MECHASKILL: AUTOMATIC COMPILATION AND EVOLUTION OF MODULAR SKILLS FOR TEXT-TO-SQL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Strong Text-to-SQL workflows offer complementary expertise, but their capabilities remain tied to workflow-specific execution logic, making them difficult to understand, reuse, and improve. Combining these strengths requires preserving the conditions under which each capability works and assessing how capabilities affect one another. We introduce MECHASKILL, a framework that compiles workflow expertise into a modular skill for a single agent and supports module-level evolution. It recovers modules with compatible interfaces from code, documentation, and execution traces, preserving the instructions, tools, and dependencies needed for reuse. We use complementary question coverage to guide composition and evaluate the complete skill to select modules and determine when they should run. We trace failures to individual modules to guide targeted revisions, then evaluate their effect on the complete skill. We evaluate MECHASKILL alongside models, agents, full workflows, and distilled skills on BIRD-147, SPIDER2-135, and BIZ-64. In our primary evaluation, MECHASKILL achieves the highest execution accuracy among the compared systems while substantially reducing inference cost relative to the strongest workflow baseline on each dataset.

1 INTRODUCTION

Strong Text-to-SQL systems use carefully designed workflows to coordinate reasoning, database access, and execution feedback (Pourreza & Rafiei, 2023; Talaei et al., 2024; Pourreza et al., 2025). The resulting procedures encode practical knowledge about how to solve the task. Different workflows offer complementary expertise, yet their capabilities are embedded in each source's execution logic, making them difficult to understand, reuse, and improve. A capability can span prompts and tools and depend on intermediate states, obscuring both its contribution and the decisions that need to change when it fails. This raises the question of how to turn workflow expertise into a reusable skill whose capabilities can be understood, combined across sources, and improved through targeted revisions.

Packaging instructions, tools, and resources as skills provides a practical basis for reuse (Anthropic, 2025). An agent can inherit a source workflow's resources and execution guidance through skill distillation (Xu et al., 2026). Modular search can explore combinations and revisions of components when their categories and interfaces are defined in advance (Shang et al., 2025). Composition across independently designed workflows introduces an additional problem: the units of reuse and their interfaces must be recovered before their joint behavior can be optimized.

The difficulty extends beyond making inputs and outputs compatible. A capability can depend on decisions made elsewhere in its source workflow, and changing those decisions can alter its usefulness. Once capabilities are combined, adding or revising one can affect both the answers produced and the computation performed by the others. A reusable skill therefore needs to preserve the conditions under which its capabilities work and assess their interactions during both composition and later revision.

We propose MECHASKILL to compile expertise from several workflows into a skill for one agent and improve it through changes to individual modules (Figure 1). The framework recovers modules from code, documentation, and execution traces, retaining their instructions, tools, and state

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.

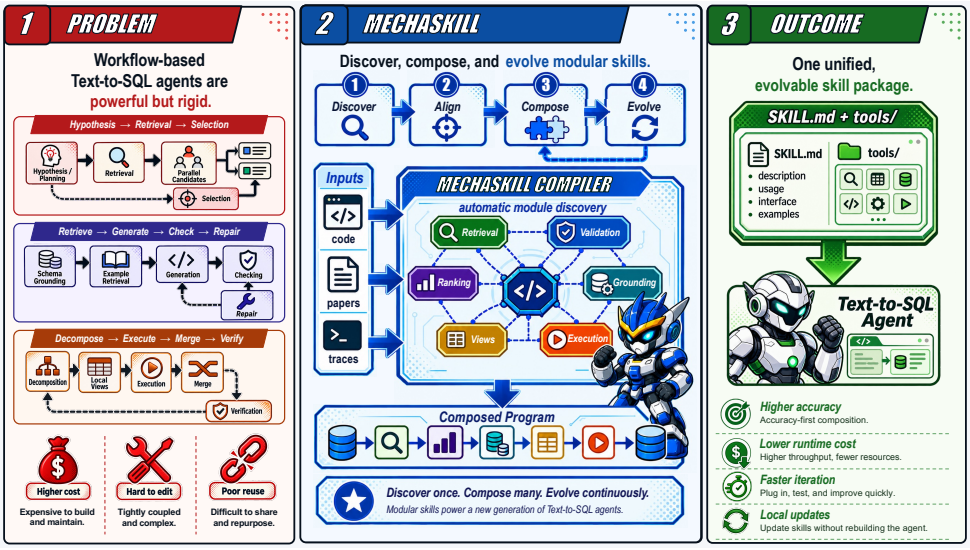


Figure 1: Overview of MECHASKILL. Discover and Align recover and organize reusable modules from heterogeneous Text-to-SQL workflows. Compose integrates complementary modules into a unified skill package for a single agent. Evolve traces errors to modules and evaluates each proposed update on the complete skill.

dependencies and defining compatible interfaces for reuse. This representation makes the role and dependencies of each capability explicit, so capabilities from different sources can be combined, inspected, and revised within one skill. We develop this approach using APEX-SQL (Cao et al., 2026), DeepEye-SQL (Li et al., 2025), and AV-SQL (Pham et al., 2026) as source workflows.

