EXACT EXPERT-STATE REUSE FOR EFFICIENT TRAINING-PLAN SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

In multi-expert adaptation, comparing assignments of training examples to experts is costly because each candidate requires a sequence of parameter updates. Different assignments can give an expert the same ordered training history, even after their global plans diverge. Factorized Counterfactual Training (FCT) exploits this repetition by computing each required expert update once and retaining both parameters and optimizer state. With independent expert updates and a fixed numerical program, it preserves candidate states, scores, and deterministic search decisions for the same requests. A controlled four-expert workload reduces updates from 3,124 to 124, with a $25.08\times$ warm speedup over whole-prefix reuse. On SciQ and ARC-Challenge across four language models, FCT avoids 36.8–46.9% of prefix-cached updates and preserves every test prediction. All 24 paired runs are faster, with median speedups of $1.40\times$ to $1.63\times$ including graph preparation and full testing on a resident model. A separate 120-second search on Qwen3-0.6B more than doubles mean candidate counts and improves mean test accuracy by 0.93 and 0.20 percentage points, respectively. These savings make it possible to compare more training plans within a fixed search budget.

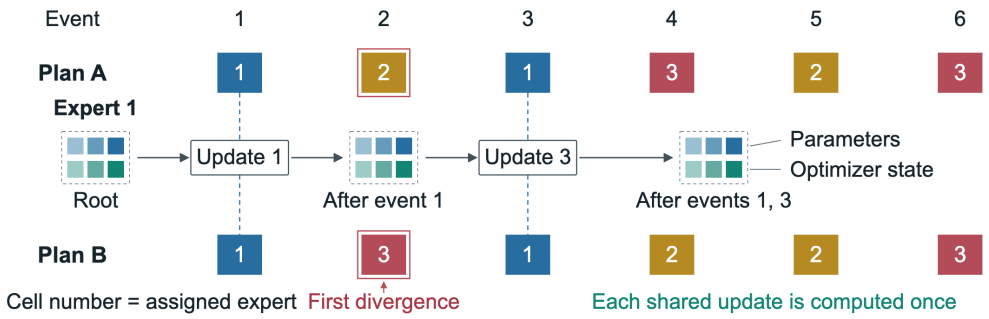

Figure 1: Different training plans can require the same expert update. The plans diverge at event 2, but both assign events 1 and 3 to expert 1. Its parameters and optimizer state can be computed once and reused. Dashed lines link matching event assignments to shared updates; solid arrows show state transitions. Cell colors and numbers identify experts. The two rows in each state icon represent parameters and optimizer state.

1 INTRODUCTION

Choosing how a model learns can require training several alternatives before committing an update. In a multi-expert system, one such choice is which expert receives each new example. Low-rank adaptation (LoRA) makes it practical to maintain several trainable experts on a shared frozen language model (Hu et al., 2021; Dettmers et al., 2023). Recent systems support concurrent adapter training, configuration selection, and agent adaptation (Zhu et al., 2025; Zuo et al., 2026; Zhu & You, 2026; Yao et al., 2026). These developments motivate a concrete computational question: how cheaply can we evaluate several assignments of a fixed training sequence to independent experts?

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.

Training-plan evaluation has dependencies that inference batching alone does not address. Each candidate plan changes the parameters and optimizer state used by later updates. Evaluating a plan therefore requires its actual update sequence, even when another plan starts from the same model. Batched training improves hardware efficiency (Ye et al., 2023; Li et al., 2026); caching a whole-bank prefix avoids repeating an identical initial assignment sequence. However, once two global prefixes diverge, an individual expert can still have exactly the same training history in both. In Figure 1, expert 1 follows the same two updates even though the plans diverge at event 2. Treating the complete expert bank as the unit of reuse repeats that expert's computation.

The relevant dependency is the ordered history of each expert. If an expert starts from the same complete state and receives the same updates in the same order, changes to other experts need not change its result. This observation allows reuse across different global training plans, including plans that share no recent whole-bank prefix. It also specifies what must be retained: parameters alone are insufficient when the optimizer has momentum. The numerical execution matters as well, because changing batch shape or reduction order can change a floating-point update even when the symbolic formula is unchanged.

We develop Factorized Counterfactual Training (FCT) for this setting. Here, counterfactual training means evaluating alternative assignments of already available training events before choosing one. FCT represents each requested expert history by a node containing its complete state, computes each required local transition once, and evaluates only the endpoints requested by the planner. Independent transitions are batched in fixed numerical blocks. This separates two sources of efficiency: reusing a transition reduces the amount of training, while concurrent execution accelerates the transitions that remain.

Our evaluation connects this dependency structure to execution cost and search utility. Controlled workloads test candidate scale, equal expert budgets, three model families, and mixed precision. SciQ and ARC-Challenge adaptation across four models yields median speedups of $1.40\times$ to $1.63\times$ with identical predictions in all 24 paired runs. A separate equal-time search on Qwen3 evaluates more candidates and improves mean accuracy on both tasks at the primary 120-second budget. A native LoRAFusion comparison (Zhu et al., 2025) distinguishes speed from output agreement across numerical programs.

FCT evaluates training plans by their distinct expert updates. We establish the complete-state factorization, count its required transitions, and implement demand-driven reuse with fixed numerical blocks. We also show that a deterministic adaptive planner retains the same decisions for the same requests. Across the measured high-reuse families, the savings extend beyond whole-prefix caching and persist with equal expert budgets. The number of global plans can therefore substantially overstate the number of distinct expert updates needed to compare them.

2 Related Work

Multi-adapter systems reduce the cost of maintaining many specialized models. LoRA trains low-rank parameter updates, while QLoRA reduces the memory required by the frozen model (Hu et al., 2021; Dettmers et al., 2023). S-LoRA and Punica organize concurrent adapter inference (Sheng et al., 2023; Chen et al., 2023a). For training, mLoRA uses pipeline parallelism, LoRAFusion combines operator and scheduling optimizations, and tLoRA shares a super-model across adapter workloads (Ye et al., 2023; Zhu et al., 2025; Li et al., 2026). These systems motivate efficient execution of many adapters. Our reuse rule concerns a different object: an entire expert-state transition that appears in multiple candidate training plans. We therefore compare both batched replay and whole-prefix reuse under a common backend, then evaluate a native LoRAFusion path separately.

Training reuse has substantial prior work in model selection. Nautilus materializes outputs of reusable frozen layers and fuses shared computations across transfer-learning trials (Nakandala & Kumar, 2022). Its trainable branches retain separate optimizers. Cerebro schedules model-selection training across data partitions (Nakandala et al., 2020), while AdaBridge combines data and computation reuse during multi-task co-evolution (Wang et al., 2024). FCT applies reuse after an update history has made both the trainable parameters and optimizer state identical. This permits merging repeated local training transitions even when the global assignments differ. The distinction is the state dependency being reused, not the general idea of caching or sharing computation.

Training-plan selection is related to, but narrower than, general optimization of a learning procedure. Hyperband allocates resources among candidate configurations (Li et al., 2016), and ALTO combines adaptive LoRA tuning with heterogeneous workload orchestration (Zuo et al., 2026). Data-selection methods use validation information or learned data relationships to choose training examples (Ren et al., 2018; Xia et al., 2024; Chen et al., 2023b); meta-learning optimizes models for subsequent adaptation (Finn et al., 2017). These methods change which training is performed. FCT keeps the planner's requested training outcomes fixed and changes their execution. Candidates with different update rules can share a transition only when the complete transition inputs and numerical semantics agree.

Continually adapting agents provide a motivation for evaluating updates before deployment. OpenTinker separates concerns in agentic reinforcement learning, including training and rollout execution (Zhu & You, 2026). MERA studies experience-based model and skill adaptation with routing (Yao et al., 2026). Our measured workload uses fixed public examples and independent expert states. It captures the cost of comparing learning assignments on arrived data, without assuming that future agent observations are independent of those assignments. This boundary will matter when applying state reuse to a planner whose actions also generate its training data.

3 Exact Reuse of Expert Training Histories

3.1 Training Plans and Complete State

We consider K experts sharing a frozen backbone W and a fixed sequence of T training events $z_1, \dots, z_T$. An event specifies its data, loss, and update configuration. A plan $r = (a_1, \dots, a_T)$ assigns event t to expert $a_t \in \{1, \dots, K\}$, optionally allowing $a_t = 0$ to skip the update. Each expert has a complete local state $s_i = (q_i, v_i)$, where q_i contains its LoRA parameters and v_i its momentum. Its update is

$$g = \nabla_{q_i} \ell(W, q_i; z_t), \qquad v_i^+ = \mu v_i + g, \qquad q_i^+ = q_i - \eta v_i^+. \tag{1}$$

Only the selected expert changes. We use $\mu = 0.6$ and $\eta = 0.02$ in the experiments. Other stateful optimizers require their corresponding moments and counters to be included in s_i; Adam, for example, carries two moment estimates (Kingma & Ba, 2014).

An expert's local history records the ordered event indices assigned to it. Write $h_i(r) = (t : a_t = i)$ in increasing order, with empty history $\varnothing$. The transition operator $U_{i,t}^{\nu}$ includes the fixed numerical program ν: precision, loss reduction, batch geometry, and update ordering. Starting from root s_i^0, the state associated with history $h = (t_1, \dots, t_m)$ is

$$S_i(h) = U_{i,t_m}^{\nu} \circ \cdots \circ U_{i,t_1}^{\nu}(s_i^0), \qquad S_i(\varnothing) = s_i^0. \tag{2}$$

The data and update program are fixed within a reuse namespace. Any randomness affecting a transition must likewise be fixed by that namespace and its event history.

Proposition 1 (Complete-state factorization)**.** *Suppose every update reads only the selected expert's complete state, its prescribed event, and immutable shared inputs. If each $U_{i,t}^{\nu}$ is deterministic, the terminal bank for plan r is*

$$S(r) = \big(S_1(h_1(r)), \dots, S_K(h_K(r))\big). \tag{3}$$

Two plans with the same history for expert i therefore have identical complete states for that expert.

The proof follows the event order. A nonselected expert keeps its state; a selected expert applies the same transition to the same local predecessor. Induction gives Equation 3. This is a statement about the specified numerical program, so its consequence is zero-tolerance numerical equality, rather than proximity under a chosen error threshold. Appendix A gives the full proof and explains why omitting optimizer state invalidates the reuse rule.

3.2 Demand-Driven Training and Endpoint Evaluation

FCT computes the union of the local histories needed by the requested plans. For a candidate family $\mathcal{R}$, define $H_i = \{h_i(r) : r \in \mathcal{R}\}$ and let $\mathrm{Pref}(H_i)$ contain every prefix of these histories. A

nonempty node $h \parallel t$ depends only on node h and event t. The required update count is

$$N_{\mathrm{FCT}}(\mathcal{R}) = \sum_{i=1}^{K} \big(|\operatorname{Pref}(H_i)| - 1\big). \tag{4}$$

The cache key includes the expert, root identity, ordered history, and update program. Roots are distinct even when experts have the same architecture. No merging is inferred from approximately equal parameters or equal validation scores.

Each request expands only missing dependencies. FCT traverses a requested history back to the nearest stored prefix, then schedules its missing transitions in dependency order. A ready transition produces one new complete state. Different ready transitions may execute together because they read immutable predecessors and write separate outputs. Candidate banks are assembled by referring to their terminal expert states, which avoids copying or retraining an entire bank whenever one expert changes. A stored state remains available for later requests from the same planner.

Endpoint evaluation is separate from training dependency expansion. Let $E_i(h; V)$ contain deterministic outputs on a fixed evaluation set V, such as losses or option probabilities. FCT computes these outputs only when a request needs history h as an endpoint, and caches them under the state and evaluation-set identities. Training ancestors still require updates, but need evaluation only if requested as endpoints. Our whole-prefix baseline uses the same endpoint policy.

Candidate scores are reconstructed from the requested endpoint vectors. In the fused search, we maximize negative mean cross-entropy,

$$F_V(r) = -\frac{1}{K|V|} \sum_{i=1}^{K} \sum_{x \in V} E_i(h_i(r); V)_x. \tag{5}$$

The public-task search instead averages expert option probabilities and scores the resulting ensemble on validation data. This nonadditive score matches its deployed readout (Appendix I). Neither score feeds into an expert's update. Both therefore retain the local training dependencies in Equation 3.

3.3 How Much Training Can Be Reused?

The complete assignment family provides an exact count of the available reuse. With a skip action, it contains $(K+1)^T$ plans. Independent replay performs $TK(K+1)^{T-1}$ updates across these plans. Whole-prefix reuse performs $(K+1)^T - 1$ updates, counting only non-skip edges. Every subset of event indices can be an expert's local history, so FCT performs $K(2^T - 1)$ updates. These counts describe the same candidate outcomes under three execution organizations. They do not change the number of requested plans.

Equal-budget families retain substantial reuse without relying on different amounts of training. If expert i must receive exactly c_i events and $\sum_i c_i = T$, the number of plans is $T!/\prod_i c_i!$. Its distinct local prefixes of length j number $\binom{T-c_i+j}{j}$. Consequently,

$$N_{\mathrm{FCT}} = \sum_{i=1}^{K} \sum_{j=1}^{c_i} \binom{T - c_i + j}{j}. \tag{6}$$

For quotas $(2, 2, 1, 1)$ over six events, 180 plans need only 52 FCT updates, compared with 522 for whole-prefix reuse. Their requested terminal expert histories number 42. Appendix B derives these expressions and separates update counts from endpoint counts.

The additive objective also permits exact subset allocation once local endpoint costs are available. A dynamic program assigns disjoint event subsets to successive experts, using each trained endpoint's loss as its cost. On the saved 180-plan family, this computation recovers the same optimum from 42 endpoint histories (Appendix B.4). It is an alternative selector, not a replacement for the training that produces those costs. Our paired comparisons hold requests fixed to measure execution reuse independently of the choice between enumeration, beam search, and exact allocation.

3.4 Fixed Numerical Blocks and Adaptive Requests

Exact reuse requires batching to preserve the transition being reused. We execute four independent expert lanes per numerical block, keeping examples per lane, tensor shapes, and per-expert reductions fixed within a paired setting. Unused lanes do not affect useful lanes. The scheduler adjusts concurrent blocks without changing a transition's numerical batch size. It retains captured execution graphs, reduces concurrency under memory pressure, and retries unfinished work after a failed allocation.

The fused implementation combines official LoRAFusion forward and input-gradient operations with our fixed-shape parameter-gradient reductions. Full-model backpropagation still passes through the frozen backbone to the trainable projections. PyTorch supplies model execution and automatic differentiation, and Triton supplies the fused operator infrastructure (Paszke et al., 2019; Tillet et al., 2019). This implementation defines the common numerical program for batched replay, whole-prefix reuse, and FCT. The native LoRAFusion adapter manager and optimizer define a separate comparison because their execution can produce different floating-point states.

Adaptive planning preserves the same reuse guarantee when its decisions are deterministic. If a planner chooses requests from previous scores with a fixed tie rule, equal scores induce the same next request. Proposition 1 then applies again. Induction preserves the request sequence and final choice at a fixed request limit. With a time limit, executors may complete different lengths of the same sequence and select different plans. The candidate family need not be known in advance, but matching histories must retain identical training events; branch-dependent data generation lies outside the factorization as stated.

4 Experiments

4.1 Experimental Setup

We evaluate the cost of reproducing prescribed training-plan outcomes. Controlled workloads use Qwen2.5-0.5B-Instruct, Qwen3-0.6B, and SmolLM2-135M-Instruct (Qwen Team, 2024; Yang et al., 2025; Allal et al., 2025). Each bank has four rank-two query/value LoRA experts, scaling factor two, and no dropout. The input panels contain eight public training examples each from SciQ and ARC-Challenge (Welbl et al., 2017; Clark et al., 2018) (Appendix D). Separate adaptation studies evaluate these three models and OLMo2-1B-Instruct (Team OLMo et al., 2024) on both complete official test sets.

The paired baselines separate batching from state reuse. Batched replay executes complete candidate training sequences using the same numerical blocks as FCT. Whole-prefix reuse stores states after common global assignment prefixes, with endpoint-only evaluation. Both share the model, roots, input order, update rule, and numerical backend with FCT in each paired run. The native comparison uses the official LoRAFusion multi-adapter path, adapter manager, and stochastic gradient descent (SGD) optimizer on one device, with a model-layout integration. It is measured on matched fixed requests, without claiming reproduction of the distributed system.

We distinguish execution speed from setup and storage costs. Warm execution uses prepared graphs; the fused resident timer includes host scheduling and transfers but excludes graph preparation, numerical comparison, and checkpoints. Cold-pool and public-task timers include method-specific graph preparation on a resident model. Ratios divide baseline time by FCT time. Controlled medians use three paired timing repeats on the same data; public-task medians use three initialization and data-sampling seeds, with cyclic method order. All runs use one accelerator with 80 GB memory; environment metadata accompanies the tables.

4.2 Candidate Scale and Equal Expert Budgets

FCT's measured advantage grows with the amount of repeated local history. For four experts and three, four, or five events, the complete candidate families contain 125, 625, and 3,125 plans. Whole-prefix reuse requires 124, 624, and 3,124 updates, while FCT requires 28, 60, and 124. The corresponding warm time ratios are $3.85\times$, $10.02\times$, and $25.08\times$ (Figure 2). Candidate states, evaluation

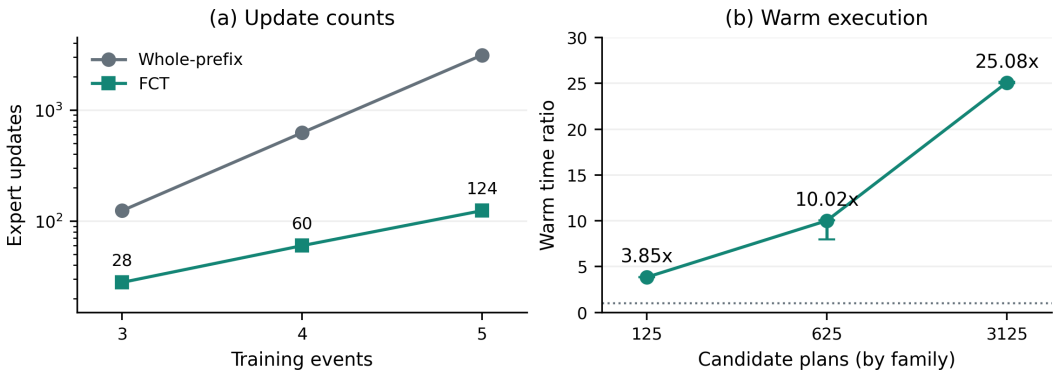

Figure 2: Local-history reuse reduces work as the candidate family grows. Left: exact update counts for four experts with an optional skip action. Right: warm whole-prefix/FCT time ratios on Qwen2.5-0.5B; positions denote the three candidate families. Markers show medians and error bars the minimum and maximum of three paired timing repeats.

