an equal budget of twenty candidate programs scored by the same proxy (§A.3), a rule-based grid, random sampling and a language-model reflective search land within 0.006 of one another, so the generator is not what the budget buys. What it cannot buy is a signal: of the thirteen operators that carry both a proxy and an official score, the best proxy in the pool, 0.7493, has the worst official accuracy, 0.6900, and the rank correlation falls from +0.496 over the full range to +0.096 where a search has to discriminate. Discovery is blocked by the objective rather than by the proposer, so a proxy that cannot rank near the top cannot certify that a better member of the family does not exist (§A.5 gives the per-generator scores and the cold-start arm).

5.5 Where the operator stops working, and why

We extended the pool to fourteen and twenty experts and re-ran the same protocol. Table 5 reports the outcome, and Fig. 6 draws the same numbers as one profile per method. The first of our two rows is not favourable to SCOPE: at fourteen tasks it sits at 0.6247 against 0.6511 for MergeOp-M1 and 0.6754 for OPCM, and at twenty tasks at 0.5027, last in the table. An unmerged base model scores 0.5565 at twenty tasks and a single-absorbed stream 0.5698, both above our operator and above MergeOp-M1 at 0.5512: the accumulation branch destroys accuracy relative to not merging at all (Fig. 11 in the appendix), while a running average, sequential TIES and OPCM sit at 0.611. Nothing else moves between the three columns: the same ten held-out orders, the same evaluator and the same merge-time data-free constraint, so the collapse is charged to the length of the stream. The repair has to find 5.1 points at fourteen tasks and 10.9 at twenty to reach the best baseline of each column. The schedule this section arrives at is in Table 4.

Damping	TA8	TALL14	TALL20
absolute step ($\kappa = 0$)	0.7003	0.6247	0.5027
fixed 0.125 (5 orders)	0.6959	0.6599*	0.5734*
fixed 0.250 (5 orders)	0.6815	0.6715*	0.6040*
length-scaled, $\kappa = 3.0$	**0.7003**	**0.6713**	**0.6070**
length-scaled, $\kappa = 2.5$	**0.7003**	**0.6698**	**0.6093**

Table 4: The damping schedule on the longer pools, eight, fourteen and twenty experts. The two fixed-strength controls were run on the five orders 100–104 and are marked; every other row is a ten-order mean over 100–109, and the highlighted cells are our best entry of each column. The eight-task column is unchanged by the exponent because $(T/8)^{\kappa} = 1$ there, where $\kappa = 2$ and $\kappa = 2.5$ reproduce the reference row of Table 3 to 2×10^{-5}; the tinted last two rows are the schedule this section arrives at.

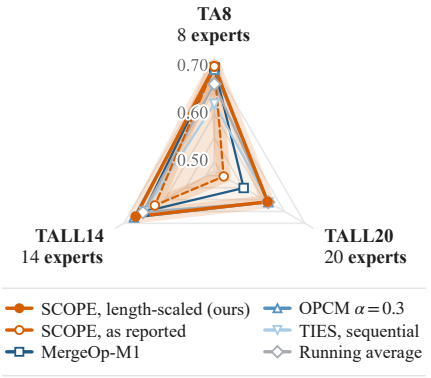

Figure 6: The same protocol on three pool sizes, one profile per method; the radius is official accuracy and the labelled rings mark 0.50, 0.60 and 0.70. Dashed: the operator of Table 1; solid: its length-scaled damping.

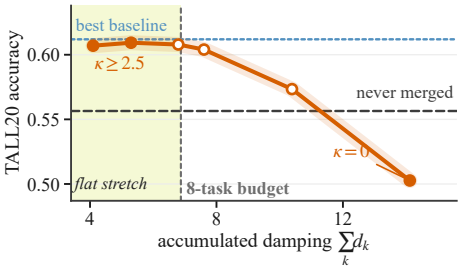

Figure 7: Twenty-task accuracy against the damping the schedule accumulates, $\sum_k d_k$; filled markers are the ten-order means of Table 4 and open markers the five-order means, and the dotted line is the best baseline of Table 5.

The damping $1/(1 + 0.05\,(k-1))$ is 0.51 at step 20 and the ratio cap is a constant fraction of the running model norm, so both are indexed by absolute position rather than by position relative to the stream; weakening them should therefore help. It does not: length-invariant variants that scale the decay and the cap by $8/T$ are bit-exact at $T = 8$ and cost 2.2 points at fourteen tasks (0.6240 → 0.6025) and 5.7 at twenty (0.5081 → 0.4508) on orders 100–104, and removing the

Table 5: Cross-task-set results, same protocol, orders 100–109. The tinted bold rows are our operator: the first is the setting of Table 1, the second the length-scaled damping of §4 with the exponent chosen per stream length ($\kappa = 3.0$ at fourteen tasks, 2.5 at twenty). The daggered cell is a five-order mean; the best entry of each column is highlighted, and ours are underlined as well.

Method	TA8 (8)	TALL14 (14)	TALL20 (20)
TIES, sequential	0.6217	0.6506	0.6119
Running average	0.6632	0.6536†	0.6110
OPCM $\alpha = 0.3$	0.6936	0.6754	0.6112
MergeOp-M1	0.6980	0.6511	0.5512
SCOPE, as reported	**0.7003**	**0.6247**	**0.5027**
SCOPE, length-scaled damping	**0.7003**	**0.6713**	**0.6093**

damping altogether costs 0.76, 6.75 and 11.80 points at eight, fourteen and twenty tasks. The schedule has to be stronger, not weaker.

So the schedule has to scale with the stream. Writing the damping as $1/(1 + c\,(T/8)^{\kappa}(k-1))$ with $c = 0.05$ leaves the eight-task operator bit-for-bit unchanged for every κ; Table 4 gives the outcome. At twenty tasks $\kappa = 2.5$ recovers 10.7 points over the operator of Table 1 and 5.8 over MergeOp-M1, winning all ten orders ($t = 31.7$ and $t = 17.3$); at fourteen tasks $\kappa = 3.0$ recovers 4.7 points and beats MergeOp-M1 on all ten ($t = 9.1$). The response is flat between $\kappa = 2$ and $\kappa = 3$, and κ selected on 100–104 wins on 105–109.

What the repair buys is parity, not an advantage: at twenty tasks its paired differences against a running average, OPCM and sequential TIES are two ties within noise and one small loss (-0.0026, $t = -2.16$), and at fourteen it is second, within half a point of OPCM. The collapse is a property of scheduling an accumulation by absolute position, and the repair removes it without producing a gain.

