ACTION-SPACE SHAPING FOR LLM AGENTS: MEASURING AND MITIGATING TOOL-SCHEMA BIAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have shown strong performance on tool-use agentic tasks when given a fixed tool schema. Yet a tool schema is not the action space of an agent; it is merely one interface representation of it. The same executable action can be exposed through many different, functionally equivalent tool definitions, and an agent that has truly learned a task should behave consistently across them. We show that current agents often do not, a phenomenon we term *schema bias*. To study this systematically, we introduce an executable transformation framework that rewrites a native tool schema using nine operators, including merging and splitting tools, altering how a single tool is expressed, and distributing one action across several dependent calls. The tasks, executable actions, and reachable states remain fixed, so any change in success is attributable to the interface alone. Evaluating eleven LLMs, including two closed models, on up to 32 schema variants, we ask how large schema bias is, how it manifests, whether the difficulty of a schema variant can be predicted without a full evaluation, and whether training removes it. We find that schema bias is substantial even for the newest models: success rates range from complete failure to 97% depending solely on the schema. To reliably estimate schema difficulty, it requires running a small sample of the target queries. Training repairs a schema variant only when that variant appears in the training data.

1 INTRODUCTION

Large language model (LLM) agents interact with external environments through tool calls. The interface presented to an agent is typically specified by a *tool schema*: a declaration of callable functions, their names and descriptions, argument structures and types, class organization. Although this schema is often treated as part of the environment, it is more accurately as one representation of the environment's action space. For example, as shown in Figure 1, one native tool *modify_order* can be expressed via many different but equivalent approaches. These alternatives differ in names, factorization, and interaction length, yet they admit a mechanical translation to the same native actions and reach the same final states. They are therefore different representations of one action space, not different tasks. An agent that has learned the task should behave consistently across them. Figure 1 also illustrates four such representations, the model-dependent gaps they produce, and how training moves them.

This non-uniqueness matters for research and practice alike. Benchmarks evaluate agents under a single schema chosen by their designers, implicitly treating one representation as canonical; reinforcement-learning environments commit to one factorization of the action space; and deployed systems merge, split, or reorganize their tools mostly for engineering reasons. If agent behavior depends on these choices, benchmark scores and training gains partly reflect an arbitrary interface rather than task difficulty or transferable capability.

This points to a distinct dimension of agent generalization. Beyond robustness to prompt wording, observation noise, or distribution shift, a capable agent should be robust to equivalent representations of the same action space. We call systematic performance variation across such representations *schema bias*, and argue that robustness to it should be treated as a property of the agent rather than left to careful benchmark design. We study schema bias through four research questions: (RQ1) how large and systematic is schema bias; (RQ2) how does the representation change the way agents fail;

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

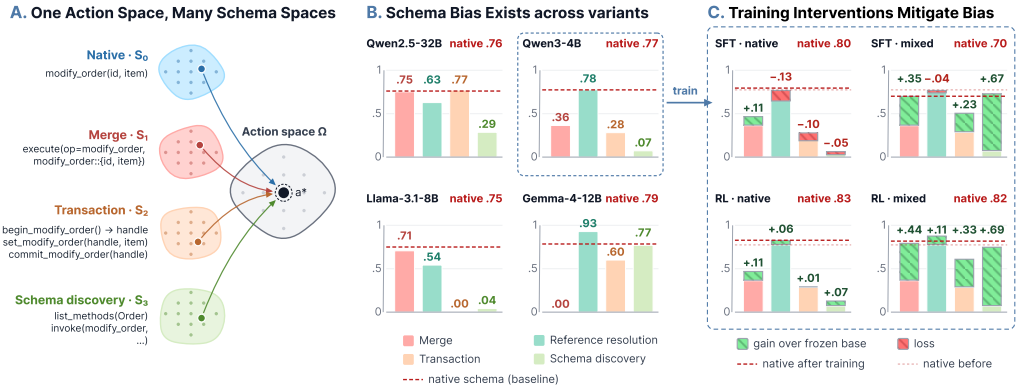


Figure 1: **Schema variants, schema bias, and training mitigation. (A)** All schema variants S_i are just different representations of the same action space Ω: a call in any of them decodes to the same native action a^* and reaches the same final state. Each space is shown with an abbreviated schema for one action. **(B)** Schema bias is large and model-specific. Success rates of four LLMs under four representative variants; the dashed line is each model's native-schema score. The same variant can match native for one model and collapse to zero for another (full grid in Figure 2). **(C)** Qwen3-4B after SFT or RL on native-only or variant-mixture data. Hatched segments are gains (green) or losses (red). Native-only data leaves most of the bias under either method; RL on variant-mixture data mitigates bias while keeping the native score (§8).

(RQ3) can the difficulty of a schema be predicted without a full evaluation; and (RQ4) can training reduce schema bias, and does the reduction transfer?

We observe that current agents exhibit large and structured schema bias. Success ranges from complete failure to near-perfect depending only on the schema, the most difficult representation differs across model families, and neither the newest open models nor two closed models are immune. Failures follow the variant: a given schema tends to break different models in the same way, while how much it costs depends on the model. Reliably ranking schema variants by difficulty requires running a small sample of the target queries. Training repairs a variant only when that variant appears in the training data; on-policy RL does so without the tax that SFT imposes on other variants, and the gains extend to new combinations of trained schema changes but not to new kinds of change. Our contributions follow the progression from measurement to mitigation.

- **Framework:** we introduce an executable framework of nine operators that transform a native tool schema, from merging and splitting tools to multi-call protocols, and verify every variant against native actions and final environment states.
- **Measurement:** we measure schema bias in eleven LLMs across up to 32 schema variants.
- **Diagnosis:** we characterize how agents fail under each variant with an automatic failure taxonomy, and evaluate ways to estimate the difficulty of a schema variant, from probes that need no task queries to small samples of the target queries.
- **Mitigation and transfer:** we compare training-free methods, supervised fine-tuning, and on-policy RL for reducing schema bias, and test how far their gains transfer to held-out domains, held-out schema changes, and real benchmarks.

Together, these results position robustness to the schema representation as a separate dimension of agent generalization: evaluations should report performance over a small set of verified-equivalent schemas, and training claims should be tested beyond the interface on which they were trained.

2 Related Work

Tool-use benchmarks such as τ-bench/τ^2-bench (Yao et al., 2024; Barres et al., 2025), AppWorld (Trivedi et al., 2024), BFCL (Yan et al., 2024; Patil et al., 2023), ToolSandbox (Lu et al., 2024), and ACEBench (Chen et al., 2025) evaluate each environment through one fixed tool schema. These schemas contain consequential representation choices: τ-bench's POMDP action space is one pa-

rameterization of its domain, AppWorld groups 457 APIs into application classes, and the Model Context Protocol (Anthropic, 2024) supports runtime disclosure of tool definitions. Our work treats such choices as variables and evaluates agents across verified-equivalent representations of the same executable action space.

Prior work on interface sensitivity primarily varies how a fixed tool is described. MetaTool (Huang et al., 2024) shows that descriptions may need to be adapted to the downstream LLM, while robustness benchmarks such as RoTBench (Ye et al., 2024) corrupt tool definitions. We study a different source of variation: invertible reparameterizations that preserve the available actions and reachable environment states.

Our framing is related to action-space design in reinforcement learning, where action removal, discretization, factorization, and temporal abstraction affect learning efficiency (Kanervisto et al., 2020; Vinyals et al., 2017; Sutton et al., 1999). We extend this perspective to tool-using language agents and ask whether a trained model behaves consistently across equivalent action representations. This differs from prompt-format sensitivity (Sclar et al., 2023; Mizrahi et al., 2024), which changes surrounding text without changing the factorization of the action space; matched surface controls help us separate these effects.

Function-calling data and training provide candidate interventions for schema bias. APIGen/xLAM (Liu et al., 2024b), ToolACE (Liu et al., 2024a), and Granite function-calling (Abdelaziz et al., 2024) synthesize or granularize training data, but do not evaluate invariance across equivalent schemas. We compare supervised fine-tuning with on-policy GRPO (Shao et al., 2024) and measure both in-distribution repair and transfer to held-out domains. The schema axes also reflect long-standing API-design choices, including fine-grained REST versus coarse RPC interfaces and opaque identifiers versus natural keys (Fielding, 2000); our experiments quantify their effects on agent performance.

3 An Executable Schema Conversion Framework

3.1 Framework components

The conversion framework has four components. First, a *native environment* provides an executable schema S_0 and an execution interface that applies actions to the environment state. Each native invocation is normalized into an operation and its typed arguments; we call this record a *native action*. Second, a library of *transformation operators* rewrites S_0 into a variant schema S_i. Third, an *execution adapter* decodes schema-specific tool calls into native actions. Finally, a *variant validator* checks that S_i executes the same native actions and reaches the same environment states as S_0. Thus, a transformation changes how an agent expresses a tool call, but not the actual executable decisions.

3.2 Schemas, native trajectories, and interaction traces

We distinguish three objects. A *native action trajectory* $a^* = (a_1, \ldots, a_K) \in \Omega^*$ records the semantic actions required by a task, which do not depend on the schema variant. A *schema* S_i specifies how those actions can be expressed. An *interaction trace* τ_i records how an agent actually invokes tools under the given schema variant S_i. Let $\mathcal{T}_i$ be the set of completed traces under S_i that can achieve the same task. The execution decoder maps an observed trace to the native actions that it executes:

$$D_i : \mathcal{T}_i \to \Omega^*, \qquad \tau_i \mapsto \hat{a}_i.$$

The decoder can be many-to-one, which means different agent traces could execute the same native trajectory. For example, a model may set transaction arguments in different orders or recover from a rejected call before committing the same action.

3.3 Transformation operators

A transformation operator $\mathcal{O}$ is a class of mechanism that rewrites the native schema; a concrete rewrite fixes a *method* m, the parameters that instantiate that mechanism. A schema variant is

$$S_i = \mathcal{O}(m, S_0), \tag{1}$$

Table 1: The schema operator space, grouped by what each mechanism changes. Each operator is a class of mechanism; a schema variant fixes its method (Eq. 1).

Family	Operators	Mechanisms
Single-call	*merge* *split*	**Tool-set partitioning**. Re-designs tool boundaries over the native operations: several functions are fused into one tool behind a discriminator, or one tool is split by a condition on its arguments. Representative methods (Figure 2): *fully merged* and *class dispatch* for merge; *fully split* and *interval split* for split.
	nest arguments *rename* *strip descriptions* *reorder arguments*	**Per-tool representation**. Rewrites the surface form of a single tool expression: parameter nesting structure, tool and parameter identifiers, presence of descriptions, and parameter order. Representative methods: *nested args* for nest and *namespaced names* for rename; strip descriptions and reorder arguments have a single method each.
Multi-call	*transaction* *reference resolution* *schema discovery*	**Cross-call protocol**. Spreads one native action over several dependent calls whose state must be carried across turns: *transaction* replaces one native action with multiple intermediate calls. *reference resolution* requires an earlier additional reference resolving call. *schema discovery* requires schema query before tool invoke.

For example, the operator merge could have different aggregation strategies, e.g. by semantic class, by domain, or at random, defined by the *method* m. The main figures report only selected representative variants. We define nine operators, grouped into three categories by what they change (Table 1). Operators in the first two categories change only how the tools are presented, so each native action is still a single tool call; operators in the third category change the structure of the interaction, so a single native action is expanded into several dependent tool calls. Full variant evaluations are shown in Appendix C.3.