During construction, MECHASKILL uses the questions solved by each source to guide composition, then evaluates the complete skill to choose modules and their invocation conditions. A module is retained for what it contributes to the shared procedure, and additional computation can be reserved for questions that need it. The module structure also lets us trace failures to the relevant decisions and revise individual modules, with each change judged by the resulting skill's performance. Composition and evolution thus use the same unit of modification while judging success at the level of the agent's final answer.

Our evaluation on BIRD-147, SPIDER2-135, and BIZ-64 compares MECHASKILL with model and agent baselines, complete workflows, and distilled skills. In the primary evaluation, MECHASKILL achieves the highest execution accuracy among the compared systems, with an average inference-cost reduction of over 80% relative to the strongest workflow baseline on each dataset. Execution profiles and diagnostic skill comparisons examine tool use, candidate selection, and the effects of local rule changes. We further evaluate the unchanged skill with another language model to test whether the composed expertise transfers beyond the model used to develop it.

The main contributions of this paper are summarized as follows:

- We propose MECHASKILL, a skill framework that discovers reusable modules from multiple Text-to-SQL workflows, composes their complementary capabilities into a unified agent skill, and supports targeted module-level evolution.
- We analyze tool-use trajectories, candidate selection, and local skill revisions, and document module interfaces and invocation rules through executable implementations and prompt templates.
- We evaluate MECHASKILL alongside model and agent baselines, full workflows, and distilled skills on public and private datasets, demonstrating accuracy–cost improvements over strong workflows and cross-model transfer of the unchanged skill.

2 Related Work

Expertise encoded in Text-to-SQL workflows. Decomposition, retrieval, grounding, and execution feedback improve SQL generation (Gao et al., 2023; Pourreza & Rafiei, 2023; Talaei et al., 2024). Diverse reasoning paths can be combined with downstream selection (Pourreza et al., 2025), while inspection and refinement tools support iterative SQL construction (Wang et al., 2024). Agentic exploration provides additional grounding for these decisions (Cao et al., 2026). Our implementation draws on APEX-SQL, DeepEye-SQL, and AV-SQL, recovering modules that can be coordinated across these sources.

Workflow transfer and reusable skills. Reusable skills package instructions, tools, and resources (Anthropic, 2025), and recent work studies when a skill-equipped agent can replace multi-agent execution (Li, 2026). Skill distillation separates capability resources from pipeline guidance and uses Behavior-Outcome Freedom to decide whether source guidance should be inherited (Xu et al., 2026). This adaptive inheritance starts from an existing source workflow; MechaSkill connects capabilities across independently designed sources and revises their joint behavior. Complementary skills can also be optimized for an ensemble (Zhu et al., 2026); our source procedures operate within one agent. Skill libraries and graphs organize skills for reuse (Wang et al., 2023; Li et al., 2026).

Modular agent search and local optimization. Modular agent search explores combinations and revisions of planning, reasoning, tool-use, and memory components with predefined interfaces (Shang et al., 2025). MechaSkill recovers module boundaries and input/output requirements from heterogeneous implementations before searching their composition. Automated search also operates over agent programs and workflows (Hu et al., 2024; Zhang et al., 2024), while skill-level optimization covers skill selection, use, and distillation (Shi et al., 2026). Evaluation feedback can guide revisions to behavior or textual variables (Shinn et al., 2023; Zhao et al., 2023; Yang et al., 2024; Yuksekgonul et al., 2024). Our updates target modules linked to source evidence. Intervention and replay identify what to edit, and evaluation of the full skill tests how the change affects the remaining modules.

3 From Workflow Expertise to a Composed Skill

APEX-SQL, DeepEye-SQL, and AV-SQL supply the recovered capabilities in Figure 2. The two public-benchmark skill packages use different subsets of these capabilities. Colors identify the source workflow; red marks changes made during composition and evolution.