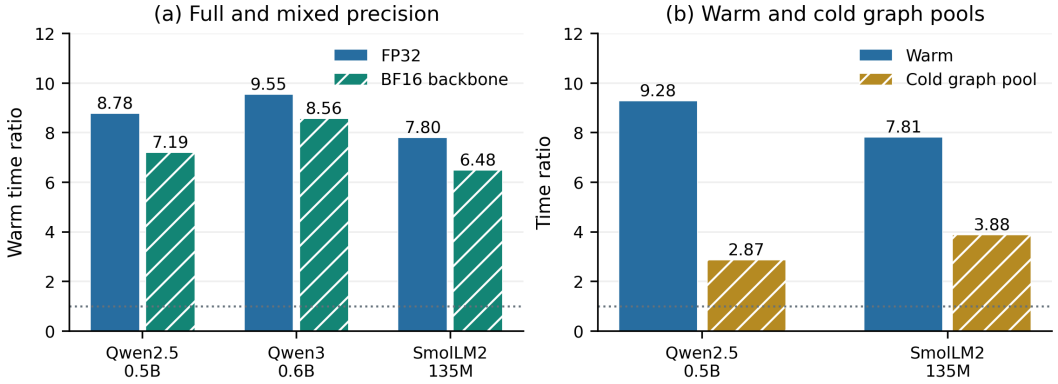

Figure 3: Savings persist with fixed expert budgets and graph preparation. All panels use the 180-plan family with quotas $(2, 2, 1, 1)$. Left: warm execution under FP32 and mixed precision. Right: a separate FP32 comparison includes independently initialized graph pools while the model remains resident. Each bar is a median of three paired baseline/FCT ratios.

columns, scores, and choices agree at zero tolerance within the prescribed program. This result tests reuse after global prefixes diverge, since the baseline already shares identical global prefixes.

The high-reuse result transfers across model families and input panels. Six model-panel combinations yield warm ratios from $8.93\times$ to $10.26\times$ for 625 plans. The second panel changes both examples and initialization. Low-reuse controls remain near parity: a SmolLM2 single-plan control has a warm ratio of $0.97\times$ and a model-time ratio of $1.00\times$ (Appendix E). Repeated training history predicts the benefit.

Equal per-expert training budgets preserve the same advantage. With quotas $(1, 1, 1, 1)$, 24 plans receive one update per expert; with $(2, 2, 1, 1)$, 180 plans give every candidate the same six-update allocation budget. For the latter family, full-precision warm ratios are $8.78\times$, $9.55\times$, and $7.80\times$ across Qwen2.5, Qwen3, and SmolLM2. With a bfloat16 (BF16) frozen backbone and 32-bit floating-point (FP32) expert states, the ratios are $7.19\times$, $8.56\times$, and $6.48\times$. Figure 3 shows that the savings do not depend on allowing some candidate plans to train less.

Graph preparation reduces the measured gains. For 180 plans, cold-pool ratios are $2.87\times$ for Qwen2.5 and $3.88\times$ for SmolLM2, compared with warm ratios of $9.28\times$ and $7.81\times$ in the same experiment. A single-plan control remains near parity. These timers exclude process startup and model download.

Table 1: Resident execution on Qwen3-0.6B with the common fused backend. Times are medians in seconds. Ratios are medians of paired ratios and need not equal ratios of median times. All three methods preserve the measured states, scores, and choices. Graph preparation, numerical comparisons, and checkpoint writes are excluded.

Workload	Batched (s)	Prefix (s)	FCT (s)	Prefix/FCT
One plan	0.334	0.333	0.333	1.01
24 fixed-budget plans	1.417	1.277	0.336	3.80
180 fixed-budget plans	12.577	6.831	0.830	8.38
Validation-driven beam	2.985	0.997	0.716	1.39

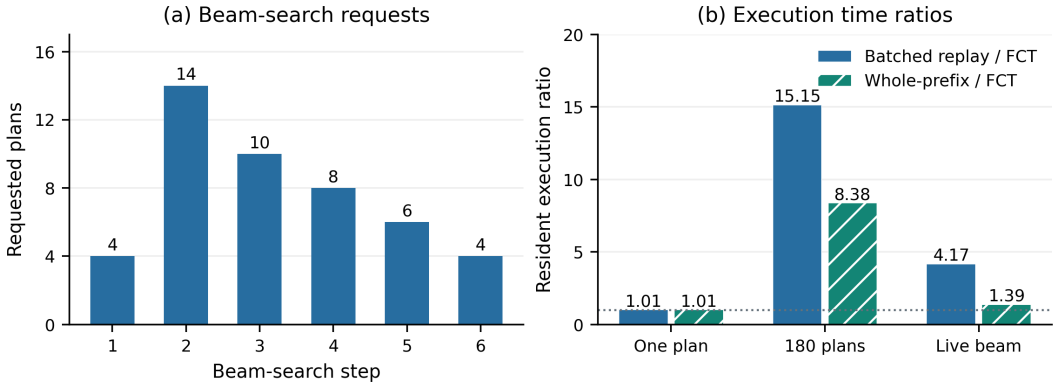

Figure 4: The same reuse mechanism supports adaptive requests. Left: actual plan counts requested by the width-four validation-driven beam. Right: resident execution ratios for a single plan, the 180-plan fixed family, and the complete live beam search under one numerical program. Every requested score and selected plan matches across these methods.

4.3 Fused Execution and Validation-Driven Search

Local-history reuse remains useful on a fused backend. For 180 fixed-budget plans, FCT is $15.15\times$ faster than batched replay and $8.38\times$ faster than whole-prefix reuse under identical numerical semantics (Table 1). One-plan times remain close. Distinct history counts thus continue to matter after operator optimization.

A validation-driven planner exercises reuse under adaptive requests. The planner performs beam search of width four over six arrived training events, with quotas $(2, 2, 1, 1)$. At each step, it evaluates feasible extensions on four local validation examples and keeps the best prefixes, breaking ties lexicographically. It requests 4, 14, 10, 8, 6, and 4 plans over its six steps. All common-backend methods return identical scores and retain the same beam at every step. The selected plan assigns the six events to experts $(4, 1, 3, 1, 2, 2)$ using one-based indices. Its final evaluation uses a disjoint local test panel.

The complete adaptive search is $4.17\times$ faster than batched replay and $1.39\times$ faster than whole-prefix reuse under the resident execution timer (Figure 4). The gain is smaller than exhaustive enumeration because the beam requests fewer repeated local histories. Identical decisions test the adaptive equivalence argument in Section 3; the equal-time study below separately tests task quality.

4.4 Public-Task Adaptation

Actual adaptation requests retain substantial reuse beyond whole prefixes. For each model, task, and seed, four searches assign 256 training examples in eight-example events after 32 initialization examples. Each search commits eight updates with quotas $(2, 2, 2, 2)$. A width-four beam scores the uniform expert probability ensemble on 64 validation examples; the final bank answers all 1,000 SciQ or 1,172 ARC-Challenge test questions. Methods start with independent empty graph pools. Appendices I and J report all 24 outcomes.

Table 2: Full-test adaptation on four backbones. Accuracy is the mean over three seeds; the ratio is the median paired method time. Both search methods have identical accuracy. Round-robin (RR) matches committed updates, not total compute.

Model	Task	Prefix/FCT	Search acc. (%)	RR acc. (%)
Qwen3-0.6B	SciQ	1.45	79.77	77.77
Qwen3-0.6B	ARC-Challenge	1.51	57.82	57.25
Qwen2.5-0.5B	SciQ	1.47	77.73	54.07
Qwen2.5-0.5B	ARC-Challenge	1.45	51.42	50.97
SmolLM2-135M	SciQ	1.47	25.13	25.03
SmolLM2-135M	ARC-Challenge	1.40	30.09	30.12
OLMo2-1B	SciQ	1.61	79.13	79.47
OLMo2-1B	ARC-Challenge	1.63	50.68	49.66

Table 3: Equal-time search on Qwen3-0.6B at 120 seconds. Counts and accuracies are three-seed means; differences use percentage points (pp). Prefix denotes whole-prefix reuse. Appendix K reports all budgets and uncertainty.

Task	Prefix candidates	FCT candidates	Prefix acc. (%)	FCT acc. (%)	Δ (pp)
SciQ	59.7	193.7	79.30	80.23	+0.93
ARC-Challenge	38.3	94.3	56.94	57.14	+0.20

FCT avoids 36.8–46.9% of prefix-cached updates and preserves every selected plan and test prediction. All 24 time ratios exceed one. Model/task medians range from $1.40\times$ to $1.63\times$, including graph preparation, search, and full testing (Table 2). The savings persist across large differences in task accuracy. Search improves six of eight group means over round-robin; the large Qwen2.5 SciQ gap includes a round-robin run that always selects one option (Appendix J.4).

4.5 Equal-Time Adaptation Search

The saved computation also permits more candidate evaluation within a fixed search budget. On Qwen3-0.6B, a best-first planner compares complete 32-event assignments by swapping expert assignments within eight-event blocks, preserving every expert's training quota. It uses the same 256-example adaptation budget and 64-example validation objective as above, with three fresh seeds. The primary 120-second optimizer timer includes cold graph preparation, validation, scheduling, and checkpoint writes. Only complete saved requests ready by the deadline can supply the selected plan; shared setup and subsequent full testing are timed separately (Appendix K).

FCT more than doubles mean candidate counts and improves mean accuracy on both tasks at this budget (Table 3). The six pairs yield four gains and two ties, with no decrease. The smaller prespecified budgets give mixed accuracy changes, and the three-seed intervals include zero. These results support a task benefit from additional search without establishing a general test-accuracy guarantee.

4.6 Comparison with Native LoRAFusion

The native LoRAFusion comparison tests a stronger system path with different numerical execution. We use its official multi-adapter operations, manager, and SGD optimizer, then apply whole-prefix reuse to the matched candidate requests. FCT achieves a $6.79\times$ resident execution ratio on the 180-plan workload and $1.29\times$ on the recorded beam requests (Table 4). For one plan, the ratio is $0.98\times$. The recorded requests come from the actual adaptive search above and are fixed for this cross-backend comparison.

Numerical differences change some native selections (Table 4). Maximum loss-column and candidate-score differences are 3.28 and 0.406; adapted prefix and FCT retain exact agreement. The backends differ in reductions and optimizer execution, without isolating a unique cause. Speed across backends and exactness within a backend therefore answer different questions.

Table 4: Matched requests across numerical programs. The native baseline is the single-device LoRAFusion path with whole-prefix reuse. Adapted prefix and FCT use the common fixed numerical program. Choice agreement counts three paired repeats, not three independent tasks.

Workload	Native/FCT	Adapted prefix/FCT	Native choice matches
One plan	0.98	1.08	3/3
180 fixed-budget plans	6.79	9.75	0/3
Recorded beam requests	1.29	1.49	3/3

Graph-pool policy also affects cost accounting. The fused run shares pools across methods; the public-task studies use independent cold pools. Appendix F reports the shared-pool costs. Neither timer includes process startup or shared model loading.

5 Discussion

Local training histories determine which updates can be shared across plans. Multi-adapter batching improves execution (Ye et al., 2023; Zhu et al., 2025; Li et al., 2026); FCT reduces the number of distinct updates. Plans can disagree about several experts while requiring the same transition for another. Reusing that transition reduces cost without changing the planner's objective.

The central planner compares candidates before committing a bank. It defines the arrived events and requests scores, while reuse follows from identical local training dependencies. Several plans can refer to one read-only state; each subsequent update creates a separate child. This organization keeps unselected candidates from modifying the committed bank.

Complete-state reuse complements optimizations within a training step. Nautilus reuses materializable frozen computations (Nakandala & Kumar, 2022), and efficient attention reduces memory traffic within a computation (Dao et al., 2022). FCT reuses prescribed updates with their optimizer states. The common fused-backend comparison tests reuse alongside operator fusion: scheduling changes while the planner continues to receive the same scores.

Application value depends on the learning procedure's requests. ALTO and Hyperband study training-resource allocation across candidates (Zuo et al., 2026; Li et al., 2016). Our successive-commitment searches avoid 36.8–46.9% of prefix-cached updates across four models without exhaustive enumeration. Under a fixed time budget, the same dependency reduction permits additional candidate evaluation. Its value remains small when a procedure requests only one plan.

Agent adaptation requires tracking dependencies in generated data. OpenTinker and MERA include interactions that generate or validate experience (Zhu & You, 2026; Yao et al., 2026). Our factorization assumes matching histories use identical arrived data. Plan-dependent rollouts, shared trainable parameters, or a global optimizer clock introduce additional dependencies. Extending reuse to such settings requires representing those dependencies in the state.

Faster execution lets the planner compare more training choices. Validation-based reweighting and data selection guide how examples contribute to training (Ren et al., 2018; Xia et al., 2024; Chen et al., 2023b). We study expert assignments: at 120 seconds, Qwen3 search compares more complete candidates and raises mean accuracy on both tasks. More candidates cannot lower the best observed validation score along a common sequence; their test-accuracy benefit remains empirical.

6 Conclusion

Different adaptation plans can share complete expert training histories. FCT computes local transitions once, retains optimizer state, and evaluates requested endpoints on demand. It preserves fixed-request outcomes while reducing cost, with median speedups of $1.40\times$ to $1.63\times$ in four-model full-test adaptation. The equal-time study further shows how those savings can support more candidate evaluation and better measured task outcomes. Ordered local histories provide a practical unit for comparing training plans efficiently.

AI USE STATEMENT

Generative artificial intelligence tools assisted research ideation, mathematical analysis, literature retrieval, experiment-code development, result analysis, manuscript drafting, and revision. Image generation assisted the layout of Figure 1, which was reconstructed as vector artwork from the specified training histories. Quantitative figures were rendered programmatically from recorded measurements. Bibliographic metadata was checked against public scholarly sources, and reported numerical results were traced to saved experiment outputs. Automated checks and model-assisted reviews were used during preparation. The submitting authors are responsible for reviewing and approving the final text, code, citations, and conclusions.

REPRODUCIBILITY STATEMENT

Appendix A gives the mathematical assumptions and proofs. Appendix D specifies model configuration, data construction, numerical execution, and evaluation. Appendix E reports complete timing ratios, including low-reuse measurements, and Appendix F separates execution from graph and storage costs. Appendices I–K provide public-task protocols, all paired outcomes, and deadline accounting. The Supplementary Material provides the complete reproduction code, plotting scripts, result reports for all tables and quantitative figures, and environment metadata.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, et al. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model. *arXiv preprint arXiv:2502.02737*, 2025. URL https://arxiv.org/abs/2502.02737.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-Tenant LoRA Serving. *arXiv preprint arXiv:2310.18547*, 2023a. URL https://arxiv.org/abs/2310.18547.

Mayee F. Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, et al. Skill-it! A Data-Driven Skills Framework for Understanding and Training Language Models. *arXiv preprint arXiv:2307.14430*, 2023b. URL https://arxiv.org/abs/2307.14430.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training Deep Nets with Sublinear Memory Cost. *arXiv preprint arXiv:1604.06174*, 2016. URL https://arxiv.org/abs/1604.06174.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, et al. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Re. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135*, 2022. URL https://arxiv.org/abs/2205.14135.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint arXiv:2305.14314*, 2023. URL https://arxiv.org/abs/2305.14314.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv preprint arXiv:1703.03400*, 2017. URL https://arxiv.org/abs/1703.03400.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, et al. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*, 2021. URL https://arxiv.org/abs/2106.09685.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

Kevin Li, Dibyadeep Saha, Avni Kanodia, and Fan Lai. tLoRA: Efficient Multi-LoRA Training with Elastic Shared Super-Models. *arXiv preprint arXiv:2602.07263*, 2026. URL https://arxiv.org/abs/2602.07263.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv preprint arXiv:1603.06560*, 2016. URL https://arxiv.org/abs/1603.06560.

Supun Nakandala and Arun Kumar. Nautilus: An Optimized System for Deep Transfer Learning over Evolving Training Datasets. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 506–520, 2022. doi: 10.1145/3514221.3517846. URL https://doi.org/10.1145/3514221.3517846.

Supun Nakandala, Yuhao Zhang, and Arun Kumar. Cerebro: a data system for optimized deep learning model selection. *Proceedings of the VLDB Endowment*, 13:2159–2173, 2020. doi: 10.14778/3407790.3407816. URL https://doi.org/10.14778/3407790.3407816.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv preprint arXiv:1912.01703*, 2019. URL https://arxiv.org/abs/1912.01703.

Qwen Team. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to Reweight Examples for Robust Deep Learning. *arXiv preprint arXiv:1803.09050*, 2018. URL https://arxiv.org/abs/1803.09050.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, et al. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. *arXiv preprint arXiv:2311.03285*, 2023. URL https://arxiv.org/abs/2311.03285.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious. *arXiv preprint arXiv:2501.00656*, 2024. URL https://arxiv.org/abs/2501.00656.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Lehao Wang, Zhiwen Yu, Sicong Liu, Chenshu Wu, Xiangrui Xu, and Bin Guo. AdaBridge: Dynamic Data and Computation Reuse for Efficient Multi-task DNN Co-evolution in Edge Systems. *arXiv preprint arXiv:2407.00016*, 2024. URL https://arxiv.org/abs/2407.00016.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing Multiple Choice Science Questions. *arXiv preprint arXiv:1707.06209*, 2017. URL https://arxiv.org/abs/1707.06209.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting Influential Data for Targeted Instruction Tuning. *arXiv preprint arXiv:2402.04333*, 2024. URL https://arxiv.org/abs/2402.04333.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Yuhang Yao, Zeyu Wang, Wanyi Chen, Tongyun Yang, Yuhang Han, Jie Xiao, et al. MERA: Model Evolution and Routing with Skill Adaptation for Agentic Systems at Scale. *arXiv preprint arXiv:2608.10333*, 2026. URL https://arxiv.org/abs/2608.10333.

Zhengmao Ye, Dengchun Li, Zetao Hu, Tingfeng Lan, Jian Sha, Sicong Zhang, et al. mLoRA: Fine-Tuning LoRA Adapters via Highly-Efficient Pipeline Parallelism in Multiple GPUs. *arXiv preprint arXiv:2312.02515*, 2023. URL https://arxiv.org/abs/2312.02515.

Siqi Zhu and Jiaxuan You. OpenTinker: Separating Concerns in Agentic Reinforcement Learning. *arXiv preprint arXiv:2601.07376*, 2026. URL https://arxiv.org/abs/2601.07376.