What the damping spends. The collapse comes not from a factor that is too weak but from a schedule that spends more of it. What the update accumulates is the sum of the per-step factors, and that sum grows with the stream: with the unscaled damping it is 6.87 over eight experts and 14.12 over twenty, so at a fixed fraction $f = k/T$ of the stream a twenty-task run is damped harder at every step and it is the tail that pays. The twenty-task accuracy tracks that sum rather than the strength of any single step, from 0.5027 at a sum of 14.12, through 0.5734 at 10.39 (fixed 0.125) and 0.6040 at 7.60 (fixed 0.25), to a plateau at every setting whose sum falls near the eight-task value (Fig. 7). Holding that budget is what the correction has to do, and holding it predicts the exponent rather than searching for it: requiring $\sum_k d_k = 6.87$ gives $\kappa \approx 2.0$ at twenty tasks and $\kappa \approx 2.6$ at fourteen, inside the flat region the response shows (§A.8 derives the sum).

The failure belongs to the schedule and not to the data. Every rule in Table 5 is merge-time data-free, so none of them fails for want of samples; what the operator that collapses cannot see is the length of the stream, and its damping was set for a stream eight experts long. Read against §3, the twenty-task table is that same constraint seen from the other end.

Additional evaluation. The page limit puts the remaining experiments in the appendix: Table 6 the protocol ladder, §A.5 the generator comparison, §A.8 the damping-schedule controls, §A.12 the provenance of every number and §A.15 the related work. Read together, they leave the sign of the headline result unchanged: the constraint that separates the sequential class from the one-shot frontier is arrival order, not the absent data.

6 Conclusion

We observe that the two constraints of continual model merging have been priced separately, and that arrival order costs a data-free rule far more than missing data does. We therefore measure the two constraints against each other and propose SCOPE, a closed-form sequential operator whose entire state is three scalars and which reads no data at merge time. A sign-conflict factor measured against the running merge, a damping scaled by stream length and a parameter-class coefficient let it track the merged model without seeing the pool. It is the strongest operator in its class that we could reproduce, and on longer streams it reaches parity rather than an advantage. This argues for measuring the order constraint on its own terms; limitations are discussed in §A.16.

Reproducibility Statement

The pools, the fine-tuned checkpoints, the zero-shot evaluator and the CLI are the released FusionBench assets (Tang et al., 2024), and every number in this paper is the macro average of zero-shot accuracy that evaluator produces on each task's held-out test split. Arrival orders are fixed permutations of the pool index, reproducible from their seed, and the same permutation is used for every method inside a comparison (§A.7 verifies this for orders 100–109). The merge operator of §4 and the baseline of §A.13 are given as source, with the constants, the configuration keys and the per-step cost listed; the offline proxy used for screening is defined in §A.3. The code, the operator sources and the per-order result files will be released.

Ethics Statement

The work studies weight-space merging of publicly released image-classification checkpoints. It uses no human subjects, no personal data and no scraped text, and the compute it consumes is four consumer GPUs. The two constraints analysed here (no data at merge time and no revisiting of an already absorbed expert) describe a deployment setting, not a security guarantee: we do not claim that the merged model preserves any privacy or safety property of the experts it combines, and we make no claim about downstream use of the merged weights.

Statement on AI Use

In this work, we used generative AI tools for two tasks with required disclosure. The constants of SCOPE and of the baselines it is compared against came out of an offline search over candidate programs in which a large language model proposed and mutated the candidates (§5.4). The experiment and analysis code was written and run with the help of an AI coding assistant working under author direction, which also helped sanity-check the experimental protocol. We have not used generative AI tools for any of the other tasks that require disclosure: we generated no synthetic data with one, developed no theoretical model or conceptual framework, formulated no mathematical claim or proof, proposed no hypothesis, translated nothing, cleaned or reformatted no dataset, and carried out no qualitative or thematic analysis with it.

Additionally, we used generative AI tools for tasks with recommended disclosure: drafting and editing parts of the manuscript for readability, and creating or editing software code (parts of the plotting script and the TikZ source of the two schematic diagrams). No figure is an image produced by a generative model, and the design, the labels and every plotted value were specified and verified by the authors. Every selection that appears in this paper was made or confirmed on official accuracy or on the held-out design orders, and every number traces to a run file (§A.12).

The authors have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

References

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Chenyu Huang, Peng Ye, Tao Chen, Tong He, Xiangyu Yue, and Wanli Ouyang. EMR-Merging: Tuning-free high-performance model merging. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. Model stock: All we need is just a few fine-tuned models. In *European Conference on Computer Vision (ECCV)*, 2024.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Lihong Lin and Haidong Kang. Unlocking the potential of continual model merging: An ODE perspective. *arXiv preprint arXiv:2605.19409*, 2026.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning (ICML)*, 2024.

Daniel Marczak, Bartłomiej Twardowski, Tomasz Trzciński, and Sebastian Cygert. MagMax: Leveraging model merging for seamless continual learning. In *European Conference on Computer Vision (ECCV)*, 2024.

Daniel Marczak, Simone Magistri, Sebastian Cygert, Bartłomiej Twardowski, Andrew D. Bagdanov, and Joost van de Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. In *International Conference on Machine Learning (ICML)*, 2025.

Michael S. Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Zihuan Qiu, Lei Wang, Yang Cao, Runtong Zhang, Bing Su, Yi Xu, Fanman Meng, Linfeng Xu, Qingbo Wu, and Hongliang Li. Null-space filtering for data-free continual model merging: Preserving stability, promoting plasticity. In *International Conference on Learning Representations (ICLR)*, 2026.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Guinan Su and Jonas Geiping. Fine, I'll merge it myself: A multi-fidelity framework for automated model merging. *arXiv preprint arXiv:2502.04030*, 2025.

Anke Tang, Li Shen, Yong Luo, Enneng Yang, Han Hu, Lefei Zhang, Bo Du, and Dacheng Tao. FusionBench: A unified library and comprehensive benchmark for deep model fusion. *arXiv preprint arXiv:2406.03280*, 2024.

Anke Tang, Enneng Yang, Li Shen, Yong Luo, Han Hu, Lefei Zhang, Bo Du, and Dacheng Tao. Merging on the fly without retraining: A sequential approach to scalable continual model merging. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yongxian Wei, Anke Tang, Li Shen, Zixuan Hu, Chun Yuan, and Xiaochun Cao. Modeling multi-task model merging as adaptive projective gradient descent. *arXiv preprint arXiv:2501.01230*, 2025.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-Merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024a.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. AdaMerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations (ICLR)*, 2024b.

Enneng Yang, Anke Tang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, and Jie Zhang. Continual model merging without data: Dual projections for balancing stability and plasticity. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *International Conference on Machine Learning (ICML)*, 2024.

