BEYOND CONTEXT: PARAMETER-EFFICIENT MODEL-LEVEL EVOLUTION OF AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents accumulate experience, and most frameworks keep it in context—editable skills, memories, and workflows that stay open to inspection and revision. That openness is worth paying for, but the payment recurs: text-carried behavior occupies the context of every call that uses it, long after it has stabilized. We organize persistent agent experience into versioned, callable domain capabilities, each encoded in a separately stored parameter-efficient update to a frozen shared backbone—a LoRA patch in our implementation. We train these patches from verified records of an agent's own work and teacher-assisted supervision; only those that pass evaluation are enabled for invocation by later agent versions. At run time the parent agent selects an enabled capability inside its own task-solving process, with no separately trained dispatch module; a deterministic harness assembles the specialist's request from the task, accepted artifacts, and runtime observations under an explicit output contract. Specialists cooperate through what they produce, sharing the backbone while keeping their patches separate; editable skills remain available for coordination and rapid revision. We evaluate a registry of eight domain patches on a Qwen3-8B backbone. Each patch specializes in its domain and improves on the base model on every evaluated benchmark that contributed no training rows. Behind the agentic interface, Ours uses patch-backed specialists with typed handoffs and outperforms the base model in every reported domain. On five constructed workflows that none of the patches was trained for, the registry grounds 274 of 500 cases against 173 on base weights, a 58% relative gain on tasks requiring two to four specialist stages to cooperate. Holding all eight patches resident on one accelerator uses 82.25% less process GPU memory than eight full-model replicas, with low runtime switching latency (32.84 ms median).

1 INTRODUCTION

Agents improve through experience, and that experience can persist in more than one form. A prominent line of work keeps it in editable context: agents interleave reasoning with action (Yao et al., 2023), revise their outputs from feedback or an episodic memory (Madaan et al., 2023; Shinn et al., 2023), extract reusable insights and workflows from past episodes (Zhao et al., 2024a; Wang et al., 2025), grow libraries of executable skills (Wang et al., 2024; Zheng et al., 2025), and maintain the context itself as an evolving playbook (Zhang et al., 2026). Experience kept this way is inspectable, quick to revise, and requires no access to model weights, which makes it the right home for changing knowledge, explicit procedures, and tool protocols.

The cost of this route is recurring rather than one-time. Retrieval avoids loading an entire library, but whatever it selects occupies the context budget of that call, and a growing collection of instructions has to stay consistent and coordinated. For behavior that has stabilized—how to shape a financial program, read a database schema, or produce a chemically valid edit—the same procedure is re-supplied on nearly every task in its domain. Weights are another place such behavior can live. Yet training a specialist is only the first step: that capability still has to reach an agent, change independently of the shared backbone, and hold up beside other specialists in multi-step work. We follow that second path end to end, turning verified domain experience into versioned parameter patches. Our central question is how independently trained capabilities can be admitted, updated, and then composed at run time as reusable components of an agent.

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.

Our answer is a registry of domain-bounded, versioned model patches attached to the agent profile. A domain patch is a parameter-efficient update specialized for a class of tasks; a patch-backed specialist executes a delegated task using that patch on the shared backbone. An offline, teacher-assisted lifecycle turns verified domain answers or execution trajectories into candidate patches, evaluates each one, and promotes those that pass to callable versions; our experiments instantiate that training with LoRA (Hu et al., 2022). At inference time the parent agent uses the base backbone to select a domain patch from compact metadata, activates it, and delegates a clean, task-specific request to a patch-backed specialist; a runtime executes the specialist's artifact where the task requires one, and the parent decides how to continue or finalize. The records that a patch-backed run leaves behind return to the collection step, so an accepted version becomes the next producer of experience. Skills keep their role in coordination and rapid adaptation; patches carry the stable domain procedure.

The design separates capability updates from task execution. Each patch can be updated on its own, leaving the shared backbone and the other patches untouched, so the library grows without asking any single patch to generalize across unrelated domains. During execution, patch-backed specialists cooperate through artifacts and observations rather than merged parameters. Because their calls share one backbone, a new domain capability costs only a parameter patch rather than an additional full-model replica, keeping marginal memory cost small. We make three contributions:

A parameter-efficient agent evolution cycle with evaluation-gated capability updates. We connect verified execution records and teacher-assisted supervision to offline patch training, evaluation-gated promotion, and subsequent agent use. Patches are independently trained, stored, and versioned on a frozen shared backbone. Admission compares candidates against the in-service patch, or the base for a new capability, and bounds regression on a declared retention panel using per-domain metrics and tolerances. Accepted updates change the callable registry without modifying the backbone or other patches; subsequent executions supply experience for further updates. LoRA provides the implementation, not the definition of the framework.

An agent-directed runtime for composing independently trained specialists through typed handoffs. The parent selects an enabled, independently trained specialist inside its own task-solving process, with no separately trained dispatch module. Separately, a deterministic harness builds the specialist's request from the task, accepted artifacts, and runtime observations under an explicit output contract, so specialists share one backbone and cooperate through what they produce rather than through merged parameters. Typed handoffs achieve the highest scores among the evaluated agentic configurations: 65.53% versus 59.16% macro-average accuracy on the single-domain panel, and 54.8% versus 42.2% grounded success on the multi-domain panel, compared with Query.

Evidence of domain specialization, multi-domain composition, and memory-efficient deployment. The same eight patches improve on the base model on every evaluated benchmark that contributed no training rows. Called directly on the eight agentic benchmarks they raise macro-average accuracy from 31.69% to 68.42%, and behind the full agent interface the typed handoff stays ahead of the base model in every domain. On five multi-domain workflows none of them was trained for, grounded success rises from 173 to 274 of 500 cases. All eight stay co-resident in under 22 GiB on a single accelerator, saving 82.25% process memory over full replicas with only 32.84 ms median switching latency.

2 Related Work

Self-evolving agents are usually organized by what gets updated—model, context, tools, or architecture—and by when learning happens (Gao et al., 2026). Cutting across both is the question of where experience persists, in context or in weights (Tao et al., 2024), and that is the axis we follow.

Non-parametric agent adaptation. Experience can change an agent's behavior without changing its weights. ReAct interleaves reasoning with actions and environmental observations (Yao et al., 2023); Self-Refine uses feedback to revise an answer (Madaan et al., 2023). Across attempts or tasks, Reflexion retains verbal feedback (Shinn et al., 2023), ExpeL extracts transferable insights (Zhao et al., 2024a), and Agent Workflow Memory induces reusable workflows (Wang et al., 2025). Voyager and SkillWeaver develop executable skill libraries (Wang et al., 2024; Zheng et al., 2025), while ACE maintains evolving contextual playbooks (Zhang et al., 2026). These mechanisms keep experience inspectable and revisable. Retrieval limits the exposed context, and executable skills need not include

their implementation in the prompt. Selected text is nonetheless paid for on every call that supplies it. Our patches complement these mechanisms by moving recurring domain behavior into parameters while skills and workflows stay editable.

Parametric agent adaptation. Parameter updates let experience influence later predictions without replaying that experience in the prompt, and surveys place them inside a cycle of acquisition, refinement, updating, and evaluation (Tao et al., 2024). Several training mechanisms move context into weights: context distillation learns behavior elicited by additional context (Snell et al., 2022), PromptIntern internalizes recurrent templates and examples (Zou et al., 2024), and on-policy distillation obtains teacher feedback on student samples (Agarwal et al., 2024). Other work stores document knowledge in trained modules (Caccia et al., 2025), generates adapters from documents with a hypernetwork (Charakorn et al., 2026), and measures how much a LoRA can memorize (Back et al., 2026). SEAL learns to generate fine-tuning data and update directives using rewards on post-update performance (Zweiger et al., 2025). We use teacher-assisted offline or periodic training to turn verified domain answers or execution trajectories into task-specific patches. What we add is the treatment of trained behavior as an independently versioned capability that an agent can invoke.

Modular model adaptation and agent orchestration. A parameter update is reusable only if it can be stored, selected, and executed independently of the others. LoRA (Hu et al., 2022) provides the parameter-efficient updates instantiated in our experiments, and adapter libraries add selection and combination on top: few-shot composition (Huang et al., 2024), input-dependent retrieval and composition (Zhao et al., 2024b), task clustering with zero-shot routing (Ostapenko et al., 2024), and per-token expert mixing (Kang et al., 2025). These approaches select or combine parameter increments for a single prediction; our specialists instead exchange typed artifacts and runtime observations across a multi-step workflow, never merging patch weights. Punica and S-LoRA address the serving layer, batching heterogeneous adapters over a shared base (Chen et al., 2024; Sheng et al., 2024). These serving mechanisms make multi-adapter residency practical; our question is which capabilities should be resident at all. Adaptive Minds (Shekar & Krishnan, 2025) is closely related: a base model selects and invokes domain LoRA adapters over multiple steps. Our emphasis is the experience-to-capability lifecycle that links verified execution records to patch training, evaluation-gated promotion, and later agent use; unlike its agent-authored sub-queries, our specialist requests are assembled by a deterministic harness.

3 METHOD

Our method operates on two timescales. Within a task, the parent selects capabilities, a deterministic harness constructs specialist requests, and the returned results inform subsequent actions. Across tasks, verified execution records and teacher-assisted supervision train candidate patches; those that pass evaluation update the patch registry, and later runs supply the next round of records. Figure 1 shows this lifecycle and a zoom-in of task execution.

3.1 OVERVIEW OF THE AGENT EVOLUTION CYCLE

Between versions, what changes is which capabilities the agent can call; the shared backbone stays fixed. At accepted update t we write the agent's configuration as $\mathcal{A}_t = (f_\theta, p_{\text{sys}}, \mathcal{T}, \mathcal{S}_t, \mathcal{R}_t)$, with f_θ the shared backbone, p_{sys} the parent system prompt, $\mathcal{T}$ the tools, $\mathcal{S}_t$ the editable skills, and $\mathcal{R}_t$ the patch registry. An update rewrites $\mathcal{R}_t$ and leaves θ untouched. An agent may start at $\mathcal{R}_0 = \varnothing$, solving tasks with its base capabilities, prompts, skills, and tools.

A patch is a parameter-efficient adaptation for a class of tasks, and a patch-backed specialist is an execution role running on the shared model with one patch active, not a full-model replica. A domain may contain multiple capabilities, each represented by its own patch. For example, chemistry may carry separate updates for structure editing and property prediction, so entries carry a capability identifier i, a domain scope d_i, and a version v. The registry we evaluate holds eight patches.