AV-SQL and APEX share a generation process. For a request with dependent ranking stages, AV-SQL grounds the relevant relations and validates views for planning. APEX-derived guidance requires materializing the first ranking boundary before computing downstream aggregates, and preserving entity identity when many-to-many joins duplicate rows. The generator uses the checked context, logical plan, and applicable rules together. The shared logical plan connects the two source modules while preserving the existing generator interface and generation stage. Appendix C.2 gives the interfaces and skill instructions.

DeepEye supplies the BIRD-147 computation. The BIRD-147 skill retains DeepEye's eight-candidate generation, SQL checks and revisions, and confidence-based fallback selection. An adapted APEX-style voting rule can first select candidates with the same execution result when enough agree. A diagnostic probe conditionally adds a compact relation plan before generation. These modules act before and after DeepEye's retained computation; Section 5.4 analyzes how recorded changes affect selection and resource use.

A module includes its use conditions. Given workflows $\mathcal{W} = \{W_1, \ldots, W_K\}$ and their code, documentation, and traces, MechaSkill constructs a package $S = (\mathcal{M}, \Pi, \mathcal{F})$. Here $\mathcal{M}$ contains the recovered modules, Π is a unified natural-language protocol in `SKILL.md`, and $\mathcal{F}$ contains callable tools. Each module is represented as

$$m = (I_m, O_m, \pi_m, F_m, d_m, p_m), \tag{1}$$

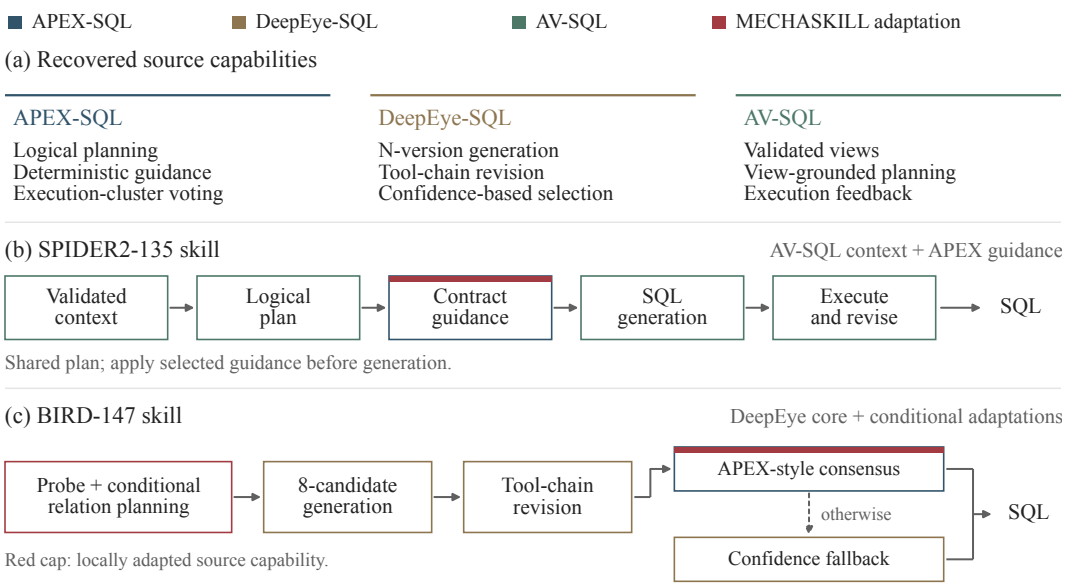


Figure 2: Module sources in two MECHASKILL packages. Colors identify recovered APEX, DeepEye, and AV-SQL capabilities; red marks composition and evolution adaptations. A red cap indicates a locally adapted source capability. (a) Source pool. (b) AV-SQL context with APEX guidance. (c) DeepEye computation with conditional planning and an APEX-style consensus guard.

where I_m and O_m specify input and output states, π_m specifies the trigger and usage procedure, F_m contains executable support, d_m records dependencies, and p_m links the module to source evidence. For the guidance module, the question and logical plan are inputs, selected rules are outputs, and the trigger determines whether guidance is needed. Several modules may share a tool or specification file; edits target the module's state transformation and use conditions.

The package runs as one editable procedure. Construction, composition, and evolution occur offline. At inference time, one agent maps a question, schema, and optional evidence to one final SQL query. Modules share observations and follow explicit dependencies and triggers, allowing module edits and reuse with another language model.

4 MECHASKILL

We construct the skills in Figure 2 by recovering state transformations, connecting their interfaces, and evaluating the joint procedure. The full procedure is given in Algorithm 1 in Appendix B.