A Appendix

A.1 The operator, in full

The listing below is the operator as it was run, with one correction of transcription: the stream-length factor of the damping schedule had been dropped from the listing, and the configuration defaults guarding the terminal trim had been shortened. It reads three state dicts and one three-scalar memory, and returns the updated state dict and memory. No other input is available to it.

```
def continual_merge_step(base_state_dict, merged_state_dict,
                         incoming_expert_state_dict, memory, config):
    step = int(config.get("step", config.get("num_seen",
               int(memory.get("num_seen", 0)) + 1)))
    num_seen = int(config.get("num_seen", step))
    raw_total = config.get("total_tasks", None)
    total_tasks = int(raw_total) if raw_total else None
    eps = 1e-8
    trim_keep_frac, trim_min_numel = 0.75, 256
    DAMP_DECAY, CONFLICT_STRENGTH, RATIO_CAP = 0.05, 0.5, 0.10
    KAPPA = float(config.get("kappa", 2.5))
    ALPHA_BIAS, ALPHA_EMBED, ALPHA_SMALL, ALPHA_LARGE = 0.1, 0.3, 0.4, 0.5

    new_memory, new_state_dict = {}, {}
    running_norm = float(memory.get("running_norm", 0.0))
    norm_count = int(memory.get("norm_count", 0))
    length_scale = (float(total_tasks) / 8.0) ** KAPPA if total_tasks else 1.0
    damping = 1.0 / (1.0 + DAMP_DECAY * length_scale * max(0, step - 1))
    do_trim = (trim_keep_frac < 1.0 - eps) and total_tasks is not None \
              and (step >= total_tasks)
    for key in incoming_expert_state_dict:
        incoming = incoming_expert_state_dict[key]
        merged, base = merged_state_dict[key], base_state_dict[key]
        if not torch.is_floating_point(incoming):
            new_state_dict[key] = merged.clone()
            continue
        delta = incoming - base
        kl = key.lower()
        is_bias_norm = ("norm" in kl or "bias" in kl
                        or "running_mean" in kl or "running_var" in kl)
        is_embed = ("embed" in kl and incoming.ndim == 2)
        is_small = incoming.numel() < 5000 and not is_bias_norm and not is_embed
        is_large_2d = (incoming.ndim == 2 and incoming.numel() >= 5000
                       and not is_embed and not is_bias_norm)

        merged_delta = merged - base
        agree = (torch.sign(merged_delta) * torch.sign(delta) > 0).float().mean().item()
        conflict_factor = 1.0 - CONFLICT_STRENGTH * (1.0 - agree)
        alpha_base = (ALPHA_BIAS if is_bias_norm else ALPHA_EMBED if is_embed
                      else ALPHA_SMALL if is_small else ALPHA_LARGE)
        alpha = alpha_base * conflict_factor * damping

        if is_large_2d:
            param_norm, delta_norm = torch.norm(merged).item(), torch.norm(delta).item()
            norm_scale = 1.0
            if running_norm > 0 and delta_norm > 0:
                norm_scale = min(1.0, max(0.1, running_norm / (delta_norm + eps)))
            if param_norm > 0 and delta_norm > 0:
                ratio = delta_norm / param_norm
                if ratio > RATIO_CAP:
                    norm_scale = min(norm_scale, RATIO_CAP / ratio)
            alpha = alpha * norm_scale
            running_norm = (delta_norm if norm_count == 0
                            else 0.9 * running_norm + 0.1 * delta_norm)
            norm_count += 1

        updated = merged + alpha * delta
        if do_trim and incoming.ndim >= 2 and incoming.numel() >= trim_min_numel:
            delta_total = updated - base
            abs_c = delta_total.abs()
            threshold = torch.quantile(abs_c.reshape(-1), 1.0 - trim_keep_frac)
            updated = base + delta_total * (abs_c >= max(threshold, eps)).to(abs_c.dtype)
        new_state_dict[key] = updated

    new_memory.update(running_norm=running_norm, norm_count=norm_count,
                      num_seen=num_seen)
    return new_state_dict, new_memory
```

The stream-length factor is one at eight experts whatever the exponent, so this listing reproduces every eight-task row with `total_tasks` set to 8; the fourteen- and twenty-task rows set `kappa` to 3.0 and 2.5 and `total_tasks` to the length of the stream. Called without `total_tasks`, as a caller who does not know the horizon must, the operator falls back to the unscaled schedule, which is the first row of Table 4. The operator this listing improves on is given, in the same format, with its own listing and the four differences between the two, in §A.13.

A.2 Full protocol ladder

Table 6: Every accuracy in this paper that comes from a same-protocol comparison inside the four classes of Fig. 3, with the protocol it was measured under. The DOP comparison of §3.3 is outside the class and appears in Table 9. Rows are grouped by whether the method sees the experts one at a time and whether data is available at merge time. Accuracy is official accuracy on the same pool, backbone and evaluator; the four sequential rows of SCOPE, MergeOp-M1, OPCM and TIES at $\lambda = 0.5$ are thirty-order means from Table 2, and every other row is a ten-order mean, the best entry of each block is highlighted, and the sequential block closes with our operator. The order-invariant rule we also measured belongs to no block – its merge never reads the running model – so it is reported on its own in Table 10. The two 0.6632 rows are a simple average and a running average, which coincide numerically because on eight experts the running mean is the simple mean.

Method	Arrival	Data at merge	Official ACC
One-shot, no data			
Task singular vectors	all at once	none	0.8307
Isotropic merging	all at once	none	0.8039
TIES $\lambda = 0.4$	all at once	none	0.7311
TIES $\lambda = 0.35$	all at once	none	0.7295
TIES $\lambda = 0.45$	all at once	none	0.7267
TIES $\lambda = 0.3$	all at once	none	0.7190
TA $\lambda = 0.3$	all at once	none	0.6755
Simple average	all at once	none	0.6632
One-shot, data available			
RegMean, 512 labels/task	all at once	512 labels	0.8285
RegMean, 256 labels/task	all at once	256 labels	0.8281
RegMean, 128 labels/task	all at once	128 labels	0.8250
RegMean, 32 labels/task	all at once	32 labels	0.8139
RegMean, 8 labels/task	all at once	8 labels	0.8018
Fisher, 256 labels/task	all at once	256 labels	0.7034
Fisher, 32 labels/task	all at once	32 labels	0.6879
Sequential, no data			
MergeOp-M1	one at a time	none	0.6984
OPCM $\alpha = 0.3$	one at a time	none	0.6947
Cumulative sum, terminal trim	one at a time	none	0.6925
Running average	one at a time	none	0.6632
Scalar recency update	one at a time	none	0.6473
TIES $\lambda = 0.50$, sign gating	one at a time	none	0.6366
TIES, sign gating	one at a time	none	0.6217
TIES, pairwise	one at a time	none	0.5420
SCOPE (ours)	one at a time	none	**0.6999**

A.3 The proxy fitness the search scored

A search over programs needs a score for candidates that have never been run on the official evaluation, and that score is a restricted version of the accuracy the rest of the paper reports. For every task in the TA8 pool we draw a stratified subsample of 256 images from that task's *training* split (subset seed 27), classify it zero-shot with the merged checkpoint, and average the eight per-task accuracies with equal weight; the fitness of a candidate is the mean of that average over the two design orders, 200 and 201. The pools, backbone, evaluator and metric are those of §5, evaluated under the pool's fixed text templates and a frozen text encoder, so the only thing that varies between two candidates is the merged image encoder. The data is what differs, and it comes from the training splits only, never