A cycle runs $\mathcal{A}_t$, builds training data from the resulting experience and from teacher assistance, and proposes one new or revised patch. Skills and patches are maintained separately, and consolidating behavior into a patch does not require deleting a skill. Updates are scheduled offline.

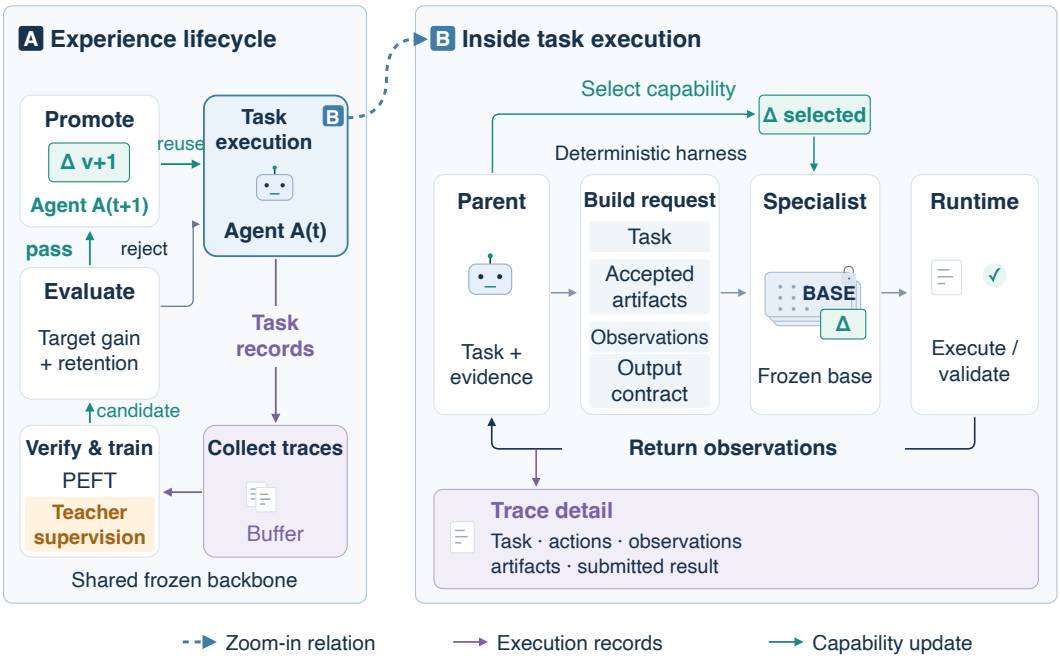


Figure 1: Experience lifecycle and task execution. (A) Verified execution traces and teacher supervision support patch training and evaluation-gated patch promotion. (B) Execution detail: the parent selects a patch, a deterministic harness builds its request, and runtime observations guide the parent's next action or final submission. Records return to experience collection.

3.2 Experience Collection and Training Data Construction

Every task the agent runs leaves a record. For task x_j executed by $\mathcal{A}_t$ the record keeps the actions taken, the environment observations they returned, and the submitted result y_j. Single-turn work reduces to (x_j, y_j); multi-step work preserves action–observation order and the artifacts produced. These records accumulate in a buffer whose producer may be an agent with no patches, or a later version already using learned capabilities.

A transcript is not yet supervision. For capability i, a verifier checks what the domain can check: labels, executable programs, database results, or molecular constraints. Records that pass the domain-specific checks can supply training targets directly; failures mark tasks for a teacher configuration—a stronger model, or an agent with skills and tools—whose outputs must pass the applicable checks before inclusion in the training set. Replay, difficulty-based selection (retaining teacher-correct, Base-wrong examples), teacher repair, and selective short rationales are per-capability recipe choices, not stages every patch passes through.

We keep the two sources labelled, since teacher-only examples can bootstrap a capability before enough agent experience exists. Retained records are serialized into sequences $\mathcal{Z}_{t,i}$ that retain source and format. Action–observation order is preserved so that each action is conditioned only on information available at that point. The domain recipe decides which of task evidence, assistant actions, and final outputs enters the sequence. Appendix F details the data sources, verification procedures, and training configurations for the eight evaluated patches.

3.3 Parameter-Efficient Capability Updates

Training adds a capability or sharpens an existing one, and never rewrites the shared backbone. A candidate $\widehat{\Delta}_i$ minimizes the standard masked autoregressive loss $-\sum_{z\in\mathcal{Z}_{t,i}}\sum_{j\in\mu(z)}\log p_{\theta;\Delta}(z_j \mid z_{<j})$, with θ frozen and $\mu(z)$ the supervised positions. The mask, and whether intermediate reasoning or tool turns are supervised, follow the actual records: every training configuration uses the Qwen3 chat template and excludes prompt tokens from the loss.

The framework asks only that an update be parameter-efficient and separately storable. Our implementation uses LoRA (Hu et al., 2022), storing the low-rank patch separately from θ.

A new capability takes a new identifier i; an existing one takes a candidate version, initialized from its current patch or trained afresh as its recipe specifies. Either way the candidate is trained toward behavior that holds under the runtime task contract without the teacher's full instructional context.

3.4 Evaluation and Agent Version Updates

Admission compares a candidate against what the agent already calls for that capability. Let Δ_i^{ref} be the current promoted patch for capability i, or no patch when i is new, and let $Q_i(\Delta)$ measure performance on fixed held-out tasks under the specified evaluation contract. The gate scores an existing capability's candidate against the patch in service rather than against the bare base:

$$Q_i(\widehat{\Delta}_i) - Q_i(\Delta_i^{\mathrm{ref}}) \geq \epsilon_i, \qquad Q_u(\widehat{\Delta}_i) - Q_u(\Delta_i^{\mathrm{ref}}) \geq -\delta_u \quad \forall u \in \mathcal{U}_i, \tag{1}$$

with $\epsilon_i > 0$ the required target gain, $\mathcal{U}_i$ a declared retention or containment panel that includes benchmarks contributing no training rows, and $\delta_u \geq 0$ the regression it tolerates. Metrics, panels, and tolerances are set per capability.

The outcome is recorded in the registry as $r_{i,v} = (i, d_i, v, \Delta_{i,v}, m_{i,v}, g_{i,v}, s_{i,v})$, where m holds the capability description and input/output contract, g the admission evidence, and s the status: `candidate`, `promoted`, `retired`, or `rolled-back`. Only promoted entries $\mathcal{R}_t^+$ are callable. Acceptance adds a capability or moves its active version pointer, giving $\mathcal{R}_{t+1}$ and hence $\mathcal{A}_{t+1}$ and leaving other entries and θ unchanged; rejection keeps $\mathcal{A}_t$ in service; rollback points back to a retained version or disables the patch. An accepted version then becomes the next producer of experience.

3.5 Agent-Directed Specialist Composition with Typed Handoffs

Execution works the same before and after an update; only the set of callable capabilities differs. Given task q, history h_k, and the metadata $\{m_r : r \in \mathcal{R}_t^+\}$ of promoted entries, the parent decides within its own reasoning whether to delegate, use another tool, or submit, bounded by tool prerequisites and output contracts. Delegation selects $r_k \in \mathcal{R}_t^+ \cup \{\perp\}$, where $\perp$ means running the step without a patch. No separately trained dispatch module makes this choice, and the order is not a fixed pipeline: the parent may revisit a capability or redirect after feedback.

Choosing parameters and choosing what to send are separate operations. In our LoRA runtime `load_lora` activates the patch for a delegated forward pass; a deterministic harness assembles the input from the original task, applicable public instructions, accepted prior artifacts, runtime observations, and the selected capability's output contract. The patch-backed specialist therefore prefills a freshly built request, not the parent's verbatim transcript, and inherits no hidden reasoning and no KV cache. This construction preserves task-relevant evidence while excluding conversational context the contract does not require. The inherited-transcript *Concat* arm in our experiments arranges that input differently, without changing the learning rule.

What comes back is a typed result or artifact, which a deterministic runtime executes or validates where the contract requires, returning rows, structures, execution feedback, or computed scores; these observations extend h_k and hand control back to the parent. Patch-backed specialists therefore cooperate through what they produce, sharing one backbone while keeping their parameter updates separate. Independently trained patches compose at run time through typed artifacts and explicit input/output contracts. The finished record returns to the collection step of Section 3.2, where execution becomes training data again.

For a fixed delegated input, a textual skill adds its body to the prompt while patch activation does not; task evidence, output instructions, and the parent's catalog stay in context either way. This is per-call accounting; trajectory totals also depend on catalog size and call count. Measured context cost appears in Section 4.3. Because delegated calls differ only in which patch is active on the shared backbone, a new capability adds a patch rather than another resident model, at the deployment cost measured in Section 4.5.

Table 1: Overview of evaluation settings and criteria.

Panel	What is tested	How it is tested	Evaluation criterion
Standalone	Specialist domain capability	Base and patch each answer once, without an agent or router.	Each benchmark's native metric.
Single-domain	Single-domain agent performance	Eight domain testbenches: direct answers or parent selection, specialist execution, and final submission.	Direct: domain answer score. Agentic: final-answer correctness and required runtime consistency.
Multi-domain	Cooperation across specialists	Five benchmark-derived workflows; 100 cases each, 2–4 specialist stages.	Success rate: final answers pass workflow checks and required artifact-consistency checks.
Deployment	Resource cost	One backbone + eight patches vs. eight replicas; single-stream timing on the shared backbone.	Resident GPU memory; switch, registration, and inference latency.

4 Experiments and Results

4.1 Setup

Table 1 summarizes four evaluation panels for eight domain patches (Table 2) sharing Qwen3-8B (Yang et al., 2025). Each patch was trained independently, without other patches or the constructed workflows. In single-domain evaluation (Table 3), *Base*, *Pure*, and *Skill* directly answer using base weights, a patch, or a textual skill, respectively. In multi-domain evaluation (Table 4), *Base* uses base-weight specialist calls and *Skill* uses routed textual skills. In both agentic panels, *Concat* passes the accumulated transcript and *Ours* supplies a typed task-specific request. *Query* uses the routing of *Ours* and a separate Base writer to compose the specialist request, appending the public contract. We additionally evaluate an unadapted Qwen3-235B-A22B-Instruct-2507-FP8 generalist under the same agentic evaluation protocols, using it for both parent and specialist turns without adapters. Appendices A–B detail configurations and data roles; Appendix C gives full standalone results; Appendices D–E specify agentic scoring; Appendix G details Query.