Zhanda Zhu, Qidong Su, Yaoyao Ding, Kevin Song, Shang Wang, and Gennady Pekhimenko. LoRAFusion: Efficient LoRA Fine-Tuning for LLMs. *arXiv preprint arXiv:2510.00206*, 2025. URL https://arxiv.org/abs/2510.00206.

Jingwei Zuo, Xinze Feng, Zien Liu, Kaijian Wang, Fanjiang Ye, Ye Cao, et al. ALTO: Adaptive LoRA Tuning and Orchestration for Heterogeneous LoRA Training Workloads. *arXiv preprint arXiv:2604.05426*, 2026. URL https://arxiv.org/abs/2604.05426.

APPENDIX CONTENTS

1. Complete-state factorization and adaptive equivalence (Section A).
2. Work counts for full, fixed-budget, and sparse requests (Section B).
3. Executor, numerical blocks, and state storage (Section C).
4. Models, inputs, evaluation, and paired measurements (Section D).
5. Complete timing-ratio results (Section E).
6. Absolute execution times and cost accounting (Section F).
7. Numerical checks and native score perturbations (Section G).
8. Relation to other training optimizations (Section H).
9. Public-task adaptation, successive commitments, and actual requests (Section I).
10. Four-model adaptation, numerical geometry, complete costs, and quality variation (Section J).
11. Equal-time search, deadline selection, complete task results, and uncertainty (Section K).

A COMPLETE-STATE FACTORIZATION

A.1 ASSUMPTIONS AND NOTATION

The factorization follows from explicit read and write dependencies. There are K experts, immutable shared inputs W, and prescribed events $z_1, \ldots, z_T$. Expert i has root s_i^0. A training event assigned to expert i applies $U_{i,t}^{\nu}$ to s_i and leaves every s_j for $j \neq i$ unchanged. An unassigned event leaves the entire bank unchanged. The transition reads no mutable state from another expert. In particular, W contains no parameter updated by any candidate plan.

Complete state means all mutable information that can affect a subsequent transition. For Equation 1, this is the parameter vector and momentum vector. With weight decay, a learning-rate schedule, gradient scaling, or a different optimizer, the corresponding configuration and mutable counters must also be represented. A counter derived deterministically from local-history length need not be stored separately, but its interpretation must remain fixed. A counter that advances when another expert trains introduces an additional dependency.

The numerical program ν defines more than a real-valued update equation. It fixes the precision of inputs, state, and reductions; the shape of the operator invocation; and the order in which arithmetic updates the state. Determinism is assumed for the specified program on its execution environment. Our finite-input numerical checks test this property for the measured workload; the mathematical statement does not establish determinism of every possible backend implementation.

For a global prefix $r_{1:t}$, let $h_i(r_{1:t})$ be the ordered subsequence of event indices assigned to expert i. Write $s_i^t(r)$ for the state produced by direct replay of that prefix. The invariant needed by the executor is

$$s_i^t(r) = S_i(h_i(r_{1:t})) \qquad \text{for all } i \text{ and } t. \tag{7}$$

It identifies states through their causal inputs, without comparing their numerical values to decide whether they may be merged.

A.2 PROOF OF PROPOSITION 1

At $t = 0$, every history is empty and both sides of Equation 7 equal s_i^0. Assume the invariant holds at $t - 1$. If $a_t = 0$, direct replay changes no expert and no history changes, so it continues to hold. If $a_t = j$, every expert $i \neq j$ remains unchanged, as does its history. For the selected expert,

$$s_j^t(r) = U_{j,t}^{\nu}(s_j^{t-1}(r)) \tag{8}$$

$$= U_{j,t}^{\nu}(S_j(h_j(r_{1:t-1}))) \tag{9}$$

$$= S_j(h_j(r_{1:t})). \tag{10}$$

The second equality uses the induction hypothesis and the third uses the definition of S_j. Hence the invariant holds at every step. At $t = T$, collecting all expert states gives Equation 3. If two plans have the same history for expert i, the right-hand sides coincide. Under the deterministic numerical program, so do the computed state values. □

The proof permits interleaving independent transitions in any order that preserves each expert's local order. It does not permit reordering two updates of the same expert. Such updates generally depend on the parameters and momentum produced by their predecessor. FCT therefore shares identical ordered histories, rather than unordered sets of examples. Events are uniquely indexed even when two events happen to contain the same text, because their configuration or prescribed position may differ.

A.3 Why Parameters Alone Are Insufficient

Equal parameters do not imply equal future training states. Consider two states with the same parameter q and different momentum values $v \neq \widetilde{v}$. For the same event, the gradients at q agree, but Equation 1 gives

$$q^+ - \widetilde{q}^+ = -\eta\mu(v - \widetilde{v}). \tag{11}$$

With nonzero learning rate and momentum, the next parameters differ. A parameter-only cache would therefore substitute a different transition outcome. The same reasoning applies to first and second moment estimates in Adam (Kingma & Ba, 2014). Complete-state reuse avoids this ambiguity by preserving the optimizer variables alongside the parameters.

Equality of validation loss is even weaker than equality of parameters. Two experts can have identical losses on a finite panel and different gradients on the next training event. FCT never uses a score, rounded tensor, or distance threshold as a merge criterion. It uses a sufficient dependency description: root, expert, numerical program, and ordered events. This rule may retain separate histories whose states happen to coincide, but it does not need to discover those coincidences to obtain the stated savings.

A.4 Adaptive Request Equivalence

Corollary 1 (Deterministic planner)**.** *Let a planner's next request be a deterministic function of its initial configuration and all previous returned scores. Suppose it uses a fixed tie rule, and every requested endpoint is evaluated by the same deterministic evaluation program. Under Proposition 1, replay and FCT induce the same request sequence, scores, and final selected plan.*

The initial request is the same because the planner's configuration is the same. For that request, complete-state factorization gives the same endpoint states; deterministic evaluation and aggregation give the same scores. Assume both executions have returned the same score sequence through request n. The planner's deterministic transition then produces the same request $n + 1$. Applying the same argument to that request extends score equality. Induction gives the full request sequence and final selection. □

The fixed-request native experiment does not invoke this corollary across backends. Its inputs are the requests recorded by the common-program beam search. Replaying those requests under a different numerical program keeps the workload comparable even if some scores change. It does not reconstruct the requests that the native backend would have generated in a live search. The main text consequently reports native score and choice differences separately.

B Work Counts and Reuse Structure

B.1 The Full Assignment Family

The complete family includes every plan in $\{0, \ldots, K\}^T$. There are $(K + 1)^T$ plans. At any fixed position, $K(K + 1)^{T-1}$ plans assign a real expert, so independent replay uses

$$N_{\text{replay}} = TK(K + 1)^{T-1}. \tag{12}$$

This expression counts logical expert updates, not padded physical lanes or model forward calls. The skip action changes no local state and contributes no gradient calculation.

Whole-prefix reuse computes each non-skip edge of the global prefix tree once. At depth t, there are $(K+1)^{t-1}$ parent prefixes and K non-skip outgoing edges from each. Therefore,

$$N_{\text{prefix}} = \sum_{t=1}^{T} K(K+1)^{t-1} = (K+1)^T - 1. \tag{13}$$

All these prefixes belong to at least one requested terminal plan. Skipped events may leave two banks numerically identical, but a prefix-keyed executor does not additionally identify all local-state dependencies across such prefixes.

An expert's possible local histories are exactly the subsets of $\{1, \dots, T\}$, in increasing order. Every subset occurs because its complement can be skipped or assigned to other experts. A nonempty history has one final transition, and all its earlier prefixes also occur. Summing over experts gives

$$N_{\text{FCT}} = K(2^T - 1). \tag{14}$$

An expert-history endpoint can be empty, so evaluating every distinct terminal history requires $K2^T$ columns. Each column contains losses over the fixed evaluation panel. The whole-prefix executor evaluates $N_{\text{prefix}} + K$ distinct endpoint references in this family.

Table 5: Exact logical work for full families with four experts and a skip action. Endpoint counts include untrained roots. Independent replay is shown to separate candidate duplication from global-prefix duplication.

Events	Plans	Replay updates	Prefix updates	FCT updates	FCT endpoints
3	125	300	124	28	32
4	625	2,000	624	60	64
5	3,125	12,500	3,124	124	128

The count separation is structural, but it is not a hardware speedup theorem. Update cost can depend on event length, model size, numerical program, and the number of ready operations. In addition, small requests may be dominated by scheduling and data movement. The measured scale experiment keeps the update semantics fixed and tests whether the logical savings survive these execution costs.

B.2 Fixed Per-Expert Budgets

The quota family removes the skip action and fixes each expert's number of updates. For quotas $c_1, \dots, c_K$ summing to T, the number of complete plans is the multinomial coefficient $T!/\prod_i c_i!$. Expert i has $\binom{T}{c_i}$ possible terminal histories. A prefix $(t_1, \dots, t_j)$ is feasible precisely when it can be extended by $c_i - j$ later indices, which requires $t_j \le T - c_i + j$. Thus the number of feasible length-j prefixes is

$$\binom{T - c_i + j}{j}. \tag{15}$$

Summing these nonempty prefixes proves Equation 6.

For quotas $(2, 2, 1, 1)$, each of the first two experts has five length-one prefixes and 15 length-two histories. Each remaining expert has six length-one histories. Hence FCT requires $2(5{+}15){+}2(6) = 52$ updates and $2(15) + 2(6) = 42$ terminal columns. A length-one history ending at event 6 is infeasible for a two-update expert because there is no later event to complete its quota. This explains why counting every subset up to the quota would overestimate the required ancestors.

Whole-prefix work can be counted by summing feasible prefix multinomials. If a prefix has used n_i updates of expert i, its multiplicity is $(\sum_i n_i)!/\prod_i n_i!$. Summing over $0 \le n_i \le c_i$, excluding the empty vector, gives the number of global nonempty prefixes. It is 64 for $(1, 1, 1, 1)$ and 522 for $(2, 2, 1, 1)$. The corresponding terminal evaluation references are 64 and 452. These references retain global-prefix identity, whereas FCT groups endpoints by their complete local history.

B.3 Sparse and Adaptive Requests

For an arbitrary request family, its actual history prefix closure determines the work. A single plan provides no repeated training transition within the plan itself. Sparse candidate sets can contain

Table 6: Logical work under fixed expert training budgets. The first pair of columns counts training transitions; the second pair counts requested endpoint evaluations. All methods evaluate the same candidates.

Quotas	Plans	Prefix updates	FCT updates	Prefix endpoints	FCT endpoints
$(1, 1, 1, 1)$	24	64	16	64	16
$(2, 2, 1, 1)$	180	522	52	452	42

a few shared histories, but their reuse need not resemble the complete family. Equation 4 applies directly without extrapolating from the number of global candidates. The sparse measurements in Section E therefore remain separate from the complete-family scale curve.

Adaptive requests add missing dependencies incrementally. If $\mathcal{R}_1, \ldots, \mathcal{R}_n$ are the requests observed so far, FCT needs the prefix closure of their union. A history already computed for an earlier request needs no further update. Evaluation caching also carries across requests when both the state and evaluation panel are unchanged. In the measured beam search, FCT performs 23 training updates and 27 validation endpoint evaluations. Whole-prefix reuse performs 46 and 50, while batched replay performs 148 and 139 under the recorded request semantics. The final selected-bank test evaluation is tracked separately.

Candidate count alone cannot estimate adaptive speedup. A narrow beam can repeatedly revisit related local histories, while a broader search can generate mostly new histories. The sequence of feasible requests also determines which independent transitions are available to execute concurrently. Both factors are recorded in the actual request trace. This is why the paper reports a measured $1.39\times$ whole-prefix ratio for the live beam instead of predicting it from the exhaustive-family ratio.

B.4 Exact Allocation for an Additive Objective

The fused workload's terminal score separates over experts. For a feasible event subset A of size c_i, let $L_i(A)$ be the sum of expert i's validation losses after training on the indices in A in their prescribed order. A complete quota plan is a partition of $\{1, \ldots, T\}$ into subsets of sizes $c_1, \ldots, c_K$. Consequently, minimizing the sum of L_i is an exact subset-allocation problem once these local costs have been computed.

Define $D_i(M)$ as the minimum total loss for assigning the events in M to experts 1 through i. The initialization is $D_0(\varnothing) = 0$, with all other states infeasible. The recurrence is

$$D_i(M) = \min_{\substack{A \subseteq M \\ |A| = c_i}} \{D_{i-1}(M \setminus A) + L_i(A)\}. \tag{16}$$

The final value is $D_K(\{1, \ldots, T\})$. Each complete partition appears in this recurrence, and each feasible recurrence path defines such a partition. Induction over i therefore proves optimality. A specified secondary ordering can resolve ties.

We checked this recurrence on the archived 180-plan fused workload without new model execution. Its 42 distinct expert endpoint histories provide all the required costs. Exact rational sums of the saved loss values yield the same minimum as explicit enumeration in all three timing repeats. The selected plan is $(1, 2, 1, 4, 2, 3)$ using one-based expert indices, and it also matches the saved floating-point score maximizer. The subset computation examines 141 feasible transitions in each repeat. These are host-side allocation transitions, not expert-training updates.

This calculation clarifies the role of the executor. The allocation algorithm can avoid explicit enumeration of global terminal scores, but obtaining every required $L_i(A)$ still entails training the corresponding complete local histories. FCT's prefix closure supplies those costs with 52 expert updates. A selector that already uses the same complete-state memoization would share this training benefit. The beam experiment instead evaluates an adaptively requested subset of histories, which can reduce the upfront endpoint work but need not find the complete family's optimum. We do not claim the beam is the best selector for an additive score.

More general bank-level scores may not admit Equation 16. Complete-state factorization still provides the terminal bank under the stated update assumptions, after which the requested bank-level

evaluation must be performed. The fused workloads use the additive score. The scale, cross-model control, quota, cold-pool, and mixed-precision workloads instead average the minimum expert loss per example. The public-task studies use the nonadditive probability-mixture objective in Equation 23. All preserve local update dependencies, while the subset-allocation recurrence applies to the additive case.

C Executor and Numerical Implementation

C.1 Demand-Driven State Construction

The executor maintains separate maps for states, evaluation columns, and global requests. State entries are immutable pairs (q, v) associated with an expert and a local history. Evaluation entries also include their evaluation-panel identity. A global plan stores references to its terminal expert states. This representation preserves the distinction between a candidate bank, a trainable local state, and the observation returned to a planner.

The following pseudocode gives the dependency logic. It omits batching, which does not change the parent relation, and uses `root` to denote the fixed complete initial state for the current expert.

```
state(expert, history):
    if history is empty:
        return root[expert]
    key = (expert, history, update_program)
    if key is stored:
        return states[key]
    parent = state(expert, history[:-1])
    child = update(parent, event[history[-1]])
    states[key] = child
    return child

evaluate_request(plans, panel):
    for plan in plans:
        for expert in experts:
            history = assigned_events(plan, expert)
            endpoint = state(expert, history)
            key = (expert, history, update_program, panel)
            if key is not stored:
                columns[key] = evaluate(endpoint, panel)
        scores[plan] = aggregate_endpoint_columns(plan)
    return scores
```

The batched executor replaces recursive calls by a ready list. It discovers missing nodes, records their predecessor references, and groups nodes whose predecessors are available. After a batch finishes, its outputs become available to their children. Every logical transition is marked complete only after its output state has been stored. Completed states and evaluation columns are restored after an interruption; an incomplete request or unfinished batch is retried. This changes repeated execution cost, not the mathematical state associated with a node.

Complete-state storage scales with the number of retained histories. If an expert has P trainable scalars and FP32 parameters and momentum, one state pair requires $8P$ bytes before container overhead. Keeping all states for N nonempty nodes and K roots uses $8P(N + K)$ bytes when experts have the same parameter count. This is a host-state storage expression. Device-resident model weights, activations, captured graphs, and transfer buffers have additional costs. We do not infer peak device memory from this expression.

C.2 Fixed Blocks and Concurrency

A numerical block always has four physical expert lanes. A useful lane contains one complete state and one prescribed training event, or one endpoint evaluation. Unused lanes are populated with valid

inputs but their outputs are ignored. Loss reduction is performed within each expert's examples and then summed across experts for gradient calculation. No gradient of one expert is added to another expert's state. These details are necessary for a useful lane's update to remain independent of its companion lanes.

The executor separates retained graph capacity from active concurrency. Graph capacity records how many captured blocks are available. Active concurrency records how many are launched for the current ready list. Reducing active concurrency leaves the captured blocks available for later requests. A memory-pressure response may release unused capacity; allocation failure reduces the unfinished launch and retries it. Previously completed nodes are not recomputed merely because a later launch needs fewer blocks.

This separation addresses a measured overhead of repeatedly preparing execution graphs. The released-capacity policy discards capacity when the active workload shrinks, causing later requests to prepare it again. Retaining capacity reduces graph capture from 449 to 104 captures in the corresponding recorded invocations, while preserving their saved outputs. Total invocation times are 554.71 and 168.46 seconds, respectively. These two invocations are an implementation diagnostic rather than a randomized causal timing experiment. Both timing catalogues are retained below.

C.3 Low-Rank Update and Full Backpropagation

Each adapted projection applies the usual low-rank update $W + \alpha BA/r$, with rank $r = 2$ and $\alpha = 4$ (Hu et al., 2021). Query and value projections are adapted throughout the pretrained model. The backbone weights are frozen, but the computation through the backbone remains differentiable with respect to the adapter parameters. Freezing the backbone therefore removes its parameter updates; it does not remove the intermediate computations required to propagate adapter gradients.

The mixed-precision setting stores backbone weights in BF16 while retaining expert master parameters, momentum, and loss accumulation in FP32. The fused path uses the official forward and input-gradient operations with fixed-shape matrix-multiplication parameter-gradient reductions. Companion rotations, repeated execution in the same graph slot, and execution in different graph slots were checked before the complete planning comparison. These tests establish finite-workload consistency for the implemented program. They do not imply equality with full-precision execution or the native reduction program.

C.4 Native Single-Device Integration

The native comparison uses the official LoRAFusion parameter-efficient fine-tuning patch, multi-adapter automatic differentiation, adapter manager, and SGD optimizer. Four physical adapters are registered across the query and value projections of Qwen3-0.6B. The integration maps the model's parameter layout to the official path and runs with degree-one execution, without distributed scheduling. Dropout remains zero. The requested training assignments and evaluation panels are identical to those used by the adapted comparison.

Native state is restored through the native manager before the prescribed local computation. The experiment records 1,812 physical adapter optimizer steps and registration of all 56 adapted projections. This verifies that the comparison invokes the native training path, rather than assigning its name to an unfused replay baseline. Its arithmetic nevertheless differs from the adapted program. Accordingly, native state and score differences are outcomes to report, not values to discard through an acceptance threshold.