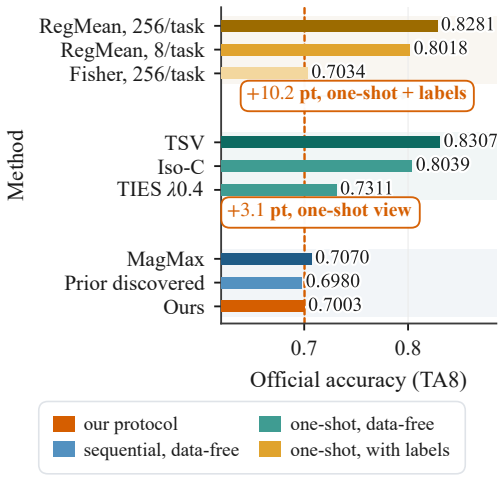

Figure 8: Accuracy against the constraints each method respects, same pool, backbone and evaluator; every row is listed with its protocol in Table 6, and the sequential bars are the ten-order means of Table 1, so both ends of each arrow are measured at the same protocol. Bars are coloured by protocol class and the horizontal axis is comparable only within a class. The two arrows leaving SCOPE are not prices of a single constraint: the first is the gap to one-shot TIES, the second the gap to one-shot RegMean at eight labels, and each crosses a different operator as well as the constraint it names.

from the held-out test splits that produce the official numbers: the two scores plotted against each other in Fig. 9 are therefore the same quantity measured on different samples.

The configuration that produced Table 8 ranks candidates on the pure-accuracy target, in which the optional penalty terms of the survival score (order variance, negative backward transfer, harness risk) are switched off, and legality and robustness enter as gates that can remove a candidate rather than as weights that move it. A candidate's rank is therefore its proxy fitness, and no tuning weight enters the comparison between the generators.

No claim in this paper rests on a proxy number. It is an internal screening score on 256 training images per task, its resolution near the top of the candidate pool is what §5.4 measures, and every number in a table of results is an official accuracy on held-out test splits.

A.4 Tables and figures deferred from §5

These carry the same numbers as the main text and are moved here for length only. Nothing in the main text depends on a row that is not reproduced there.

Table 7: The remaining settings of SCOPE. All four members of the family are above the MergeOp-M1 of Table 1; the spread inside the family is nearly as large as the gap from its default setting to that baseline. The tinted last row is the default of the main text.

Operator setting	Official ACC	Std
SCOPE, keep 0.80	0.7002	–
SCOPE, keep 0.85	0.7001	–
SCOPE, damping 0.03	0.6986	–
SCOPE (default)	**0.7003**	**0.0041**

A.5 The generator comparison in full

All three generators begin from the same skeleton operator with a budget of twenty candidate programs scored by the same proxy fitness on orders 200 and 201 (§A.3). Warm-started, they are indistinguishable, all landing between 0.7424 and 0.7446, and Table 8 gives the per-generator scores. Cold-started, the language-model arms fall four to ten points below a hand-written baseline, but that arm is not like-for-like: the cold runs search an empty program space rather than the skeleton's, so we read them only as evidence that this operator space does not yield to search from scratch on this budget, not as evidence about the generator. The reason a warm start does not separate the generators

Table 8: Equal-budget program generation from the same warm start (§5.4). Proxy fitness is computed on the design orders 200 and 201; it is an internal screening score, not a result, and appears here only to compare the generators.

Generator	Start	Best proxy fitness
Rule-based grid (no LLM), 4×5 points	skeleton	0.7446
Random sampling (no LLM), ten hyper-parameters	skeleton	0.7424
Language-model reflective search, seed 1	skeleton	0.7446
Language-model reflective search, seed 2	skeleton	0.7483
Language-model reflective search, seed 3	skeleton	0.7446
Language-model search, from scratch	empty	0.6458–0.7021

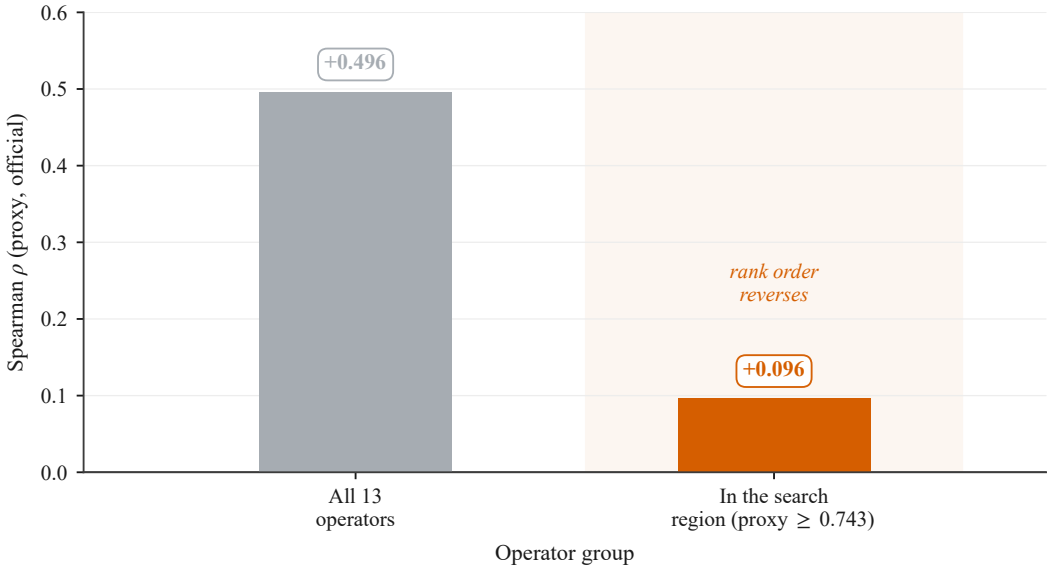

Figure 9: Spearman correlation between the proxy fitness used during search and official accuracy, over the thirteen operators in our library. Across the whole range the proxy is informative; inside the band where candidates are actually separated it is not, and the best proxy there has the worst official accuracy.

is the fitness signal (Fig. 9). Over the full range the Spearman correlation between proxy and official accuracy is $+0.496$, but restricted to the seven operators with proxy ≥ 0.743, the band where a search actually has to discriminate, it is $+0.096$. That subsample is selected on the very quantity being correlated and is truncated by construction, so part of the drop is a selection effect rather than a finding. The claim that does not depend on the correlation is the ranking miss: the best proxy in the pool, 0.7493, has the worst official accuracy, 0.6900.

A.6 Per-order accuracies

SCOPE on orders 100–109: 0.7022, 0.6996, 0.6938, 0.6971, 0.7064, 0.7038, 0.6998, 0.7014, 0.6941, 0.7049, mean 0.7003, standard deviation 0.0041. Over orders 100–129: mean 0.6999, standard deviation 0.0035. The prior operator over the same thirty orders: mean 0.6984, standard deviation 0.0034. Twenty-task per-order values for SCOPE: 0.5193, 0.5121, 0.5094, 0.5019, 0.4980 over orders 100–104.