Tool-set partitioning changes the grouping or splitting of the native functions. The aggregation operator *merge* moves a tool-call decision from the function name into an argument. Which functions go together is the method m: *fully merged* combines all functions of a domain into one tool, *class dispatch* groups the functions of each class across the whole catalog, and further methods group functions by semantic similarity or at random, merge only part of a domain, or change how the merged tool takes its arguments. The splitting operator *split* breaks a tool apart by the value range of one argument: *fully split* bakes every enumerable argument into the tool name, and *interval split* does the same for a non-enumerable argument by cutting its range into intervals. Figure 2 includes only these two merge methods and two split methods; the remaining methods are described in Appendix C.1 and evaluated in Appendix C.3.

Per-tool representation operators modify a tool in place. We define four: *nest* wraps a tool's arguments into one nested object; *rename* changes the tool identifiers (function names), either by prefixing them with their class name (namespaced names) or by replacing them with opaque identifiers; *strip descriptions* and *reorder arguments* change only the textual presentation. Nest changes the parameter-schema tree, whereas the other three change its labels or ordering; none of them changes the selected native operation or the minimum call path.

Cross-call dependency operators expand the same native action into a sequence of tool calls with sequential dependencies: a later call consumes what an earlier call returned. We define three. *Transaction* replaces one native action with several intermediate calls that open a transaction, write the arguments one call at a time, and commit; such protocols are usually motivated by external requirements such as formality, validation, or auditability. *Reference resolution* removes one argument's value from the tool and adds a resolver: the agent first resolves the user-provided value to a handle and then passes that handle in place of the value. *Schema discovery* requires the agent to first list a class's methods and argument names, and to call only with the names it was given.

3.4 Validating schema variants

We compare each schema variant S_i with the native schema S_0 on the same task. Each task specifies a user query and a target native trajectory a^*. The operator maps a^* to a variant trace τ_i^*. During variant validation, the execution adapter runs trace τ_i^* and reconstructs native actions. S_i passes validation only if the reconstructed actions match a^*. The schema transformation framework is not

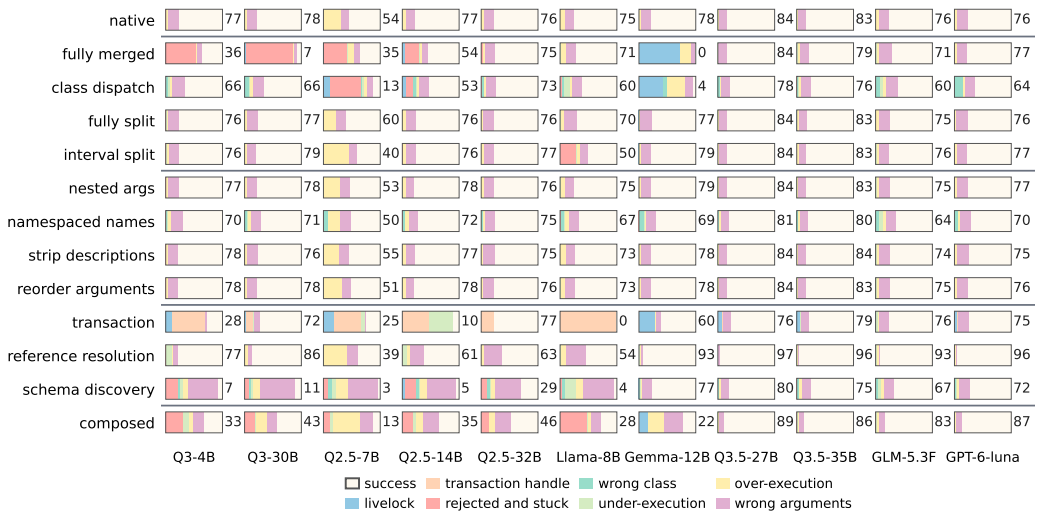


Figure 2: **The model-by-schema performance grid: exact success and failure mode composition.** Each cell is one model (column) on one verified-equivalent schema (row), 2,748 identical tasks. The number is the exact success rate (%), represented as the white portion of composition bar. The coloured segments are the absolute rates of the automatically detected failure modes analysed in §6. Row labels are the representative variant names following Table 1. The two rightmost columns are closed models evaluated through their APIs. The remaining variants are reported in Appendix Figure 4.

tied to a particular benchmark. We also apply the same framework to both our synthetic controlled environment and open-sourced tool-use benchmarks, such as τ^2-bench and BFCL.

4 Experimental Setup

The experiments follow the four research questions: we first measure a model-by-schema performance landscape (RQ1) and analyze its failure modes (RQ2), then test ways to estimate schema difficulty cheaply (RQ3), and finally train models to reduce schema bias and test whether the gains transfer (RQ4). Every comparison within an environment holds the tasks, the native execution interface, and the scorer fixed.

In brief, the controlled synthetic environment has 12 domains, 168 native operations, and 2,748 task queries; τ^2-bench and BFCL serve as real-benchmark validation. Nine open-weight models are evaluated on all 32 variants and two closed models on the thirteen representative variants of Figure 2. An episode succeeds when the native actions it executes match the gold actions exactly, and calls that violate the active schema are rejected and returned to the model as an error it can recover from. Appendix B gives the full setup: the synthetic domains, query statistics, and query construction, the real-benchmark adapters, the models and their settings, and the evaluation protocol.

5 RQ1: Measuring Schema Bias

Our first research question asks how much task success varies across equivalent schema variants, and whether that variation follows reproducible patterns rather than evaluation noise. We derive 32 verified-equivalent variants (the native schema and 31 transformed variants) from one native action space and evaluate all of them on nine open-weight models, and the representative variants also on two closed models. Figure 2 reports the representative variants (the rest are in Appendix Figure 4) together with how the failed episodes are distributed over failure modes, which §6 analyses. With tasks and native semantics fixed, success still ranges from complete failure to near-perfect depending only on the schema, and the same variant can be catastrophic for one model and harmless for another. No single variant wins on every model, and each model family is sensitive to different rep-

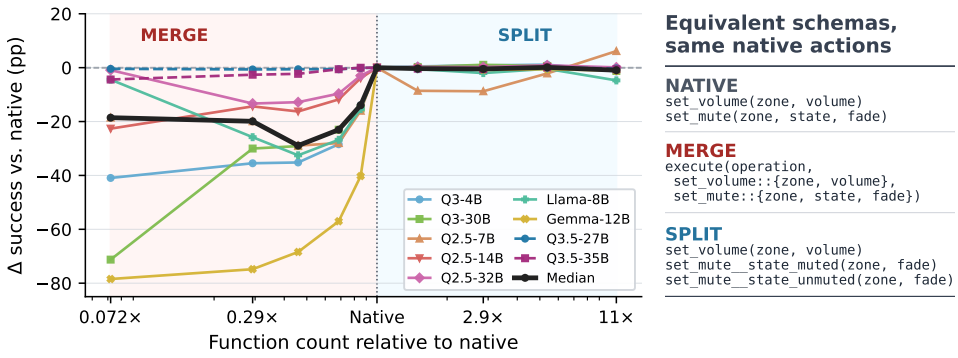


Figure 3: **Merging is generally more harmful than splitting.** Left: colored lines show each model's change from its native-schema success rate. The black line is the median across the nine models. The x-axis represents ratio of function count between variant and native schema. We sample ten variants to span the granularity ladder from full merge to full split. Each ratio is the query-weighted mean across 12 schema domains. Right: a compact and abbreviated schema example shows how merging and enum splitting transform the native schema.

resentations; the newest Qwen3.5 generation is the most robust but not immune. The closed models are not immune either: equivalent schemas move both by 32 points, and their weak spots are the variants that expose a large catalog (class dispatch and namespaced names) rather than the cross-call protocols that break most open models. The rest of this section organises this variation into recurring patterns by operator family.

5.1 Recurring structural patterns

Results across the nine-model by 32-variant grid are model-dependent, but recurring patterns organize much of the variation. We group them by the operator family of Table 1 and, because no pattern holds for every model, report the main trend together with its exceptions. All results use the default evaluation protocol (Appendix B.4).

Tool-set partitioning: *merging is generally more harmful than splitting.* Tool Merging and splitting are two opposite and complementary directions, but they do not show similar challenges to LLMs. Figure 3 compares a representative ten-step granularity ladder across all nine models, spanning $0.07\times$ to $10.6\times$ the native function count. Splitting largely preserves native-schema performance, whereas fully merging tools reduces success by a median of 19 points. *Different merge methods cause further challenges.* The merged tools above use the flat argument structure; Appendix C.1 re-expresses the same merged arguments in the nested and union forms. No structure is safe for every model: the nested form costs every model 19 to 73 points, and even the otherwise robust Qwen3.5 models, which lose under 7 points on the flat merge, fall to 0.20–0.25 on it.

Besides, which functions are merged together matters as well: at a fixed granularity, grouping semantically related operations is consistently better than random or deliberately mismatched grouping, and the gap widens as the groups get larger. Mixing argument structures within one schema variant adds a further cost: for all but one of the nine models, the mixed catalog scores below the average of its constituent structures (Appendix D.2, D.1).

Per-tool surface changes pose limited challenges for most models. Nesting the arguments, removing descriptions, permuting argument order, and namespacing tool names leave every model close to its native score; the one exception is that opaque function names hurt the smallest Qwen2.5 model, and that sensitivity vanishes with scale (Appendix D.3).

Cross-call protocols are the hardest operators for most models. Spreading one action over several dependent calls produces the largest losses in the landscape: the transaction protocol and schema discovery collapse most earlier-generation models, often to near zero, and only the newest generation absorbs them. Reference resolution is the exception in the other direction: the models that handle the extra resolver call score at or above their native rate, so a cross-call protocol is not harmful in itself, but whether a model follows it is model-specific (§6 analyses the failure modes).

Table 2: The seven failure modes. Every failed episode is assigned the first mode whose rule fires. The first four surface as explicit runtime errors or routing errors; the last three are silent failure: the episode ends with the wrong set of native actions or the wrong argument values.

Failure mode	Rule	What went wrong
Livelock	The episode hits the turn limit.	The agent loops on retries or clarifications without committing an action.
Transaction handle	A transaction flag fires: a write or commit names a transaction that was never opened, an opened transaction is never committed, a commit precedes the required writes, or a transaction is committed twice.	The begin / write / commit protocol is broken.
Rejected and stuck	At least one call was rejected as off-schema and fewer native actions executed than required.	The agent calls a name or argument the schema does not expose and cannot recover.
Wrong class	A call is routed through a dispatcher or namespace of the wrong class.	The operation is chosen from the wrong group.
Under-execution	Fewer native actions executed than the task requires, with no rejection.	Required steps are skipped.
Over-execution	More native actions executed than the task requires.	Extra or repeated actions are taken.
Wrong arguments	The required actions execute with at least one wrong argument value.	The right functions are called with the wrong values.

5.2 Real tool-use benchmark validation

To test whether schema bias survives outside the synthetic grid, we evaluate all nine models on the airline and retail domains of τ^2-bench under six equivalent schemas: native, fully merged, nested args, and the three cross-call protocols. Schema bias persists in the real environment, and it is domain-dependent on top of being model-dependent. On airline, most earlier-generation models score higher under some alternative schema than under the native one, and which alternative helps differs by model; on retail, the native schema is the best or near-best choice for almost every model, fully merged collapses every model to near zero, and the cross-call protocols cost most models a large share of their native score. The newest models are not exempt: the strongest model, Qwen3.5-27B, loses under every alternative schema in both domains, and its mixture-of-experts sibling collapses under fully merged (Appendix D.5, Table 9).