4.1 RECOVERING CAPABILITY BOUNDARIES FROM SOURCE EVIDENCE

Connect operations to the states they change. For each workflow, we build a program graph from code, documentation, and execution traces on training examples. Code supplies operations, interfaces, and executable checks; documentation describes their roles; traces show each operation's inputs and outputs. Nodes represent operations and stored states. Edges encode data, control, validation, and repair dependencies, with producer–consumer links following program-dependence representations (Ferrante et al., 1987). Each node records source locations and trace segments for checking that a proposed boundary preserves the original behavior.

Recover a reusable transformation. Discovery groups trace operations by signature and proposes boundaries with stable inputs and outputs, serializable states, and outputs used by later steps. A recurring candidate must appear in at least two questions and two databases. Cached boundary states supply replay inputs, and source matching locates the implementation. Operations that only forward or reformat data are excluded. A language model uses this evidence to keep, merge, split,

or reject each candidate. A module may combine behavior distributed across functions, or a source component may yield several distinct modules.

The retained boundary is compiled into instructions, a tool, or both. Instructions specify activation, actions, observation handling, and stopping conditions; executable support preserves database checks, retrieval functions, dependencies, and required state conversions. AV-SQL's view construction becomes a procedure that proposes a task-specific CTE, checks it through `validate_views`, and exposes checked context to planning. Read-only and identifier checks remain executable. APEX's guidance stage contributes rules through a deterministic question-and-plan interface. DeepEye contributes the candidate-pool, checker, and selection interfaces that retain candidate order and execution observations across these stages.

On a fixed trace, reproducing boundary states and branch decisions in dependency order preserves the final output (Appendix A). This checks that compilation preserves recorded behavior before composition changes module interactions.

4.2 Composing Capabilities at Shared Decision Points

Align module interfaces and conditions for use. We organize modules along a shared task timeline: question understanding, grounding, context and planning, candidate generation, execution and revision, and final selection. The timeline locates related transformations; compatible inputs, outputs, and dependencies determine legal connections. Structural comparisons identify equivalent interfaces, contained operations, and producer–consumer relations. Equivalent alternatives can replace one another, overlapping work can be removed, and dependent operations retain their required order. Triggers specify when an applicable module executes. Joint execution then determines whether the connected behaviors help or conflict.

The composed skill connects AV-SQL context to APEX guidance through the logical plan. After checking views, the agent constructs a plan from the grounded schema and question. The guidance module takes this plan and the question, then returns applicable rules, such as materializing dependent ranking stages and preventing duplicate counts in many-to-many joins. The existing generator receives context, plan, and rules together. Schema inspection and execution checks are shared, and selecting the guidance requires no model call.

Use complementary coverage to choose what to explore. For source W_k on training set D_{train}, let $C_k = \{x \in D_{\text{train}} : W_k \text{ solves } x\}$. Exclusive successes suggest useful source combinations, and their union defines a training-set source-output oracle:

$$A_{\text{oracle}} = \frac{|\bigcup_k C_k|}{|D_{\text{train}}|}. \tag{2}$$

Training-set coverage guides composition proposals; we score each complete skill on an independent dev set G. Correctness labels are used offline, while inference uses the question, database evidence, and module triggers. Composition can produce answers absent from fixed source outputs by changing the procedure before generation (Appendix A).

Select the complete procedure. Starting from an executable package, search proposes additions, removals, substitutions, merges, valid reorderings, and refinements to triggers, usage instructions, or tool parameters. Static checks enforce interfaces and dependencies before execution. Compared candidates use the same dev questions, evaluator, model settings, and execution limits. Dev execution accuracy ranks first; exact ties are broken by recorded resource use and then package complexity. The search records each parent, edit, score, and decision, caches evaluated configurations, and stops at its evaluation budget or when no valid neighboring configuration improves the score. Appendix B specifies the ordering and search record.

4.3 Revising Decisions within the Composed Skill

Locate the boundary responsible for a failed decision. Attribution traces errors on training examples back through the states used by later steps, checking for omitted evidence, unmet input or output requirements, and choices that discard a correct candidate. We record the relevant module's

input, output, source references, and the later decision that a revision should change. A target can be an invocation rule as well as a tool implementation.

We hold recorded upstream states fixed and substitute a cached valid output, a prior output, or a compatible alternative at the boundary; a contribution can also be removed when dependencies permit. We rerun later steps under the same settings. If the answer is corrected across repeated replays, the boundary is a candidate for revision. For selection, this tests whether a different choice among available candidates can correct the answer before changing candidate generation.