A.7 The DOP comparison, and why it is same-protocol

§3.3 reports DOP under our protocol, which is worth little unless it really saw the same orders and the same evaluator. Four checks say it did.

- The continual-merge harness of the benchmark release already carries aliases for order seeds 100–109; they are the permutations we generate, element by element.

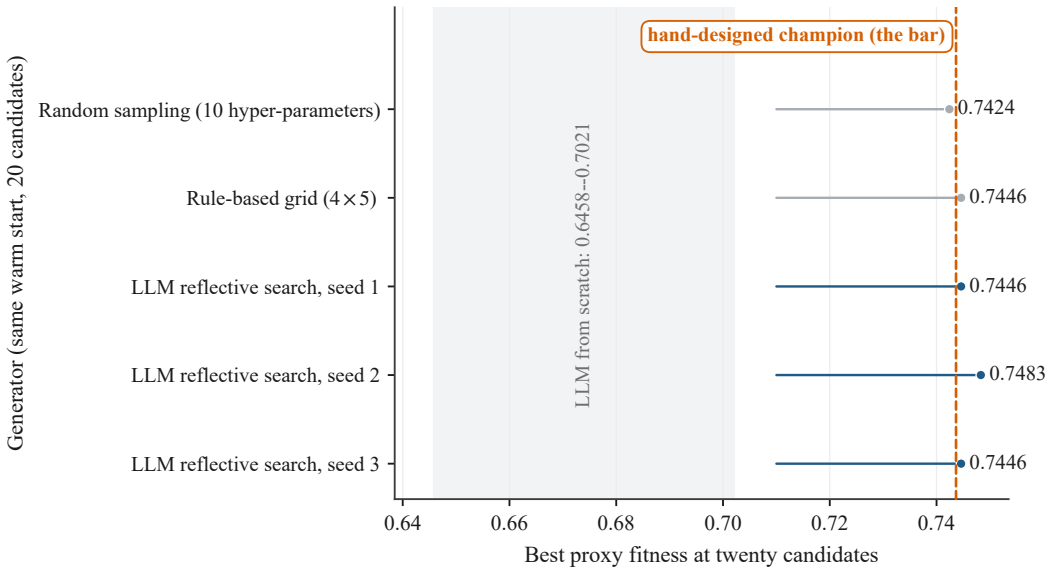

Figure 10: Equal-budget generation from the same warm start (Table 8). The two no-language-model generators and the three language-model runs land within six tenths of a point of one another, and four of the five are at or above the hand-designed bar; random sampling falls 0.1 points short. The shaded band is what the same search reaches from scratch, four to ten points below the bar. The start, not the generator, is what the budget buys.

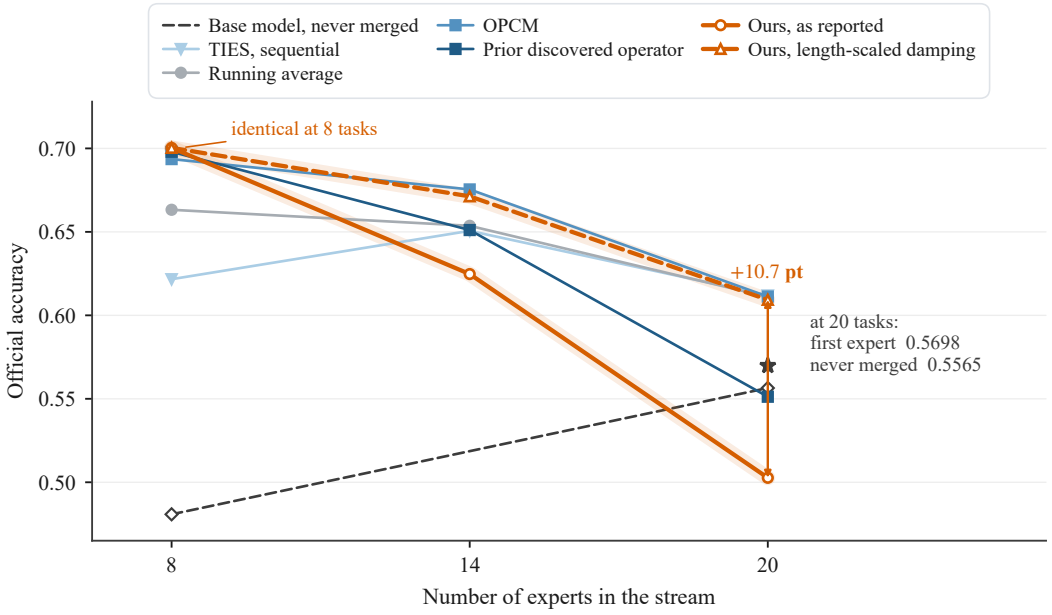

Figure 11: Longer streams, ten orders, same protocol (§5.5). At twenty tasks the accumulation branch falls below a base model that is never merged, while averaging and projection hold near 0.611; scaling the damping by stream length restores that level without exceeding it.

- Our generator reproduces them independently: for all ten seeds, `random.seed(s)` with `random.shuffle(x)` agrees with the harness's `random.Random(s).shuffle(idx)`.
- The evaluation is the same TA8 task pool, task for task, with the same number of evaluation batches per task (SUN397 156, Stanford Cars 63, RESISC45 50, EuroSAT 22, SVHN 204, GTSRB 99, MNIST 79, DTD 15).

- Two independent runs settle it: the released default, which shuffles with seed 100, and our order-aligned run at seed 100 give the same per-order value, 0.794540, digit for digit.

Table 9: DOP under the protocol of this paper: same pool, backbone, evaluator and arrival orders as every other row, orders 100–109. Values are the final per-order entries of `program_report.json`, whose intermediate overwrites are not used; mean 0.779756, standard deviation 0.014639.

Order	100	101	102	103	104	105	106	107	108	109	mean
DOP	0.7945	0.7847	0.7816	0.7627	0.7964	0.7514	0.7962	0.7844	0.7810	0.7646	**0.7798**

Cost. One aligned order takes 17.5 minutes end to end: 8.1 minutes loading the eight experts, 6.0 minutes merging them and 3.3 minutes evaluating. Our own merge is 27.25 seconds per arrival and 191 seconds per order end to end, so DOP's merge step is $\approx 1.9\times$ ours and its end-to-end cost $\approx 5.5\times$. The gap is the inner optimisation, not the state.

Pipeline self-check. The sequential TIES number we quote at $\lambda = 0.30$ is a ten-order mean, 0.6217; over the thirty orders of Table 2 it is 0.621809. Two different blocks of the protocol agree to three decimals.

A.8 Damping-schedule controls

A fixed damping of 0.125 is exactly what the scaled rule uses at twenty tasks with $\kappa = 1$ (Table 4); at that strength the twenty-task value is 0.5734, 3.5 points short of $\kappa = 2$, and it costs 0.44 points at eight tasks. A fixed damping of 0.25 does get most of the twenty-task gain (0.6040 on the five orders, against 0.6079 for $\kappa = 2$) but costs 1.9 points at eight tasks, where the operator is supposed to leave the main result alone. The only setting we found that is strong at twenty tasks and free at eight is the one whose strength depends on the length of the stream.