6 RQ2: Failure Modes Under Equivalent Schemas

We analyze how schema representation changes the type of error an agent makes. Across failed episodes we observe seven recurring failure modes, listed in Table 2: four explicit ones (livelock, transaction handle, rejected and stuck, wrong class) and three silent ones (under-execution, over-execution, wrong arguments). The labels are automatic; in a blinded audit (Appendix E.3), three LLM proxies reach a majority label on 88% of sampled failed episodes.

Failure modes cluster on certain variants and models. Figure 2 shows how the failure modes are distributed across models and representative schemas. The same native tasks produce different error types as the schema changes. Some failure modes are largely consistent across models: fully merged catalogs fail as rejected-and-stuck, the transaction protocol as broken transaction handles, and schema discovery as silent wrong arguments. Two models stand out: Qwen2.5-7B tends to over-execute under almost every variant, and Gemma-4-12B livelocks on any dispatcher. Further observations are in Appendix E.1.

Some failures invoke correct actions but use the wrong tool calls. A model sometimes invokes the correct native action but fails to write it in the presented schema: the call is off-schema, yet it decodes to the right native action. We call these *schema-form failures* and measure them by re-running each variant with off-schema calls executed as the native action they decode to rather than rejected (Appendix E.2); the resulting delta in success is the schema-form gap, shown in Figure 5.

Schema-form failures concentrate on variants that rename or regroup tools and on the earlier model generations. Fully merged has by far the largest gap, driven by the Qwen3 and smaller Qwen2.5 models, which keep calling native tool names; composed variants show a clear gap for every earlier model except Gemma-4-12B, and on interval split almost all of Llama-3.1-8B's loss is schema-form. The recovered episodes are mostly rejected-and-stuck failure modes. The two Qwen3.5 models and Gemma-4-12B show almost no schema-form failures on any variant.

7 RQ3: Predicting Schema Difficulty Without a Full Evaluation

An environment designer who exposes a tool schema also decides how hard the environment is for every agent that will use it. RQ1 showed that this difficulty is large and model-specific, so before committing to a schema the designer needs to know how a given model will fare on it. Running the full query set on every candidate schema answers this reliably but expensively. We therefore ask what cheaper evidence suffices to rank candidate schemas for a model, comparing estimators by what they may observe: the model on the schema alone, or the model with a sample of the target task queries. Each estimator scores every variant, and we measure how well the scores rank one model's variants by the Spearman correlation ρ with the full-set success rates, computed within each model and then averaged over the nine models.

Probing the model without task queries is not enough. We try two query-free probes, both built from a few dozen simple instructions that are written automatically from the schema's own tool descriptions and argument values, without any task queries: the likelihood the model assigns to the correct call sequence for each instruction, and a schema compliance probe, which runs the model on the instructions and records how often it follows the schema's calling convention. Both rank variants only moderately well ($\rho = 0.52$ for the likelihood and 0.57 for the compliance probe). Further to this, we note that schema difficulty is task-conditional, for example, models often follow a protocol on isolated calls yet fail it inside multi-turn tasks (Appendix F). Difficulty is therefore a joint property of the model, the schema, and the task queries.

A small sample of target queries is the reliable estimate. Running the model on about one hundred queries per candidate schema ranks the variants nearly as the full set does ($\rho = 0.81$, and 0.92 with 800 queries), at a small fraction of the cost. Even 36 queries, as many as the probe instructions, reach $\rho = 0.72$, so the probes fall short because they omit the tasks, not because they are small. However, whether such an estimate transfers across task distributions is uncertain. For example, a variant ranking measured on our synthetic tasks does not predict the ranking on τ^2-bench, whereas a ranking measured on τ^2-bench predicts the ranking on BFCL moderately well (Appendix F). A designer should therefore sample from the environment's own task distribution.

8 RQ4: Reducing Schema Bias Through Training

In this section, we ask whether training can remove schema bias and whether the gains carry over to domains and schemas that were not trained on. We compare three levers of increasing cost on Qwen3-4B: Two training-free methods, variant-specific instructions and decoding constraints; Supervised Fine-Tuning (SFT) with LoRA and on-policy GRPO (Shao et al., 2024). Both training methods use the same 2,000 queries from eight training domains, either on a single schema or rotating over seven variants including native (*mixed*), and every model is evaluated on the twelve representative variants, including four held-out domains (settings in Appendix G). Table 3 compares all methods on the twelve representative variants.

Training-free methods fix habits that one sentence can describe. The constraint decoding method forces the first call to use tools provided in the schema. Among all variants, it turns out to help almost only on the fully merged schema. The instruction method, which describes each variant's calling convention in one sentence, clearly improves fully merged and schema discovery. Neither method has a noticeable effect on the other variants.

SFT repairs the trained variants but can tax the others. SFT on native-schema demonstrations keeps native performance but does not repair the hard variants, so the repair comes from seeing a variant during training rather than from more tool-use practice. SFT on mixed data lifts the trained variants (red frames in Table 3), while already-robust variants, trained or not, lose accuracy to different extents. This tax depends more on how the demonstrations are phrased than on how many there are (Appendix G).

On-policy RL repairs the trained variants without the tax. With either native-only or mixed data, RL keeps every variant at or above its untrained score. As with SFT, native-only data leaves the hard variants unrepaired, whereas mixed data gives the largest gains on the trained variants of any method, making it the most effective mitigation we test. The repair needs three conditions: the

Table 3: Change in success from each mitigation on Qwen3-4B over the twelve representative variants of Figure 2. *Base* is the untrained model; every other column show delta success rate from it. changes of at least 0.03 (about 2.5 standard errors) are shaded green or red, darker for larger changes. *Instruction* appends variant-specific calling convention describing instructions to input prompt (Table 11); *Decoding* forces the first call to be schema-valid via constrained decoding. SFT and RL columns are results averaged from three seeds; *Mixed* data rotate over seven schema variants. Red frames mark the variants each training set contains.

		Training-free		SFT		RL (GRPO)	
Variant	Base	Instruction	Decoding	Native	Mixed	Native	Mixed
Native	0.77	–	0.00	+0.02	−0.08	**+0.05**	+0.04
Tool-set partitioning							
Fully merged	0.36	+0.28	+0.29	+0.11	+0.34	+0.11	**+0.43**
Class dispatch	0.66	+0.01	−0.02	−0.26	0.00	+0.05	**+0.05**
Fully split	0.76	0.00	0.00	+0.05	−0.06	**+0.06**	+0.05
Interval split	0.76	+0.01	−0.04	−0.09	−0.07	**+0.05**	+0.05
Per-tool representation							
Nested args	0.77	+0.01	+0.01	+0.02	−0.08	**+0.06**	+0.04
Namespaced names	0.70	0.00	0.00	+0.03	+0.01	**+0.07**	+0.06
Strip descriptions	0.78	0.00	0.00	0.00	−0.10	**+0.04**	+0.03
Reorder arguments	0.78	+0.01	0.00	+0.02	−0.08	**+0.06**	+0.04
Cross-call dependency							
Transaction	0.28	+0.06	−0.11	−0.10	+0.23	+0.02	**+0.34**
Reference resolution	0.77	+0.03	+0.07	−0.13	−0.04	+0.06	**+0.11**
Schema discovery	0.07	+0.45	+0.02	−0.05	+0.67	+0.06	**+0.68**

variant must appear in the training data, the model must be free to move far from its initialization (no KL penalty), and the environment must reject invalid calls (Appendix G).

Gains transfer to recombinations of familiar schema changes, not to new ones. Both training methods keep part of their gain on the four held-out domains, and RL keeps more than SFT. To test generalization across schemas, we run an additional SFT experiment that trains on seven schema variants and evaluates on changes held out from training (Appendix G): the gains carry over to new combinations of the trained changes and to closely related untrained variants, such as finer interval splits, but not to kinds of operator the model never saw. On real benchmarks whose tools we rewrite with the same operators, the gain appears only where the rewrite matches a trained variant (transaction on τ^2-bench retail), while overall τ^2-bench and BFCL scores barely change (Appendix G). Training therefore reduces the specific schema biases it targets, but does little to make the model robust to schema changes in general.

9 Conclusion

Tool schemas are representations of an agent's action space, yet current evaluations usually treat one representation as canonical. We introduced an executable framework that changes the schema while preserving native actions and final environment states. We show that success varies substantially across equivalent schemas, that this schema bias persists even for the top-ranked models, and that it is neither a small formatting effect nor a single universal penalty: it varies systematically with the model and the schema variant. The failures follow the variant: a given schema tends to break different models in the same way, while how much it costs depends on the model. Reliably ranking schema variants by difficulty requires running a small sample of the target queries. Training repairs a variant only when that variant appears in the training data; on-policy RL does so without the tax that SFT imposes on untrained variants, and the gains extend to new combinations of trained changes but not to new kinds of schema change.

Finally, we argue that schema bias should be taken into account wherever tool calling is involved: when designing the tool schemas of an environment, when training models, and when evaluating tool-calling ability. Robustness to the schema representation should become a first-class criterion for tool-using agents.

AI USE STATEMENT

We used large language models to polish the writing of this paper. All content, analyses, and conclusions are the authors' own, and the authors checked every edit. Separately, LLMs are the subject of our experiments, and an LLM rewrote the templated evaluation queries under a grounding check that keeps the gold calls fixed (Appendix B).

ETHICS STATEMENT

This work evaluates and trains language-model agents in a synthetic tool-use environment and on public benchmarks. It involves no human subjects, no personal or sensitive data, and no real-world side effects: every tool call is executed in a simulated environment. Closed models were accessed through their public APIs under the providers' terms of use. We do not foresee specific ethical risks beyond those common to research on language-model agents.

REPRODUCIBILITY STATEMENT

Every variant passes reference-trace decoding and state-equivalence checks before evaluation. Across approximately 50,000 reference-trajectory checks, these tests identified three transformation-implementation errors before model evaluation and none afterward. We release the checks, environment adapters, domain specifications, transformation framework, the registry of all 32 schema variants, and the model–schema evaluation grids. Each reported configuration uses the full set of 2,748 queries with state-based scoring. One full configuration costs approximately 5 GPU-minutes with data-parallel evaluation on eight RTX 5090 GPUs, and all grids are resumable by model–schema configuration. At temperature 0.7, three independent seeds preserve the expected high–medium–low ordering for both Qwen3-4B and Llama-3.1-8B; seed standard deviations are at most 0.008, with episode-bootstrap confidence intervals reported per cell. The phrasing-and-volume SFT study completes all 160 planned evaluations (two phrasings, four data sizes, two seeds, and ten variants), each on the full query set. The two SFT-then-RL runs each use 19,200 rollout episodes and 200 optimizer steps, costing 24.2 and 24.5 GH200 GPU-hours.

The ranking analysis uses the full 9×32 model–variant matrix, with all 288 cells complete and paired at the episode level. The two additional training studies of Appendix G.6 train and evaluate every planned run (36 GRPO runs and 48 checkpoints, respectively) without selecting checkpoints or seeds, and bind training data, checkpoints, and evaluation outputs to recorded digests so that incomplete runs cannot enter the results.

REFERENCES

Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Bhargav, Maxwell Crouse, Chulaka Gunasekara, et al. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks. *arXiv preprint arXiv:2407.00121*, 2024.

Anthropic. Introducing the model context protocol. `https://www.anthropic.com/news/model-context-protocol`, 2024.