4.2 Standalone Specialization Reaches Beyond Its Training Source

Table 2 shows that the eight registry entries are weight-level specialists whose capability is not confined to the benchmark family that produced their training data.

On held-out splits of their own training families, the patches advance backbone performance by substantial margins, reaching $+33.98$ pp on WikiSQL Dev (52.40% to 86.38%) and $+58.64$ pp on EHRSQL Valid (29.06% to 87.70%), while MedMCQA improves by $+4.52$ pp over an already strong 58.91% base. On a stratified 200-problem sample from the official APPS test split, broad coding improves from 11.0% to 22.5% ($+11.5$ pp).

On benchmarks that supplied no training rows, SQL lifts KaggleDBQA from 20.00% to 38.92% ($+18.92$ pp), and cybersecurity achieves a $+10.90$ pp gain on exact CVSS-vector prediction alongside $+7.56$ pp on CTIBench MCQ. Trained on APPS, Magicoder, CodeFeedback, and Bash-command data (Appendix F), the broad-coding patch improves HumanEval+ from 76.83% to 81.10% ($+4.27$ pp) and improves DS-1000 execution by $+2.91$ pp. Similarly, the data-analyst patch, trained on synthetic DataFrame tasks, achieves 96.41% on retained V3 generation while transferring to DS-1000 execution ($+3.49$ pp).

The law patch gains $+19.66$ pp on its train-family panel and improves MMLU Law ($+0.91$ pp) and the 146-task LegalBench macro ($+0.68$ pp). Chemistry editing achieves up to $+51.00$ pp on TOMG-Bench substitution and advances related molecular editing tasks in MolLangBench ($+3.50$ pp) and ChEBI captioning (Token F1 increases from 0.1734 to 0.2025; Appendix C).

All 60 reported evaluation rows in Appendix C improve on the matched base.

Table 2: Standalone specialization across eight domains. Accuracy (%); parentheses: gains over Base. Full 60-row results in Appendix C. MolLang edit is a related task; other transfer benchmarks supplied no training rows.

	Train-family		Transfer evaluation	
Domain	Benchmark	Accuracy	Benchmark	Accuracy
SQL	WikiSQL	86.4 (+34.0)	KaggleDBQA	38.9 (+18.9)
Finance	ConvFinQA	64.5 (+58.1)	FinanceReasoning	50.8 (+11.7)
Medicine	EHRSQL	87.7 (+58.6)	MMLU-Medical	81.0 (+2.9)
Cybersecurity	CyberBench	63.0 (+23.2)	CTIBench	65.4 (+7.6)
Law	Legal panel	56.5 (+19.7)	LegalBench	62.8 (+0.68)
Chemistry	TOMG subst	70.0 (+51.0)	MolLang edit	10.5 (+3.5)
Broad coding	APPS	22.5 (+11.5)	HumanEval+	81.1 (+4.27)
Data analyst	DataFrame Gym V4	65.9 (+40.3)	DS-1000	27.3 (+3.49)

4.3 Single-Domain Evaluation: Capability and Handoff Effects

In Table 3, called directly, the base backbone reaches 31.69% macro-average accuracy and the patches 68.42%. Adding parent selection, tool interaction, and finalization yields 58.92% when the specialist inherits the parent's transcript and 65.53% under our typed handoff (with the model-written *Query* arm at 59.16%). All eight rows of Ours exceed the direct base, with broad coding reaching 133/164 against 126/164 on HumanEval+. These end-to-end scores combine expert capability with routing and submission behavior.

Compared with the unadapted 235B generalist, Ours achieves higher accuracy on six of eight domains and a higher macro-average (65.53% versus 53.69%). The generalist remains stronger on broad coding and cybersecurity, while the largest advantages of Ours occur in SQL, law, and chemistry editing.

The largest Ours–Concat differences occur in chemistry editing and law. Concatenation is slightly ahead on SQL (77.11 versus 76.07) and medicine (76.20 versus 76.00). On chemistry editing and law it is the difference between keeping the patch's capability and losing much of it: concatenation scores 47.00 and 35.60, our typed handoff 70.00 and 53.60, against direct-patch scores of 70.00 and 54.00. The stage counts place the loss inside the specialist forward. Under concatenation the chemistry specialist produces an RDKit-valid edit on 91 of 100 cases and satisfies the substitution constraint on 47; under our typed handoff it is valid on 99 and constraint-satisfying on 70. The law specialist's exact answer rises from 89 to 134 of 250 under the same change, and finance (74.79 to 76.18) and the data analyst (56.82 to 63.01) move the same way by smaller margins. A typed handoff that gives the specialist less, but exactly what its contract needs, returns chemistry to its direct-patch score and law to within 0.40 pp of it.

The model-written *Query* arm first selects a specialist, then asks the Base model to write its request. Its 59.16% macro-average accuracy is below Ours at 65.53%, although it scores higher on data analysis (66.28 versus 63.01), cybersecurity (30.90 versus 28.30), and chemistry (71.00 versus 70.00), and lower on the other five domains. The macro difference is $+6.37$ pp. Query's corresponding-specialist coverage ranges from 90.39% in finance to 100% in three domains; data analysis reaches 1,085/1,195. Appendix G reports coverage, writer cost, and run conditions.

The *Skill* baseline supplies domain instructions without a patch, adding 128–209 prompt tokens per call. It improves six of eight benchmarks, ties on coding, and lowers chemistry editing by 3.00 pp, reaching 38.64% macro-average accuracy. Finance gains most (24.51% to 50.42%), while chemistry falls to 18.00% against Ours at 70.00%. Ours exceeds Skill on all eight benchmarks.

4.4 Multi-Domain Evaluation: Specialist Composition

Table 4 reports the five constructed workflows. Ours grounds 274/500 (54.8%), exceeding Base (173/500, 34.6%), routed Skill (202/500, 40.4%), and unadapted Qwen3-235B-A22B-Instruct-2507-FP8 (253/500, 50.6%; Appendix B)—a 58% relative gain (+101 cases) over Base. Against the 235B generalist, Ours gains 21 grounded successes overall ($+4.2$ pp), with higher scores in medicine, cybersecurity, and law, while the generalist performs better in finance and chemistry.

Table 3: Single-domain agentic evaluation on adapted domain testbenches, with direct-generation baselines and unadapted *235B* (Appendix B). Query, Concat, and Ours use patch-backed specialists and parent finalization. *Base* and *Pure* answer directly without and with a patch; *Concat* gives the specialist the full transcript, *Ours* a typed handoff. *Query* selects the specialist before a Base call writes its request (Appendix G). *Skill* is the context-level baseline, and *Tok.* its added prompt tokens.

		Primary metric (%)							
Specialist / agentic testbench	N	Base	Skill	Pure	Query	Concat	Ours	235B	Tok.
SQL / WikiSQL execution	1,726	46.47	47.86	76.13	42.53	77.11	76.07	42.64	+128
Finance / TAT-QA program	718	24.51	50.42	87.47	66.43	74.79	76.18	72.01	+209
Medicine / PubMedQA	1,000	34.70	44.60	79.20	75.50	76.20	76.00	65.40	+140
Cybersecurity / CTIBench CVSS	1,000	20.70	21.90	33.50	30.90	27.60	28.30	32.00	+167
Chemistry / TOMG substitution	100	21.00	18.00	70.00	71.00	47.00	70.00	52.00	+133
Broad coding / HumanEval+	164	76.83	76.83	81.10	79.88	76.22	81.10	87.80	+129
Law / ECtHR-A	250	3.60	13.20	54.00	40.80	35.60	53.60	29.60	+147
Data analyst / DataFrame Gym V4	1,195	25.69	36.32	65.94	66.28	56.82	63.01	48.03	+148
Macro avg.	—	**31.69**	**38.64**	**68.42**	**59.16**	**58.92**	**65.53**	**53.69**	—

Table 4: Multi-domain evaluation: grounded successes on five constructed 100-case workflows (parentheses: differences from Base). All arms retain the workflow scaffold: *Base* (base weights, no skills), *Skill* (textual skills, no adapters), *Query* (patches with model-written queries), *Concat* (patches with full parent history and output contracts), *Ours* (patches with typed handoffs), and unadapted *Qwen3-235B-A22B-Instruct-2507-FP8*. Protocols: Appendices B, E, G.

Workflow	Base	Skill	Query	Concat	Ours	235B
Finance → SQL → Finance	3	19 (+16)	0 (−3)	53 (+50)	46 (+43)	**82** (+79)
Chemistry → SQL → Data analyst → Chemistry	44	47 (+3)	43 (−1)	30 (−14)	64 (+20)	**75** (+31)
SQL → Medicine	39	41 (+2)	49 (+10)	50 (+11)	**51** (+12)	47 (+8)
SQL → Cybersecurity → CVSS	24	30 (+6)	32 (+8)	31 (+7)	**33** (+9)	21 (−3)
Law → SQL → Law	63	65 (+2)	**87** (+24)	54 (−9)	80 (+17)	28 (−35)
Total	**173**	**202** (+29)	**211** (+38)	**218** (+45)	**274** (+101)	253 (+80)
Rate	**34.6%**	**40.4%**	**42.2%**	**43.6%**	**54.8%**	50.6%

The model-written *Query* arm grounds 211/500 (42.2%), compared with 274/500 under typed handoffs. Its performance varies across workflows: law reaches 87/100, above the typed arm's 80, whereas finance scores 0/100 despite invoking all required specialists, executing the retrieval and calculation, and faithfully submitting the runtime result on every case.

Concat and Query compare full-history and model-written requests with typed handoffs at the system level. To isolate the role of specific task information, we intervene at 30 SQL, finance, and law handoffs with fixed experts and upstream states (Appendix G.2). Seven cases show both recovery when omitted public information is restored to Query and failure when the same information is removed from Typed. In finance, restoring the prior calculation definition raises specialist correctness from 0/10 to 5/10; removing it lowers Typed from 4/10 to 0/10. These paired results support task-state preservation as a mechanism of effective handoff. Chemistry shows a separate execution bottleneck: 47/100 Query trajectories exhaust the eight-step budget (Appendix G).

The parent chooses among all eight entries at each step rather than following a supplied path (Appendix B), and the workflows were constructed after the patches were trained, so the panel tests multi-domain use of independently trained patches without workflow-specific joint training.

Ours exceeds Base and routed Skill in all five workflows (+72 cases over Skill), and grounds 274/500 cases against Concat's 218/500 (+56 cases), with largest gains in chemistry (64 versus 30) and law (80 versus 54); finance favors Concat (53 versus 46), while medicine and cybersecurity differ by only one and two cases.