Why the strength has to grow. With $d_k = 1/(1 + c\,(T/8)^{\kappa}(k-1))$ the damping is indexed by absolute step, so at a fixed fraction $f = k/T$ of the stream it behaves as $1/(1 + cf\,T\,(T/8)^{\kappa}) = 1/(1 + 8^{\kappa} cf\,T^{\kappa+1}/8^{\kappa+1})$, i.e. the same constant suppresses the tail of the stream as $(T/8)^{\kappa+1}$. That is why weakening the constant looked like the fix: relative to its own stream, a twenty-task run is damped harder at every step. What the operator accumulates is not a per-step factor but the sum of them, and that sum grows with the length of the stream: 6.87 at $T = 8$ against 14.12 at $T = 20$ with the same constant. The twenty-task value tracks the sum rather than the strength at any single step (14.12 $\rightarrow$ 0.5027, 10.39 (fixed 0.125) $\rightarrow$ 0.5734, 7.60 (fixed 0.25) $\rightarrow$ 0.6040, 6.80 ($\kappa = 2$) $\rightarrow$ 0.6079), so it is the eight-task budget that has to be held, and holding it predicts the exponent instead of searching for it: requiring $\sum_k d_k = 6.87$ gives $\kappa \approx 2.0$ at twenty tasks and $\kappa \approx 2.6$ at fourteen, and the exponents §5.5 reports, 2.5 and 3.0, sit inside the flat region of that response, at sums of 5.3 and 6.2 against 6.9. Multiplying the constant by $(T/8)^{\kappa}$ is the correction that holds the budget, which is why it is invisible at $T = 8$ and decides the longer pools.

A.9 Re-verifying the constant family

The constants of SCOPE were chosen offline with the proxy fitness of §5.4, so the caveat in §A.16 rests on whether that choice reproduces. Under the proxy it does. On the original calibration grid (orders 300 and 301) and on a fresh one (orders 400 and 401), the same constant, `c_large2d_1p2`, is the argmax, at 0.749267 and 0.758545; the four candidate strengths of that grid are ordered $1p2 > 1p45 > 0p85 > 0p7$ in both runs, position by position. Under the official evaluator on the fresh orders the argmax moves to `c_bias_norm_0p7` (0.706158), with `c_large2d_1p2` fifteenth of seventeen. The family is reproducible under the design proxy, and the proxy gives no guarantee about the official ranking, which is why the caveat is tightened rather than dropped, and why no claim in either direction is made from these runs. They lie outside the held-out block and no number here enters a table or a conclusion.

A.10 The data-channel arm: accounting and reading

Dropping C2 while keeping C1 admits one further arm: an operator that still sees one expert per step and never revisits one, but may keep statistics computed from labelled samples. RegMean merging is the natural instance of it. Our tables contain no row for that arm, so we define the cell here, and we state the two things that have to travel with its accuracy.

State accounting. C2 forbids a merge step that reads samples and a state that grows with the stream; the arm we measured fails the first clause. Its state is an accumulator of 934.3 MB, printed once per arrival and identical at each of the eight steps, so on this pool it is set by the width of the network rather than by the stream length: seven orders of magnitude above the 68 bytes that the operators of Table 1 carry, and flat in T. Per-task statistics, which do grow linearly in T, remain absent from our tables. Where this arm is reported, the state size travels with the accuracy.

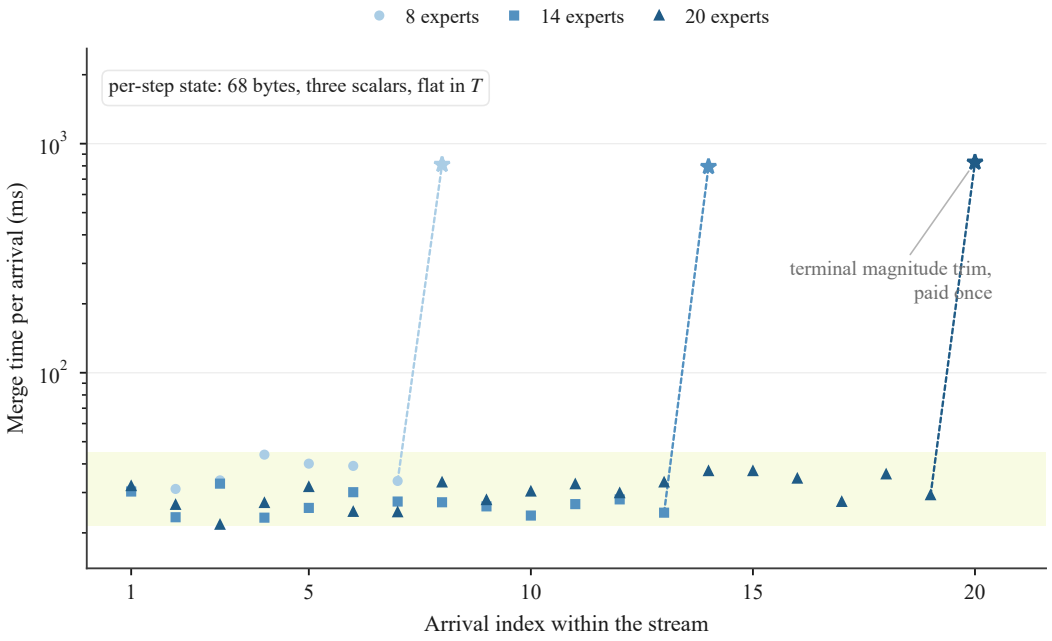

Figure 12: Cost of one arrival, measured on the operator of §4 at 8, 14 and 20 experts. A per-step merge takes between 22 and 44 milliseconds and does not grow with the stream; the star at the end of each run is the magnitude trim, which costs about 0.8 seconds and is paid once. What travels between steps is three scalars, 68 bytes, at every stream length, which is exactly what makes the fourth cell of Fig. 3 the expensive one.

Reading. At a fixed labelled budget per task (the 256 samples per task of §3.2), the arm separates the two relaxations that §3.2 makes at once, and it does so without the one-shot view that the RegMean curve also buys. We pre-registered two outcomes and read four settings of the arm's non-diagonal reduction rather than its best one, because a single point cannot separate a ceiling from its tuning: 0.6823 with no reduction, 0.7961 at 0.5, 0.8273 at 0.95 and 0.0496 at 1.0. The first outcome holds. At 0.95 the arm reaches 0.8273 against 0.8281 for one-shot RegMean at the same labelled budget, so the order constraint is a data-free phenomenon and nearly all of the 10.9 to 18.9 points of §3.1 are the price of data-freedom rather than of streaming; the second outcome, an arm that stays near the sequential data-free level of 0.6999, is not what we observe. The setting at 1.0 collapses to 0.0496, and the reason is in the parameterisation rather than in the arm: the reduction is a coefficient multiplying the off-diagonal entries, so 1.0 leaves the full Gram matrix in the inversion and the arm inverts a matrix that is numerically singular. All four numbers are official accuracies from the task pool's evaluator, but they come from its default evaluation of the eight experts rather than from orders 100–109: they are comparable with the RegMean data curve of §3.2, which uses the same evaluation, and not with Table 1.