Edit locally and evaluate the resulting composition. The edit changes the identified calling condition, procedure, handling of observations, or tool implementation while retaining unrelated modules. The adapted APEX-style guard protects a unique leading cluster containing at least four of eight executable candidates; otherwise it uses the retained DeepEye selector. The edit changes a decision condition while preserving the candidate interface. Section 5.4 analyzes recorded selection and planner revisions.

Let $\widehat{S}_t = S_t[m \leftarrow \widehat{m}]$ be the updated package. An acceptance check Pass_G combines structural validity with the accuracy-first comparison on the independent dev set G:

$$S_{t+1} = \begin{cases} \widehat{S}_t, & \mathrm{Pass}_G(\widehat{S}_t, S_t), \\ S_t, & \text{otherwise.} \end{cases} \tag{3}$$

Dev-set evaluation of the full skill records corrected answers, new errors, and resource use. Accuracy improvements are retained; exact ties use resource use and complexity, and a separate efficiency phase sets a minimum accuracy requirement. Search resumes from the accepted package until its stopping condition holds. Appendix B.3 specifies the allowed edits and repeated execution checks, and Appendix A establishes nondecreasing accepted accuracy on G under this acceptance rule.

5 Experiments

5.1 Experimental Setting

Datasets and splits. We evaluate frozen skills on BIRD-147 (the AdaSkill subset of official BIRD dev) (Li et al., 2023; Xu et al., 2026), SPIDER2-135 (native SQLite) (Lei et al., 2025), and BIZ-64 (private Chinese business questions). Development data comprise 1,188 cleaned BIRD train/dev examples, excluding BIRD-147 and its error-analysis records; 2,000 selected SPIDER 1.0 training examples (Yu et al., 2018); and 300 private examples. We split these data 4:1 for training and internal dev selection. Appendix B.5 gives the dataset and split details.

Compared methods. The bare model generates SQL directly; the base agent adds database inspection and SQL execution. APEX-SQL (Cao et al., 2026) uses planning and execution voting; DeepEye-SQL (Li et al., 2025) combines diverse candidates, revision, and confidence-based selection; AV-SQL (Pham et al., 2026) grounds generation in validated views. We evaluate each full workflow, its single-source distilled skill, and MechaSkill, which composes and evolves modules from this source pool. APEX replaces its unavailable reward model with execution voting and a model tie-break.

Models and metrics. We construct and evolve skills with Qwen3.6-27B, then evaluate the same skills with Qwen3.5-35B-A3B without further changes. Each system submits one SQL query, scored by execution accuracy (EX). With N questions, batch time T, and g GPUs, Cost is $gT/(3600N)$ GPU-hours/question and Time is T/N seconds/question. For 27B, resource measurements use four B200 GPUs with method-specific concurrency; Time measures batch throughput. Appendix D details the settings and measurement runs.

5.2 Accuracy and Cost of Reusing Workflow Expertise

MechaSkill achieves the highest 27B EX on all three datasets, outperforming both the full workflows and their individually distilled skills (Table 1). Combining source capabilities improves the accuracy available from any single distilled skill, including on the private business questions.

Table 1: 27B results: EX (%), Cost (10^{-3} GPU-h/question), and Time (batch s/question).

	BIRD-147			SPIDER2-135			BIZ-64		
Method	EX ↑	Cost ↓	Time ↓	EX ↑	Cost ↓	Time ↓	EX ↑	Cost ↓	Time ↓
Bare model	59.9	2.12	1.91	31.9	4.50	4.05	46.9	5.31	4.78
Base agent	63.3	4.07	3.66	29.6	15.29	13.76	51.6	13.51	12.16
APEX-SQL	64.6	91.81	82.63	51.9	47.84	43.05	65.6	80.82	72.74
DeepEye-SQL	70.7	352.29	317.06	49.6	570.19	513.17	68.8	676.56	608.90
AV-SQL	61.9	49.93	44.94	51.1	78.07	70.27	62.5	124.75	112.28
APEX distilled skill	63.9	4.61	4.15	44.4	9.53	8.58	60.9	12.94	11.65
DeepEye distilled skill	64.6	5.52	4.97	51.1	11.87	10.68	70.3	15.88	14.29
AV-SQL distilled skill	63.9	4.62	4.16	50.4	8.77	7.90	64.1	8.27	7.44
MECHASKILL	**71.4**	61.50	55.35	**55.6**	9.96	8.96	**71.9**	17.53	15.78

MECHASKILL reduces inference cost by an average of 86.4% relative to the strongest workflow on each dataset while improving accuracy (Figure 3). On SPIDER2-135 and BIZ-64, it improves both accuracy and cost over every source workflow. Individual distilled skills remain useful at lower budgets; composition extends the accuracy–cost frontier to higher accuracy.