D Controlled-Workload Protocol

D.1 Models and Expert Initialization

The three backbones cover two Qwen families and a smaller SmolLM2 model (Qwen Team, 2024; Yang et al., 2025; Allal et al., 2025). Their public model identifiers are listed below. Each uses four LoRA experts with rank two, scale two, zero dropout, and no trainable biases. All experiments retain the pretrained language-model computation.

- `Qwen/Qwen2.5-0.5B-Instruct`
- `Qwen/Qwen3-0.6B`
- `HuggingFaceTB/SmolLM2-135M-Instruct`

Root states are shared exactly within each paired comparison. In the fused search, the initial adapter parameters first receive one prescribed warm update, with separate input examples in the four lanes. The resulting parameters and nonzero momentum form the stored roots used by all methods. The six searched events begin from these roots. This initialization ensures that the comparison includes meaningful optimizer state, rather than relying on all-zero momentum throughout the workload.

The scalar update uses learning rate 0.02 and momentum 0.6, with no dropout or plan-dependent optimizer schedule. These values define the measured training procedure; the experiments do not search them for a favorable score. Different public input panels also use their prescribed initialization seeds. The second-panel comparison changes examples and initialization jointly, so its effect cannot be attributed to one of those changes alone.

D.2 Input Panels and Local Splits

Input panels are formed from the public training splits of SciQ and ARC-Challenge (Welbl et al., 2017; Clark et al., 2018). Each panel selects eight examples from each source at a fixed offset. The scale workload uses offset 100; cross-model workloads use offsets 200 and 300. Panel indices 0 through 7 denote SciQ examples, and 8 through 15 denote ARC examples. The main mixed-precision and fused comparisons use the offset-200 panel. These small panels make many exact alternative training histories computationally accessible.

For the fused search, the six training indices are $(0, 8, 1, 9, 2, 10)$ in that order. Validation uses $(4, 5, 12, 13)$, and final local testing uses $(6, 7, 14, 15)$. The warm root update uses $(0, 8, 1, 9)$ across the four experts. Validation and testing are disjoint from the searched training events, but all are local partitions of public training data. Earlier scale and quota workloads use their recorded training and eight-example evaluation panels. The reproducibility records preserve these distinctions.

Each example is formatted as a question followed by lettered answer choices and an answer prompt. The model's chat template produces the input sequence, without enabling a reasoning mode. The target is the single token corresponding to the correct option letter. Cross-entropy is computed from the final-position full-vocabulary logits, rather than renormalizing only over the option letters. Left padding uses a fixed panel length, with explicit attention masks and position indices. Labels are required to encode as a single token in the chosen tokenizer.

D.3 Candidate Families

The complete families enumerate all assignments to four experts and an optional skip action over three, four, or five events. The quota families enumerate all assignments with counts $(1, 1, 1, 1)$ or $(2, 2, 1, 1)$ and contain 24 or 180 plans. A single-plan control measures an instance with no within-request training reuse. Additional sparse sets retain the original smaller request families without expanding them into full enumeration. Their timing results are reported independently.

The beam planner uses width four and quotas $(2, 2, 1, 1)$ over six events. At each step, it extends every retained prefix by each expert whose quota remains available. It evaluates all feasible extensions, orders them by negative mean validation cross-entropy, and breaks ties by the assignment sequence. The top four survive, or all candidates survive when there are fewer than four. Only the selected terminal bank is evaluated on the local test panel. FCT changes the executor used for these requests, not the search objective or tie rule.

The native comparison fixes the requests observed in this beam run. Every method receives the same candidate list at each recorded step. This protocol separates execution differences from the possibility that different numerical scores would have produced a different later candidate list. Table 4 reports final selection agreement on the recorded requests. It does not compare two independently evolving native and adapted beams.

D.4 Timing and Numerical Checks

Each controlled condition has three paired timing repeats. Methods use the same root states and inputs within a repeat. Later fused comparisons cycle the method order across repeats. Medians summarize observed paired ratios; a ratio of medians is a different statistic. The tables explicitly identify the timer, because model execution, warm request execution, cold graph preparation, and partially accounted invocation costs cannot be pooled into one speedup estimate. The separate public-task learning seeds and full-test protocol appear in Appendix I.

Numerical checks compare full local parameter and momentum tensors for matching histories during execution. Endpoint columns, all candidate scores, and selections are retained in the result records. Selected state pairs are also archived for comparison after the run. The saved state pairs do not include every intermediate gradient or every full trajectory. We therefore distinguish online full-state comparisons from the finite saved-state checks reported in Section G.

Device-utilization samples are diagnostic measurements, not a substitute for elapsed time. They mix intervals of model execution, scheduling, graph preparation, comparison, and storage. The mean of irregularly sampled utilization values is not a time-weighted occupancy integral. The executor adapts concurrency to available memory and useful-work throughput, but the measured invocations do not establish sustained 95% utilization. The manuscript's efficiency claims are based on paired elapsed-time measurements.

E Complete Timing-Ratio Results

The following tables retain all 196 recorded summary ratios from the consolidated system measurements, including values below one. Model denotes the original model-computation timer. Warm denotes execution with prepared graphs; execution denotes the later fused resident-request timer. Cold includes an independently initialized graph pool with a resident model. Accounted denotes a ratio of recorded partial cost sums in shared evolving pools. It is not a full-process speedup or a bound on one. Each row retains its own workload and timer semantics.

The initial dependency-reuse measurements evaluate more intermediate nodes in sparse cases. Endpoint-only measurements remove those unrequested evaluations for both methods. These are scientifically different evaluation policies, so their sparse timing ratios should not be pooled. Likewise, the released-capacity and retained-capacity fused runs are separate invocations. Their labels describe the implementation difference without treating them as independent learning trials.

E.1 Initial dependency-reuse measurements

Table 7: Initial dependency-reuse measurements. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen2.5, panel 100			
5^4 plans	Prefix	Model	9.7491
5^4 plans	Prefix	Warm	9.6086
Sparse set 1	Prefix	Model	1.0947
Sparse set 1	Prefix	Warm	1.0841
Sparse set 2	Prefix	Model	1.0932
Sparse set 2	Prefix	Warm	1.0903
Sparse set 3	Prefix	Model	1.0936
Sparse set 3	Prefix	Warm	1.0990
Sparse set 4	Prefix	Model	0.9999
Sparse set 4	Prefix	Warm	0.9976

E.2 Endpoint-only evaluation and scale

Table 8: Endpoint-only evaluation and scale. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen2.5, panel 100			
5^3 plans	Prefix	Model	3.9632
5^3 plans	Prefix	Warm	3.8520
5^4 plans	Prefix	Model	9.7548
5^4 plans	Prefix	Warm	10.0181
5^5 plans	Prefix	Model	24.3040
5^5 plans	Prefix	Warm	25.0805
Sparse set 1	Prefix	Model	1.0010
Sparse set 1	Prefix	Warm	0.9933
Sparse set 2	Prefix	Model	1.0004
Sparse set 2	Prefix	Warm	1.0216
Sparse set 3	Prefix	Model	1.0006
Sparse set 3	Prefix	Warm	1.0014
Sparse set 4	Prefix	Model	0.9999
Sparse set 4	Prefix	Warm	0.9950
One plan, $T = 8$	Prefix	Model	0.9995
One plan, $T = 8$	Prefix	Warm	1.0052

E.3 Additional models and input panels

Table 9: Additional models and input panels. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen2.5, panel 200			
5^4 plans	Prefix	Model	9.7654
5^4 plans	Prefix	Warm	9.9107
Sparse set 1	Prefix	Model	0.9996
Sparse set 1	Prefix	Warm	1.0164
Sparse set 2	Prefix	Model	0.9999
Sparse set 2	Prefix	Warm	0.9937
Sparse set 3	Prefix	Model	1.0012
Sparse set 3	Prefix	Warm	1.0178
Sparse set 4	Prefix	Model	0.9997
Sparse set 4	Prefix	Warm	1.0015
One plan, $T = 8$	Prefix	Model	0.9998
One plan, $T = 8$	Prefix	Warm	0.9913
Qwen2.5, panel 300			
5^4 plans	Prefix	Model	9.7760
5^4 plans	Prefix	Warm	9.9044
Sparse set 1	Prefix	Model	0.9993
Sparse set 1	Prefix	Warm	0.9982
Sparse set 2	Prefix	Model	0.9994
Sparse set 2	Prefix	Warm	1.0007
Sparse set 3	Prefix	Model	1.0005
Sparse set 3	Prefix	Warm	1.0036
Sparse set 4	Prefix	Model	0.9996
Sparse set 4	Prefix	Warm	0.9937
One plan, $T = 8$	Prefix	Model	1.0004
One plan, $T = 8$	Prefix	Warm	1.0011

Input / workload	Baseline	Timer	Ratio
Qwen3, panel 200			
5^4 plans	Prefix	Model	9.6081
5^4 plans	Prefix	Warm	10.0385
Sparse set 1	Prefix	Model	0.9997
Sparse set 1	Prefix	Warm	0.9954
Sparse set 2	Prefix	Model	1.0010
Sparse set 2	Prefix	Warm	1.0148
Sparse set 3	Prefix	Model	0.9995
Sparse set 3	Prefix	Warm	1.0043
Sparse set 4	Prefix	Model	0.9998
Sparse set 4	Prefix	Warm	1.0090
One plan, $T = 8$	Prefix	Model	0.9998
One plan, $T = 8$	Prefix	Warm	1.0083
Qwen3, panel 300			
5^4 plans	Prefix	Model	9.7895
5^4 plans	Prefix	Warm	10.2589
Sparse set 1	Prefix	Model	0.9999
Sparse set 1	Prefix	Warm	1.0017
Sparse set 2	Prefix	Model	1.0010
Sparse set 2	Prefix	Warm	1.0076
Sparse set 3	Prefix	Model	0.9997
Sparse set 3	Prefix	Warm	0.9769
Sparse set 4	Prefix	Model	1.0004
Sparse set 4	Prefix	Warm	1.0248
One plan, $T = 8$	Prefix	Model	1.0008
One plan, $T = 8$	Prefix	Warm	0.9965
SmolLM2, panel 200			
5^4 plans	Prefix	Model	9.2441
5^4 plans	Prefix	Warm	8.9300
Sparse set 1	Prefix	Model	0.9999
Sparse set 1	Prefix	Warm	0.9926
Sparse set 2	Prefix	Model	0.9988
Sparse set 2	Prefix	Warm	1.0001
Sparse set 3	Prefix	Model	1.0003
Sparse set 3	Prefix	Warm	1.0085
Sparse set 4	Prefix	Model	0.9989
Sparse set 4	Prefix	Warm	0.9698
One plan, $T = 8$	Prefix	Model	1.0011
One plan, $T = 8$	Prefix	Warm	1.0099
SmolLM2, panel 300			
5^4 plans	Prefix	Model	9.6927
5^4 plans	Prefix	Warm	9.4354
Sparse set 1	Prefix	Model	0.9995
Sparse set 1	Prefix	Warm	1.0152
Sparse set 2	Prefix	Model	1.0012
Sparse set 2	Prefix	Warm	1.0079
Sparse set 3	Prefix	Model	1.0007
Sparse set 3	Prefix	Warm	1.0087
Sparse set 4	Prefix	Model	0.9996
Sparse set 4	Prefix	Warm	0.9987
One plan, $T = 8$	Prefix	Model	1.0001
One plan, $T = 8$	Prefix	Warm	0.9667

E.4 Fixed expert budgets in full precision

Table 10: Fixed expert budgets in full precision. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen2.5, panel 200			
Quota (1,1,1,1)	Prefix	Model	3.6310
Quota (1,1,1,1)	Prefix	Warm	3.4531
Quota (2,2,1,1)	Prefix	Model	9.4147
Quota (2,2,1,1)	Prefix	Warm	8.7788
One plan, $T = 8$	Prefix	Model	0.9997
One plan, $T = 8$	Prefix	Warm	1.0032
Qwen3, panel 200			
Quota (1,1,1,1)	Prefix	Model	3.5558
Quota (1,1,1,1)	Prefix	Warm	3.3827
Quota (2,2,1,1)	Prefix	Model	9.1358
Quota (2,2,1,1)	Prefix	Warm	9.5493
One plan, $T = 8$	Prefix	Model	0.9993
One plan, $T = 8$	Prefix	Warm	0.9914
SmolLM2, panel 200			
Quota (1,1,1,1)	Prefix	Model	3.2296
Quota (1,1,1,1)	Prefix	Warm	2.8630
Quota (2,2,1,1)	Prefix	Model	8.4694
Quota (2,2,1,1)	Prefix	Warm	7.8011
One plan, $T = 8$	Prefix	Model	1.0002
One plan, $T = 8$	Prefix	Warm	0.9825

E.5 Model-resident cold graph pools

Table 11: Model-resident cold graph pools. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen2.5, panel 200			
Quota (1,1,1,1)	Prefix	Model	3.6314
Quota (1,1,1,1)	Prefix	Warm	3.3939
Quota (1,1,1,1)	Prefix	Cold	3.8339
Quota (2,2,1,1)	Prefix	Model	9.4232
Quota (2,2,1,1)	Prefix	Warm	9.2780
Quota (2,2,1,1)	Prefix	Cold	2.8662
One plan, $T = 8$	Prefix	Model	1.0002
One plan, $T = 8$	Prefix	Warm	1.0078
One plan, $T = 8$	Prefix	Cold	1.0037
SmolLM2, panel 200			
Quota (1,1,1,1)	Prefix	Model	3.2295
Quota (1,1,1,1)	Prefix	Warm	2.8726
Quota (1,1,1,1)	Prefix	Cold	3.7806
Quota (2,2,1,1)	Prefix	Model	8.4768
Quota (2,2,1,1)	Prefix	Warm	7.8127
Quota (2,2,1,1)	Prefix	Cold	3.8829
One plan, $T = 8$	Prefix	Model	1.0003
One plan, $T = 8$	Prefix	Warm	0.9891
One plan, $T = 8$	Prefix	Cold	1.0087

E.6 Mixed-precision execution

Table 12: Mixed-precision execution. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen2.5, panel 200			
Quota (1,1,1,1)	Prefix	Model	2.9818
Quota (1,1,1,1)	Prefix	Warm	2.6241
Quota (2,2,1,1)	Prefix	Model	7.8258
Quota (2,2,1,1)	Prefix	Warm	7.1941
One plan, $T = 8$	Prefix	Model	0.9998
One plan, $T = 8$	Prefix	Warm	1.0533
Qwen3, panel 200			
Quota (1,1,1,1)	Prefix	Model	3.1116
Quota (1,1,1,1)	Prefix	Warm	2.9212
Quota (2,2,1,1)	Prefix	Model	7.3197
Quota (2,2,1,1)	Prefix	Warm	8.5615
One plan, $T = 8$	Prefix	Model	0.9996
One plan, $T = 8$	Prefix	Warm	1.0100
SmolLM2, panel 200			
Quota (1,1,1,1)	Prefix	Model	2.7119
Quota (1,1,1,1)	Prefix	Warm	2.4858
Quota (2,2,1,1)	Prefix	Model	7.0290
Quota (2,2,1,1)	Prefix	Warm	6.4832
One plan, $T = 8$	Prefix	Model	0.9997
One plan, $T = 8$	Prefix	Warm	0.9852

E.7 Fused execution with released graph capacity

Table 13: Fused execution with released graph capacity. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen3, panel 200			
One plan, $T = 6$	Batched	Execution	0.9999
One plan, $T = 6$	Batched	Accounted	1.0417
One plan, $T = 6$	Prefix	Execution	1.0004
One plan, $T = 6$	Prefix	Accounted	1.0051
Quota (1,1,1,1)	Batched	Execution	2.5438
Quota (1,1,1,1)	Batched	Accounted	9.3939
Quota (1,1,1,1)	Prefix	Execution	2.8963
Quota (1,1,1,1)	Prefix	Accounted	3.1529
Quota (2,2,1,1)	Batched	Execution	15.7576
Quota (2,2,1,1)	Batched	Accounted	34.4279
Quota (2,2,1,1)	Prefix	Execution	8.9554
Quota (2,2,1,1)	Prefix	Accounted	30.9691
Live beam	Batched	Execution	3.4925
Live beam	Batched	Accounted	1.9379
Live beam	Prefix	Execution	1.5629
Live beam	Prefix	Accounted	1.5501

E.8 Fused execution with retained graph capacity

Table 14: Fused execution with retained graph capacity. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen3, panel 200			
One plan, $T = 6$	Batched	Execution	1.0104
One plan, $T = 6$	Batched	Accounted	1.0268
One plan, $T = 6$	Prefix	Execution	1.0100
One plan, $T = 6$	Prefix	Accounted	0.9975
Quota (1,1,1,1)	Batched	Execution	4.2210
Quota (1,1,1,1)	Batched	Accounted	2.8515
Quota (1,1,1,1)	Prefix	Execution	3.8033
Quota (1,1,1,1)	Prefix	Accounted	2.6524
Quota (2,2,1,1)	Batched	Execution	15.1503
Quota (2,2,1,1)	Batched	Accounted	15.6794
Quota (2,2,1,1)	Prefix	Execution	8.3777
Quota (2,2,1,1)	Prefix	Accounted	8.8131
Live beam	Batched	Execution	4.1663
Live beam	Batched	Accounted	1.8809
Live beam	Prefix	Execution	1.3914
Live beam	Prefix	Accounted	1.1181

E.9 Native single-device path comparison

Table 15: Native single-device path comparison. Each entry is a median paired baseline/FCT ratio.

Input / workload	Baseline	Timer	Ratio
Qwen3, panel 200			
One plan, $T = 6$	Native prefix	Execution	0.9803
One plan, $T = 6$	Native prefix	Accounted	0.8768
One plan, $T = 6$	Adapted prefix	Execution	1.0768
One plan, $T = 6$	Adapted prefix	Accounted	1.1229
Quota (2,2,1,1)	Native prefix	Execution	6.7930
Quota (2,2,1,1)	Native prefix	Accounted	8.7721
Quota (2,2,1,1)	Adapted prefix	Execution	9.7515
Quota (2,2,1,1)	Adapted prefix	Accounted	8.4568
Fixed beam requests	Native prefix	Execution	1.2867
Fixed beam requests	Native prefix	Accounted	1.1871
Fixed beam requests	Adapted prefix	Execution	1.4885
Fixed beam requests	Adapted prefix	Accounted	1.1666

F Absolute Timing and Cost Accounting

F.1 What Each Timer Includes

Resident request execution includes the model's forward and backward computations, state transfer, and host-side request scheduling. Graph preparation is timed separately because a graph may be reused across several methods and requests. Numerical comparisons used for the experimental checks and checkpoint writes are also excluded from the primary resident timer. This scope measures the executor's cost for requests on an available model, rather than an end-user job that also downloads weights and initializes its environment.