Victor Barres et al. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Chen Chen et al. ACEBench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*, 2025.

Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California, Irvine*, 2000.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. MetaTool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*, 2024.

Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. Action space shaping in deep reinforcement learning. In *IEEE Conference on Games (CoG)*, 2020.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. ToolACE: Winning the points of LLM function calling. *arXiv preprint arXiv:2409.00920*, 2024a.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024b.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024.

Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of what art? a call for multi-prompt LLM evaluation. *Transactions of the Association for Computational Linguistics*, 2024.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design. *arXiv preprint arXiv:2310.11324*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. volume 112, pp. 181–211, 1999.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Junjie Ye, Sixian Wu, Guanyu Li, Tao Gui, Qi Zhang, and Xuanjing Huang. RoTBench: A multi-level benchmark for evaluating the robustness of large language models in tool learning. *arXiv preprint arXiv:2401.08326*, 2024.

A LIMITATIONS

Our framework covers nine operators and 32 schema variants, but the space of equivalent schemas is larger: other directions, such as how tool results and errors are reported or how state is exposed across calls, may reveal biases that our variants do not. The mitigation results are bounded in the same way. Training removes the bias only for the schema changes it covers and their combinations, not for new kinds of change, so every new interface design still has to be evaluated, and if necessary trained for, on its own; schema bias cannot be fixed once for all. Finally, most measurements come from a controlled synthetic environment and a finite set of models, so the size of the effect on other benchmarks and model families remains to be established.

Table 4: The twelve synthetic domains, native operation counts, and the intervention-training split used in RQ4. Each domain defines a grouped native tool schema (system context, callable operations, and typed arguments); the full specification is released with the artifact.

Domain	Representative operations	Ops	RQ4
Car cabin controller	windows, sunroof, seats, cabin climate	18	train
Home HVAC / thermostat	modes, zones, holds, fan settings	13	train
Multi-room media player	playback, sources, queues, rooms	15	train
Smart kitchen appliances	oven, stove, timers, presets	13	train
Smart laundry	wash/dry cycles, soil and water levels	13	train
Home security	locks, alarms, sensors, access codes	13	train
Smartwatch & fitness	workouts, goals, hydration, notifications	13	train
Car navigation	routes, waypoints, ETA, traffic	13	train
Calendar & scheduling	events, invites, reminders, availability	15	held out
Smart home controller	lights, scenes, rooms, automations	16	held out
Smart TV	inputs, apps, search, playback	13	held out
Robot vacuum	cleaning runs, maps, rooms, schedules	13	held out
Total		**168**	

B EXPERIMENTAL SETUP

This appendix gives the full setup summarized in §4.

B.1 SYNTHETIC ENVIRONMENT AND QUERIES

The controlled environment has twelve synthetic domains with 168 native operations (Table 4). Each domain defines a native tool schema, namely a system context, callable operations with typed arguments, and multi-step routines used for compositional queries, from which the tool catalog and the query sets are derived. The environment implements every operator of Table 1. The released artifact includes the native schemas, the query files, and the transformation registry.

The 2,748 task queries come in three tiers: 585 *single* queries requiring one native call, 1,747 *compound* queries combining several independent operations in one utterance, and 416 *compositional* queries chaining dependent steps within a domain, where later actions reuse entities established earlier. Table 5 summarizes the catalog and the queries. Every tool argument is required and 66% of arguments are enumerated, which is what makes the split operators (baking enum values into tool names) and the merge operators (routing by an `operation` enum) well defined. The model–schema landscape uses all 2,748 queries in all twelve domains. For training in §8 only, eight domains supply the training queries and four are held out to test transfer (1,789 and 959 evaluation queries). The same query and target native actions are held fixed across all schema variants.

How the queries are built. The domains are written by hand as a declarative specification modelled on real device and web interfaces (home automation, media players, calendars, in-car controls). Gold calls are sampled programmatically with a fixed seed: a single query calls one tool with sampled argument values, a compound query samples two to six independent calls within a domain, and a compositional query instantiates a hand-written multi-step routine whose later calls depend on earlier ones. Each gold call list is first rendered as a templated request and then rewritten by gemini-3.5-flash in two passes, both conditioned on the gold calls and never changing them. The first pass writes a natural request for exactly those calls; the second adds a persona and situational context, one or two distractor remarks that map to no tool, filler, and non-linear ordering, which roughly triples the length. Every concrete gold value must remain recoverable from the final text: the 49 queries (1.8%) whose grounding falls below 0.9 keep the templated phrasing, and mean grounding over all queries is 0.995.

Table 5: Statistics of the controlled synthetic environment: the native tool catalog (top) and the 2,748 evaluation queries (bottom). Every query is a single user utterance whose target is one to six native calls within one domain; there are no multi-turn dialogues. Word counts are for the colloquial queries used in all experiments, with the templated source query in parentheses.

Native catalog	
Domains	12
Native tools	168 (13–18 per domain)
Arguments	397 (1–4 per tool, mean 2.4; all required)
enumerated (string enum)	261 (2–19 values, mean 3.8)
free string / integer / boolean	43 / 92 / 1
Queries	
Queries	2,748 (198–265 per domain)
Gold native calls	7,818 (1–6 per query, mean 2.8)
single: one call	585
compound: 2 / 3 / 4 / 5 / 6 independent calls	441 / 599 / 503 / 202 / 2
compositional: 3 / 4 dependent calls	144 / 272
Tools covered by gold calls	168 of 168 (18–97 calls per tool, median 47)
Arguments per gold call	2.4
Query length, words	11–149, median 80 (templated: 9–31 mean)
single / compound / compositional, mean	66 / 84 / 88 (9 / 28 / 31)

B.2 Real benchmarks

We test external validity on the retail and airline environments of τ^2-bench and on BFCL multi-turn episodes. For τ^2-bench, our adapter changes the exposed schema while leaving the benchmark's native execution interface, recorded trajectory, and scorer unchanged. For BFCL, we align episodes by turn and report state-independent turns separately from later turns whose executable state is unavailable. Both benchmarks keep their native task splits and serve only as validation targets for transfer.

B.3 Models

Nine open-weight models are evaluated on all 32 variants: seven *earlier-generation* models from three families, from 4B to 32B parameters (Qwen3-4B, Qwen3-30B-A3B, Qwen2.5-7B/14B/32B, Llama-3.1-8B, and Gemma-4-12B), and two *newer-generation* Qwen3.5 models (27B and 35B-A3B), which use that generation's native tool-call format with thinking enabled. Qwen3.5-4B and 9B additionally appear on the merge argument structures (Table 6). Two closed models are evaluated through their APIs on the thirteen representative variants of Figure 2, with the same queries and scorer: gpt-6-luna (OpenAI Responses API, reasoning effort medium) and GLM-5.3-Flash (default settings), one run each in September 2026.

B.4 Evaluation protocol

Every model and schema variant is evaluated on the same 2,748 queries. An episode succeeds when the multiset of native actions it executes equals the gold multiset exactly, so the scoring rule does not depend on the schema used to express the actions. The environment rejects tool calls that violate the active schema and returns a recoverable error, so the agent can retry after reading the feedback. Unless noted, results use this state-based exact scoring with invalid calls rejected, and deterministic decoding.

C SCHEMA VARIANTS

C.1 METHODS OF EACH OPERATOR

Section 3.3 names only the representative method of each operator (Table 1). This section lists every method evaluated in the paper; Appendix C.2 shows an excerpt of each, and Appendix C.3 reports their scores.

Merge: which functions go together. *Fully merged* puts all functions of a domain into one dispatcher. *Class dispatch* puts the functions of each class, across the whole 168-function catalog, into one dispatcher per class (twelve dispatchers). *Grouped merges* split a domain into dispatchers of about two, four, or eight functions each, formed by semantic similarity, at random, or by deliberately mixing unrelated functions. *Partial merges* fuse only part of a domain and form the merge side of the granularity ladder (Figure 3); they are labelled by the resulting function count relative to native.

Merge: how the merged tool takes its arguments. All three argument structures decode to the same native call; they differ only in where the model states which function it means and where it places that function's arguments. In the *flat* structure, the dispatcher has an `operation` enum and one optional field for every argument of every member function, prefixed by the function name (`set_thermostat__zone`). In the *nested* structure, it has the `operation` enum and a single `arguments` object whose properties are the union of the member functions' arguments, the shape used by JSON-RPC and MCP. In the *union* structure, there is no `operation` field: the dispatcher has one argument object per member function, and the model selects the function by filling exactly one of them. Averaged over domains, the fully merged flat, nested, and union schemas have about 1,300, 830, and 1,815 tokens and 34, 2, and 14 top-level arguments. JSON-Schema `oneOf` is not used because tool-call parsers do not support it reliably, which would confound parser and model failures.

Split. *Fully split* bakes every enumerable argument into the tool name, so `set_mute(zone, state)` becomes `set_mute__state_muted(zone)` and `set_mute__state_unmuted(zone)`; the split side of the granularity ladder bakes in progressively more arguments, up to $10.6\times$ the native function count. *Interval split* does the same for one numeric argument by cutting its range into two, three, or five intervals (one, two, or four cut points).

Per-tool representation. *Nest* packs a subset of a tool's arguments into one nested object (one level deep in all our variants). *Rename* either prefixes tool names with their class as a namespace (*namespaced names*) or replaces them with opaque strings (*opaque names*). The *whole catalog* control is not a rename: it exposes all 168 functions with their native names and serves as the control for class dispatch and namespaced names, which expose the same catalog. *Strip descriptions* and *reorder arguments* have a single method each.

Cross-call protocols. *Transaction* applies the open–write–confirm sequence either to every function or to a subset, and can be composed with reference resolution so that the written arguments are handles rather than values. *Reference resolution* and *schema discovery* have a single method each.

C.2 EXCERPTS OF EACH METHOD

Each listing is the climate-domain tool `set_thermostat` unless noted. Blue marks what the operator changes relative to native; gray `...` omits unchanged fields or sibling tools. Flatten-arguments is omitted because the native parameters are already flat.