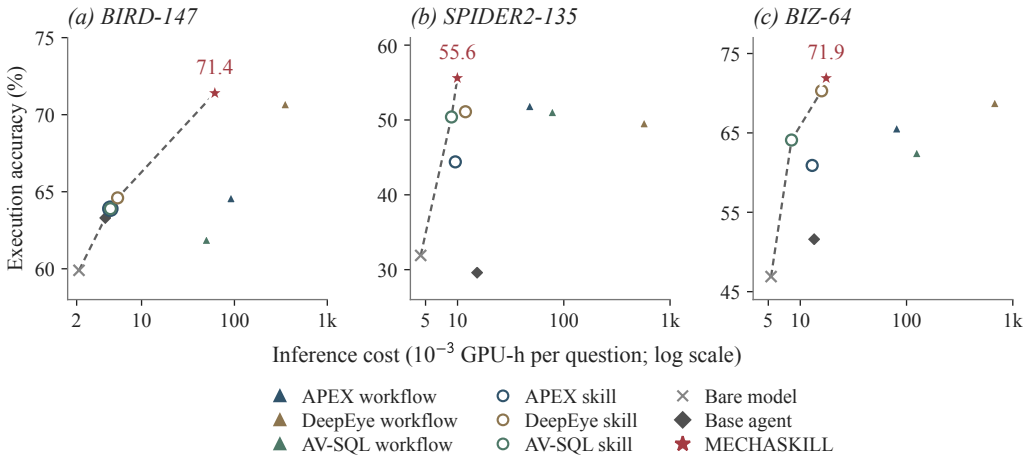


Figure 3: Accuracy–cost comparison with 27B. Markers distinguish system types; dashed lines connect nondominated systems. Cost values are taken from Table 1.

5.3 EXECUTION BEHAVIOR OF COMPOSED SKILLS

We measure both agent turns that invoke tools and total model calls, including those made inside tools (Figure 4). MECHASKILL reuses the source capabilities in Figure 2 with fewer model requests than the full APEX and DeepEye workflows on both datasets (Appendix D.3).

An initial candidate check on BIRD-147 determines whether relation planning or further selection is needed. The planner runs on about a quarter of questions, and the selector on fewer still. Input and output tokens also fall relative to DeepEye (Appendices D.6 and D.7).

Generation and selection must be assessed together. On the 16-question diagnostic panel, configurations with complete candidate coverage still differ in final accuracy: selection can discard a correct candidate (Appendix D.4).

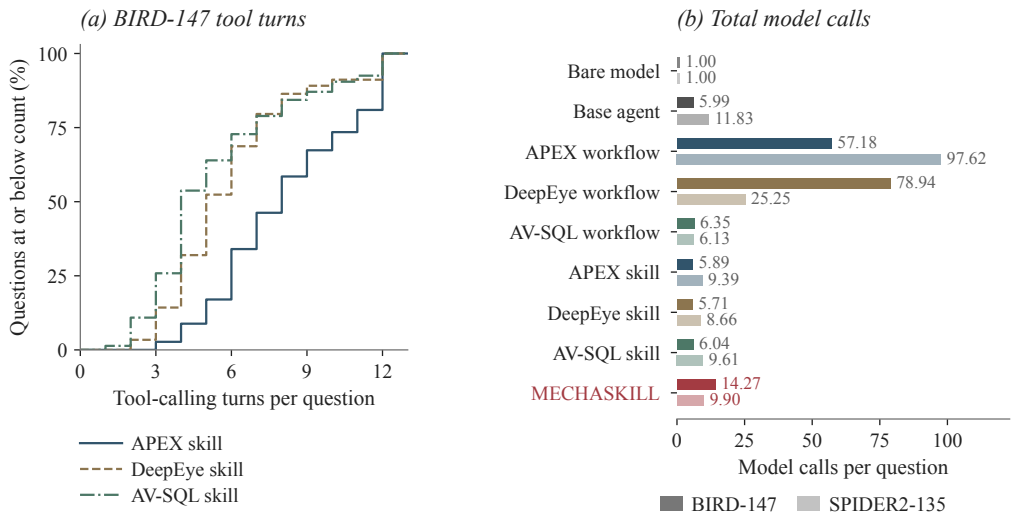


Figure 4: Tool-use trajectories and model requests with 27B. (a) Source-skill tool-turn distributions on BIRD-147. (b) Mean total model requests, including calls inside tools, on BIRD-147 (darker) and SPIDER2-135 (lighter). Cohorts are detailed in Appendix D.6.