Table 5: Deployment cost: resident memory of shared and replicated specialists, and single-stream latency of the shared-backbone patch bank.

(a) Resident memory

Metric	Shared	Replicas
Process GPU memory (MiB)	**22,202**	125,104
Framework-allocated memory (GB)	21.63	131.07
Logical weight payload (GB)	21.64	131.18
Summed peak host RSS (GB)	10.72	84.61

(b) Single-stream latency

Operation	Median
Warm patch switch	32.84 ms
Cold patch registration	1.887 s
Base inference path	94.07 ms
Patch inference path	145.78 ms

Setup: BF16 on 64 GiB accelerators. Shared: 1 backbone + 8 patches on 1 GPU; replicas: 8 backbones on 8 GPUs. The same snapshot measured 15,638 MiB for the backbone alone. Latencies use the patch bank only, without batching. Base and patch inference paths each use a full 512-token prefill with no cache carried across calls; switch and registration timings measure adapter operations.

4.5 Deployment Cost: One Backbone Instead of Eight

Every expert is a low-rank increment on weights the others already share, so a backbone of P bytes with N resident patches of sizes δ_i occupies roughly $P + \sum_{i=1}^{N} \delta_i$ bytes, against roughly NP for separately hosted specialists, counting weights only and excluding activations, KV cache, and framework overhead; the relative saving grows with library size toward a limit set by the adapter-to-backbone ratio. Table 5 measures both sides: one backbone with eight resident patches occupied 22,202 MiB of process GPU memory on a single accelerator, against 125,104 MiB for eight independently resident backbones across eight accelerators, the layout that hosting one full model per specialist implies—an 82.25% reduction, or $5.63\times$ smaller, consolidating eight processes on eight devices into one process on one.

A single backbone alone occupied 15,638 MiB in the same snapshot, so making all eight specialists simultaneously available added 6,564 MiB of process-level cost rather than seven further backbone replicas; in storage terms the eight patches total 5.24 GB beside a 16.40 GB backbone. Every dispatch in the agentic panel and every composed workflow ran without holding more than one backbone.

Selecting a resident adapter takes 32.84 ms at the median; cold registration costs 1.887 s, paid once at load. With a full 512-token prefill, patch inference adds 51.71 ms over the base path (Table 5). These measurements establish a memory-residency advantage, not a throughput advantage; the single-stream latencies characterize the evaluated implementation and probe configuration.

5 Conclusion

Across a registry of eight domain patches on one shared backbone, specialists acquire capability that reaches beyond the families that trained them. Typed handoffs improve single-domain macro-average accuracy over full-history and model-written requests, with gains varying by domain. Handoff comparisons and paired input interventions further show that preserving task-critical information helps specialists apply their learned capabilities during agent execution. The specialists also compose across constructed multi-domain workflows, where the registry grounds 274 of 500 cases against 173 on base weights. Because each is a low-rank increment, all eight stay resident on one accelerator in 82.25% less process GPU memory than eight full-model replicas. These callable patches complement editable skills: skills support procedures that must be read, revised, or bound to a tool, while patches carry stable, verified domain behavior consolidated offline with a teacher and admitted after evaluation. Together, they let an agent coordinate through editable context and reuse domain capabilities by loading the corresponding patches.

Future work. Extending this architecture to larger backbones would test whether stronger base reasoning carries into specialist composition, and a broader registry would test whether the admission gate holds as domains accumulate. Alternating evolution cycles are a further direction: skills adapt in context to new demands, verified trajectories consolidate into patches, and redundant skills are retired once capability preservation and workflow compatibility have been checked. Each of these extends the same shift this paper argues for—from experience that is re-supplied to experience that is loaded.

Reproducibility Statement

Appendix A specifies the experimental configuration and deployment measurement setup, and Appendix B describes evaluation-panel construction. Appendix F documents the domain-specific training sources, data construction and verification procedures, and hyperparameters. Appendices D and E detail the single-domain and multi-domain execution procedures, including inputs, output contracts, and scoring rules. Appendix G further specifies the model-written Query condition and the paired input interventions used to analyze handoff behavior.

AI Use Statement

We used generative AI tools for literature search, manuscript editing, and language refinement. LLM-generated teacher supervision was used in training-data construction, as described in Section 3.2. The domain-specific verification procedures and experimental evidence are described in Sections 3.2 and 4. The authors take full responsibility for the final manuscript, including its claims, references, and results.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.13649.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, S. K. Hong, Youngjune Gwon, and Sungjin Ahn. Understanding LoRA as knowledge memory: An empirical analysis. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.01097.

Lucas Caccia, Alan Ansell, Edoardo Ponti, Ivan Vulić, and Alessandro Sordoni. Training plug-n-play knowledge modules with deep context distillation. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2503.08727.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026. URL https://arxiv.org/abs/2602.15902.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *Proceedings of Machine Learning and Systems*, 2024. URL https://arxiv.org/abs/2310.18547.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://arxiv.org/abs/2109.00122.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. ConvFinQA: Exploring the chain of numerical reasoning in conversational finance question answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022. URL https://arxiv.org/abs/2210.03849.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. URL https://arxiv.org/abs/2507.21046.

Neel Guha, Julian Nyarko, Daniel Ho, Christopher Ré, Adam Chilton, Aditya K, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel Rockmore, Diego Zambrano, Dmitry Talisman, Enam Hoque, Faiz Surani, Frank Fagan, Galit Sarfaty, Gregory Dickinson, Haggai Porat, Jason Hegland, Jessica Wu, Joe Nudell, Joel Niklaus, John Nay, Jonathan Choi, Kevin Tobia, Margaret Hagan, Megan Ma, Michael Livermore, Nikon Rasumov-Rahe, Nils Holzenberger, Noam Kolt, Peter Henderson, Sean Rehaag, Sharad Goel, Shang Gao, Spencer Williams, Sunny Gandhi, Tom Zur, Varun Iyer, and Zehua Li. LegalBench: A collaboratively built benchmark for measuring legal reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/89e44582fd28ddfea1ea4dcb0ebbf4b0-Abstract-Datasets_and_Benchmarks.html.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoRAHub: Efficient cross-task generalization via dynamic LoRA composition. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2307.13269.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, 2019. URL https://arxiv.org/abs/1909.06146.

Junmo Kang, Leonid Karlinsky, Hongyin Luo, Zhen Wang, Jacob Hansen, James Glass, David Cox, Rameswar Panda, Rogerio Feris, and Alan Ritter. Self-MoE: Towards compositional large language models with self-specialized experts. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.12034.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2211.11501.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. EHRSQL: A practical text-to-sql benchmark for electronic health records. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/643e347250cf9289e5a2a6c1ed5ee42e-Abstract-Datasets_and_Benchmarks.html.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.01210.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.17651.

Oleksiy Ostapenko, Zhan Su, Edoardo Ponti, Laurent Charlin, Nicolas Le Roux, Lucas Caccia, and Alessandro Sordoni. Towards modular LLMs by building and reusing a library of LoRAs. In *International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/ostapenko24a.html.

Pavan C. Shekar and Aswanth Krishnan. Adaptive Minds: Empowering agents with LoRA-as-tools. *arXiv preprint arXiv:2510.15416*, 2025. URL https://arxiv.org/abs/2510.15416v2. Version 2, revised June 2026.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Proceedings of Machine Learning and Systems*, 2024. URL https://arxiv.org/abs/2311.03285.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022. URL https://arxiv.org/abs/2209.15189.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024. URL https://arxiv.org/abs/2404.14387.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2305.16291.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2409.07429.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2024a. URL https://arxiv.org/abs/2308.10144.

Ziyu Zhao, Leilei Gan, Guoyin Wang, Wangchunshu Zhou, Hongxia Yang, Kun Kuang, and Fei Wu. LoraRetriever: Input-aware LoRA retrieval and composition for mixed tasks in the wild. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024b. URL https://arxiv.org/abs/2402.09997.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017. URL https://arxiv.org/abs/1709.00103.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021. URL https://arxiv.org/abs/2105.07624.

Jiaru Zou, Mengyu Zhou, Tao Li, Shi Han, and Dongmei Zhang. PromptIntern: Saving inference costs by internalizing recurrent prompt during large language model fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024. URL https://arxiv.org/abs/2407.02211.

Adam Zweiger, Jyothish Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/6b41e04c41726e2a60e456d0a2b961ab-Abstract-Conference.html.

A Experimental Configuration

Registry and training scale. Frozen training scale ranges from 3,462 rows for the data analyst to 78,176 for medicine, at LoRA rank 64 for seven domains and rank 32 for law. Each patch is the promoted artifact for its domain; data recipes, verifiers, and gate settings are domain-specific rather than one shared configuration. Representative benchmark sources include WikiSQL (Zhong et al., 2017), FinQA and ConvFinQA (Chen et al., 2021; 2022), TAT-QA (Zhu et al., 2021), PubMedQA (Jin et al., 2019), EHRSQL (Lee et al., 2022), APPS (Hendrycks et al., 2021), HumanEval+ (Liu et al., 2023), DS-1000 (Lai et al., 2023), and LegalBench (Guha et al., 2023). Training recipes combine checked model outputs, teacher-generated targets, and reference-label supervision in domain-specific proportions; Appendix F details the sources and construction. Panel construction and scoring details are given in Appendices B and D.

Scoring harnesses for the single-domain panel. In the single-domain panel (Table 3), the textual skill arm uses one deterministic direct forward per case with thinking disabled and, like the two direct model arms, no agentic wrapper. Broad coding is scored with the official EvalPlus HumanEval+ harness, tabular data analysis with the frozen DataFrame Gym execution evaluator with execution and submission rules detailed in Appendix D, and the remaining six domains with their frozen benchmark-native scorers.

Multi-domain workflow shapes. The five constructed workflows of Table 4 range from a two-stage SQL-to-medicine chain to a four-stage chemistry workflow that passes through SQL and the data analyst before chemistry validates the result.

Deployment probe and platform. The single-stream probe uses a 512-token prompt with one generated token and no batching, on a single GPU with 64 GiB of device memory. The microbenchmark uses eight such devices with Torch 2.1.2, Transformers 4.57.1, and PEFT 0.15.2.