Native. Default original tool schema.

```
{
  "name": "set_thermostat",
  "description": "Set a zone thermostat mode and target temperature.",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", "downstairs", "
    basement", "whole_house"]},
```

```
    "mode": {"type": "string", "enum": ["heat", "cool", "auto", "eco", "
    off"]},
    "temperature": {"type": "integer"}
  },
  "required": ["zone", "mode", "temperature"]
}
```

Merge, flat argument structure. Thirteen named tools become one dispatcher. Every member's arguments appear as optional namespaced fields on that one tool; hard validation rejects arguments that do not belong to the chosen `operation`.

```
{
  "name": "execute",
  "parameters": {
    "operation": {"type": "string", "enum": ["set_thermostat", "
    set_temp_range", ...]},
    "set_thermostat::zone": {"type": "string", "enum": ["upstairs", ...],
    "optional": true},
    "set_thermostat::mode": {"type": "string", "enum": ["heat", ...], "
    optional": true},
    "set_thermostat::temperature": {"type": "integer", "optional": true},
    "set_temp_range::heat_to": {"type": "integer", "optional": true},
    ...
  },
  "required": ["operation"]
}
```

Merge, nested argument structure. Same members and `operation` enum. Arguments sit in one shared object whose properties are the *union* of member parameters, not a per-operation copy.

```
{
  "name": "execute",
  "parameters": {
    "operation": {"type": "string", "enum": ["set_thermostat", ...]},
    "arguments": {
      "zone": {"type": "string", "enum": ["upstairs", ...]},
      "mode": {"type": "string", "enum": ["heat", ...]},
      "temperature": {"type": "integer"},
      "heat_to": {"type": "integer"},
      ...
    }
  }
}
```

Merge, union argument structure. There is no `operation` enum. The agent selects an operation by which object is present and leaves the others unset.

```
{
  "name": "execute",
  "parameters": {
    "set_thermostat": {
      "zone": {"type": "string", "enum": ["upstairs", ...]},
      "mode": {"type": "string", "enum": ["heat", ...]},
      "temperature": {"type": "integer"}
    },
    "set_temp_range": {
      "zone": {"type": "string"},
      "heat_to": {"type": "integer"},
      "cool_to": {"type": "integer"}
    },
    ...
  }
```

```
}
```

Merge, class dispatch (catalog scope, class grouping, flat). Same flat argument structure, one dispatcher per class, named by the class. On this single-class catalog that is one tool; a 12-class catalog has 12.

```
{
  "name": "climate",
  "parameters": {
    "operation": {"type": "string", "enum": ["set_thermostat", "
    set_temp_range", ...]},
    "set_thermostat::zone": {"type": "string", "optional": true},
    ...
  },
  "required": ["operation"]
}
```

Merge, semantic grouping. This is still a flat merge. Similar names and parameter sets are clustered, then each cluster is merged, so climate's 13 tools become three dispatchers rather than one. The argument structure matches merge-flat (an `operation` enum plus namespaced optional arguments). Random and anti-semantic grouping keep these three dispatcher sizes and scramble membership.

```
{
  "name": "thermostat_humidity_ops",
  "parameters": {
    "operation": {"enum": ["set_thermostat", "set_thermostat_hold",
      "set_temp_range", "set_humidity", "set_humidity_alert",
      "set_dehumidifier", "set_air_purifier", "set_vent", "
    set_zone_priority"]},
    "set_thermostat::zone": {"type": "string", "optional": true}
  }
}
{
  "name": "mode_fan_ops",
  "parameters": {
    "operation": {"enum": ["set_fan_mode", "set_away_mode", "
    set_hvac_schedule"]}
  }
}
{
  "name": "filter_reminder_ops",
  "parameters": {
    "operation": {"enum": ["set_filter_reminder"]}
  }
}
```

Split by enum value (fully split). `mode` is removed as an argument and baked into the function name. The catalog replaces one tool with five; two are shown.

```
{
  "name": "set_thermostat__mode_heat",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", ...]},
    "temperature": {"type": "integer"}
  },
  "required": ["zone", "temperature"]
}
{
  "name": "set_thermostat__mode_cool",
  "parameters": { ... }
}
```

Split by numeric predicate (interval split). `set_thermostat` becomes two tools that still take `temperature`, but each accepts only one side of the cut.

```
{
  "name": "set_thermostat__temperature_low",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", ...]},
    "mode": {"type": "string", "enum": ["heat", ...]},
    "temperature": {"type": "integer", "exclusiveMaximum": 20}
  }
}
{
  "name": "set_thermostat__temperature_high",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", ...]},
    "mode": {"type": "string", "enum": ["heat", ...]},
    "temperature": {"type": "integer", "minimum": 20}
  }
}
```

Nest arguments. The native three-argument object becomes one required wrapper. The inner fields are unchanged.

```
{
  "name": "set_thermostat",
  "parameters": {
    "options": {
      "zone": {"type": "string", "enum": ["upstairs", ...]},
      "mode": {"type": "string", "enum": ["heat", ...]},
      "temperature": {"type": "integer"}
    }
  },
  "required": ["options"]
}
```

Rename, namespaced names. Only the function name changes: the class name is prefixed as a namespace; arguments stay native.

```
{
  "name": "climate__set_thermostat",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", ...]},
    "mode": {"type": "string", "enum": ["heat", ...]},
    "temperature": {"type": "integer"}
  }
}
```

Rename, opaque names. The name becomes an opaque identifier; the description and arguments stay native, so the description is the remaining routing signal.

```
{
  "name": "fn_1748c5",
  "description": "Set a zone thermostat mode and target temperature.",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", ...]},
    "mode": {"type": "string", "enum": ["heat", ...]},
    "temperature": {"type": "integer"}
  }
}
```

Strip descriptions. Function and parameter descriptions are removed; names and types stay.

```
{
  "name": "set_thermostat",
  "description": "",
  "parameters": {
    "zone": {"type": "string", "enum": ["upstairs", "downstairs", "
    basement", "whole_house"]},
    "mode": {"type": "string", "enum": ["heat", "cool", "auto", "eco", "
    off"]},
    "temperature": {"type": "integer"}
  }
}
```

Reorder arguments. Names and types are unchanged. The declared property order, and therefore `required`, becomes `temperature`, `mode`, `zone`.

```
{
  "name": "set_thermostat",
  "parameters": {
    "temperature": {"type": "integer"},
    "mode": {"type": "string", "enum": ["heat", ...]},
    "zone": {"type": "string", "enum": ["upstairs", ...]}
  },
  "required": ["temperature", "mode", "zone"]
}
```

Transaction. One native call becomes five dependent calls. Nothing is executed until confirm. The `mode` and `temperature` writes match the `zone` write.

```
{
  "name": "begin_set_thermostat",
  "parameters": {}
}
{
  "name": "set_set_thermostat__zone",
  "parameters": {
    "txn_id": {"type": "string"},
    "zone": {"type": "string", "enum": ["upstairs", ...]}
  },
  "required": ["txn_id", "zone"]
}
{
  "name": "set_set_thermostat__mode",
  "parameters": { "txn_id": {"type": "string"}, "mode": {"type": "string
    ", ...} }
}
{
  "name": "set_set_thermostat__temperature",
  "parameters": { "txn_id": {"type": "string"}, "temperature": {"type": "
    integer"} }
}
{
  "name": "commit_set_thermostat",
  "parameters": {
    "txn_id": {"type": "string"}
  },
  "required": ["txn_id"]
}
```

Reference resolution. `zone` is removed from `set_thermostat`. The agent must first call a resolver with a free-form query and then pass the returned handle.

```
{
  "name": "resolve_set_thermostat__zone",
  "parameters": {
    "query": {"type": "string"}
  },
  "required": ["query"]
}
{
  "name": "set_thermostat",
  "parameters": {
    "mode": {"type": "string", "enum": ["heat", ...]},
    "temperature": {"type": "integer"},
    "zone_ref": {"type": "string"}
  },
  "required": ["zone_ref", "mode", "temperature"]
}
```

Schema discovery. The upfront catalog has no `set_thermostat` and no argument types. `list_methods` returns those native parameter schemas at runtime; `invoke` accepts an untyped `arguments` object.

```
{
  "name": "list_methods",
  "parameters": {
    "class": {"type": "string", "enum": ["climate"]}
  }
}
{
  "name": "invoke",
  "parameters": {
    "class": {"type": "string", "enum": ["climate"]},
    "method": {"type": "string"},
    "arguments": {"type": "object"}
  }
}
```

Runtime result of `list_methods({"class": "climate"})`, abbreviated:

```
{
  "methods": [
    {
      "name": "set_thermostat",
      "parameters": {
        "zone": {"type": "string", "enum": ["upstairs", ...]},
        "mode": {"type": "string", "enum": ["heat", ...]},
        "temperature": {"type": "integer"}
      }
    },
    ...
  ]
}
```

C.3 Scores of all methods

Figure 2 shows one representative method per operator. Figure 4 reports the remaining methods in the same format on all nine models, and Table 6 gives the fully merged schema under each argument structure, including two further newer-generation models (Qwen3.5-4B and 9B). Within an operator, the score moves with the method about as much as it moves across operators, which is why the main figures fix one method per operator and the tables below vary one factor at a time.

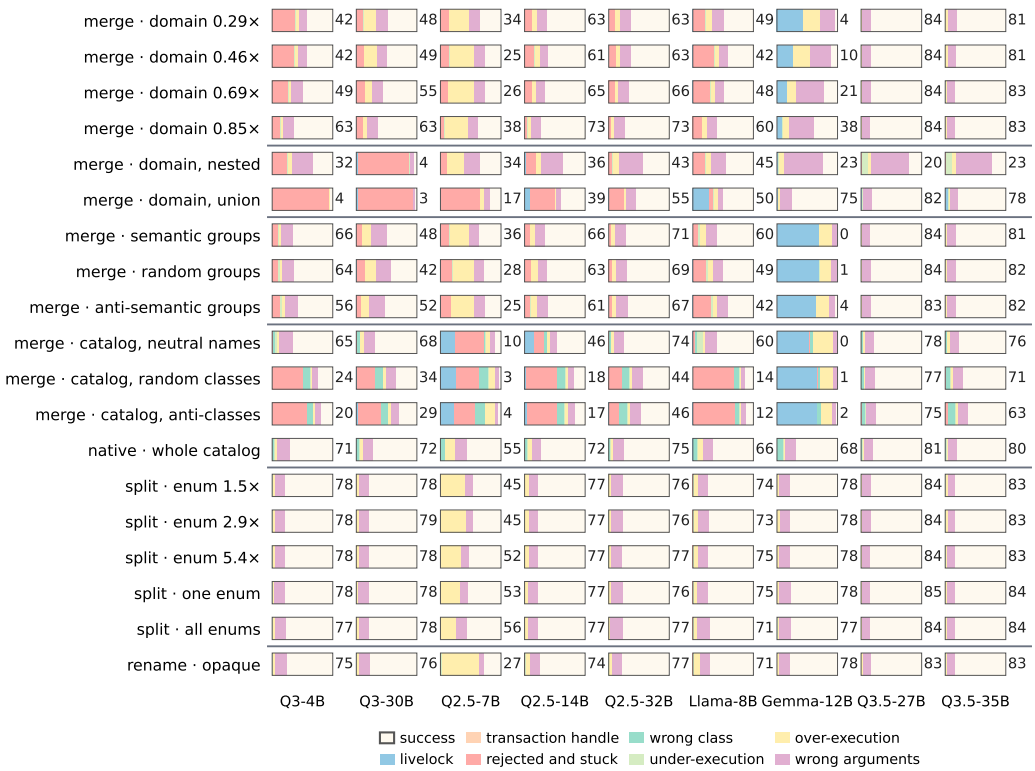


Figure 4: **Scores of the remaining methods of each operator**, in the format of Figure 2 (nine models, 2,748 queries). Row labels give the operator and method (Table 1): partial merges along the granularity ladder (function count relative to native), the nested and union argument structures of the fully merged schema, the three grouping rules at a fixed group size, the membership and naming controls of class dispatch together with the whole-catalog control, the split ladder, and opaque names.

Table 6: Merge penalty by argument-structure encoding. Exact success on the fully merged schema (one dispatcher per domain, the same member sets as the fully merged row in Figure 2) under three typed single-call argument structures, next to each model's native score from the same run. 2,748 queries per cell; a native re-run after the grid stays within 0.3 points. Bold marks each model's lowest argument structure.

model	native	flat	nested	union
Qwen3-4B	.774	.367	.324	**.036**
Qwen3-30B-A3B	.774	.070	.044	**.031**
Qwen2.5-7B	.531	.357	.341	**.174**
Qwen2.5-14B	.774	.554	**.361**	.389
Qwen2.5-32B	.763	.754	**.430**	.547
Llama-3.1-8B	.739	.688	**.454**	.497
Gemma-4-12B	.785	**.000**	.232	.752
Qwen3.5-4B	.829	.800	**.253**	.713
Qwen3.5-9B	.832	.767	**.235**	.364
Qwen3.5-27B	.845	.839	**.202**	.824
Qwen3.5-35B-A3B	.835	.790	**.228**	.778

D ADDITIONAL RESULTS ON MEASURING SCHEMA BIAS (RQ1)

D.1 ARGUMENT STRUCTURES AND MIXED CALLING CONVENTIONS

Argument structure of a merged tool. Table 6 shows that no argument structure is safe for every model: nested is the worst structure for most models and costs every newer-generation model 58 to 64 points, while flat is Gemma-4-12B's worst structure (it scores zero) and union is worst for the two Qwen3 models and Qwen2.5-7B. The same ordering holds for the semantic grouped merge (about 3.3 dispatchers per domain): nested costs 29 to 60 points on every model, union 0 to 32, and flat under 30 for every model except Gemma-4-12B (78). Under the union structure, Qwen3.5-9B produced malformed tool calls on 24.8% of requests; these are scored as model failures.