5.4 How Skill Edits Change Decisions

A post-evaluation replay of an earlier BIRD-147 skill shows how a selection-rule edit changes the final SQL (Figure 5). For a request to find the gas station with the highest revenue, the selector chooses `SUM(Amount * Price)`. The revised rule requires explicit support for multiplication and allows a direct aggregate even when its column name is absent from the request. Replaying selection on the same candidate pool chooses `SUM(Price)` and returns the correct station. Generation and execution remain fixed; the edit changes how candidates are judged (Appendix D.8).

Question: Which gas station has the highest amount of revenue?

(a) Historical selection edit — BIRD-147 · question 1527

Do not reconstruct Amount * Price or a similar formula without explicit support.

```
- if unsupported_factors and direct_metric_alternatives:
+ if not explicit_formula and direct_metric_alternatives:
```

(b) Final SQL from the same candidate pool

Before

```
SELECT GasStationID
FROM transactions_1k
GROUP BY GasStationID
ORDER BY SUM(Amount * Price) DESC
LIMIT 1
```

GasStationID = 3250 · Incorrect

After

```
SELECT g.GasStationID
FROM gasstations g
JOIN transactions_1k t
  ON g.GasStationID =
     t.GasStationID
GROUP BY g.GasStationID
ORDER BY SUM(t.Price) DESC
LIMIT 1
```

GasStationID = 4347 · Correct

Figure 5: Selection-rule replay on an earlier BIRD-147 skill. Holding the 11-candidate pool fixed, the revised rule replaces an unsupported product with a direct aggregate.

Across archived diagnostic variants on both datasets, we count both corrected answers and new errors on previously solved questions (Figure 6).

On BIRD-147, invoking the planner only when candidates disagree on table relations or row grain preserves accuracy while reducing planner invocations and new errors. Selective planning and the

consensus rule (Section 4.3) each add two correct answers alone; together they add five. This comparison captures an interaction between planning and selection beyond their individual gains.

Repeated runs on SPIDER2-135 show why these edits must be evaluated in the complete skill. The calendar-year rule improves accuracy in one run and reduces it in a repeat, whereas the session-endpoint rule retains a gain in both. Correcting some answers can introduce errors on other questions. Each skill edit therefore needs to be checked on previously solved questions (Appendix D.4).

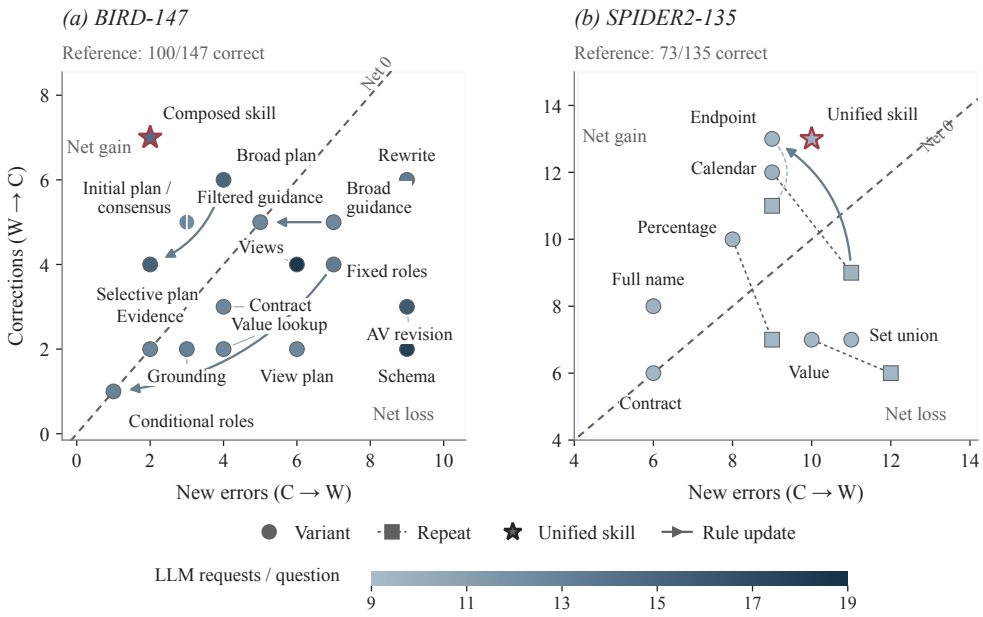


Figure 6: Corrected answers and new errors across archived skill edits with 27B. Reference skills solve 100/147 questions on BIRD-147 and 73/135 on SPIDER2-135. Stars mark the composed or unified skills; arrows connect rule updates; squares and dotted links show repeated runs. Color gives mean model requests. Panel-specific axis ranges preserve readability.