B Panel Construction

Standalone protocol and row roles. Each patch answers once against the matched base backbone, without an agent or router. Benchmark scorers grade the answer and execute generated artifacts where the metric requires. A row is *train-family* when its testbench shares a source family with the training data, *never-trained* when the benchmark supplied no training rows, *related task* when it is an untrained task within a benchmark that supplied training data for another task, and *retention* when it checks whether an earlier version's capability is preserved.

APPS test subset. The coding train-family evaluation samples 200 problems from the official APPS test split, stratified as 40 introductory, 120 interview, and 40 competition problems (seed 20260925). All sampled problems use standard input/output. Both arms receive identical prompts and generate once with greedy decoding, thinking disabled, a 16,384-token context limit, and a 4,096-token output limit. Base disables the adapter and the coding arm enables it in the same model instance; paired generations use the same GPU, alternate arm order, and start with fresh prefills. The model sees the question, starter code, and calling convention; tests and reference solutions remain outside the prompt. The original APPS execution harness requires all tests to pass, with a 4-second per-test limit and an 11-second outer timeout. Only code-fence formatting is stripped; answers are not repaired. All 200 problems remain in the denominator, including timeouts and output-limit cases.

Single-domain agentic arms and cross-panel protocols. The six 8B arms in Table 3 use the same eight domain testbenches within that panel. Their generation modes and handoffs are defined in the table caption; Appendix D details the single-domain execution and scoring procedure, and Appendix G details Query.

The standalone and agentic panels are reported separately: a benchmark name does not establish identical task inventories, prompts, or scoring contracts. The HumanEval+ and DataFrame Gym V4 direct columns reproduce the standalone values, whereas PubMedQA uses exact answers in the standalone panel and parsed judgment JSON in the agentic panel. The task inventory, invocation sequence, and domain-specific scoring rules are detailed in Appendix D.

Agentic diagnostics. Besides grounded success, the agentic panel records four stage-level submetrics—parse validity, specialist correctness, runtime faithfulness, and finalization validity—which localize where a trajectory loses cases. The per-domain inputs, specialist artifacts, and success criteria are in Appendix D; the per-domain scoring harnesses are in Appendix A.

Multi-domain workflow construction. The five workflows are constructed from benchmark material rather than drawn from an official suite. Each has 100 cases, two to four specialist stages, and a deterministic runtime. Scoring requires verifier-confirmed success under the workflow's typed contract. The five 8B arms share the parent, tool set, workflow, and runtime: Ours uses the patch registry, Base runs specialist forwards with base weights, and Skill uses routed textual skills. Query adds a Base query writer as specified in Appendix G; Concat supplies the accumulated parent history and public output contracts.

Each of the five workflows has a latent dependency graph that is used to build its cases but is never shown to the parent and never enforced as a call sequence; at every decision the parent may choose any of the eight registry entries, so routing remains part of end-to-end task execution rather than a supplied solution path. Appendix E describes the shared execution procedure and illustrates artifact dependencies in each workflow.

235B generalist baseline. The unadapted Qwen3-235B-A22B-Instruct-2507-FP8 baseline is evaluated under the identical agentic execution environment (vLLM 0.21.0, thinking disabled, greedy decoding, eight-step budget) across both multi-domain workflows and single-domain testbenches. It supplies parent routing and execution turns directly using its generalist weights without adapters. In multi-domain evaluation (Table 4), it achieves 253/500 grounded successes (50.6%), trailing Ours (274/500, 54.8%) by 21 cases (+4.2 pp overall). In single-domain agentic evaluation across the domain testbenches, the unadapted 235B generalist achieves: SQL 42.64% (736/1,726; Ours 76.07%), Finance 72.01% (517/718; Ours 76.18%), Medicine 65.40% (654/1,000; Ours 76.00%), Cybersecurity 32.00% (Ours 28.30%), Chemistry 52.00% (52/100; Ours 70.00%), Broad coding 87.80% (144/164; Ours 81.10%), Law 29.60% (74/250; Ours 53.60%), and Data analyst 48.03% (574/1,195; Ours 63.01%). Across all eight domain benchmarks, the 235B generalist reaches an 8-domain macro-average of 53.69%, compared to 65.53% under Ours (8B Base: 31.69%). The 8B backbone equipped with specialized typed-handoff patches thus outperforms the unadapted 235B generalist on six of the eight domains, establishing an overall margin of $+11.85$ pp.

C Complete Standalone Results

Table 6 reports all 60 evaluation rows in the complete standalone panel analyzed in Section 4.2. Metrics are benchmark-native; the ChEBI SMILES-to-caption row is Token F1 rather than a percentage. The LegalBench evaluation generated 85,154 responses across 147 tasks; its reported macro average covers 146 automatically scored tasks and excludes the manual-evaluation task, `rule_qa`. For this row, N denotes total generated responses. Aggregate rows and their subgroups both appear here and must not be summed together. Role labels follow Appendix B; rows marked *subgroup* are components of the aggregate row above them.

Table 6: Complete standalone results for all eight patches against the matched base backbone, including a ChEBI captioning probe for the chemistry patch. ChEBI Token F1 is on a 0–1 scale; other values are percentages, with differences in percentage points.

Role	Testbench	N	Metric	Base	Patch	Δ
SQL						
Train-family	BIRD MiniDev	500	Exec. acc.	41.00	50.40	+9.40
Train-family	Spider Dev	1,034	Exec. acc.	68.86	76.31	+7.45
Train-family	WikiSQL Dev	8,421	Exec. acc.	52.40	86.38	+33.98
Train-family	SParC Dev	1,199	Exec. acc.	65.64	76.81	+11.17
Train-family	CoSQL Dev	982	Exec. acc.	64.15	70.98	+6.83
Never-trained	KaggleDBQA	185	Exec. acc.	20.00	38.92	+18.92
Finance						
Train-family	ConvFinQA Dev	1,003	Strict prog./ans.	6.38	64.51	+58.13
Train-family	FinQA Dev	883	Strict prog./ans.	7.36	64.78	+57.42
Train-family	MultiHiertt Dev	842	Strict prog./ans.	1.78	41.69	+39.91
Train-family	TAT-QA Dev	718	Strict prog./ans.	20.33	79.67	+59.34
Never-trained	BizFinBench v2 Quant EN	1,000	Answer acc.	31.50	37.60	+6.10
Never-trained	FinanceReasoning	2,238	Answer acc.	39.10	50.76	+11.66
Medicine						
Train-family	MedMCQA	4,183	Exact ans.	58.91	63.42	+4.52
Train-family	MedQA	1,273	Exact ans.	63.24	67.71	+4.48
Train-family	PubMedQA	1,000	Exact ans.	54.50	79.30	+24.80
Train-family	EHRSQL Valid	1,163	Exec. acc.	29.06	87.70	+58.64
Train-family	EHRSQL Test	1,167	Exec. acc.	28.19	85.78	+57.58
Never-trained	MMLU-Medical (Aggregate)	945	Acc.	78.10	80.95	+2.86
Never-trained subgroup	MMLU-Medical (Anatomy)	135	Acc.	68.89	75.56	+6.67
Never-trained subgroup	MMLU-Medical (Clinical Knowledge)	265	Acc.	79.62	81.51	+1.89
Never-trained subgroup	MMLU-Medical (College Medicine)	173	Acc.	74.57	78.03	+3.47
Never-trained subgroup	MMLU-Medical (Medical Genetics)	100	Acc.	81.00	87.00	+6.00
Never-trained subgroup	MMLU-Medical (Professional Medicine)	272	Acc.	82.35	82.72	+0.37
Cybersecurity						
Train-family	CyberBench (exact-gradable holdout)	3,000	Task-aware exact	39.80	63.00	+23.20
Never-trained	CTIBench (MCQ)	2,500	Acc.	57.80	65.36	+7.56
Never-trained	CTIBench (CWE mapping 2021)	1,000	Acc.	50.00	51.90	+1.90
Never-trained	CTIBench (CWE mapping recent)	1,000	Acc.	50.70	52.40	+1.70
Never-trained	CTIBench (CVSS vector)	1,000	Exact vector	22.40	33.30	+10.90
Law						
Train-family	Legal panel (micro agg.)	1,750	Acc.	36.80	56.46	+19.66
Train-family subgroup	Legal panel (ContractNLI)	250	Acc.	47.6	58.4	+10.8
Train-family subgroup	Legal panel (CaseHOLD)	250	Acc.	59.6	64.4	+4.8
Train-family subgroup	Legal panel (ECtHR-A)	250	Acc.	5.2	52.4	+47.2
Train-family subgroup	Legal panel (ECtHR-B)	250	Acc.	2.0	39.6	+37.6
Train-family subgroup	Legal panel (LEDGAR)	250	Acc.	62.4	68.4	+6.0
Train-family subgroup	Legal panel (SCOTUS)	250	Acc.	8.8	24.4	+15.6
Train-family subgroup	Legal panel (UNFAIR-ToS)	250	Acc.	72.0	87.6	+15.6
Never-trained	MMLU Law (Aggregate)	1,763	Acc.	54.623	55.530	+0.907
Never-trained	LegalBench official (146 tasks)	85,154	Task macro	62.118	62.799	+0.681

Table 6: Complete standalone results (continued).