Cold-pool measurements answer a separate question. Each compared method starts with its own empty graph pool while the model remains resident. Its timed interval includes the necessary graph

preparation and execution. This removes cross-method graph sharing from that comparison. It does not include process creation, package installation, network transfer, or model loading. The cold-pool ratios in Figure 3 therefore provide measured setup-inclusive evidence for graph initialization, with the stated boundary.

Partially accounted costs cannot be converted into a rigorous full-process speedup. Let $A \leq A_{\text{total}}$ and $B \leq B_{\text{total}}$ be recorded cost sums for two methods. Neither $A/B \leq A_{\text{total}}/B_{\text{total}}$ nor the reverse follows without further information about the omitted terms. For example, adding an unmeasured cost only to the numerator increases the total ratio, while adding it only to the denominator decreases it. This is why the accounted ratios remain separately labeled in the full tables.

F.2 Paired Resident Times

The following table expands the resident fused comparison to individual repeats. It reports the actual time of each method rather than reconstructing absolute times from a speedup. This preserves variation due to execution order and short-request overhead. Training states and planner outcomes remain fixed across these repeats.

Table 16: Common fused program. Resident execution times in seconds.

Workload	Repeat	Batched	Prefix	FCT
One plan	1	0.361509	0.332947	0.333373
24 quota plans	1	1.511197	1.275760	0.327194
180 quota plans	1	12.410134	6.830507	0.834360
Live beam	1	2.941736	0.982157	0.721134
One plan	2	0.333183	0.366534	0.333936
24 quota plans	2	1.408165	1.283373	0.340446
180 quota plans	2	12.595375	6.818857	0.813928
Live beam	2	2.984776	0.996842	0.716415
One plan	3	0.333545	0.333416	0.330123
24 quota plans	3	1.417493	1.277208	0.335815
180 quota plans	3	12.576684	6.965235	0.830128
Live beam	3	3.173049	1.053837	0.686438

Table 17: Matched native requests. Resident execution times in seconds.

Workload	Repeat	Native	Adapted prefix	FCT
One plan	1	0.413644	0.431400	0.377690
180 quota plans	1	7.133321	10.331979	1.043816
Recorded requests	1	0.998271	1.186886	0.775841
One plan	2	0.367044	0.403195	0.374436
180 quota plans	2	6.913757	7.534380	1.028478
Recorded requests	2	1.012687	1.148971	0.771890
One plan	3	0.366329	0.374455	0.373754
180 quota plans	3	7.066601	10.144315	1.040279
Recorded requests	3	1.009807	1.152273	0.792699

The retained-graph fused invocation spends 46.20 seconds in graph capture, 87.17 seconds in request execution, and 26.01 seconds in checkpoint calls. The recorded pre-packaging invocation takes 168.46 seconds. The listed components do not sum to the full invocation because initialization, numerical checks, and other host work also occur. The sampled peak device allocation reported by the telemetry is 43.15 GiB, shared across the comparison. It is not a per-method peak-memory estimate.

The native comparison records 99 request units across methods and repeats, with 7,050 logical operations and 7,392 physical lane operations. The latter includes padding. It prepares 114 graph

captures and records 62.64 seconds of preparation within a 182.93-second pre-packaging invocation. No allocation failure is recorded in this run. These invocation-level measurements describe the comparison infrastructure; per-request execution ratios remain the quantities reported in the main results.

F.3 Low-Reuse and Host-Cost Measurements

Single-plan controls help distinguish computational saving from incidental timing variation. With no duplicated local history, FCT and whole-prefix reuse perform the same logical training work. Several model-time ratios are consequently near one. Warm times can still differ because state representation, host bookkeeping, transfers, and short-duration measurement overhead contribute differently. The SmolLM2 single-plan warm ratio in the second panel is 0.9667, indicating a small overhead.

Near-parity model time is consistent with the absence of reusable training work in this control. Its model-time ratio is 1.0001, while the wider warm timer includes additional host costs. The measurements do not isolate one host component as the source of the difference. Fixed-budget experiments separately record storage-related costs and retain near-parity single-plan controls. These observations place the benefit in workloads that repeat local histories.

G Numerical Evidence and Score Perturbations

G.1 Evidence at Each Numerical Level

The numerical evaluation proceeds from operator behavior to full-model state and then to planning decisions. Repeated layer-level execution tests compare the same inputs in the same graph slot, different slots, and permuted companion lanes. The fixed-reduction candidate satisfies the tested equality conditions. The official-kernel layer comparison contains nonidentical outputs, including 58 same-slot comparison records. No intermediate trace establishes a unique internal instruction as the cause of all differences.

Full-model tests then exercise backpropagation through the pretrained Transformer. The saved roots and gradients permit reconstruction of six vector parameter-and-momentum transitions, corresponding to 24 expert transitions. Across these traces, 2,688 gradient blocks are nonzero. Companion rotations are nontrivial, so equality is not explained by unused trainable projections or a no-update path. These checks connect the fixed-reduction implementation to the actual expert-training state used by the planner.

The fused planning experiment compares all matching intermediate states online and retains all requested loss columns and scores. Post-run aggregation from the saved columns covers 2,259 scores and 36,144 column elements, with zero discrepancy. The saved complete-state comparisons cover 42 expert parameter-and-momentum pairs. The separate native experiment permits recomputation of 2,043 scores and comparison of 32,688 column elements, along with 36 saved adapted expert pairs. These counts describe distinct evidence types rather than a single number of independent trials.

Native saved-state coverage is smaller than the adapted online comparison. Of 36 archived native expert states, 33 have matching histories available for the corresponding saved adapted comparison. Native columns for a repeated history also vary in some records. The native and adapted numerical programs therefore cannot be treated as interchangeable. Our primary exactness statement remains within the common adapted program, where both full-state comparison and endpoint reconstruction support it.

G.2 A Decision-Relevant Perturbation Bound

The effect of a backend change depends on relative score perturbations across candidates. On a fixed candidate set, let $F(r)$ be the adapted score, $N(r)$ the native score, and $\Delta(r) = N(r) - F(r)$. Let r_F maximize F and r_N maximize N, using a fixed tie rule. Since $F(r_F) \geq F(r_N)$ and $N(r_N) \geq$

$N(r_F)$,

$$0 \le F(r_F) - F(r_N) \tag{17}$$
$$\le \Delta(r_N) - \Delta(r_F) \tag{18}$$
$$\le \max_r \Delta(r) - \min_r \Delta(r). \tag{19}$$

The last quantity measures the range of perturbations on that candidate set. A uniform score shift can be large while preserving every ordering, so maximum absolute score error alone does not determine selection changes.

A sufficient condition for a unique adapted winner to remain the native winner follows immediately. Define its adapted margin as

$$m = F(r_F) - \max_{r \ne r_F} F(r). \tag{20}$$

If $m > \max \Delta - \min \Delta$, then for every competitor r,

$$N(r_F) - N(r) \ge m - (\max \Delta - \min \Delta) > 0. \tag{21}$$

This is a sufficient condition, not a necessary one. When it fails, the winner may still agree. Equal or nearly equal candidate scores therefore require attention to the actual score ordering rather than a universal drift threshold.

G.3 Observed Native Decisions

The fixed-request records contain 24 candidate sets across three repeats: three single-plan sets, three 180-plan sets, and 18 beam-step sets. Native and adapted maximizers differ in the three 180-plan sets; all beam-step maximizers agree. The largest adapted-score loss from choosing the native maximizer is 0.0007722083. Equation 19 holds for all sets using exact rational representations of the stored floating-point values. This comparison concerns candidate scores, not population-level task quality.

Table 18: Native selections on fixed candidate sets. Regret is $F(r_F) - F(r_N)$, evaluated with the adapted score. All beam-step maximizers agree despite nonzero score perturbations.

Request family	Repeat	Candidate sets	Changed maximizers	Maximum regret
One plan	1,2,3	3	0	0
180 plans	1,2,3	3	3	0.0007722083
Beam steps	1	6	0	0
Beam steps	2	6	0	0
Beam steps	3	6	0	0

The 180-plan family has an adapted winner margin of 0.0007722083. Native perturbation ranges are approximately 0.6548, 0.5443, and 0.5443 in its three repeats. Thus the sufficient margin condition does not certify its winner, consistent with the observed changes. The beam sets also fail this sufficient condition in cases where their maximizers still agree. The diagnostic explains sensitivity to relative score changes without attributing those changes to a single operator or claiming that the native choices have worse test performance.

H Relation to Other Training Optimizations

H.1 Reuse, Batching, and Model Selection

The method operates on a different dependency level from adapter batching. Multi-adapter serving systems such as S-LoRA and Punica share a base model while executing different adapter requests (Sheng et al., 2023; Chen et al., 2023a). Training systems such as mLoRA, LoRAFusion, and tLoRA organize the computation and resources required by trainable adapters (Ye et al., 2023; Zhu et al., 2025; Li et al., 2026). FCT reduces the set of complete local transitions demanded by a training-plan search. The remaining transitions can use such execution techniques, subject to their numerical semantics.

Frozen-computation reuse and complete-state reuse also have different validity conditions. Nautilus defines materializable computations through frozen dependencies and then optimizes their execution and storage (Nakandala & Kumar, 2022). In FCT, a trained expert state is reusable because the entire ordered update history is the same. An expert's later activations may depend on its adapter even though the shared backbone weights are frozen. The method therefore does not assume that arbitrary backbone activations can be reused across different trained expert states.

Model-selection systems optimize a broader decision problem. Cerebro targets throughput of model-selection training (Nakandala et al., 2020), Hyperband allocates training resources among configurations (Li et al., 2016), and ALTO adapts LoRA tuning and workload orchestration (Zuo et al., 2026). FCT supports the subproblem in which candidate transitions share their complete inputs. Changing rank, optimizer settings, or learning rate generally creates a different transition namespace. Reuse is still possible within any group that retains the same dependencies, but the formulas for a single shared update rule should not be applied to unrelated trials.

H.2 Memory and Recomputation

Caching complete local states exchanges storage for avoided training. Gradient checkpointing makes another compute-memory tradeoff by recomputing activations during backpropagation (Chen et al., 2016). These choices are compatible in principle: a state transition may internally use checkpointing while its completed output is retained for another plan. Their combined cost depends on the model, number of histories, available memory, and recomputation schedule. The present measurements retain their declared state and graph policies; they do not establish an optimal cache-eviction policy.

The frozen-backbone and low-rank configuration makes local-state retention practical in the measured setting. Its state cost scales with trainable adapter size rather than the full backbone. QLoRA addresses a complementary memory component by quantizing frozen weights (Dettmers et al., 2023). Quantization would define another numerical program, requiring within-program comparisons before reusing its transitions. Likewise, operator changes such as efficient attention can alter execution cost without changing the symbolic dependency structure (Dao et al., 2022). Exact floating-point agreement still needs to be evaluated for the specific implementation.

H.3 From Arrived Experience to Interactive Learning

Arrived experience separates the training-plan decision from data generation. A central planner can evaluate alternative assignments on already collected examples, select one bank, and then commit its complete expert states. This is the setting formalized in Section 3. Recent agent-training systems motivate this separation of training, validation, and rollout concerns (Zhu & You, 2026; Yao et al., 2026). They do not by themselves establish that every agent workload has repeated local training histories.

Interactive learning adds dependencies that must be modeled explicitly. If an updated expert changes the next action, that action changes a rollout, and the rollout supplies the next training event, a matching index sequence is insufficient. The actual data and relevant interaction state must agree as well. Similarly, a trainable shared backbone couples expert transitions. Extending the state representation can recover valid reuse for some such systems, but may reduce its frequency. Measuring that frequency is more informative than assuming that the complete-family savings transfer unchanged.

The current method evaluates a prescribed learning choice accurately and efficiently. It does not select better data by itself, as studied by validation reweighting, LESS, or Skill-it (Ren et al., 2018; Xia et al., 2024; Chen et al., 2023b), and does not optimize an initialization for fast adaptation as in model-agnostic meta-learning (Finn et al., 2017). Its contribution is to make repeated exact training outcomes available at lower execution cost when their local dependencies coincide. This separation enables future studies to attribute improvements either to a better learning procedure or to a more efficient executor.

I PUBLIC-TASK ADAPTATION AND ACTUAL REQUEST REUSE

I.1 TASK PROTOCOL AND ALLOCATION BUDGET

The public-task experiment measures a complete adaptation workflow on two separate science-question tasks. We first describe Qwen3-0.6B with SciQ and ARC-Challenge (Yang et al., 2025; Welbl et al., 2017; Clark et al., 2018); Appendix J completes the four-model matrix under the same protocol. Each task has three initialization and training-sample seeds, labeled 1, 2, and 3 in the tables. They correspond to 20260908, 20260909, and 20260910. The official test set is shared across seeds: 1,000 questions for SciQ and 1,172 for ARC-Challenge.

Each seed uses 288 distinct training examples without replacement. The first 32 initialize four experts through one eight-example update per expert. The remaining 256 form 32 events of eight examples each, in the sampled order. The four rank-two query/value LoRA experts retain separate FP32 parameters and momentum on a BF16 frozen backbone. The learning rate is 0.02, momentum is 0.6, LoRA scaling is two, and dropout is zero. These settings remain fixed across methods, blocks, and tasks. A separate sample of 64 official validation examples supplies the planning objective.

The search commits four consecutive blocks of eight events. At each depth, a width-four beam expands all feasible assignments of the next event to an expert with remaining quota. Candidates are ranked by validation score, with lexicographic tie breaking. Every complete candidate assigns exactly two events to each expert. The best final bank becomes the complete starting bank for the next block, including its momentum. All methods commit 32 updates after initialization, and each expert receives 64 adaptation examples. Search evaluates extra candidate updates, whereas round-robin allocation commits the cyclic sequence without search. Thus the quality reference matches training exposure and committed updates, not total computation.

Answer prediction uses a fixed multiple-choice interface. The prompt contains the question, lettered options, and an answer cue, wrapped in the model's chat template with thinking disabled. Training minimizes full-vocabulary cross-entropy for the correct answer-letter token. Evaluation normalizes next-token logits over the available answer letters. SciQ options are rotated by their fixed dataset-row index, balancing the correct-letter position without consulting model outputs; ARC preserves its listed option order, including questions with different option counts. No test question is dropped or truncated. All training and validation IDs are disjoint from test IDs.

This is a declared low-data adaptation protocol on complete public test splits. The prompt, option normalization, and training budget are specified here so that the results can be reproduced; they do not imply equivalence to an existing leaderboard's prompting or adaptation settings. Test labels enter only the final accuracy and loss computation. Validation scores determine assignments and bank commitments; test accuracy never selects an expert, seed, block, or hyperparameter.

I.2 A FIXED ENSEMBLE OBJECTIVE

The planner evaluates the same ensemble that supplies the final answer. For available answer a and question x, let $p_i(a \mid x, S_i)$ be expert i's normalized option probability at state S_i. Define

$$\bar{p}(a \mid x, S) = \frac{1}{4}\sum_{i=1}^{4} p_i(a \mid x, S_i), \qquad \widehat{a}(x, S) = \arg\max_a \bar{p}(a \mid x, S). \tag{22}$$

The validation objective maximizes the log likelihood of this probability mixture:

$$G_V(S) = \frac{1}{|V|}\sum_{(x,y)\in V} \log \bar{p}(y \mid x, S). \tag{23}$$

Negative log likelihood (NLL) on test data uses the negative of the same average. Every requested expert probability vector can be evaluated and cached separately; the final mixture is reconstructed in a fixed expert order.

This objective is nonadditive across experts. For example, two experts with correct-answer probabilities $(0.9, 0.1)$ have a larger arithmetic mean than $(0.4, 0.4)$, but a smaller geometric mean. Averaging probabilities before taking a logarithm therefore ranks these pairs differently from averaging

individual log probabilities. Duplicating each pair gives the same distinction with four experts. Equation 23 differs from the mean-expert loss in Equation 5, so the additive subset allocation in Appendix B.4 does not directly optimize it. The uniform ensemble and beam search define this application; the execution contribution remains complete-state reuse.

Corollary 2 (Successive bank commitments)**.** *Under Proposition 1, a deterministic readout and planner preserve the same scores, selected bank, and next block's complete root under FCT and whole-prefix reuse. This equivalence extends to any fixed number of successive blocks.*

Within a block, equal complete expert states give equal option probabilities. The fixed mixture and score aggregation then give equal scores, so Corollary 1 gives the same requests and winner. Both executors commit the same complete winning bank. Taking this bank as the next root extends the argument by induction over blocks. Each new root defines a fresh local-history namespace; an event index alone never licenses reuse across different block roots. Applying the same readout to the final equal banks preserves all test predictions. □

I.3 Observed Reuse and Method Costs

Every measured block contains reusable local histories despite a changing committed bank. Figure 5 reports the avoided fraction

$$\rho_b = 1 - \frac{N_{\mathrm{FCT},b}}{N_{\mathrm{prefix},b}}, \tag{24}$$

where both counts are reconstructed from that block's actual beam requests. The full-run fractions in Table 19 use sums of block counts, rather than an unweighted average of block percentages. No complete candidate family is enumerated to create the measured reuse.

Table 19: All Qwen3 public-task paired results. Updates count candidate training after common initialization. Method time includes its graph preparation, search, transfers, scheduling, and full-test evaluation, with model and roots resident. The ratio is prefix time divided by FCT time.

Task	Seed	Prefix upd.	FCT upd.	Saved (%)	Prefix (s)	FCT (s)	Ratio
SciQ	1	301	165	45.18	66.40	46.31	1.434
SciQ	2	328	202	38.41	78.02	53.68	1.453
SciQ	3	317	198	37.54	76.90	50.75	1.515
ARC-Challenge	1	305	183	40.00	102.06	71.28	1.432
ARC-Challenge	2	311	179	42.44	104.66	69.44	1.507
ARC-Challenge	3	306	185	39.54	103.64	68.64	1.510

Time savings remain positive for all six pairs. SciQ ratios range from 1.434 to 1.515, and ARC-Challenge ratios from 1.432 to 1.510. The per-task medians are 1.453 and 1.507, corresponding to 31.2% and 33.7% less method time. The time reduction is smaller than the update-count reduction because both methods retain full-test evaluation and other request costs. Table 20 separates search from final testing instead of attributing every saved second to gradient computation.

Methods use independent empty graph pools and cyclic execution order. The orders for SciQ seeds 1, 2, and 3 are prefix/FCT/round-robin, FCT/round-robin/prefix, and round-robin/prefix/FCT. ARC shifts these orders by one position. All six pairs complete within one invocation; no method timing spans a resumed session. Capacity selection fixes eight examples per expert lane. Training and validation lengths are padded to 128 tokens for SciQ and 192 for ARC; the longest ARC test shape is 256. Each pair uses the same numerical geometry while useful-work concurrency can vary.