A.11 Implementation details

Pools and evaluator: FusionBench, CLIP ViT-B/32 classification pools (`clip-vit-base-patch32_TA8`, `_TALL14`, `_TALL20`) with the matching task pools. Evaluation is zero-shot classification of the merged checkpoint on each task's test split, macro-averaged. PyTorch 2.5.1. Runs used four RTX 3090 GPUs with queueing on a per-GPU lock, so that at most one evaluation occupies a device at a time. Per-order cost tracks the merge step: MagMax's

merge is a single element-wise pass, so one of its merged models is produced and evaluated in about 3 minutes at eight experts, 4 at fourteen and 5 at twenty; our own merge is 27.25 seconds per arrival and 191 seconds per order end to end, and DOP's inner optimisation brings its aligned order to 17.5 minutes (§A.7). Orders are fixed permutations of the pool index and are identical across methods and across runs.

A.12 Provenance of the numbers

Every number in the tables and figures of this paper traces to a run directory holding the per-order JSON produced by the evaluator, with the order index, the seed and the operator hash recorded alongside. Aggregates are computed from those files; no number in this paper was transcribed from a slide, and none was averaged across protocols.

The operator we improve on in Table 1 is machine designed, and the metadata record frozen with it survives: it pins the SHA-256 of the operator source and records `code_origin` = `llm_generated`, `generation_invalid` = `false`, and the mechanism family `tensor_routing`. The pinned hash is the hash of the source file we re-ran to check that the baseline numbers reproduce.

A.13 MergeOp-M1, in full

MergeOp-M1 is the third row of Table 1 and the baseline every comparison in this paper is against: the machine-designed operator whose frozen metadata §A.12 describes it. It was reported without a definition, so we give the definition and the source here. It sits in the same family as SCOPE: one closed-form pass over the state dict per arrival, three scalars of state, no data, and Equation 2 with the four per-tensor factors of Equation 3. Four settings separate it from SCOPE, and together they are the whole of our contribution to it:

- *Damping.* $d_k = 1/(1 + 0.1\,(k-1))$: a fixed strength indexed by absolute step, with no $(T/8)^{\kappa}$ factor. SCOPE halves the constant and scales it by stream length (§5.5).
- *No terminal trim.* The magnitude trim on the last step is absent, and the operator has no code path for it.
- *Step counter.* The default step is one, so the caller passes the index; SCOPE recovers it from the memory when it is not passed.
- *Fallback coefficient.* Tensors outside the four classes take 0.1; SCOPE gives them the large-matrix coefficient, 0.5. On this pool that class is one tensor, the patch-embedding convolution that §4.2 names as taking the last class without being a large two-dimensional matrix.

Everything else is shared: the parameter classes $a_j \in \{0.1, 0.3, 0.4, 0.5\}$ under the same name-and-shape rule, the sign-conflict factor $c_k[j] = 1 - \frac{1}{2}(1 - \mathrm{agree})$ computed against the running merge, and the norm stabilisation $s_k[j]$ of the large two-dimensional matrices with the same running estimate ρ, the same floor 0.1 and the same ratio cap 0.10. Under the official evaluator this operator averages 0.6980 over the ten orders of Table 1 and 0.6984 over the thirty of Table 2; it is what §5.1 and §5.5 compare SCOPE against. The listing below is the frozen source, verbatim; its SHA-256 is the hash pinned in §A.12.

```
def continual_merge_step(base_state_dict, merged_state_dict, incoming_expert_state_dict,
    memory, config):
    step = int(config.get('step', 1))
    num_seen = int(config.get('num_seen', step))
    total_tasks = int(config.get('total_tasks', num_seen))
    new_memory = {}
    running_norm = float(memory.get('running_norm', 0.0))
    norm_count = int(memory.get('norm_count', 0))
    new_state_dict = {}
    # per-layer alphas
    alpha_bias = 0.1
    alpha_embed = 0.3
    alpha_small = 0.4
    alpha_large = 0.5
    alpha_fallback = 0.1
    # damping factor: decreases with step
    damping = 1.0 / (1.0 + 0.1 * (step - 1))
    for key in incoming_expert_state_dict:
        incoming = incoming_expert_state_dict[key]
        merged = merged_state_dict[key]
        base = base_state_dict[key]
```

```
        # skip non-float tensors
        if not torch.is_floating_point(incoming):
            new_state_dict[key] = merged.clone()
            continue
        # compute delta
        delta = incoming - base
        # detect parameter type
        key_lower = key.lower()
        is_bias_norm = ('norm' in key_lower or 'bias' in key_lower or 'running_mean' in key_lower or 'running_var' in key_lower)
        is_embed = ('embed' in key_lower and len(incoming.shape) == 2)
        numel = incoming.numel()
        is_small = numel < 5000 and not is_bias_norm and not is_embed
        is_large_2d = (len(incoming.shape) == 2 and numel >= 5000 and not is_embed and not is_bias_norm)
        # conflict attenuation: sign agreement between (merged-base) and delta
        merged_delta = merged - base
        sign_agree = (torch.sign(merged_delta) * torch.sign(delta)) > 0
        agree_frac = sign_agree.float().mean().item() if sign_agree.numel() > 0 else 1.0
        conflict_factor = 1.0 - 0.5 * (1.0 - agree_frac)  # 1.0 if full agree, 0.5 if full disagree
        # choose alpha base
        if is_bias_norm:
            alpha_base = alpha_bias
        elif is_embed:
            alpha_base = alpha_embed
        elif is_small:
            alpha_base = alpha_small
        elif is_large_2d:
            alpha_base = alpha_large
        else:
            alpha_base = alpha_fallback
        # apply conflict and damping
        alpha = alpha_base * conflict_factor * damping
        # special handling for large 2D: bounded norm-stabilized scaling
        if is_large_2d:
            param_norm = torch.norm(merged).item()
            delta_norm = torch.norm(delta).item()
            # norm-stabilized scale using running average
            norm_scale = 1.0
            if running_norm > 0 and delta_norm > 0:
                norm_scale = min(1.0, max(0.1, running_norm / (delta_norm + 1e-8)))
            # further bound by per-layer ratio cap
            if param_norm > 0 and delta_norm > 0:
                ratio = delta_norm / param_norm
                if ratio > 0.1:
                    norm_scale = min(norm_scale, 0.1 / ratio)
            alpha = alpha * norm_scale
            # update running norm (exponential moving average)
            if norm_count == 0:
                running_norm = delta_norm
            else:
                running_norm = 0.9 * running_norm + 0.1 * delta_norm
            norm_count += 1
        # perform merge
        updated = merged + alpha * delta
        new_state_dict[key] = updated
    # update memory
    new_memory['running_norm'] = running_norm
    new_memory['norm_count'] = norm_count
    new_memory['num_seen'] = num_seen
    return (new_state_dict, new_memory)
```

A.14 AN ORDER-INVARIANT MERGE, MEASURED OUTSIDE THE CLASS

The comparison of §5.1 is over rules that fold the stream in arrival order and decide what to do with the running model. A per-element maximum over task vectors is the mirror case: it carries no running merged model at all, so its output does not depend on the order in which the experts arrive, and such a row says nothing about the constraint the tables are about. We measured it anyway, on the same pool, backbone, evaluator and orders, and record the grid here rather than in Table 1 or Table 5: one merged model per cell, one element-wise pass, and no running state to be order-sensitive about.