Role	Testbench	N	Metric	Base	Patch	Δ
Chemistry Editing						
Train-family	MolLangBench (recognition)	100	Acc.	48.0	50.0	+2.0
Related task	MolLangBench (molecular edit)	200	Acc.	7.0	10.5	+3.5
Train-family	TOMG-Bench (add group)	100	Exec. edit	26.0	35.0	+9.0
Train-family	TOMG-Bench (delete group)	100	Exec. edit	22.0	26.0	+4.0
Train-family	TOMG-Bench (substitute group)	100	Exec. edit	19.0	70.0	+51.0
Never-trained	ChEBI (SMILES-to-caption)	100	Token F1	0.1734	0.2025	+0.0291
Broad Coding						
Train-family	APPS	200	Strict test pass	11.00	22.50	+11.50
Never-trained	HumanEval (base)	164	pass@1	84.76	87.80	+3.05
Never-trained	HumanEval+ (plus)	164	pass@1	76.83	81.10	+4.27
Never-trained	MBPP (base)	200	pass@1	80.00	81.00	+1.00
Never-trained	MBPP+ (plus)	200	pass@1	68.00	69.00	+1.00
Never-trained	DS-1000 (supported libs)	172	Exec. pass	23.84	26.74	+2.91
Never-trained	DSCodeBench (frozen pilot)	199	Exec. pass	22.11	26.13	+4.02
Data analyst						
Train-family	DataFrame Gym V4 (26 families)	1,195	Hidden exec.	25.69	65.94	+40.25
Retention	DataFrame Gym V3 (8-family retention)	362	Hidden exec.	66.02	96.41	+30.39
Never-trained	HumanEval (base)	164	pass@1	84.76	85.98	+1.22
Never-trained	HumanEval+ (plus)	164	pass@1	76.83	79.27	+2.44
Never-trained	MBPP (base)	200	pass@1	80.00	81.50	+1.50
Never-trained	MBPP+ (plus)	200	pass@1	68.00	70.00	+2.00
Never-trained	DS-1000 (supported libs)	172	Exec. pass	23.84	27.33	+3.49
Never-trained subgroup	DS-1000 (Pandas)	29	Exec. pass	10.34	13.79	+3.45
Never-trained	DSCodeBench	200	Exec. pass	22.00	25.00	+3.00

D SINGLE-DOMAIN AGENTIC EVALUATION: TESTBENCH ADAPTATION AND EXECUTION

The eight rows of Table 3 are eight domain-specific testbenches adapted to a single-domain agent interface. Each task targets one domain-specialist stage and does not require a multi-domain chain of specialist artifacts; it may still involve multiple model forwards and runtime operations. The wrapper adds capability selection, a patch-backed specialist call, domain-specific processing of its output, and a final submission by the parent. It does not supply the correct specialist label or the gold answer as part of the task. Task provenance and agentic wrapping are separate: for example, DataFrame Gym is our generated synthetic execution suite, to which the same selection–execution–submission interface is applied. The law row here uses ECtHR-A; the seven-task legal aggregate in Table 6 belongs to the separate standalone panel.

Single-domain invocation sequence (Table 3).

1. **Present the task.** Package the question and its public inputs (such as a table, schema, abstract, or function specification) with the shared eight-capability catalog. Reference answers and hidden tests stay with the evaluator, outside the generation prompt.
2. **Select and invoke.** The parent selects a capability. A valid call activates the selected adapter for specialist generation. Concat provides the accumulated agent transcript; Ours provides a separate, task-specific request with its output contract; Query inserts a separate Base writer to compose the request (Appendix G). A wrong selection is not silently replaced by the intended specialist.
3. **Process the artifact.** The domain harness parses the output and, where applicable, executes SQL or code, checks a molecular edit, or computes a result from a structured prediction. A well-formed output and a correct answer are distinct checks.
4. **Submit and score.** The specialist adapter is inactive for the parent's final submission. The evaluator grades that submission according to the domain-specific rule below, including agreement with the executed artifact where required.

The Base, Pure, and textual Skill columns instead use direct generation, without this routing and finalization loop. They retain the applicable domain scorer.

D.1 Domain-Specific Inputs and Scoring

Table 7 makes explicit which artifact is generated, what the runtime can check, and what the final evaluator scores in each domain.

Table 7: How the eight single-domain testbenches enter the agent interface. Inputs are model-visible task information; success refers to the final agentic submission scored by the domain harness.

Domain / task	Input and specialist artifact	Processing and success criterion
SQL / WikiSQL	Question and database schema → SQL query.	Execute against the linked database; require a valid final numeric answer consistent with the execution result and reference answer.
Finance / TAT-QA	Financial table, text, and question → calculation program.	Execute the program; compare its numeric result with the reference and require the final answer to reproduce it under the task's scale convention.
Medicine / PubMedQA	Question and abstract → judgment JSON.	Parse the final `yes/no/maybe` submission and compare with the gold label, with specialist evidence and justification logged for diagnostic tracking.
Cyber / CTIBench	Vulnerability description → CVSS v3.1 vector.	Parse the vector and compute score/severity deterministically; require reference-vector correctness and a final submission faithful to the runtime assessment.
Chemistry / TOMG	Starting SMILES and requested substitution → candidate SMILES.	RDKit checks validity and the task's edit/similarity constraints; require a valid final submission preserving the accepted candidate.
Broad coding / HumanEval+	Function specification → Python solution.	EvalPlus executes the candidate against base and extended tests; success requires both test sets to pass and submission of the same candidate ID.
Law / ECtHR-A	Case facts → violated-article labels.	Validate and canonicalize labels, then issue an assessment ID; require exact gold-label agreement and faithful final submission under the benchmark's legal criteria.
Data analyst / DataFrame Gym V4	Tabular task specification → Python function.	Execute against hidden DataFrame cases; require the candidate to pass and the final submitted candidate ID to match.

The WikiSQL subset. The SQL row of Table 3 reports $N = 1{,}726$ rather than the 8,421 development questions of the standalone panel. Starting from the full development split, we execute each question's gold SQL and keep the question only when the result is exactly one row and one column holding a finite number; strings, multi-row or multi-column results, NaN, and infinities are excluded. Exactly 1,726 questions satisfy this, and they form the agentic SQL testbench. The rule is deterministic, depends only on the gold annotation, and was applied before any model was run. It exists because the agentic contract requires the parent to submit a single checkable value; the other seven testbenches already yield one such value per case and need no equivalent filter.

E MULTI-DOMAIN EVALUATION: EXECUTION PROCEDURE AND FIVE EXAMPLES

Multi-domain execution procedure (Table 4). Unlike the single-domain panel, these constructed workflows require outputs from one specialist stage to support later stages. The shared procedure is:

1. **Present the workflow task.** Give the parent the task, public inputs, and eight-capability catalog, without the latent dependency graph or an intended specialist sequence.
2. **Select and execute a specialist.** The parent chooses the next capability. Base uses base weights, while Skill adds textual instructions. Ours, Query, and Concat use the selected adapter with a typed request, a request drafted by a separate Base writer, or the accumulated parent history, respectively, together with the applicable public output contracts.
3. **Return and reuse artifacts.** The runtime processes the result and returns artifacts and observations to the parent. Accepted outputs become available to subsequent requests; the parent selects the next capability and may revisit one already called.
4. **Submit and verify.** The parent submits the final answer. Success requires the workflow's verifier checks, including the required consistency with supporting artifacts, rather than merely a completed sequence of calls.

These examples show how artifacts from one specialist become inputs to another in the five workflow families of Table 4. For each family, we select the lowest-index successful case from archived patch-backed runs and summarize its recorded actions and runtime outputs. The routes below are observed sequences, not solution paths supplied to the agent. Questions are abridged, and displayed decimal values are rounded; full-precision values remain in the records.

E.1 FINANCE: REQUEST FIELDS, RETRIEVE ROWS, CALCULATE

Case: `open-finance-stability30:0001`. **Task:** Use Aster Manufacturing's linked financial database to determine its 2023 post-restructuring operating margin and the percentage-point change from 2022.

Observed route: Finance $\rightarrow$ SQL $\rightarrow$ Finance $\rightarrow$ submit.

1. The finance specialist returns a `financial_data_request` for 2022 and 2023 revenue, operating profit before special items, and restructuring charges.
2. The SQL specialist maps those requested fields onto the database's `income_statement` columns. Execution returns $(2022, 1250, 210, 22.5)$ and $(2023, 1310, 238, 18)$ in that field order.
3. The finance specialist emits executable programs computing $(238 - 18)/1310 \times 100$ and its difference from $(210 - 22.5)/1250 \times 100$. Runtime returns a 2023 margin of 16.793893% and a change of $+1.793893$ percentage points.
4. The agent submits those two runtime values. The final submission is faithful, correct, and grounded.

The intermediate data request connects financial interpretation to schema selection; the final calculation uses the values returned by the database query.

E.2 MEDICINE: RETRIEVE EVIDENCE BEFORE GIVING A JUDGMENT

Case: `medicine-evidence:0000`. **Task:** Retrieve the study for request key `MED-EVID-0000`, answer whether mitochondria play a role in lace-plant leaf remodelling during programmed cell death, and cite the retrieved study identifier.

Observed route: SQL $\rightarrow$ Medicine $\rightarrow$ invalid submission action $\rightarrow$ submit.

1. SQL requests `study_id, abstract` from `studies`, filtered by the request key. Runtime returns study `21645374` and its abstract; the source sample is PubMedQA `labeled:21645374`.

2. The medicine specialist produces a structured assessment with `answer=yes` and `evidence_study_ids=[21645374]`. The assessment is accepted by the workflow's runtime checks.
3. The next agent action fails the single-tool-block parser. A subsequent `submit_biomedical_answer` supplies the same answer and study identifier and is recorded as grounded success.

Unlike the single-domain PubMedQA row, this constructed workflow requires the abstract to be retrieved through SQL and the final answer to cite the retrieved record.

E.3 CYBERSECURITY: RETRIEVE A DESCRIPTION, PREDICT A VECTOR, EXECUTE SCORING

Case: `cyber-sql-cvss-open:0002`. **Task:** Retrieve assessment `ctibench-cti_vsp-00020`, determine its CVSS v3.1 vector, and submit the runtime-computed score and severity.

Observed route: SQL → Cybersecurity → submit; CVSS computation occurs in Runtime after the vector is generated.

1. SQL retrieves `assessment_id, cve_prompt` from `vulnerability_assessments` for the requested identifier. The returned description concerns unsanitized command execution in the `network` package before version 0.7.0.
2. The cybersecurity specialist produces `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`.
3. Runtime parses that vector and returns score `9.8`, severity `Critical`, and a content-bound assessment ID.
4. The agent submits the vector, score, severity, and the same assessment ID. The final answer is faithful and grounded.

The specialist predicts the vector; the numerical CVSS calculation is a deterministic runtime operation, not a third learned specialist.

E.4 LAW: SEARCH FOR PRECEDENTS, THEN ASSESS THE RETRIEVED EVIDENCE

Case: `law-precedent-grounded:0000`. **Task:** In a constructed ECtHR-style application, an applicant is held for twelve days before appearing before a judge. Identify violated articles and support the answer with identifiers from the linked precedent database.

Observed route: Law → SQL → Law → submit.

1. The law specialist emits a `precedent_search_request` with Article 5 candidates and search terms concerning detention and judicial review.
2. SQL retrieves Article 5 rows, ordered by `violation_found`. Runtime returns `SUP-0000-1`, a supporting row, and `CON-0000-1`, a contrasting no-violation row. These identifiers belong to the constructed database, not an official case citation system.
3. The law specialist returns a `legal_assessment` identifying article `5` and citing only `SUP-0000-1`; Runtime accepts the structured assessment.
4. The agent submits the same article and supporting identifier. The final submission is faithful, correct, and grounded.

The two law calls have different roles: the first formulates a retrieval request, while the second assesses the evidence actually returned by SQL.