The wider method-wall timer also favors FCT in every pair. It records the work surrounding the primary timer, including the comparison performed by whichever paired method runs second. It is therefore useful as a total observed method cost, while the primary timer isolates the deployed execution work. The combined four-model invocation takes 3,535.90 seconds before packaging. Shared dataset preparation takes 7.40 seconds; tokenization and model loading total 20.15 and 17.89 seconds across the four backbones. These totals include the cases in Appendix J, not just Qwen3, and are not one method's latency.

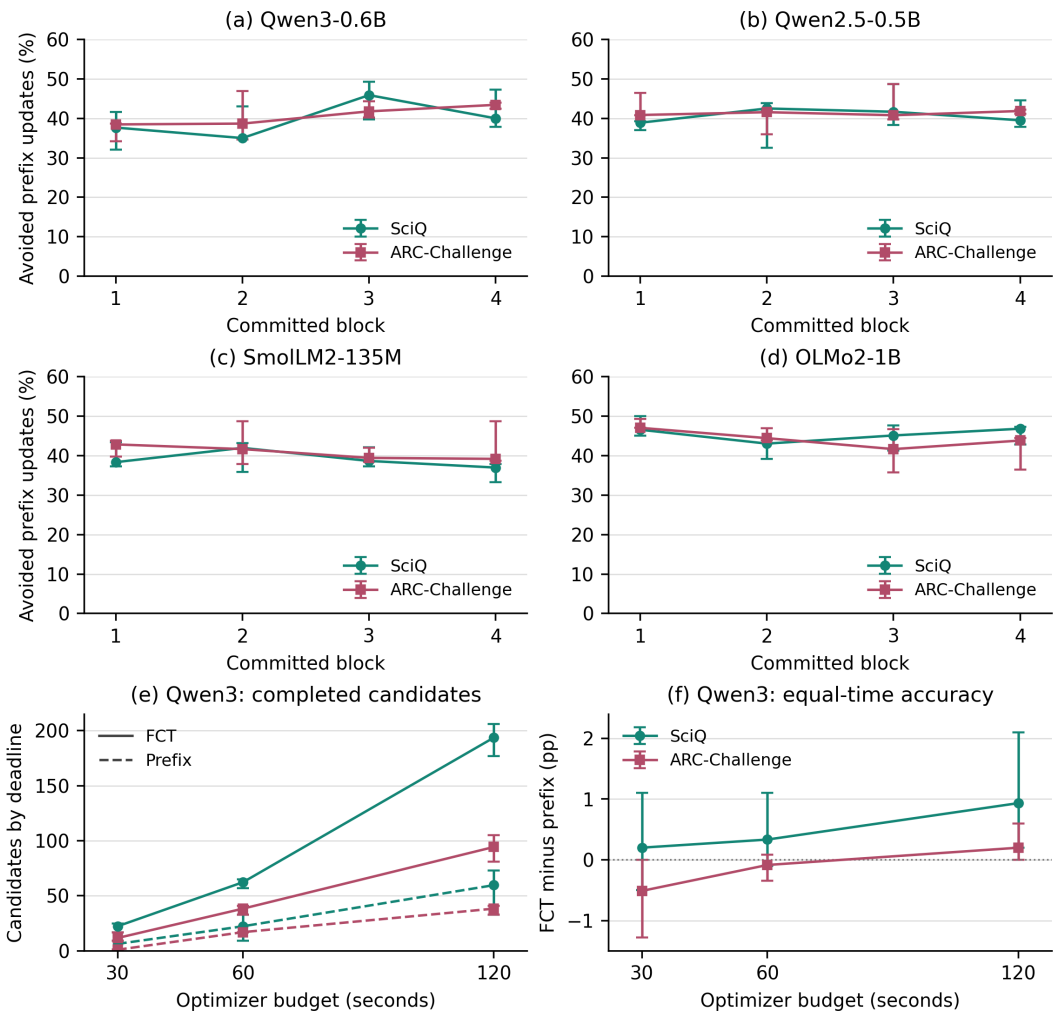

Figure 5: Actual search requests support reuse and additional candidate evaluation. (a–d) Avoided prefix updates after successive commitments across four models, with median markers. (e–f) Separate Qwen3 equal-time search: completed candidate counts and paired accuracy differences, with mean markers (Appendix K). All error bars show the three-seed minimum and maximum, not confidence intervals. Colors and shapes identify tasks; solid and dashed lines in (e) identify FCT and prefix reuse.

The resource samples support useful device activity without implying continuous saturation. Across 6,997 samples from the four-model invocation, mean utilization is 83.50%; graph preparation, host work, and storage remain visible. Concurrency adapts to useful-work throughput and memory, while the numerical batch remains fixed. These samples describe the measured execution; the paired elapsed times determine the efficiency claim.

I.4 Task Quality and Seed Variation

The searched allocation improves average test accuracy on both tasks while retaining seed-dependent variation. Table 21 reports every seed, including the ARC decrease. Prefix and FCT have identical predictions, so their quality occupies one column. Search improves SciQ accuracy by 1.10, 2.60, and 2.30 percentage points. ARC changes are -0.85, $+0.43$, and $+2.13$ points. These comparisons evaluate the chosen allocation; they do not attribute a quality improvement to changing the executor.

Accuracy and NLL need not move together. SciQ seed 1 increases accuracy by 1.10 points but raises NLL from 0.5889 to 0.6964. Conversely, ARC seed 1 lowers NLL from 1.0537 to 1.0085

Table 20: Method cost components in seconds. Search and test sum to the primary method time. Capture is already included in those columns. Save records checkpoint calls. Wall time additionally includes numerical comparisons and bookkeeping inside the method, excluding shared setup and outer packaging.

Task	Seed	Method	Search (s)	Test (s)	Capture (s)	Save (s)	Wall (s)
SciQ	1	Prefix	59.55	6.86	5.16	2.28	69.36
SciQ	1	FCT	38.44	7.87	8.30	2.36	50.33
SciQ	1	RR	2.60	8.58	3.68	0.52	11.85
SciQ	2	Prefix	71.16	6.86	13.06	2.73	82.54
SciQ	2	FCT	45.80	7.88	9.89	2.70	57.02
SciQ	2	RR	2.60	7.95	3.14	0.34	11.06
SciQ	3	Prefix	70.05	6.85	13.47	2.63	80.20
SciQ	3	FCT	43.91	6.84	7.42	2.78	55.25
SciQ	3	RR	2.55	7.98	3.11	0.31	10.98
ARC-Challenge	1	Prefix	88.98	13.07	8.58	2.44	106.40
ARC-Challenge	1	FCT	58.18	13.10	8.68	2.76	74.69
ARC-Challenge	1	RR	3.25	13.64	3.97	0.55	17.62
ARC-Challenge	2	Prefix	90.53	14.14	9.64	2.53	107.87
ARC-Challenge	2	FCT	56.36	13.08	7.68	2.51	73.73
ARC-Challenge	2	RR	3.20	13.64	3.92	0.43	17.43
ARC-Challenge	3	Prefix	89.58	14.06	9.52	2.53	106.85
ARC-Challenge	3	FCT	56.58	12.06	5.63	2.64	73.13
ARC-Challenge	3	RR	3.23	14.45	4.80	0.49	18.34

Table 21: Full-test quality for every seed. Accuracies and differences are percentages and percentage points (pp). Win/loss counts questions answered correctly only by search or only by round-robin, respectively. Lower NLL is better. Prefix and FCT are identical in every row.

Task	Seed	Search acc.	RR acc.	Δ (pp)	Search NLL	RR NLL	Win/loss
SciQ	1	79.20	78.10	+1.10	0.6964	0.5889	65/54
SciQ	2	80.40	77.80	+2.60	0.6844	0.7259	60/34
SciQ	3	79.70	77.40	+2.30	0.5605	0.6824	70/47
ARC-Challenge	1	57.08	57.94	-0.85	1.0085	1.0537	65/75
ARC-Challenge	2	57.00	56.57	+0.43	1.0093	1.0740	71/66
ARC-Challenge	3	59.39	57.25	+2.13	1.0118	1.0160	58/33

while accuracy falls by 0.85 points. Validation likelihood selects the allocation; held-out accuracy depends on which option has the largest probability. Reporting both measures retains this distinction without treating a mean gain as a uniform quality improvement. The three seeds provide descriptive variation; no population-level significance claim is made.

I.5 Request Records and Numerical Evidence

The archived requests expose the actual reuse frequency at each decision depth. Table 23 lists all 24 committed blocks. With width four and eight events, the prefix-update count per block is at most $\sum_{t=0}^{7} \min(4, 4^t) \min(4, 8 - t) = 92$. This bound follows from at most four retained parents and at most the number of experts with remaining quota. The table reports observed counts, which can be smaller because feasible children depend on the retained beam.

Table 22: Mean and sample standard deviation over the three initialization and data-sampling seeds. All seeds share the same official test questions. These standard deviations describe training-run variation on fixed tests, not independent test-set uncertainty or confidence intervals.

Task	Allocation	Accuracy (%)	NLL
SciQ	Search	79.77 ± 0.60	0.6471 ± 0.0752
SciQ	RR	77.77 ± 0.35	0.6657 ± 0.0700
ARC-Challenge	Search	57.82 ± 1.36	1.0099 ± 0.0017
ARC-Challenge	RR	57.25 ± 0.68	1.0479 ± 0.0294

Table 23: Actual requests by committed block. The eight comma-separated counts are the numbers of candidate prefixes requested at successive depths; their sum is the prefix update count. Saved updates are relative to that sum.

Task	Seed	Block	Requests by depth	FCT upd.	Saved (%)
SciQ	1	1	4,16,14,13,11,9,6,4	45	41.56
SciQ	1	2	4,16,14,13,11,9,8,4	45	43.04
SciQ	1	3	4,16,14,12,11,8,4,4	37	49.32
SciQ	1	4	4,16,14,12,9,8,5,4	38	47.22
SciQ	2	1	4,16,14,12,11,10,7,4	53	32.05
SciQ	2	2	4,16,15,13,11,10,7,4	52	35.00
SciQ	2	3	4,16,16,14,12,11,8,4	46	45.88
SciQ	2	4	4,16,15,14,13,11,8,4	51	40.00
SciQ	3	1	4,16,14,12,11,9,7,4	48	37.66
SciQ	3	2	4,16,15,12,10,8,6,4	49	34.67
SciQ	3	3	4,16,15,13,12,11,8,4	50	39.76
SciQ	3	4	4,16,15,14,13,9,7,4	51	37.80
ARC-Challenge	1	1	4,16,15,12,11,8,5,4	46	38.67
ARC-Challenge	1	2	4,16,15,13,11,8,7,4	51	34.62
ARC-Challenge	1	3	4,16,15,13,10,10,7,4	44	44.30
ARC-Challenge	1	4	4,16,15,12,9,8,5,4	42	42.47
ARC-Challenge	2	1	4,16,13,12,11,10,8,4	48	38.46
ARC-Challenge	2	2	4,16,14,14,12,9,8,4	43	46.91
ARC-Challenge	2	3	4,16,15,12,10,9,7,4	46	40.26
ARC-Challenge	2	4	4,16,14,12,10,9,6,4	42	44.00
ARC-Challenge	3	1	4,16,15,12,10,9,6,4	50	34.21
ARC-Challenge	3	2	4,16,15,12,9,9,6,4	46	38.67
ARC-Challenge	3	3	4,16,15,13,12,9,6,4	46	41.77
ARC-Challenge	3	4	4,16,14,13,11,8,6,4	43	43.42

Numerical evidence is recorded at both request and final-output levels. Runtime checks compare paired request lists, endpoint probabilities, validation scores, beam choices, and final full-test predictions. Saved final banks provide 24 paired expert parameter-and-momentum states across the six Qwen3 cases. Intermediate complete-state comparisons are recorded online, with selected block witnesses retained. The paired checks use zero tolerance; reproducing a logarithm reduction in another software environment is a separate calculation and is not the criterion for executor equality.

These results establish actual reuse for this fixed-data adaptation workflow. They complement the three-model execution evidence and the separate native-backend comparison. They do not measure a native backend on this new task protocol, a production request distribution, or branch-dependent agent rollouts. Those settings change the evidence needed to assess reuse frequency. Within the measured workflow, independent global search paths repeatedly require the same local training histories, and exploiting that repetition lowers the cost of obtaining the same final answers.

J FOUR-MODEL PUBLIC-TASK ADAPTATION

J.1 MATCHED DATA AND MODEL-SPECIFIC REQUESTS

The complete application matrix contains four models, two tasks, and three seeds per combination. Qwen2.5-0.5B-Instruct, SmolLM2-135M-Instruct, and OLMo2-1B-Instruct extend the Qwen3 study in Appendix I (Qwen Team, 2024; Allal et al., 2025; Team OLMo et al., 2024). The OLMo2 checkpoint is `allenai/OLMo-2-0425-1B-Instruct`. The family paper describes its architecture and training recipe; the official model card identifies this April 2025 checkpoint.[1] Exact model revisions and numerical environments accompany the reproduction data.

All four models use the same example identities and learning budgets for each corresponding seed. The training, validation, and test IDs, answer labels, option counts, and option order match. Each model uses its own tokenizer and chat template, so token sequences can differ. Each also generates its own width-four beam requests and commits its own four winning banks. No model supplies another model's plans. The readout, validation objective, learning rate, momentum, and per-block quotas remain those in Appendix I.

Full benchmark coverage refers to the official test splits. Every run evaluates all 1,000 SciQ or 1,172 ARC-Challenge questions, with no test truncation or omission. The adaptation budget remains 32 initialization and 256 subsequent training examples. Thus the matrix tests the same low-data workflow across architectures and tasks, without changing the training budget to favor a model. The 24 training runs share the same test questions within a task; they are not 24 independent test samples.

J.2 FIXED TENSOR SHAPES WITHIN EACH PAIR

The fused kernel must cover each projection's actual column domain. Its column loads and stores are unmasked, so the output-column tile must divide every affected projection width. We select the largest power-of-two divisor shared by the default tile and these widths. This deterministic shape rule gives forward/backward column tiles of 64/64 for SmolLM2 and 128/128 for Qwen2.5, while OLMo2 uses the default 256/128. No projection is padded or cropped, and no learning parameter is changed.

Table 24: Projection geometry for the three additional models. Shapes are output by input widths. Column tiles are fixed within a model; row and reduction tiles remain unchanged. Exactness compares paired executors using the same geometry.

Model	Query shape	Value shape	Forward tile	Backward tile
SmolLM2-135M	576×576	192×576	64	64
Qwen2.5-0.5B	896×896	128×896	128	128
OLMo2-1B	2048×2048	2048×2048	256	128

Qwen2.5 retains its frozen query/value bias as a BF16 addition after the fused projection. SmolLM2 and OLMo2 have no bias in these projections. These choices define each model's numerical program; changing the tile need not preserve bits across programs. Within each paired run, prefix reuse and FCT use identical tile geometry, update rules, and roots. Training and evaluation retain eight examples per expert lane while useful-work concurrency adapts independently. This preserves the numerical dependency required by Proposition 1.

Padded lengths follow each model's tokenized inputs. SciQ uses 128 tokens for training and validation; test batches are padded by length, with a maximum of 128. For ARC-Challenge, Qwen2.5 and SmolLM2 training uses 256, compared with 192 for Qwen3 and OLMo2. All models use 192 for ARC validation and a maximum test shape of 256. Corresponding batches have the same shapes within each pair; different tokenizers can yield different shapes.

[1] `https://huggingface.co/allenai/OLMo-2-0425-1B-Instruct`

J.3 Complete Cost Matrix

The cost advantage holds in every model/task group and every seed. Table 25 reports all eight group medians and their observed ranges. The median ratios correspond to 28.7–38.7% less primary method time. Across the 24 complete runs, FCT avoids 36.81–46.88% of prefix-cached updates. Both executors nevertheless produce identical requested scores, committed states, and final test predictions. Figure 5 shows the repeated local histories across successive commitments.

Table 25: All eight model/task timing groups. Ratios divide prefix by FCT cost. Min–max covers three paired seeds, not a confidence interval. Time saved is $100(1 - 1/r)$ for the median primary ratio r. Wall uses the wider method-level timer.

Model	Task	Median	Min–max	Time saved (%)	Wall ratio
Qwen3-0.6B	SciQ	1.453	1.434–1.515	31.2	1.448
Qwen3-0.6B	ARC-Challenge	1.507	1.432–1.510	33.7	1.461
Qwen2.5-0.5B	SciQ	1.470	1.335–1.499	32.0	1.435
Qwen2.5-0.5B	ARC-Challenge	1.452	1.445–1.485	31.1	1.438
SmolLM2-135M	SciQ	1.475	1.367–1.558	32.2	1.375
SmolLM2-135M	ARC-Challenge	1.402	1.356–1.421	28.7	1.357
OLMo2-1B	SciQ	1.610	1.587–1.615	37.9	1.550
OLMo2-1B	ARC-Challenge	1.631	1.494–1.741	38.7	1.587

All 24 paired cases are measured in one four-model invocation. Each method completes without interruption, starting with its own empty graph pool. Shared model loading and preparation are outside the primary timer, and the cyclic method order rotates over seeds and tasks. Each model/task group contains exactly three paired seed measurements.

Primary time includes search, transfers, scheduling, method-specific graph preparation, and full-test evaluation. The tables below separate search and test; their sum is the primary time. Capture is already included in these quantities. The wider wall timer additionally records method-level numerical comparisons, checkpoint calls, and bookkeeping. All 24 wall ratios also exceed one, although this timer includes order-dependent comparison work. The recorded invocation metadata and packaging time are not substitutes for per-method execution costs.

J.4 Qwen2.5: Execution Savings and Allocation Variation

Qwen2.5 obtains median cost ratios of $1.470\times$ on SciQ and $1.452\times$ on ARC-Challenge. Tables 26 and 27 retain every seed. Search accuracy on SciQ remains between 76.9% and 78.7%, whereas round-robin varies from 25.0% to 77.8%. On ARC-Challenge the mean search advantage is 0.46 percentage points, with both positive and negative seed differences.

Table 26: Qwen2.5-0.5B paired execution results. Updates count candidate training; time includes independent graph preparation, search, and full testing.

Task	Seed	Prefix upd.	FCT upd.	Saved (%)	Prefix (s)	FCT (s)	Ratio
SciQ	1	307	194	36.81	43.81	32.82	1.335
SciQ	2	316	182	42.41	43.37	29.51	1.470
SciQ	3	308	179	41.88	47.61	31.76	1.499
ARC-Challenge	1	308	181	41.23	66.87	46.27	1.445
ARC-Challenge	2	312	178	42.95	67.47	46.46	1.452
ARC-Challenge	3	303	178	41.25	66.43	44.72	1.485

Table 27: Qwen2.5-0.5B full-test quality. Accuracy and differences use percentages and percentage points (pp). Win/loss counts questions correct only under search or only under round-robin. Lower NLL is better. Both search executors are identical.