Mixing argument structures in one catalog. In the granularity ladder, partial merges, which mix one dispatcher with native named tools, score below both endpoints for Llama and Qwen2.5 (Llama: 0.71, 0.42, and 0.75 for fully merged, partial, and native), and the same dip appears on τ^2-bench airline. Partial merges, however, change the function count and the merged membership at the same time as they mix conventions. The matched test in Table 7 isolates the mixing: at a fixed partition into dispatchers (about $0.3\times$ and $0.5\times$ the native function count), every catalog exposes the same groups, names, and function count and differs only in whether all dispatchers use one argument structure or the three structures rotate across groups. The mixed catalogs have between 1,618 and 1,904 tokens, inside the range of the homogeneous ones (1,349 to 2,137). Mixing costs more than the average of its parts: H is negative with an interval excluding zero in 16 of the 18 model-by-granularity cells, and the two exceptions (Qwen2.5-14B) are within one point of zero. A mixture is nevertheless almost always better than its worst structure used throughout; the only significant exception is Qwen3-4B at $0.5\times$ (-0.065 $[-0.080, -0.048]$). Mixing protocols across calls behaves differently: a catalog in which half the functions use the transaction protocol interpolates between native and fully transactional (0.36, within $[0.28, 0.77]$, on Qwen3-4B).

D.2 WHICH OPERATIONS SHARE A DISPATCHER

Within a domain. Three grouped merges with identical group count and sizes, differing only in which operations share a dispatcher, follow the order semantic, then random, then anti-semantic for five of the seven earlier-generation models, with a semantic-minus-anti-semantic difference of 3.9 (Qwen2.5-32B) to 18.2 points (Llama-3.1-8B). Two exceptions qualify the pattern: all three groupings score near zero for Gemma-4-12B, and Qwen3-30B-A3B reverses the order (anti-semantic 0.52, semantic 0.48). The two newer-generation models stay within one point across the three groupings. On Qwen3-4B, semantic minus anti-semantic grows with group size (3.8, 9.3, and 12.6 points at two, four, and eight functions per dispatcher): semantic grouping improves as groups grow, until everything is merged into one dispatcher, while anti-semantic grouping steadily worsens. Short of full merging, most of the loss attributed to coarser grouping is explained by which operations are grouped together.

Across the whole catalog: membership, not naming. When all 168 functions are exposed at once, plain and namespaced names keep one tool per function and score nearly identically (0.71 and 0.70 on Qwen3-4B), so the gap between namespaced names and native in Figure 2 is the cost of the larger catalog, not of the prefix. Class dispatch (about 14 functions in each of 12 class-named dispatchers) scores 0.66, far above one fully merged dispatcher (0.36), but that comparison changes the number of dispatchers, their membership, and their names at once. Table 8 holds the flat structure, twelve dispatchers, and the group-size profile fixed and varies only membership and naming: the class groups with class names, the same groups with neutral names (`group_01`–`group_12`, with descriptions of matched length but no class information), and five random and five anti-class partitions (fixed seeds; anti-class partitions spread each class across dispatchers as evenly as possible), also with neutral names. Membership matters: class groups beat random groups by $+6$ (Qwen2.5-7B) to $+46$ points (Llama-3.1-8B), with intervals above zero for seven of eight models. Naming matters little: at most $+10$ points (Qwen3-8B), indistinguishable from zero for Qwen3-4B, and slightly negative for Qwen3-30B-A3B and Qwen2.5-32B, the two models whose dominant merge failure is calling the wrong dispatcher name. Gemma-4-12B is again the exception: every arrangement scores below 0.04 because it fails on any dispatcher with an `operation` enum.

Table 7: Matched convention-mixture test on all nine models. At each granularity target, every arm exposes the same dispatcher groups, names, and function count; the three homogeneous arms use one argument structure throughout, and the mixed score averages three balanced rotations of the structures across groups. H is mixed minus the mean of the three homogeneous arms, with a paired task bootstrap 95% interval (2,748 tasks).

model	ratio	flat	nested	union	mixed	H [95% CI]
Qwen3-4B	0.3×	.612	.383	.583	.411	−.115 [−.124, −.105]
	0.5×	.600	.417	.555	.352	−.172 [−.183, −.161]
Qwen3-30B-A3B	0.3×	.621	.185	.624	.296	−.181 [−.192, −.170]
	0.5×	.619	.212	.580	.240	−.231 [−.242, −.219]
Qwen2.5-7B	0.3×	.333	.133	.309	.202	−.056 [−.067, −.046]
	0.5×	.332	.100	.316	.144	−.105 [−.115, −.095]
Qwen2.5-14B	0.3×	.640	.427	.610	.556	−.004 [−.012, .005]
	0.5×	.657	.471	.607	.584	.005 [−.003, .014]
Qwen2.5-32B	0.3×	.698	.475	.689	.598	−.023 [−.030, −.016]
	0.5×	.717	.493	.722	.628	−.016 [−.023, −.009]
Llama-3.1-8B	0.3×	.560	.425	.500	.464	−.031 [−.039, −.022]
	0.5×	.591	.496	.574	.484	−.069 [−.078, −.061]
Gemma-4-12B	0.3×	.032	.374	.775	.265	−.128 [−.136, −.121]
	0.5×	.023	.468	.787	.246	−.180 [−.189, −.171]
Qwen3.5-27B	0.3×	.838	.292	.841	.609	−.049 [−.054, −.043]
	0.5×	.833	.427	.838	.665	−.034 [−.040, −.028]
Qwen3.5-35B-A3B	0.3×	.818	.338	.823	.606	−.054 [−.060, −.048]
	0.5×	.821	.446	.817	.632	−.062 [−.070, −.055]

Table 8: Whole-catalog class-grouping controls. All arms expose the 168 native operations through twelve flat dispatchers with the same group-size profile. M (membership) is class-neutral minus the mean of five random-neutral partitions; N (naming) is class-semantic minus class-neutral. Paired task bootstrap 95% intervals over 2,748 tasks. Qwen3-8B, not one of the nine main models, is included in this control only.

model	class-sem.	class-neut.	random	anti-class	M [95% CI]	N [95% CI]
Qwen3-4B	.662	.655	.233	.211	+.421 [.404, .438]	+.007 [−.004, .019]
Qwen3-8B	.636	.532	.198	.198	+.335 [.317, .352]	+.103 [.085, .123]
Qwen3-30B-A3B	.668	.683	.283	.260	+.400 [.384, .415]	−.015 [−.025, −.004]
Qwen2.5-7B	.176	.098	.036	.031	+.062 [.052, .073]	+.078 [.064, .091]
Qwen2.5-14B	.523	.460	.172	.156	+.288 [.271, .306]	+.063 [.046, .081]
Qwen2.5-32B	.730	.741	.442	.439	+.299 [.285, .313]	−.012 [−.020, −.002]
Llama-3.1-8B	.610	.603	.141	.120	+.461 [.444, .479]	+.007 [−.005, .020]
Gemma-4-12B	.036	.005	.019	.022	−.014 [−.018, −.011]	+.032 [.025, .039]

D.3 PER-TOOL CHANGES

Nesting the arguments, removing descriptions, and permuting argument order each move success by at most 4 points on every one of the nine models (Figure 2), and the few larger moves go in both directions (Qwen2.5-7B gains 1.3 points without descriptions and loses 2.4 with permuted arguments). Opaque names (Figure 4) lower Qwen2.5-7B from 0.54 to 0.27 but change every other model by at most 3 points; within Qwen2.5, the drop shrinks from 27 points at 7B to 3 at 14B and disappears at 32B. Names and descriptions are redundant routing signals: descriptions are unnecessary when names are informative (Qwen3-4B scores 0.78 without them), and names are unnecessary when descriptions remain (opaque names cost Qwen3-4B 2 points), except for the smallest Qwen2.5 model.

Table 9: Schema bias on τ^2-bench: task success of nine models under six verified-equivalent schemas in the airline (50 tasks) and retail (114 tasks) domains. In the cell marked †, one task's model request timed out on all four attempts and the task is scored as a failure; every other cell finished all tasks without error. Fully merged collapses the domain's tools into one dispatcher and nested args groups each tool's arguments into objects (the same operators as the corresponding rows of Figure 2); the three protocols are the cross-call operators of Table 1. Bold marks each model's best schema per domain.

	airline						retail					
model	native	merged	nested	txn.	ref.	disc.	native	merged	nested	txn.	ref.	disc.
Qwen2.5-7B	.24	.28	.14	.18	**.30**	**.30**	**.08**	.04	.06	.06	.04	.03
Qwen2.5-14B	.20	**.44**	.16	.22	.42	.42	**.33**	.05	.30	.17	.16	.04
Qwen2.5-32B	.32	**.46**	.30	.22	.44	.42	**.42**	.05	**.42**	.28	.20	.05
Qwen3-4B	.26	**.46**	.28	.30	.38	.40	**.28**	.09	.20	.13	.12	.06
Qwen3-30B-A3B	.30	**.46**	.38	.28	.40	.42	.39	.05	**.40**	.37	.28	.04
Qwen3.5-27B	**.72**	.40	.68	.54	.56	.52	**.46**	.04	.20	.38	.31	.25
Qwen3.5-35B-A3B	**.56**	.04	.38	.40	.18	.16	.19	.00	.25	.20	**.28**	.02
Llama-3.1-8B	.30	.38	.32	.32	**.46**	**.46**	.06	.04	.05	**.08**	.04	.05
Gemma-4-12B	.46	.00	**.48**	.42	.38	.42	**.48**	.00	.42	.31	.26†	.21

D.4 Ranking instability

On the seven earlier-generation models and all 32 schema variants (224 cells, each with the same 2,748 queries), 23 of the 31 transformed variants change the top-ranked model relative to native. Nineteen of the 21 model pairs swap order under at least one variant, and 14 pairs have a swap whose paired bootstrap interval excludes zero. On all nine models (288 cells), Qwen3.5-27B ranks first on every variant except the transaction protocol and the nested merge, and 32 of 36 model pairs swap somewhere (27 with bootstrap support). These counts measure how often rankings change; they do not imply that every swap is large or that it carries over to another benchmark.