E.5 CHEMISTRY: PROPOSE, RETRIEVE, AGGREGATE, AND CHECK

Case: `chemistry-sql-data:0000`. **Task:** For series `MMP-0000`, replace chlorine in `Clc1ccccc1` with hydroxyl while preserving the aromatic core and reducing RDKit Crippen logP by at least 0.3. Report a candidate, supporting pair IDs, the mean change and a two-sided 95% confidence interval, and the candidate's executed property change.

Observed route: Chemistry → SQL → Data analyst → Chemistry → invalid submission action → submit.

1. Chemistry proposes `Oc1ccccc1`, which is parsed as a valid molecule candidate.
2. SQL selects `pair_id, delta_logp` from `mmp_evidence` for series `MMP-0000` and that candidate. Runtime returns twelve supporting rows, `MMP-0000-P01` through `MMP-0000-P12`.
3. The data specialist supplies `solve(df)` to compute the mean, sample standard deviation, and interval using $\bar{x} \pm 1.96s/\sqrt{n}$. Execution returns mean -0.9478 and interval $[-0.976244, -0.919356]$. This is the normal-approximation interval implemented by the recorded program.
4. Chemistry confirms the candidate. RDKit returns original logP 2.34, candidate logP 1.3922, and change -0.9478, satisfying the requested reduction.
5. One agent action fails the single-tool-block parser. The next action submits the candidate, all twelve pair IDs, the executed interval and mean, and the RDKit change. The final submission is faithful, correct, and grounded.

The trace separates three objects that could otherwise be conflated: the proposed molecular structure, statistics over retrieved evidence rows, and the property change computed for the actual candidate. The final submission binds them together without merging specialist parameters.

F DOMAIN-SPECIFIC TRAINING RECIPES

This appendix describes the eight patches used in the evaluated registry. All adapt Qwen3-8B with a frozen backbone. The recipes differ in source data, supervision, filtering, and replay: the framework does not require a single distillation procedure. Table 8 summarizes the frozen training sets, and Table 9 provides the corresponding training hyperparameters; the following descriptions distinguish generated targets from reference labels and execution checks from weaker format or syntax checks.

Table 8: Training data for the eight evaluated patches. Counts refer to the frozen dataset of the listed training stage, not to the number of evaluation cases or all examples seen during earlier stages. The finance patch continues an earlier adapter.

Patch	Rows	Sources and supervision
SQL	11,662	BIRD, Spider, WikiSQL, SParC, and CoSQL train splits; verified teacher SQL, gold SQL, and Base-correct retention examples.
Finance	15,450	ConvFinQA, FinQA, MultiHiertt, and TAT-QA train-family replay, plus generated financial-formula and formula-chain tasks.
Medicine	78,176	MedMCQA, MedQA, PubMedQA, and EHRSQL training splits; the frozen medical hard-example, answer-only pilot.
Cybersecurity	11,432	CyberBench source tasks; hard examples serialized as task-native answers, without teacher reasoning.
Chemistry	9,296	TOMG-Bench OpenMolIns editing and MolLangBench recognition train splits; 7,902 teacher-correct/Base-wrong examples and 1,394 Base-correct replay anchors.
Broad coding	3,605	APPS, Magicoder, CodeFeedback, and Bash-command tasks; 2,407 direct teacher targets, 298 response-level repairs, and 900 replay examples.
Law	20,806	ContractNLI and LexGLUE training tasks, C-Eval/CMMLU legal anchors, short teacher rationales, and task-native replay.
Data analyst	3,462	Synthetic DataFrame Gym tasks; 778 V3 examples, 1,507 V4 teacher-correct/Base-wrong examples, and 1,177 Base-correct replay examples.

SQL: multisource SQL supervision. The SQL recipe mixes BIRD verified targets, Spider teacher repairs, Spider Base-correct retention examples, and gold SQL from WikiSQL, SParC, and CoSQL. Before deduplication, construction samples 3,000 BIRD verified examples, 2,500 WikiSQL examples, 1,000 examples each from SParC and CoSQL, and 3,000 Spider Base-correct retention examples, and includes all available Spider teacher repairs. Deduplication by system–instruction pair yields 11,662 rows. When a Spider repair and a retention example share a prompt, the verified teacher repair takes priority. Targets are SQL statements rather than explanations. Thus the data combine model-produced solutions and released reference queries; Base-correct selection does not imply that the retained target is always the Base output. Only the source families' training splits supply supervision; their held-out splits and KaggleDBQA are evaluated separately. The selected SQL adapter is trained directly on the base backbone.

Finance: formula-chain training with replay. The initial finance patch uses audited teacher solutions from the four financial training families in Table 8. Each target is a compact JSON object with `program` and `answer` fields. The selected patch continues this adapter using replay from its training set and additional generated formula and formula-chain examples, for 15,450 rows in its frozen training stage. The formula candidates undergo deterministic contract checks and review by the Qwen3-235B-A22B-Instruct-2507-FP8 teacher. Data construction excludes duplicate instructions. The initial and continuation stages each use one epoch, with learning rates 10^{-4} and 10^{-5}, respectively.

Table 9: Training configurations for the selected stages. All stages run for one epoch. Across all eight training configurations, LoRA targets all linear layers, dropout is 0.05, computation is BF16, the scheduler is cosine, and prompt tokens are excluded from the loss. Batch is per device; Acc. is gradient accumulation, not global batch size. Finance lists the continuation stage.

Patch	r / α	Learning rate	Cutoff	Batch	Acc.	Warmup	Seed
SQL	64 / 128	10^{-4}	20,480	1	4	0.05	20260810
Finance	64 / 128	10^{-5}	16,384	1	4	0.05	20260810
Medicine	64 / 128	5×10^{-5}	20,480	1	4	0.05	20260811
Cybersecurity	64 / 128	5×10^{-5}	4,096	1	4	0.05	20260811
Chemistry	64 / 128	10^{-5}	1,024	2	4	0.03	20260813
Broad coding	64 / 128	5×10^{-5}	8,192	1	4	0.05	20260815
Law	32 / 64	5×10^{-6}	8,192	1	4	0.05	20260815
Data analyst	64 / 128	3×10^{-5}	8,192	1	4	0.05	20260822

Medicine: a multisource hard-example pilot. The selected medical patch uses 78,176 examples from the training splits of MedMCQA, MedQA, PubMedQA, and EHRSQL. The medical pilot uses answer-only supervision. These sources span short medical answers and healthcare SQL; their evaluation uses exact answers and SQL execution equivalence, respectively. Training runs for one epoch with 2,443 optimizer steps. MMLU-Medical supplies no SFT rows; its frozen 945-task evaluation set has zero task-ID and exact normalized prompt overlap with the training set.

Cybersecurity: task-native answer supervision. The patch trains on 11,432 hard CyberBench examples with task-native answer-only targets. CTIBench is excluded from training. The CyberBench evaluation is a same-source, untrained-task-ID holdout, whereas CTIBench tests transfer across benchmark sources.

Chemistry: executable molecular-edit distillation. The molecular-editing patch uses TOMG-Bench OpenMolIns add/delete/substitute tasks and MolLangBench recognition training tasks. Qwen3-235B-A22B-Instruct-2507-FP8 provides two candidates per task. Selection retains 7,902 teacher-correct examples that the Base misses and 1,394 Base-correct anchors. Editing targets are final SMILES strings checked with RDKit, including the requested functional-group change and a Tanimoto similarity threshold of 0.35; recognition targets use exact integer checks. There is no long reasoning target. The frozen set has no duplicate task IDs or prompt–output pairs, and no formatted example exceeds its 1,024-token cutoff.

Broad coding: direct targets, response repair, and replay. The coding patch uses four tracks: general Python, scientific Python, Python debugging, and Bash. Candidate sources are APPS training problems, Magicoder Python and shell tasks, CodeFeedback debugging tasks, and a Bash-command corpus. Qwen3-235B-A22B-Instruct-2507-FP8 generates direct final-code targets and repairs of Base attempts, supplemented by Base-correct APPS replay examples. Here repair means conditioning the teacher on the original task and Base attempt, then training on its final code; it is not a token-distribution distillation loss. General-Python teacher targets and replay pass execution checks. Bash receives syntax and denylist checks; debugging and scientific Python receive static syntax checks. Section F.1 traces this mixture through model selection and deployment as a callable coding capability.

Law: balanced supervision and anti-forgetting replay. The selected law patch is a fresh rank-32 adapter trained on 20,806 rows. Its data inherit the 17,753-row mixture of an earlier law patch: balanced hard examples from ContractNLI and LexGLUE, Base-correct anchors, LEDGAR-derived binary contract tasks, and 916 Base-correct C-Eval/CMMLU legal anchors. The selected mixture adds short teacher rationales and task-native replay. UNFAIR-ToS replay balances positive gold examples with previously correct negative examples; other replay uses unused training tasks answered correctly by both Base and the earlier law patch. The builder excludes task-ID and exact prompt overlap with the declared evaluation sets and checks the token cutoff. LegalBench supplies no training rows.

Data analyst: synthetic execution tasks with retention. DataFrame Gym V4 constructs tasks across 26 Pandas, NumPy, SciPy, and scikit-learn operation families. Reference functions must pass four hidden execution cases; normalized prompts separate training candidates from the 1,195-task holdout. Qwen3-235B-A22B-Instruct-2507-FP8 supplies final Python `solve` functions, without long reasoning. The selected training set combines 1,507 teacher-correct/Base-wrong V4 examples, 1,177 Base-correct V4 replay examples, and 778 frozen V3 examples. It is trained directly on the base model. The V3 holdout serves as a retention panel. This synthetic training source is separate from the external coding benchmarks used to evaluate transfer.

F.1 Broad Coding Case Study: From Base Experience to a Deployed Capability

Experience and training. The coding patch connects Base-generated attempts to an independently trained patch and subsequent agent use. Its 3,605 unique training tasks comprise 900 Base-correct APPS code outputs retained after strict execution checks, 298 teacher repairs conditioned on the task and Base attempt, and 2,407 direct teacher targets. The repair fraction is 25% of the 1,193 scientific Python tasks. Verification follows the track-specific checks above. A new rank-64 LoRA is trained on Qwen3-8B for one epoch at learning rate 5×10^{-5}; the longest formatted training example is 4,224 tokens, within the 8,192-token cutoff.