Task	Seed	Search acc.	RR acc.	Δ (pp)	Search NLL	RR NLL	Win/loss
SciQ	1	78.70	77.80	+0.90	0.5831	0.7034	54/45
SciQ	2	77.60	59.40	+18.20	0.7183	1.1088	229/47
SciQ	3	76.90	25.00	+51.90	0.8526	1.5460	553/34
ARC-Challenge	1	48.46	52.22	-3.75	1.2073	1.1414	85/129
ARC-Challenge	2	53.67	51.11	+2.56	1.0813	1.1430	103/73
ARC-Challenge	3	52.13	49.57	+2.56	1.1200	1.2616	144/114

The large SciQ mean difference includes an answer-position collapse in round-robin. In seed 3, that bank predicts option A for all 1,000 questions, giving 25.0% accuracy on the balanced answer positions. The searched bank answers 76.9% correctly in the same run. Table 28 reconstructs the answer counts from saved final probabilities. This observation explains why the aggregate gap is unusually large; it does not identify the optimization cause or establish a general quality improvement. The execution saving remains a separate paired result with the search outcome held fixed.

Table 28: Qwen2.5-0.5B SciQ predicted answer positions. Each row contains all 1,000 predictions. The test set has 250 correct answers at each position. FCT represents both identical search executors.

Seed	Allocation	A	B	C	D
1	FCT	272	235	251	242
1	RR	315	212	268	205
2	FCT	207	214	334	245
2	RR	116	129	265	490
3	FCT	314	246	244	196
3	RR	1000	0	0	0

Table 29: Qwen2.5-0.5B mean and sample standard deviation across three training seeds on each fixed official test set.

Task	Allocation	Accuracy (%)	NLL
SciQ	Search	77.73 ± 0.91	0.7180 ± 0.1347
SciQ	RR	54.07 ± 26.80	1.1194 ± 0.4214
ARC-Challenge	Search	51.42 ± 2.67	1.1362 ± 0.0645
ARC-Challenge	RR	50.97 ± 1.33	1.1820 ± 0.0690

Table 30: Qwen2.5-0.5B cost components in seconds. Search plus test is the primary method time. Capture is included, not added. Wall also includes method-level comparisons and bookkeeping.

Task	Seed	Method	Search (s)	Test (s)	Capture (s)	Save (s)	Wall (s)
SciQ	1	Prefix	39.01	4.80	6.41	1.54	45.95
SciQ	1	FCT	28.02	4.80	6.15	1.56	35.82
SciQ	1	RR	1.73	6.59	2.55	0.35	8.81
SciQ	2	Prefix	38.55	4.82	4.75	1.64	46.49

Task	Seed	Method	Search (s)	Test (s)	Capture (s)	Save (s)	Wall (s)
SciQ	2	FCT	24.69	4.82	4.04	1.57	31.67
SciQ	2	RR	1.71	6.54	2.47	0.31	8.72
SciQ	3	Prefix	42.81	4.80	9.91	1.54	49.78
SciQ	3	FCT	26.96	4.80	6.59	1.46	34.68
SciQ	3	RR	1.68	6.54	2.45	0.25	8.61
ARC-Challenge	1	Prefix	57.49	9.38	6.99	1.48	69.85
ARC-Challenge	1	FCT	36.94	9.33	6.17	1.65	48.56
ARC-Challenge	1	RR	2.52	10.20	3.80	0.38	13.32
ARC-Challenge	2	Prefix	58.10	9.37	7.01	1.51	69.65
ARC-Challenge	2	FCT	37.11	9.35	7.17	1.53	49.48
ARC-Challenge	2	RR	2.48	10.19	3.77	0.32	13.16
ARC-Challenge	3	Prefix	57.06	9.37	7.00	1.44	68.58
ARC-Challenge	3	FCT	35.33	9.39	5.49	1.42	47.63
ARC-Challenge	3	RR	2.51	10.17	3.75	0.38	13.23

Table 31: Qwen2.5-0.5B actual search blocks. The eight request counts sum to the prefix update count; saved updates use that sum as denominator.

Task	Seed	Block	Requests by depth	FCT upd.	Saved (%)
SciQ	1	1	4,16,13,12,10,8,6,4	46	36.99
SciQ	1	2	4,16,14,12,12,10,8,4	54	32.50
SciQ	1	3	4,16,14,12,9,8,6,4	45	38.36
SciQ	1	4	4,16,15,12,12,10,8,4	49	39.51
SciQ	2	1	4,16,15,12,12,9,7,4	48	39.24
SciQ	2	2	4,16,15,13,12,10,8,4	46	43.90
SciQ	2	3	4,16,13,12,9,8,6,4	42	41.67
SciQ	2	4	4,16,16,15,13,9,6,4	46	44.58
SciQ	3	1	4,16,13,12,9,8,6,4	44	38.89
SciQ	3	2	4,16,16,15,12,8,5,4	46	42.50
SciQ	3	3	4,16,14,13,11,8,4,4	38	48.65
SciQ	3	4	4,16,16,13,12,10,7,4	51	37.80
ARC-Challenge	1	1	4,16,14,12,9,8,4,4	38	46.48
ARC-Challenge	1	2	4,16,14,12,11,8,6,4	48	36.00
ARC-Challenge	1	3	4,16,15,13,11,8,5,4	45	40.79
ARC-Challenge	1	4	4,16,16,14,13,11,8,4	50	41.86
ARC-Challenge	2	1	4,16,15,13,11,8,5,4	45	40.79
ARC-Challenge	2	2	4,16,15,12,11,9,6,4	45	41.56
ARC-Challenge	2	3	4,16,15,12,11,8,4,4	38	48.65
ARC-Challenge	2	4	4,16,15,14,13,11,8,4	50	41.18
ARC-Challenge	3	1	4,16,12,12,9,8,6,4	42	40.85
ARC-Challenge	3	2	4,16,15,13,11,8,5,4	44	42.11
ARC-Challenge	3	3	4,16,15,12,9,8,5,4	44	39.73
ARC-Challenge	3	4	4,16,15,14,13,10,7,4	48	42.17

J.5 SmolLM2: Reuse with Small Quality Differences

SmolLM2 retains execution savings even when allocation changes have little effect on accuracy. Median cost ratios are $1.475\times$ on SciQ and $1.402\times$ on ARC-Challenge. Mean search and round-robin accuracy are 25.13% and 25.03% on SciQ, and 30.09% and 30.12% on ARC-Challenge. This smallest model therefore supplies a useful contrast to Qwen2.5: similar reuse can accompany very different task quality.

Table 32: SmolLM2-135M paired execution results. Updates count candidate training; time includes independent graph preparation, search, and full testing.

Task	Seed	Prefix upd.	FCT upd.	Saved (%)	Prefix (s)	FCT (s)	Ratio
SciQ	1	325	203	37.54	30.26	20.52	1.475
SciQ	2	300	185	38.33	28.83	18.50	1.558
SciQ	3	306	181	40.85	29.15	21.32	1.367
ARC-Challenge	1	309	189	38.83	46.72	34.46	1.356
ARC-Challenge	2	316	180	43.04	47.08	33.58	1.402
ARC-Challenge	3	311	176	43.41	45.31	31.88	1.421

Table 33: SmolLM2-135M full-test quality. Accuracy and differences use percentages and percentage points (pp). Win/loss counts questions correct only under search or only under round-robin. Lower NLL is better. Both search executors are identical.

Task	Seed	Search acc.	RR acc.	Δ (pp)	Search NLL	RR NLL	Win/loss
SciQ	1	25.20	25.20	+0.00	1.5102	1.5115	8/8
SciQ	2	25.10	25.20	-0.10	1.5214	1.5240	11/12
SciQ	3	25.10	24.70	+0.40	1.5159	1.5164	8/4
ARC-Challenge	1	30.12	30.38	-0.26	1.4544	1.4557	25/28
ARC-Challenge	2	29.69	30.29	-0.60	1.4640	1.4652	10/17
ARC-Challenge	3	30.46	29.69	+0.77	1.4562	1.4560	27/18

SciQ predictions concentrate on option B for both allocations. Table 34 shows that this preference persists across the three seeds, consistent with the near-chance accuracy of the declared answer-letter protocol. The fixed prompt and learning settings retain this outcome in the comparison. Complete-state reuse lowers the cost of reproducing these outcomes just as it does for the higher-accuracy models.

Table 34: SmolLM2-135M SciQ predicted answer positions. Each row contains all 1,000 predictions. The test set has 250 correct answers at each position. FCT represents both identical search executors.

Seed	Allocation	A	B	C	D
1	FCT	95	776	40	89
1	RR	98	775	36	91
2	FCT	72	790	48	90
2	RR	79	797	45	79
3	FCT	111	781	37	71
3	RR	109	778	39	74

Table 35: SmolLM2-135M mean and sample standard deviation across three training seeds on each fixed official test set.

Task	Allocation	Accuracy (%)	NLL
SciQ	Search	25.13 ± 0.06	1.5158 ± 0.0056
SciQ	RR	25.03 ± 0.29	1.5173 ± 0.0063
ARC-Challenge	Search	30.09 ± 0.38	1.4582 ± 0.0051
ARC-Challenge	RR	30.12 ± 0.37	1.4589 ± 0.0054

Table 36: SmolLM2-135M cost components in seconds. Search plus test is the primary method time. Capture is included, not added. Wall also includes method-level comparisons and bookkeeping.

Task	Seed	Method	Search (s)	Test (s)	Capture (s)	Save (s)	Wall (s)
SciQ	1	Prefix	27.54	2.72	7.51	1.32	32.26
SciQ	1	FCT	17.81	2.71	4.72	1.55	23.45
SciQ	1	RR	1.54	4.62	2.83	0.33	6.65
SciQ	2	Prefix	26.11	2.72	7.55	1.48	31.66
SciQ	2	FCT	15.78	2.72	3.79	1.61	20.75
SciQ	2	RR	1.54	4.65	2.86	0.25	6.59
SciQ	3	Prefix	26.44	2.72	7.46	1.52	31.33
SciQ	3	FCT	18.61	2.71	6.71	1.45	24.23
SciQ	3	RR	1.52	4.65	2.84	0.24	6.57
ARC-Challenge	1	Prefix	40.41	6.31	9.51	1.45	49.74
ARC-Challenge	1	FCT	28.14	6.32	8.60	1.56	36.66
ARC-Challenge	1	RR	2.13	7.32	3.86	0.37	9.97
ARC-Challenge	2	Prefix	40.77	6.31	9.47	1.36	49.12
ARC-Challenge	2	FCT	27.26	6.33	8.65	1.33	36.45
ARC-Challenge	2	RR	2.10	7.28	3.79	0.30	9.87
ARC-Challenge	3	Prefix	39.01	6.29	7.59	1.42	47.41
ARC-Challenge	3	FCT	25.56	6.33	7.72	1.44	34.86
ARC-Challenge	3	RR	2.15	7.29	3.85	0.37	9.97

Table 37: SmolLM2-135M actual search blocks. The eight request counts sum to the prefix update count; saved updates use that sum as denominator.

Task	Seed	Block	Requests by depth	FCT upd.	Saved (%)
SciQ	1	1	4,16,16,15,13,9,6,4	52	37.35
SciQ	1	2	4,16,15,14,13,9,6,4	47	41.98
SciQ	1	3	4,16,15,13,13,11,7,4	52	37.35
SciQ	1	4	4,16,16,14,10,8,6,4	52	33.33
SciQ	2	1	4,16,14,12,11,8,4,4	45	38.36
SciQ	2	2	4,16,14,13,12,9,6,4	50	35.90
SciQ	2	3	4,16,14,13,11,8,6,4	44	42.11
SciQ	2	4	4,16,14,12,10,8,5,4	46	36.99
SciQ	3	1	4,16,15,13,11,8,5,4	43	43.42
SciQ	3	2	4,16,15,12,12,10,8,4	46	43.21
SciQ	3	3	4,16,13,12,10,9,7,4	46	38.67
SciQ	3	4	4,16,14,13,10,8,5,4	46	37.84
ARC-Challenge	1	1	4,16,16,15,12,9,7,4	50	39.76
ARC-Challenge	1	2	4,16,13,13,10,8,6,4	46	37.84
ARC-Challenge	1	3	4,16,14,13,12,9,6,4	48	38.46
ARC-Challenge	1	4	4,16,15,12,10,8,5,4	45	39.19
ARC-Challenge	2	1	4,16,15,14,11,8,5,4	44	42.86
ARC-Challenge	2	2	4,16,15,14,11,8,4,4	39	48.68
ARC-Challenge	2	3	4,16,16,15,14,11,8,4	51	42.05
ARC-Challenge	2	4	4,16,15,12,10,8,6,4	46	38.67
ARC-Challenge	3	1	4,16,15,14,11,9,7,4	45	43.75
ARC-Challenge	3	2	4,16,15,14,13,10,8,4	49	41.67
ARC-Challenge	3	3	4,16,13,12,9,8,5,4	43	39.44
ARC-Challenge	3	4	4,16,13,13,10,9,7,4	39	48.68

J.6 OLMo2: Cross-Architecture Evidence

OLMo2 supplies the largest group median speedup, $1.631\times$ on ARC-Challenge, with $1.610\times$ on SciQ. Its complete cases avoid 42.66–46.88% of prefix-cached updates. Mean search accuracy is 79.13% on SciQ and 50.68% on ARC-Challenge, compared with 79.47% and 49.66% under round-robin. Thus the cost benefit persists with both signs of the quality difference.

Table 38: OLMo2-1B paired execution results. Updates count candidate training; time includes independent graph preparation, search, and full testing.

Task	Seed	Prefix upd.	FCT upd.	Saved (%)	Prefix (s)	FCT (s)	Ratio
SciQ	1	320	170	46.88	84.71	52.61	1.610
SciQ	2	305	169	44.59	81.37	51.29	1.587
SciQ	3	312	178	42.95	87.09	53.92	1.615
ARC-Challenge	1	293	168	42.66	119.00	79.67	1.494
ARC-Challenge	2	298	168	43.62	129.50	79.42	1.631
ARC-Challenge	3	298	163	45.30	135.27	77.67	1.741

Table 39: OLMo2-1B full-test quality. Accuracy and differences use percentages and percentage points (pp). Win/loss counts questions correct only under search or only under round-robin. Lower NLL is better. Both search executors are identical.

Task	Seed	Search acc.	RR acc.	Δ (pp)	Search NLL	RR NLL	Win/loss
SciQ	1	79.20	79.30	-0.10	0.5696	0.5682	9/10
SciQ	2	78.80	79.90	-1.10	0.5707	0.5661	11/22
SciQ	3	79.40	79.20	+0.20	0.5698	0.6017	9/7
ARC-Challenge	1	51.96	48.89	+3.07	1.1500	1.1752	85/49
ARC-Challenge	2	49.49	49.15	+0.34	1.1934	1.1939	82/78
ARC-Challenge	3	50.60	50.94	-0.34	1.1424	1.1492	37/41

Table 40: OLMo2-1B mean and sample standard deviation across three training seeds on each fixed official test set.

Task	Allocation	Accuracy (%)	NLL
SciQ	Search	79.13 ± 0.31	0.5700 ± 0.0006
SciQ	RR	79.47 ± 0.38	0.5787 ± 0.0200
ARC-Challenge	Search	50.68 ± 1.24	1.1619 ± 0.0275
ARC-Challenge	RR	49.66 ± 1.12	1.1728 ± 0.0225

Table 41: OLMo2-1B cost components in seconds. Search plus test is the primary method time. Capture is included, not added. Wall also includes method-level comparisons and bookkeeping.

Task	Seed	Method	Search (s)	Test (s)	Capture (s)	Save (s)	Wall (s)
SciQ	1	Prefix	76.95	7.76	4.48	2.06	87.44
SciQ	1	FCT	44.85	7.76	3.94	2.33	56.61
SciQ	1	RR	2.36	8.20	1.65	0.43	11.12
SciQ	2	Prefix	73.59	7.78	4.53	2.36	85.40

Task	Seed	Method	Search (s)	Test (s)	Capture (s)	Save (s)	Wall (s)
SciQ	2	FCT	43.52	7.76	3.33	2.32	54.23
SciQ	2	RR	2.35	8.17	1.61	0.37	11.05
SciQ	3	Prefix	79.34	7.75	8.92	2.22	89.98
SciQ	3	FCT	46.18	7.75	3.93	2.44	58.06
SciQ	3	RR	2.34	8.21	1.63	0.33	11.05
ARC-Challenge	1	Prefix	103.38	15.62	6.51	2.15	122.85
ARC-Challenge	1	FCT	64.11	15.56	5.81	2.46	82.76
ARC-Challenge	1	RR	3.22	15.17	2.34	0.40	19.00
ARC-Challenge	2	Prefix	113.93	15.57	14.90	2.13	132.30
ARC-Challenge	2	FCT	63.80	15.62	5.31	2.24	83.36
ARC-Challenge	2	RR	3.16	16.08	3.20	0.33	19.76
ARC-Challenge	3	Prefix	119.70	15.56	21.43	2.13	138.07
ARC-Challenge	3	FCT	62.08	15.60	5.91	2.11	81.46
ARC-Challenge	3	RR	3.21	16.06	3.24	0.44	19.89

Table 42: OLMo2-1B actual search blocks. The eight request counts sum to the prefix update count; saved updates use that sum as denominator.