Table 10: The MagMax merge coefficient on the three pools, all under the protocol of Table 6: one merge-time data-free pass per arrival, no calibration data, one merged model. We report the released default $\alpha = 0.5$ for every pool; the best point of our grid is in bold. Values are the official accuracies recorded in the `summary.csv` of each released run directory.

α	TA8 (8)	TALL14 (14)	TALL20 (20)
0.2	0.6246	–	–
0.3	0.6649	0.6638	0.6166
0.4	0.6923	**0.6730**	**0.6186**
0.5	0.7070	0.6703	0.6123
0.6	**0.7110**	0.6594	0.6000
0.7	0.7031	0.6375	0.5820

Each cell is one merged model rather than a mean over orders, because the merge is a per-element maximum and therefore invariant to arrival order; on TA8 two different arrival orders give the same value to six decimals, 0.707002. The same pipeline scores the unmerged pre-trained model on TALL20 at 0.5565, the base-model row of §5.5. The grid is reported here, at the coefficient we swept, rather than as a row of the sequential block, because a rule that never reads the running model is not a like-for-like comparison with one that folds the stream.

A.15 Related work

One-shot model merging. RegMean and Fisher merging weight task vectors by statistics computed from data (Jin et al., 2023; Matena & Raffel, 2022) rather than by geometry, and EMR-Merging elects a representative task vector instead of tuning (Huang et al., 2024; Jang et al., 2024). Two recent methods act on the spectrum of the task matrices: task singular vectors flatten them onto a shared basis (Gargiulo et al., 2025) and isotropic merging separates common and task-specific subspaces (Marczak et al., 2025). AdaMerging learns merge coefficients with a gradient objective on unlabelled test samples (Yang et al., 2024b). All of these require the pool at once, the constraint we study rather than assume away; FusionBench provides the pools, evaluator and implementations that make same-protocol comparison possible (Tang et al., 2024).

Continual model merging. MagMax pairs sequential fine-tuning with a merge rule (Marczak et al., 2024): task vectors are formed against the pre-trained model, and the merge keeps the value of largest magnitude element by element. We evaluate that rule from the released implementation, under the same pool and evaluator as every other row of Table 1, not the fine-tuning half of the recipe, which trains on each task's data. OPCM projects each incoming task vector onto the subspace orthogonal to the dominant directions of the accumulated task vector and scales the result adaptively (Tang et al., 2025); null-space filtering (Qiu et al., 2026) also arrives one expert at a time and reads no data at merge, but solves a LoRA gate per arrival rather than one pass of arithmetic, so it leaves the closed-form class §3.3 defines. Every comparison in our tables is a rule we re-ran ourselves under a single protocol, so numbers published under other harnesses are cited but not placed on our scale, and §A.16 measures that offset on a method both works run; adaptive projective gradient descent frames merging as a constrained optimisation solved without data (Wei et al., 2025); and ODE-M models the merge as a trajectory in parameter space driven by a small calibration set at every step (Lin & Kang, 2026). DOP is the closest published work in setting (data-free, sequential, stability and plasticity as orthogonal projections solved by multi-gradient descent), and §3.3 runs it: outside the C1–C2 class because its merge computes gradients, at $\approx 1.9\times$ our merge cost and $\approx 5.5\times$ end to end.

Automated algorithm discovery. Language models have been used to propose and refine programs, scoring candidates by a proxy objective and keeping the best: evolutionary program search over model-written code (Romera-Paredes et al., 2024; Novikov et al., 2025), LLM-driven heuristic design (Liu et al., 2024), black-box optimisation with the model itself as the optimiser (Yang et al., 2024a), and automated search over merging strategies and their hyper-parameters (Su & Geiping, 2025). Our contribution to that line is negative and specific: under an equal budget and a shared warm start a language-model search is no better than a grid over the same hyper-parameters, and the reason is the fitness signal rather than the generator (§5.4). ODE-M attacks the same design gap by modelling the merge as a trajectory in parameter space; its numbers come from a different protocol, so we do not compare against it directly.

A.16 LIMITATIONS

Stream length, and the size of the effect. The operator's advantage is an eight-task result. At fourteen tasks it is ahead of MergeOp-M1 and behind OPCM, and at twenty the repaired operator is indistinguishable from a running average and slightly behind sequential TIES. What carries over is the diagnosis rather than the advantage: at twenty tasks the accumulation branch (our operator and MergeOp-M1 alike) scores below a base model that is never merged, while averaging and projection methods hold near 0.611. The mean gain over the baseline is 0.15 points, small and consistent across orders.

Scope of the evaluation. One backbone, one modality, classification tasks: none of the conclusions has been tested on a language model or on generative tasks, and since the twenty-task pool contains fourteen of the tasks in the fourteen-task pool, the two columns of Table 5 are not independent.

The level of the published class. Two published methods sit in the setting of Eq. 1 at no data: DOP, which we run and price (§3.3), and NUFILT (Qiu et al., 2026), which reports 83.6 on this pool and backbone at eight tasks from its own harness. That number is not placed on our scale, because OPCM is run by both works on the same eight tasks and the two harnesses disagree on it: we report 0.6936 at $\alpha = 0.3$ against 0.755 there, and the gap holds at fourteen and twenty experts (0.6754 against 0.719, 0.6112 against 0.657). A six-point offset on a shared rule means the two harnesses differ by more than the merge rules they compare, so the 0.70 of §5.1 is the level this protocol gives this class rather than a claim about the best number the class can reach.

The constants were searched, not predicted. Every constant but the damping exponent was chosen offline by the proxy of §5.4, and §A.3 shows that proxy to be unreliable: under the official evaluator the argmax of the same family changes, so a better member may exist than the one we froze. The exponent was selected on one block of orders and reported on a disjoint one, which makes it a diagnosis rather than a blind prediction. Three measurements would bound the claim and we have none of them: a caller that supplies a wrong horizon (T is assumed exact everywhere), the traversal order the shared ρ is sensitive to (§4), and per-arrival curves, which would separate what the operator accumulates from what it forgets.

The cell we fill outside our tables. Every number in our tables is either merge-time data-free or carried by an $O(1)$ state; the arm that keeps arrival order but may read data is excluded for those samples, not for a growing state (934.3 MB against 68 bytes for Table 1).