Historical selection. The repair-fraction comparison covers 0%, 25%, 50%, 75%, and 100% of scientific Python targets. The 25% candidate is selected as the balanced Python-code expert: it improves all five primary metrics over Base (HumanEval, HumanEval+, MBPP, MBPP+, and supported-library DS-1000). The 0% candidate scores higher on HumanEval+ (137/164 versus 133/164), but lower on DS-1000 (38/172 versus 46/172); larger repair fractions fall below Base on MBPP+. These panels participate in version selection. Table 6 reports the matched direct-generation results against Base, including the positive gain on the 199-task DSCodeBench pilot, which uses three test cases per task.

APPS test evaluation. After version selection, the deployed patch is evaluated on the stratified 200-problem APPS test subset described in Appendix B. Strict execution success increases from 22/200 (11.0%) for Base to 45/200 (22.5%) for the patch. Paired outcomes show 26 problems changing from incorrect to correct and three from correct to incorrect, a net gain of 23. Table 6 reports this additional direct-generation result.

Agent deployment. The selected coding patch is used by the coding specialist in Table 3. Under the typed-handoff protocol described in Appendix D, the agent achieves 133/164 strict successes: both EvalPlus test suites pass and the parent submits the exact candidate identifier. The parent autonomously chooses whether to invoke the specialist or attempt the task directly; the run includes 153 specialist calls and 11 parent direct attempts. This reaches the same aggregate score as direct patch generation, documenting successful integration of the learned capability. The matched gains over Base in Table 6 concern direct generation; the agent result measures performance after selection, execution, and final submission.

G MODEL-WRITTEN QUERY: PROTOCOL AND DIAGNOSTICS

Select first, then write. Query retains the autonomous specialist selection of Ours. After a selection, a separate Base-model writer, with no tools and a 2,048-token output budget, composes the request sent verbatim to the selected specialist. The public output contract and the configured schema, parsing, evidence, and label requirements are supplied separately; these may include task constraints, not just formatting. No omitted dynamic evidence is automatically restored, and queries are neither corrected nor resampled. This is a variant of the runtime used by Ours, not an evaluation of the original Adaptive Minds implementation.

Single-domain selection uses the same eight-tool catalog and 96-token routing budget as Ours; multi-domain selection retains the autonomous eight-step policy. The inherited system prompts, model weights, specialist budgets, runtime, and scorers are unchanged. The routing procedure is retained, not the realized decisions from an earlier run: routes are generated anew. In multi-domain finance, the writer receives a multi-message input constructed from the workflow state, together with the parent system prompt.

Invocation coverage and correctness. Table 10 reports coverage over every test case, including direct attempts and incomplete trajectories. Single-domain final correctness is 3,371/6,153 (54.79% pooled); the main table instead reports the equally weighted eight-domain macro average of 59.16%. HumanEval+ requires both base and plus tests to pass with the submitted candidate ID matching; DataFrame candidates are execution-scored. Multi-domain final correctness and grounded success both equal 211/500, while remaining distinct recorded events.

Table 10: Query invocation coverage and correctness. Counts are cases, not calls. Any means at least one specialist was invoked. Required means the corresponding domain specialist for single-domain tasks and all necessary specialists for multi-domain workflows. Candidate correctness is recorded before final submission for single-domain tasks; a dash means it is not reported here.

Domain / workflow	N	Any	Required	Candidate	Final
Single-domain					
SQL	1,726	1,725	1,725	735	734
Finance	718	710	649	495	477
Medicine	1,000	997	986	755	755
Cybersecurity	1,000	1,000	1,000	309	309
Chemistry	100	100	100	71	71
Broad coding	164	154	154	131	131
Law	250	250	250	102	102
Data analyst	1,195	1,185	1,085	795	792
Multi-domain					
Finance–SQL–Finance	100	100	100	—	0
Chemistry–SQL–Data analyst–Chemistry	100	100	53	—	43
SQL–Medicine	100	100	100	—	49
SQL–Cybersecurity–CVSS	100	100	100	—	32
Law–SQL–Law	100	100	100	—	87

G.1 Cost, Failure Traces, and Comparison Conditions

Additional writer cost. The writer adds 6,121 single-domain and 1,598 multi-domain model requests. Its single-domain input/output totals are 9,217,919/1,523,308 tokens; multi-domain totals are 4,292,735/498,570 tokens. Overall request counts, including the parent and specialists, are 24,829 and 5,407, respectively. Query is therefore not a compute-matched comparison with a runtime that has no writer call.

Cross-step definition loss in finance. All 100 finance cases invoke Finance and SQL, execute retrieval and calculation, and submit an answer faithful to the runtime; none is correct. In case 0, the first specialist defines adjusted operating margin as operating profit before special items minus restructuring charges, divided by revenue. The retrieved 2023 values are 170, 14, and 920, respectively. The next query carries these values but refers only to the earlier definition without restating it. The specialist computes $170/920 \times 100 = 18.47826\%$ rather than $(170 - 14)/920 \times 100 = 16.95652\%$. The program executes and the parent faithfully submits its result. This trace illustrates how an omitted cross-step definition directly propagates into downstream calculation error.

Other execution and budget outcomes. Chemistry reaches all required specialists and final submission on 53 cases, of which 43 succeed; 47 cases exhaust the eight-step budget. Each of the other four workflows reaches all required specialists and submission on all 100 cases. All 5,407 multi-domain requests succeed without empty output, generation length termination, or context-limit failure. While request-level formatting succeeds, traces reveal that task failures stem from domain-level logic and evidence handling. Single-domain execution records 94 writer length terminations, 20 routing length terminations, and three at other candidate stages, with no empty recorded output.

Execution conditions and verification. For the Query arm in Tables 3 and 4, execution fidelity and scoring consistency were verified across 6,653 terminal records: 6,153 single-domain tasks and 500 multi-domain tasks, confirming consistent request bindings and deterministic scoring reproduction across all 500 multi-domain trajectories. Concat inputs and output contracts follow the shared execution procedure specified in Appendix B.

G.2 Paired Interventions on Specialist Inputs

We test whether preserving specific task information changes specialist correctness at a fixed handoff. The diagnostic uses 30 archived Query handoffs—ten each for single-domain SQL, multi-domain finance calculation, and multi-domain law assessment—and generates four specialist outputs per handoff. The metric is correctness of the specialist artifact under the original runtime and scorer: SQL execution equivalence, both financial values, or legal labels with valid retrieved evidence. Parent finalization is outside this diagnostic; Tables 3 and 4 report end-to-end performance separately.

Selection and controlled execution. SQL uses the first ten tasks in the frozen task order. Finance and law use the first ten tasks, in original index order, that reached the target stage, taking the first eligible handoff per task. Law index 1 did not reach that stage and is skipped under this rule. Selection precedes the new generations. All four arms replay the same historical Query upstream state and use the same expert weights, system prompt, GPU, output budget, runtime, and scorer. The original writer output is fixed. Expert outputs are generated anew, with arm order rotated across cases, thinking disabled, fresh prefill, and no cross-call KV reuse. Output budgets are 512 tokens for SQL and finance and 640 for law. Finance retains its original multi-message input construction and parent system prompt in all four arms.

Four input arms. *Query* uses the archived request; *Query+* restores target fields missing from it; *Typed–* removes those same fields from the complete typed request; *Typed* retains them. Restored text comes from public task inputs or prior runtime state. SQL intervenes only on the database schema, leaving the question in Typed unchanged. Finance intervenes on the prior `finance_request.calculation` field, removing all copies in Typed– while preserving retrieved values and the output contract. Law restores missing `precedent_id`, `holding_summary`, and `facts_summary` fields from retrieved rows, or blanks the corresponding cells in every Typed copy. Public row indices, articles, and violation flags link the restored fields to their rows; these anchors can repeat existing Query content. Missing text is detected after whitespace normalization; paraphrases may retain meaning, and the law intervention tests the fields jointly. Five SQL and five law cases already contain the target fields and serve as no-intervention controls.

Table 11: Specialist-stage correctness at fixed handoffs (correct/total). Query+ restores selected public fields; Typed– removes the same fields. The lower panel restricts the upper panel to cases with actual input changes. The ten unchanged controls are correct in every arm.

Handoff	Query	Query+	Typed–	Typed
All selected handoffs				
SQL	7/10	8/10	5/10	10/10
Finance calculation	0/10	5/10	0/10	4/10
Law assessment	8/10	10/10	8/10	10/10
Cases with actual interventions				
SQL	2/5	3/5	0/5	5/5
Finance calculation	0/10	5/10	0/10	4/10
Law assessment	3/5	5/5	3/5	5/5

Paired changes and execution checks. Table 11 reports both the full sample and the subset with input changes. Restoration changes incorrect to correct on one SQL, five finance, and two law cases; removal changes correct to incorrect on five, four, and two, respectively. Seven of the twenty intervened cases exhibit both changes: one SQL, four finance, and two law cases. No restoration changes a correct answer to an incorrect one. All 30 regenerated Query outputs match their historical outputs verbatim, as do all 20 within-input repeated-output pairs in the ten no-intervention controls. The 120 requests have no empty outputs or generation-length terminations. One Query+ finance program fails execution and remains an incorrect result.

Information recovery and remaining errors. In the five intervened SQL cases, removing the schema lowers Typed from 5/5 to 0/5, while restoring it raises Query from 2/5 to 3/5. Schema recovery alone does not repair every change to the question: in `wikisql-dev-000038`, restoration replaces a nonexistent column with a valid one, but the resulting query still fails execution equivalence. Some original queries contain writer-generated SQL or legal answers, so they also carry answer choices made before specialist execution.

In finance, a post-hoc component check using the frozen retrieved values shows that Query+ improves the 2023 adjusted-margin calculation from 3/10 to 9/10, while complete correctness reaches 5/10. Typed obtains the margin on 10/10 but both requested values on 4/10. In case 0, restoring the definition corrects the margin from 18.47826% to 16.95652%, yet the annual-change program remains incorrect. Thus retaining the definition improves the target calculation, while composing the two yearly results remains a separate source of error. Query+ also exceeds Typed by one case in this diagnostic.

Law obtains 8/10 with Query and 10/10 with Typed; restoring the missing fields recovers two Query cases. In `law_0000`, Query cites an absent precedent identifier, Query+ cites the retrieved supporting record, and Typed– fails with an empty citation. The end-to-end Query advantage over Ours on law in Table 4 (87/100 versus 80/100) is not reproduced at these fixed states. This diagnostic supports the role of specific task information in specialist correctness; its selected upstream states and stage-level metric do not quantify that mechanism's contribution to the full-panel end-to-end differences.