D.5 Schema bias on τ^2-bench

Table 9 reports the τ^2-bench results behind §5.2: nine models, two domains, and six schemas. We rewrite τ^2-bench's tool definitions with our operators and decode every call back to the native tool, so the benchmark's own scorer is unchanged. The model under test also acts as the user simulator and as the natural-language judge, so absolute scores are not comparable with the τ^2-bench leaderboard; only within-model contrasts across schemas matter. The spread is large: the gap between a model's best and worst schema is 16 to 52 points on airline and, for the models above the retail floor, 22 to 48 points on retail. The two domains disagree on the direction of the effect. Fully merged lifts Qwen2.5-14B, Qwen2.5-32B, Qwen3-4B, and Qwen3-30B-A3B by 14 to 24 points above native on airline but drops every model to at most 0.09 on retail; reference resolution and schema discovery lift Llama-3.1-8B by 16 points on airline, while they cut the retail scores of the Qwen2.5 and Qwen3 models by 11 to 37 points. Nested args is the least harmful operator, within 8 points of native in 15 of 18 model–domain cells. Gemma-4-12B scores zero under fully merged in both domains, matching its merge profile on the synthetic grid. Qwen3.5-27B is the only model whose native schema is best in both domains.

D.6 Thinking in the newer generation

Turning off thinking on Qwen3.5, with the same queries and scorer as Figure 2, shows where that generation's robustness comes from. The improvement on fully merged is mostly in the weights (a drop of 4 points without thinking), as is most of the improvement on the transaction protocol (a drop of 13 to 17 points). Schema discovery at 4B still depends on thinking (a drop of 52 points); at 9B the drop is only 10 points. This is a test-time ablation of the generation contrast in §5, not a training intervention.

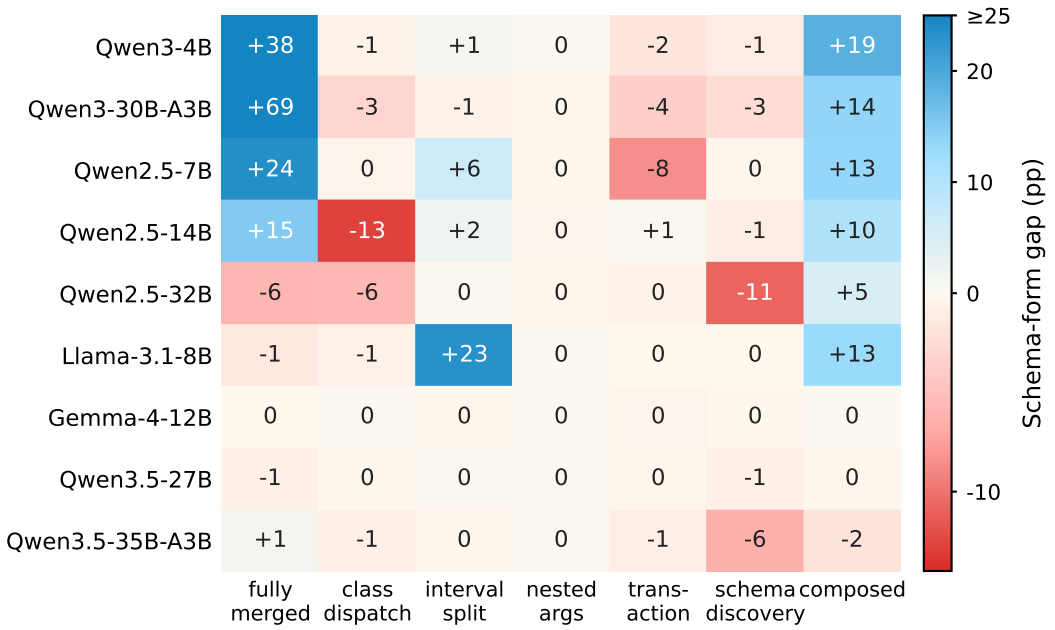


Figure 5: **Schema-form gap per model and variant.** Change in success (percentage points) when off-schema calls are executed as the native action they decode to rather than rejected. Positive values (blue) are schema-form failures; negative values (red) mean rejection with retry does better than executing the decoded call. The colour scale saturates at $+25$; the printed number is the true gap. Nested args is a near-zero control; interval split uses one threshold.

E Additional results on failure modes (RQ2)

E.1 Failure modes across models and scales

Within Qwen2.5, scale turns frequent, explicit failures into rarer, silent ones: at 7B, 65% of failed episodes end rejected and stuck; at 32B, failures fall to 25% of episodes and 72% of them are wrong arguments. Larger models in this family fail less often, but their remaining errors are less likely to surface as rejections. Under the transaction protocol, six of the seven earlier-generation models concentrate their failures on unused transaction handles (44 to 100% of failures); only Gemma-4-12B livelocks instead (70%). Under schema discovery, every model is dominated by silent wrong arguments (54 to 72%). Reference resolution has no shared mode: failure rates range from 7% (Gemma-4-12B) to 61% (Qwen2.5-7B) and split among wrong arguments, under-execution, and over-execution. Final success can also hide rejected first attempts: Qwen3.5-35B-A3B scores 0.79 on fully merged but still has 38% of its first attempts rejected, against 10% for its dense 27B sibling.

E.2 Schema-form gaps

Figure 5 compares two scorings of the same episodes: invalid calls rejected (the default) or executed as the native call they decode to. The gap is largest on fully merged: 69 points for Qwen3-30B-A3B, 38 for Qwen3-4B, 24 for Qwen2.5-7B, and 15 for Qwen2.5-14B. On Qwen3-4B, 544 of the 2,748 composed-variant episodes succeed when invalid calls are executed, and every one of them decodes to the gold native names; under fully merged, 66% of calls are still rejected, typically because the model calls a native name instead of the dispatcher. On composed variants the gap is 5 to 19 points for the earlier-generation models except Gemma-4-12B, whose gap is near zero. Llama-3.1-8B gains 23 points on interval split, where the other models stay within 7 points. Negative gaps, where rejection followed by a retry does better, are largest for Qwen2.5-14B on class dispatch (-13), Qwen2.5-32B on schema discovery (-11), and Qwen3.5-35B-A3B on schema discovery (-6). No newer-generation or Gemma cell has a positive gap above 2 points.

E.3 AUDIT OF THE FAILURE LABELS

The seven failure modes of Table 2 are assigned by rules, without manual annotation. To bound their reliability, we sample 150 failed episodes approximately evenly across model–variant strata, remove the model, variant, outcome, automatic label, and decoder flags, and ask three independent LLM labelers (Qwen3, Qwen2.5, and Llama) to assign one of the seven modes to each episode. This is an audit against LLM proxies, not human validation.

A majority label (at least two of the three labelers agree) exists for 88% of the episodes. Two modes cannot be recovered by the labelers: transaction handle and rejected-and-stuck are defined by decoder flags that the blinded episodes omit, and no labeler ever chose them, although 45 of the 150 automatic labels fall in these two modes. On the majority-labelled episodes whose automatic label is one of the other five modes, the automatic label matches the majority in 77%. We therefore use the modes to compare aggregate error distributions rather than to interpret individual episodes.

F ESTIMATING SCHEMA DIFFICULTY (RQ3)

Every estimator scores each variant for a model, and ρ is the Spearman correlation between these scores and the full 2,748-query success rates, computed within each model over the variants that estimator scores and then averaged over the nine models; scores are never pooled across models.

Probes that do not use the task queries. The schema compliance probe runs 36 single-operation instructions per variant, derived from the schema (three per domain, no task queries), and records compliance, the fraction of probes that execute at least one native action with no rejection. On the 13 probed variants, compliance ranks each model's variants with mean $\rho = 0.57$ (range 0.00 to 0.88). The likelihood probe converts the same instructions' gold native calls into each variant's correct call sequence, executes it to obtain the tool responses, and scores the log-probability the model assigns to its own turns without generating; ranked by this score, $\rho = 0.52$ (range 0.24 to 0.75). Table 10 shows why the probes fall short: six of the seven earlier-generation models follow the schema-discovery protocol on isolated calls (compliance 0.67–1.00) yet score 0.03–0.29 on the task queries, and Qwen3-4B follows the transaction protocol in isolation (0.97) but scores 0.28 on tasks; conversely, the Qwen2.5 models fail the isolated transaction probe (0.00–0.03), while Qwen2.5-32B scores 0.77 on transaction tasks. The newer-generation models comply on almost every probe and also score 0.75 to 0.84 on the corresponding tasks.

Samples of the target queries. Subsampling a variant's queries (40 draws per size) ranks a model's variants with $\rho = 0.72$ at 36 queries, 0.81 at 100, and 0.92 at 800; the lower end of each range is set by the newer-generation models, whose variants differ by only a few points. The correlation uses every variant with episode-level logs (27 for Qwen3-4B and the newer-generation models, 17–19 for the others).

Across benchmarks. Difficulty rankings do not carry over from the synthetic tasks to τ^2-bench for the two models with the most different schema profiles ($\rho = -0.12$ to -0.25 for Qwen3-4B and 0.25 to 0.31 for Qwen2.5-32B, with intervals that include zero). Between τ^2-bench and the state-independent turns of BFCL, a stratified sample of variants reaches $\rho = 0.62$ and 0.76, and a six-variant sample 0.72 and 0.79.

Table 10: Isolated-call compliance vs. full-task exact success (compliance / exact) on the variants with the largest task-conditional gaps. Compliance is the fraction of 36 schema-derived single-operation probes executed with no rejection and no task query; exact is the full 2,748-query score of Figure 2.

model	transaction	schema discovery	opaque names	fully merged
Qwen3-4B	.97 / .28	.81 / .07	1.00 / .75	.33 / .36
Qwen3-30B-A3B	1.00 / .72	.89 / .11	1.00 / .76	.00 / .07
Qwen2.5-7B	.00 / .25	.67 / .03	1.00 / .27	.36 / .35
Qwen2.5-14B	.03 / .10	.33 / .05	1.00 / .74	.22 / .54
Qwen2.5-32B	.00 / .77	.92 / .29	1.00 / .77	.25 / .75
Llama-3.1-8B	.03 / .00	.78 / .04	1.00 / .71	1.00 / .71
Gemma-4-12B	1.00 / .60	1.00 / .77	1.00 / .78	.97 / .00
Qwen3.5-27B	1.00 / .76	1.00 / .80	1.00 / .83	1.00 / .84
Qwen3.5-35B-A3B	1.00 / .79	1.00 / .75	1.00 / .83	.83 / .79

G Reducing schema bias through training (RQ4)

G.1 Training-free methods

Table 11 lists the one-sentence instruction used for each variant and the decoding setting. On models other than Qwen3-4B, the schema-discovery instruction adds about 45 points for the models that follow it and almost nothing for the others. Constraining the first call raises Gemma-4-12B on fully merged from 0.00 to 0.50 and Qwen3-30B-A3B from 0.07 to 0.74, but lowers Qwen3.5-4B by 12 points. The transaction instruction adds at most 10 points (Qwen2.5-7B from 0.25 to 0.35, Qwen2.5-14B $+8$) and changes the remaining models by at most 4 points.

G.2 Training setup

All training uses Qwen3-4B-Instruct-2507 and the same 2,000 queries from the eight training domains (Table 4), phrased as templated requests. They are sampled afresh with a different seed, and any query whose domain and gold-call multiset match an evaluation query is rejected, so no evaluation task appears in training. SFT traces are oracle call sequences for these queries under the training variant, replayed through the environment to check that they reproduce the gold native actions with no rejection. RL uses the same queries as prompts; the reward is exact success plus partial credit for recalled native actions, minus penalties for spurious actions and protocol violations. Table 12 lists the hyperparameters.

G.3 What training data, KL penalty, and feedback contribute

Table 13 collects the ablation runs behind §8. These ablations were run on an earlier six-variant mixture (fully merged, transaction, reference resolution, schema discovery, opaque names, and a one-argument enum split); the *Mixed* columns of Table 3 use the seven-variant mixture described in §8.