Task	Seed	Block	Requests by depth	FCT upd.	Saved (%)
SciQ	1	1	4,16,14,13,11,8,4,4	37	50.00
SciQ	1	2	4,16,16,13,11,10,7,4	46	43.21
SciQ	1	3	4,16,16,14,13,11,8,4	45	47.67
SciQ	1	4	4,16,14,13,10,10,8,4	42	46.84
SciQ	2	1	4,16,15,12,10,8,4,4	39	46.58
SciQ	2	2	4,16,15,12,11,8,4,4	45	39.19
SciQ	2	3	4,16,15,15,13,9,6,4	45	45.12
SciQ	2	4	4,16,14,13,10,8,7,4	40	47.37
SciQ	3	1	4,16,12,12,10,8,5,4	39	45.07
SciQ	3	2	4,16,14,12,9,8,5,4	41	43.06
SciQ	3	3	4,16,16,16,16,12,8,4	54	41.30
SciQ	3	4	4,16,15,13,10,9,6,4	44	42.86
ARC-Challenge	1	1	4,16,12,12,8,8,4,4	36	47.06
ARC-Challenge	1	2	4,16,13,12,10,8,5,4	40	44.44
ARC-Challenge	1	3	4,16,15,15,13,8,6,4	52	35.80
ARC-Challenge	1	4	4,16,15,12,8,8,5,4	40	44.44
ARC-Challenge	2	1	4,16,12,12,9,8,4,4	35	49.28
ARC-Challenge	2	2	4,16,15,13,13,10,8,4	44	46.99
ARC-Challenge	2	3	4,16,14,12,9,8,5,4	42	41.67
ARC-Challenge	2	4	4,16,14,12,9,9,6,4	47	36.49
ARC-Challenge	3	1	4,16,12,12,8,8,4,4	36	47.06
ARC-Challenge	3	2	4,16,14,13,12,11,8,4	46	43.90
ARC-Challenge	3	3	4,16,16,13,10,8,4,4	40	46.67
ARC-Challenge	3	4	4,16,14,12,9,8,6,4	41	43.84

J.7 Summary Across Models

The three additional models contribute 72 committed blocks and 19,548 paired final predictions. The Qwen3 evidence in Appendix I completes the 24-case matrix. Runtime checks report zero-tolerance agreement in requested probabilities, scores, complete states, and final predictions. The saved request traces, block outcomes, and test results accompany the measurements, while complete-state witnesses remain in the training artifacts.

The matrix separates consistent execution gains from model-dependent learning outcomes. Mean search accuracy improves in six of eight groups, with small decreases for OLMo2 on SciQ and

SmolLM2 on ARC-Challenge. All groups retain lower cost for the identical search result. Three seeds quantify initialization and sample variation within this fixed workflow, while the two related science-question tasks delimit its application coverage. These measurements establish repeated local histories in actual adaptation requests, without requiring a uniform quality advantage from the planner.

K Equal-Time Adaptation Search

K.1 Search Protocol

The equal-time comparison tests whether faster plan evaluation can improve the selected bank within a fixed search budget. It complements the identical-outcome comparisons in Appendices I and J. Those studies stop after prescribed searches and successive commitments. Here both executors receive the same planner, data, and time allowance, so their completed candidate counts may differ. The measured contrast changes the executor's reuse unit while keeping the learning rule fixed.

This study uses Qwen3-0.6B on SciQ and ARC-Challenge with three fresh seeds. Seeds 1, 2, and 3 in this section denote 20260911, 20260912, and 20260913. Each pair uses 32 initialization examples, 256 distinct adaptation examples, and 64 validation examples. The complete official tests contain 1,000 SciQ and 1,172 ARC-Challenge questions. The model, answer interface, option handling, fixed numerical blocks, and independent cold graph pools follow Appendix I. Expert parameters and momentum use FP32, the frozen backbone uses BF16, and all other update settings remain unchanged. These six pairs form a separate workload from the earlier 24 pairs.

Every candidate specifies the full 32-event training horizon. Four consecutive eight-event blocks each assign two events to each of the four experts. Thus every candidate trains on the same 256 examples and gives each expert eight updates, in addition to its common initialization. The initial complete plan assigns experts cyclically. Unlike the earlier blockwise beam, this search compares alternative complete plans from a common root and commits only the selected final bank. No candidate gains quality by receiving more committed training data or updates.

The planner expands complete plans by quota-preserving pair swaps. Within an eight-event block, it exchanges the expert assignments at two positions assigned to different experts. There are $\binom{8}{2} - 4\binom{2}{2} = 24$ such neighbors per block and 96 across four blocks. The best-scoring unexpanded complete plan supplies a seeded ordering of its previously unseen neighbors. The planner requests batches of at most eight, ranks results by Equation 23, and resolves score ties lexicographically. The seed fixes proposal order before either executor runs. The supplied implementation and saved traces specify this deterministic search; FCT introduces no new search objective.

The primary budget is 120 seconds of model-resident optimizer wall time. Prespecified 30- and 60-second snapshots describe earlier progress from the same traces. The timer includes candidate training, validation, independent graph preparation, scheduling, transfers, numerical comparisons performed during search, and checkpoint writes. Shared model loading, data preparation, capacity selection, initialization, and post-selection testing lie outside this budget and are reported below. Equal optimizer time therefore does not mean equal total process duration.

K.2 Deadline Selection and Exact Common Requests

Only completed saved requests can contribute a bank at a deadline. Let $\tau_m(j)$ be the time when executor m has durably saved request j, including its best complete candidate so far, called the incumbent. For budget B, selection uses

$$j_m(B) = \max\{j : \tau_m(j) \le B\}, \qquad r_m(B) = \arg\max_{r \in \mathcal{R}_{1:j_m(B)}} G_V(S(r)), \tag{25}$$

with the same lexicographic tie rule. The initial complete plan is ready before the first deadline in every pair. A request that starts before B but finishes after it is ineligible at B. Its completion may contribute to a later prespecified budget. Search ends after the atomic request that reaches the final deadline; this tail is reported as overrun and never changes the earlier selection.

The common request sequence remains numerically exact even when its completed lengths differ. Induction over the deterministic planner gives the same next candidate batch whenever the com-

pleted scores agree. Runtime comparisons report zero-tolerance agreement in complete parameters, momentum, option probabilities, scores, and planner state for every common request. Saved request traces record these comparisons and each candidate's readiness. The equal-time comparison therefore selects from different-length prefixes of one candidate sequence.

A longer completed prefix cannot lower the best observed validation score, but it need not improve test accuracy. If $j_{\mathrm{FCT}}(B) \geq j_{\mathrm{prefix}}(B)$, the prefix baseline's eligible candidates are a subset of FCT's, so Equation 25 gives

$$G_V(S(r_{\mathrm{FCT}}(B))) \geq G_V(S(r_{\mathrm{prefix}}(B))). \tag{26}$$

This inequality concerns the validation objective on a fixed set. Neither the factorization nor candidate-set inclusion implies a corresponding test-accuracy inequality. The measured time points below test that separate question directly.

Test labels cannot affect any retained selection. Both search traces and all deadline selections are frozen before full testing begins. Selected banks that coincide are tested once, giving 24 distinct bank evaluations across the six pairs. Each bank answers the complete official test set with the readout in Equation 22. Repeated seeds share the same test questions, so these evaluations are not independent datasets. All results below use the frozen primary budget and include every pair.

K.3 Accuracy and Candidate Counts

The primary budget yields a positive mean accuracy difference on both tasks. SciQ increases from 79.30% to 80.23%, with paired changes of $+2.10$, $+0.20$, and $+0.50$ percentage points. ARC-Challenge increases from 56.94% to 57.14%, with paired changes of 0, $+0.60$, and 0. These are four gains and two ties. Mean paired candidate-count ratios are $3.75\times$ on SciQ and $2.51\times$ on ARC. These means of ratios differ from ratios of mean counts. The full-test NLL means also decrease on both tasks (Table 45).

Table 43: SciQ results at every prespecified budget. Accuracy is in percent. Candidate counts include the initial complete plan; only saved requests ready by the deadline count. Lower NLL is better.

Budget (s)	Seed	Prefix plans	FCT plans	Prefix acc.	FCT acc.	Δ (pp)	Prefix NLL	FCT NLL
30	1	1	25	77.40	78.50	+1.10	0.5938	0.5743
30	2	9	17	80.50	80.00	-0.50	0.5387	0.5328
30	3	9	25	78.70	78.70	+0.00	0.6931	0.6931
60	1	9	65	78.50	78.50	+0.00	0.5743	0.5743
60	2	25	57	80.00	79.90	-0.10	0.5328	0.5370
60	3	33	65	78.40	79.50	+1.10	0.5938	0.6678
120	1	33	198	78.50	80.60	+2.10	0.5743	0.5173
120	2	73	177	79.90	80.10	+0.20	0.5370	0.5261
120	3	73	206	79.50	80.00	+0.50	0.6678	0.5604

Table 44: ARC-Challenge results at every prespecified budget. Accuracy is in percent. Candidate counts include the initial complete plan; only saved requests ready by the deadline count. Lower NLL is better.

Budget (s)	Seed	Prefix plans	FCT plans	Prefix acc.	FCT acc.	Δ (pp)	Prefix NLL	FCT NLL
30	1	1	17	57.25	57.00	-0.26	1.0218	1.0139
30	2	1	9	56.31	56.31	+0.00	1.0201	1.0270
30	3	1	9	57.85	56.57	-1.28	1.0313	1.0360
60	1	17	41	57.00	57.00	+0.00	1.0139	1.0139
60	2	17	41	56.31	56.40	+0.09	1.0270	1.0311

Budget (s)	Seed	Prefix plans	FCT plans	Prefix acc.	FCT acc.	Δ (pp)	Prefix NLL	FCT NLL
60	3	17	33	57.94	57.59	-0.34	1.0390	1.0196
120	1	33	105	57.00	57.00	+0.00	1.0139	1.0139
120	2	41	97	56.40	57.00	+0.60	1.0311	1.0226
120	3	41	81	57.42	57.42	+0.00	1.0372	1.0372

Shorter budgets expose variation in the relation between validation search and test accuracy. At 30 and 60 seconds, SciQ's mean changes are $+0.20$ and $+0.33$ points, while ARC's are -0.51 and -0.09 points. Some individual SciQ seeds also decrease. FCT completes more candidates at all measured deadlines, yet additional validation-selected alternatives do not uniformly help the test metric. Figure 5(e–f) shows candidate growth and signed accuracy differences together.

Table 45: Primary 120-second uncertainty. Intervals use paired seed differences and a Student t distribution with two degrees of freedom. NLL columns are seed means; SD is the sample standard deviation of accuracy differences. Both intervals include zero.

Task	Mean Δ (pp)	SD	95% interval	Prefix NLL	FCT NLL
SciQ	+0.93	1.02	$[-1.60, 3.47]$	0.5930	0.5346
ARC-Challenge	+0.20	0.34	$[-0.66, 1.06]$	1.0274	1.0245

The three seeds support a directional result, with substantial uncertainty in its magnitude. For paired accuracy differences d_1, d_2, d_3 on each fixed official test set, we report the conventional interval

$$\bar{d} \pm t_{0.975,2} \frac{s_d}{\sqrt{3}}, \qquad s_d^2 = \frac{1}{2} \sum_{i=1}^{3} (d_i - \bar{d})^2. \tag{27}$$

This interval assumes independent training seeds and approximately normal paired differences; three observations give little information about that approximation. Both intervals include zero. Pooling the four nontied primary differences gives a descriptive two-sided sign-test value of 0.125, but pools shared-task measurements. We therefore report observed means and all signed changes without a population-level significance claim.

K.4 Completion Times and Selected Assignments

The saved times make the budget boundary explicit for every result. Tables 46 and 47 report the last eligible request's readiness and retained validation score. An incumbent can persist across several requests, so its saved readiness need not be the time at which it first became the best candidate. Test win/loss counts identify changes at the question level without treating questions as independent training runs.

Table 46: SciQ deadline readiness and validation scores. Ready time is the saved request containing the incumbent, which may have been selected earlier. Higher validation scores are better. Win/loss counts test questions correct only under FCT/prefix, respectively.

Budget (s)	Seed	Prefix ready (s)	FCT ready (s)	Prefix val. score	FCT val. score	Win/loss
30	1	4.41	27.62	-0.6160	-0.5527	27/16
30	2	28.69	23.69	-0.5535	-0.5315	13/18
30	3	27.69	28.75	-0.7195	-0.7195	0/0
60	1	34.09	57.80	-0.5527	-0.5527	0/0
60	2	52.39	56.45	-0.5315	-0.5165	24/25

Budget (s)	Seed	Prefix ready (s)	FCT ready (s)	Prefix val. score	FCT val. score	Win/loss
60	3	59.62	58.26	-0.6976	-0.6861	49/38
120	1	107.57	118.37	-0.5527	-0.5183	50/29
120	2	118.39	118.67	-0.5165	-0.4925	18/16
120	3	113.18	119.23	-0.6861	-0.6048	40/35

Table 47: ARC-Challenge deadline readiness and validation scores. Ready time is the saved request containing the incumbent, which may have been selected earlier. Higher validation scores are better. Win/loss counts test questions correct only under FCT/prefix, respectively.

Budget (s)	Seed	Prefix ready (s)	FCT ready (s)	Prefix val. score	FCT val. score	Win/loss
30	1	5.52	25.15	-0.9695	-0.9330	28/31
30	2	5.41	22.13	-1.1016	-1.0711	41/41
30	3	6.17	25.60	-1.1063	-1.1040	35/50
60	1	55.88	52.50	-0.9330	-0.9330	0/0
60	2	58.30	58.33	-1.0711	-1.0523	38/37
60	3	57.55	57.55	-1.0889	-1.0710	54/58
120	1	99.80	117.35	-0.9330	-0.9330	0/0
120	2	107.93	116.68	-1.0523	-1.0442	27/20
120	3	111.10	119.24	-1.0685	-1.0685	0/0

The selected assignments show how equal-time choices differ while training exposure stays fixed. Tables 48 and 49 list all six primary pairs. Every row assigns each expert twice within every eight-event block. Identical plans have identical complete states and test outcomes. Different plans can still tie in accuracy, since accuracy records only the final answer choice.

Table 48: SciQ selected complete assignments at 120 seconds. Digits identify experts 1 through 4; each eight-event block contains each digit twice.

Seed	Method	Events 1–8	Events 9–16	Events 17–24	Events 25–32
1	Prefix	12341234	12341234	42311234	12341234
1	FCT	12341234	12341234	42211334	12341234
2	Prefix	12314234	12341234	12341234	12341234
2	FCT	12314234	12341234	12341234	32141234
3	Prefix	12341234	13241234	12341234	12341234
3	FCT	12341234	12343214	12341324	21341234

Table 49: ARC-Challenge selected complete assignments at 120 seconds. Digits identify experts 1 through 4; each eight-event block contains each digit twice.

Seed	Method	Events 1–8	Events 9–16	Events 17–24	Events 25–32
1	Prefix	12341234	12341234	12441233	12341234
1	FCT	12341234	12341234	12441233	12341234
2	Prefix	12341234	32141234	12341234	12341234
2	FCT	12341234	12143234	12341234	12341234

Seed	Method	Events 1–8	Events 9–16	Events 17–24	Events 25–32
3	Prefix	12341234	12341234	12321434	12341234
3	FCT	12341234	12341234	12321434	12341234

K.5 Matched Work and Complete Cost Accounting

The common-request control measures execution cost for the same candidate set. It compares both methods through the last request that appears in both completed traces, including a final late request when present. FCT reduces update counts and yields observed common-trace time ratios from $1.61\times$ to $3.23\times$. These ratios include graph preparation and storage under each executor. The primary candidate counts instead use only deadline-eligible requests. Comparing the two addresses both cost reduction and additional search capacity.

Table 50: Matched-request control over the common completed trace, including the final atomic request even if late. Every compared complete state, probability vector, and score agrees exactly. This control is distinct from deadline-eligible candidate counts.

Task	Seed	Plans	Prefix updates	FCT updates	Prefix time (s)	FCT time (s)	Ratio
SciQ	1	41	663	358	132.35	41.04	3.225
SciQ	2	81	1501	777	129.19	75.57	1.709
SciQ	3	81	1372	715	122.82	70.46	1.743
ARC-Challenge	1	41	663	358	124.70	52.50	2.375
ARC-Challenge	2	49	958	503	123.14	66.80	1.843
ARC-Challenge	3	49	834	446	127.32	79.34	1.605

Both methods include their own graph-pool and request-bookkeeping costs. Execution order is prefix then FCT for SciQ seeds 1 and 3 and ARC seed 2, and the reverse for the other three pairs. Each measured search completes in one uninterrupted invocation. Table 51 retains the actual atomic overrun: it ranges from 2.83 to 12.36 seconds for prefix reuse and 0.09 to 10.81 seconds for FCT. These times are observable costs, even though the late results are ineligible for the 120-second decision.

Table 51: Actual search elapsed time. Overrun is the portion after 120 seconds; late requests cannot affect the 120-second selection. Capture is included in elapsed time. Each method uses an independent cold graph pool.

Task	Seed	Method	Elapsed (s)	Overrun (s)	Capture (s)
SciQ	1	Prefix	132.36	12.36	79.72
SciQ	1	FCT	120.66	0.66	8.38
SciQ	2	Prefix	129.20	9.20	12.71
SciQ	2	FCT	120.68	0.68	7.94
SciQ	3	Prefix	122.83	2.83	14.72
SciQ	3	FCT	120.09	0.09	8.53
ARC-Challenge	1	Prefix	124.71	4.71	47.36
ARC-Challenge	1	FCT	120.99	0.99	6.30
ARC-Challenge	2	Prefix	123.15	3.15	18.73
ARC-Challenge	2	FCT	120.84	0.84	8.49
ARC-Challenge	3	Prefix	127.33	7.33	12.77
ARC-Challenge	3	FCT	130.81	10.81	12.62

Table 52: Costs outside the optimizer budget, in seconds. Encoding, capacity probing, and initialization are shared within each pair. Banks counts distinct selected banks tested on the full official split. Postselect includes test computation, selected-state restoration, and post-selection checks.

Task	Seed	Encode	Probe	Init.	Banks	Test	Postselect
SciQ	1	0.55	2.08	1.23	3	24.05	31.40
SciQ	2	0.52	1.82	1.21	5	39.71	50.01
SciQ	3	0.50	1.80	1.21	5	40.96	52.11
ARC-Challenge	1	0.79	1.85	1.29	2	26.36	31.41
ARC-Challenge	2	0.76	1.85	1.30	4	54.37	62.60
ARC-Challenge	3	0.76	1.86	1.38	5	69.21	78.25

Graph preparation contributes to the variation between paired time ratios. Prefix reuse spends 79.72 seconds in capture for SciQ seed 1, compared with 8.38 seconds for FCT; the corresponding ARC times are 47.36 and 6.30 seconds. Capacity records include graph eviction and recapture under memory pressure. The measured benefit therefore combines fewer updates with the graph-pool costs of executing them; it is not a pure update-count ratio.

Shared and post-selection work is reported outside the optimizer timer. Dataset and tokenizer preparation take 12.11 seconds and model loading takes 1.34 seconds once for the invocation. Pair-specific encoding, capacity probing, and initialization appear in Table 52. Post-selection testing includes all distinct retained banks from both methods and all three deadlines. Its cost therefore covers the full experiment, rather than one deployed plan. The complete invocation takes 1,853.89 seconds before packaging. Bundled measurements retain complete assignments, candidate counts, ready times, validation scores, NLL, and signed test outcomes for regenerating every table and panel.

The combined evidence supports two uses of exact reuse. With fixed requests, it reduces execution cost while preserving outcomes across the four-model study. With fixed optimizer time, it expands the evaluated candidate set and improves the measured primary task means in this Qwen3 study. The latter evidence adds practical utility to the execution result while leaving broad task-quality generalization open.