SFT. Training on one variant repairs it: transaction-only data lift transaction from 0.28 to 0.57, and schema-discovery-only data lift schema discovery from 0.07 to 0.74. The tax on other variants depends on the data: transaction-only data leave native unchanged and also lift fully merged to 0.75, whereas schema-discovery-only data lower native by 16 points and reference resolution by 21. Native-only data leave the hard variants unrepaired. Seeds differ widely on mixed data (native 0.60, 0.70, and 0.81 for the three seeds of the six-variant mixture, and 0.42 to 0.84 for the four seeds of the seven-variant mixture). On Qwen2.5-7B, which starts farther from its ceiling, the same mixed data raise every evaluated variant (native from 0.54 to 0.78, two seeds).

RL. On transaction data, GRPO raises the reward from -0.22 to near its ceiling within about fifteen updates and holds it for the remaining updates, with no truncated episodes. Native-only and mixed RL keep every already-robust variant at or above its untrained score; transaction-only RL with KL$=0$ is the exception, lowering reference resolution from 0.77 to 0.63. Repairing schema discovery

Table 11: Training-free interventions used in §8. *Instruction* appends the listed sentence for each variant to the default system prompt, which already requires using only the provided tools; the native schema needs no instruction. *Decoding* sets vLLM `tool_choice=required` on the first turn only, so the first completion is grammar-constrained to a schema-valid tool name and arguments; later turns use the default `auto` policy.

Variant	Instruction
Fully merged	All operations go through a single dispatcher tool. Set `operation` to the operation you need and pass each of its arguments under the key `<operation>::<argument>`.
Class dispatch	Each class has one dispatcher tool named after the class. Call the tool of the right class, set `operation` to the method you need, and pass each argument under the key `<method>::<argument>`.
Fully split	The values of enumerable arguments are part of the tool names. Call the tool whose name contains the values you need and pass only the remaining arguments.
Interval split	Numeric arguments are split into ranges across several tools. Call the tool whose name gives the range that contains your value, and still pass the value itself.
Nested args	Some arguments are grouped into nested objects. Put each argument inside the object that the tool's parameter schema defines for it.
Namespaced names	Tool names are prefixed with their class name. Call each tool by its full prefixed name, using the class you intend to act on.
Strip descriptions	The tools come without descriptions. Infer what each tool does from its name and its argument names.
Reorder arguments	The order of arguments in the tool definitions is arbitrary. Fill every argument by its name, not by its position.
Transaction	Open a transaction and read the handle it returns, pass that exact handle to every subsequent write, and finish by confirming the transaction. Do not invent or reuse handles, and do not leave a transaction unconfirmed.
Reference resolution	Some arguments take a handle instead of a raw value. First call the matching resolver tool with the user's value, then pass the handle it returns.
Schema discovery	Retrieve a class's available methods and argument names before the first call to that class, and use only the returned names. Do not guess argument names.

Table 12: Training hyperparameters (§8). The base model is Qwen3-4B-Instruct-2507 unless noted; the Qwen2.5-7B run uses the same LoRA recipe.

method	configuration
LoRA SFT	Rank r=32, α=64, dropout 0.05; targets `q,k,v,o,gate,up,down_proj`; loss on assistant turns only. Two epochs, learning rate 2×10^{-4} with cosine decay and 3% warmup, AdamW, bf16, batch size 1 with gradient accumulation 8. Maximum sequence length 16k tokens (shorter cutoffs truncated transaction traces), 32k for the seven-variant mixture (the namespaced-names and fully split tool lists reach about 22k tokens), and 8k for the phrasing-and-volume study. Seeds 42 to 45.
GRPO	On-policy GRPO in slime with Megatron (tensor parallel 2) and SGLang rollouts. Learning rate 1×10^{-6} (constant), eight samples per prompt, 24 prompts per rollout, global batch 96, KL coefficient in $\{0, 0.01, 0.1\}$ (low-variance estimator), clipping 0.2/0.28, no entropy bonus, rollout temperature 1.0, at most 2,048 generated tokens per turn and 12 turns per episode; episodes and rollout contexts are capped at 11k tokens, or 32k for the seven-variant mixture. Every run uses 100 rollouts (19,200 episodes, 200 optimizer steps); an SFT-then-RL run takes about 24 GPU-hours on GH200 GPUs.

needs both the variant in the data and no KL penalty: mixed data with KL$=0$ reach 0.75, while mixed data with KL$=0.01$ reach 0.25, schema-discovery-only data with KL$=0.01$ reach 0.12, and transaction-only data with KL$=0$ reach 0.20. The rejection signal, not its wording, drives learning: on transaction-only data, the schema-discovery variant, which is not trained, reaches 0.49 with explained rejections and 0.49 with bare refusals, but only 0.09 when invalid calls are executed instead of rejected. Starting GRPO from the mixed SFT adapter reaches the same scores as mixed RL from the untrained model even with KL$=0.01$, because SFT has already moved the model away from its initialization.

G.4 Phrasing and volume of SFT data

To separate how demonstrations are phrased from how many there are, we train LoRA adapters on templated or colloquially rephrased versions of the same traces (same gold calls) at 500, 1,000,

Table 13: Training ablations on Qwen3-4B: exact success on the full 2,748-query set for six of the twelve representative variants, by training method, training data, and KL coefficient (n = seeds averaged). *Mixed* data rotate over fully merged, transaction, reference resolution, schema discovery, opaque names, and a one-argument enum split. The last three RL rows change the environment or the reward: rejections without an explanation, no rejections (invalid calls are executed as their decoded native call), and a reward of exact success only. The SFT→RL rows start GRPO from the mixed SFT adapter.

Training data	n	Native	Fully split	Fully merged	Ref. res.	Trans.	Schema disc.
Untrained		0.77	0.76	0.37	0.77	0.28	0.07
SFT							
native only	2	0.79	0.81	0.47	0.64	0.18	0.02
transaction only	2	0.81	0.79	0.75	0.83	0.57	0.35
schema discovery only	2	0.61	0.55	0.59	0.57	0.39	0.74
mixed	3	0.70	0.71	0.69	0.74	0.50	0.70
RL, KL$= 0$							
native only	1	0.83	0.82	0.48	0.83	0.30	0.13
transaction only	2	0.78	0.76	0.55	0.63	0.77	0.20
mixed	2	0.81	0.80	0.79	0.87	0.65	0.75
RL, KL$= 0.01$							
transaction only	1	0.77	0.76	0.57	0.81	0.74	0.49
schema discovery only	1	0.81	0.81	0.68	0.74	0.37	0.12
mixed	2	0.82	0.82	0.80	0.88	0.54	0.25
transaction only, bare refusals	1	0.77	0.74	0.48	0.81	0.75	0.49
transaction only, no rejections	1	0.79	0.78	0.46	0.81	0.76	0.08
transaction only, exact-match reward	1	0.78	0.77	0.50	0.66	0.72	0.11
SFT→RL							
mixed, KL$= 0.01$	2	0.81	0.80	0.79	0.86	0.67	0.75

2,000, and 4,000 examples, with two seeds each, and evaluate them on ten variants in the four held-out domains (160 evaluations). Colloquial data win at every size: the mean over the ten variants is 0.680 versus 0.447 at 500 traces, 0.711 versus 0.569 at 1,000, 0.698 versus 0.473 at 2,000, and 0.691 versus 0.607 at 4,000. More data do not help beyond 1,000 colloquial traces.

G.5 TRANSFER

Table 14 summarizes three transfer tests.

Held-out domains. With the seven-variant mixture, SFT keeps on average 43% of its training-domain gain on transaction (8 to 61% across four seeds) and 67% on schema discovery in the four held-out domains, while RL keeps 65% and 71% (three seeds). Held-out transaction success reaches 0.38 after SFT and 0.49 after RL.

Held-out schema changes. A separate LoRA adapter is trained on 2,000 traces rotating over seven changes (fully merged, one-cut interval split, nested args, transaction, reference resolution, opaque names, and a one-argument enum split) and evaluated on changes that were not trained: new combinations of the trained changes (transaction composed with reference resolution, and a catalog mixing several operators), a stronger version of a trained change (interval split with four cut points instead of one), new operators (splitting every enum argument, and semantic grouped merges), and new class-level structures (class dispatch and schema discovery). Relative to the untrained model, the trained changes improve by 14.6 points, new combinations by 8.8, the stronger version by 11.2, new operators by -2.4, and new class-level structures by 1.2 (exact success; recall -3.5).

Real benchmarks. Two seeds of SFT followed by RL (six-variant mixture) gain 9.2 and 10.7 points over the untrained model on the synthetic grid. On τ^2-bench, with tools rewritten by our operators, the task-weighted change is $+1.1$ and -0.7 points; retail under the transaction rewrite is the only cell

Table 14: Transfer of training gains on Qwen3-4B (Appendix G.5). Changes are in exact success relative to the untrained model; in the last row, pairs of values are two seeds of SFT followed by RL.

held out	evaluated on	result
Domains	transaction and schema discovery, four held-out domains	SFT keeps 43% (transaction) and 67% (schema discovery) of its gain; RL keeps 65% and 71%
Schema changes (SFT)	new combination / stronger version / new operator / class-level structure	$+8.8$ / $+11.2$ / -2.4 / $+1.2$ points (trained changes $+14.6$)
Benchmarks	τ^2-bench task-weighted / τ^2-bench retail, transaction rewrite / BFCL state-independent	$+1.1$, -0.7 / $+10.5$, $+3.5$ / $+0.6$, $+1.5$ points

whose paired interval excludes zero ($+10.5$ and $+3.5$). On BFCL, state-independent turns change by $+0.6$ and $+1.5$ points and state-dependent turns by -0.8 and -0.7.

G.6 Additional training studies

Two further studies test whether training recipes found on the synthetic tasks hold on other targets. Neither changes the conclusions of §8; both show that the effect of a recipe depends on the target.

Interface factors derived from failures. We isolate three interface factors behind recurring failures on τ^2-bench, each holding the task semantics fixed: whether a policy constraint is stated in the system prompt or next to the tool it governs; whether an entity is found with one composite lookup or a chain of three lookups; and whether the tool list shows every state transition or only those currently legal. On Qwen3-4B (160 tasks per factor, five realizations each), placing the policy next to the tool has no effect (0.0 points), chained lookups lower exact success by 86.1 points ($[-88.5, -83.6]$), and showing only legal transitions raises it by 8.8 points ($[6.5, 11.0]$). Training on the two effective factors (GRPO, three 2×2 designs, three seeds, 36 runs) shows that factors do not combine additively: covering the relevant task semantics and exposing the new interface interact negatively (-17.6 points, $[-27.0, -9.1]$), and none of the corresponding effects on τ^2-bench excludes zero.

Training for an airline domain. We train Qwen3-4B on a generated airline environment (48 runs across three designs: training-data properties, interface choices, and training order) and evaluate on held-out generated airline tasks and on τ^2-bench airline. Multi-turn training data lower held-out success by 14.5 points ($[-22.6, -5.2]$) and deeper queries raise it by 4.2 points ($[0.5, 7.9]$); no interface choice has an effect that survives Holm correction on either target. The best training order reverses between targets: SFT alone beats RL from the untrained model by 39.6 points ($[22.5, 52.7]$) on the generated tasks but trails it by 9.7 points ($[-19.5, -3.1]$) on τ^2-bench airline, where SFT followed by RL beats SFT alone by 10.8 points ($[1.5, 19.0]$). A training recipe should therefore be validated on the target distribution.