5.5 Skill Transfer and Generalization Across Models

Module composition, instructions, and triggers developed with 27B are reused unchanged with 35B-A3B, without further selection or revision using the new model.

On both public benchmarks, the transferred skill remains more accurate than the strongest distilled skill and the base agent. DeepEye retains the highest accuracy in this evaluation. The gains over individual source skills thus carry over to a model that did not guide composition or evolution. Appendix D.2 reports the full public and private results and resource measurements.

6 Discussion and Conclusion

MechaSkill recovers workflow procedures with explicit states, dependencies, and invocation rules. In the primary evaluation, the composed skills improve accuracy and cost over each dataset's strongest workflow; gains over distilled skills transfer to another model on both public benchmarks.

The method requires accessible implementations and representative traces; external indexes, models, and services remain dependencies. Repeated dev-set selection risks overfitting. Broader evaluation on unseen databases, controlled module ablations, and tasks with less direct correctness feedback are directions for further study.

AI USE STATEMENT

We used OpenAI Codex to assist with manuscript drafting and language polishing, literature discovery and reference checking, research ideation and execution, and the development of the mathematical proofs in Appendix A. Assistance with research execution included code editing, plotting, and interpretation of experimental results. AI tools were not used to generate synthetic datasets. The authors take responsibility for the final text, references, proofs, and empirical results.

ETHICS STATEMENT

Executing generated SQL can expose sensitive data or impose load on production databases. Our experiments use sandboxed, read-only benchmark databases, enforce statement and timeout guards, and log every execution. A compiled skill can preserve source-system mistakes as well as capabilities, so tracing its sources and reproducing recorded behavior do not guarantee correct answers. Deployments require access control, query review, and domain-specific privacy safeguards.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 define the module interfaces, composition objective, and update rule, with proofs in Appendix A. Appendices B and C provide algorithmic details, data sources and the split protocol, prompt and skill instructions, module schemas, and runtime parameters. Section 5 and Appendix D specify the evaluation settings and report exact result counts and resource measurements. BIZ-64 is private; its construction and evaluation protocol are described in Appendix D.5.

REFERENCES

Anthropic. Equipping agents for the real world with Agent Skills, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Bowen Cao, Weibin Liao, Yushi Sun, Dong Fang, Haitao Li, and Wai Lam. APEX-SQL: Talking to the data via agentic exploration for text-to-SQL. *arXiv preprint arXiv:2602.16720*, 2026. doi: 10.48550/arXiv.2602.16720. URL https://arxiv.org/abs/2602.16720.

Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987. doi: 10.1145/24039.24041. URL https://doi.org/10.1145/24039.24041.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023. doi: 10.48550/arXiv.2308.15363. URL https://arxiv.org/abs/2308.15363.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. doi: 10.48550/arXiv.2408.08435. URL https://arxiv.org/abs/2408.08435.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. SPIDER 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/46c10f6c8ea5aa6f267bcdabcb123f97-Abstract-Conference.html.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. DeepEye-SQL: A software-engineering-inspired text-to-SQL framework. *arXiv preprint arXiv:2510.17586*, 2025. doi: 10.48550/arXiv.2510.17586. URL https://arxiv.org/abs/2510.17586.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*,

volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

Xiaoxiao Li. When single-agent with skills replace multi-agent systems and when they fail. *arXiv preprint arXiv:2601.04748*, 2026. doi: 10.48550/arXiv.2601.04748. URL https://arxiv.org/abs/2601.04748.

Xiaoyuan Li, Moxin Li, Keqin Bao, Yubo Ma, Wenjie Wang, Dayiheng Liu, and Fuli Feng. SkillGraph: Skill-augmented reinforcement learning for agents via evolving skill graphs. *arXiv preprint arXiv:2605.12039*, 2026. doi: 10.48550/arXiv.2605.12039. URL https://arxiv.org/abs/2605.12039.

Minh Tam Pham, Trinh Pham, Tong Chen, Hongzhi Yin, Quoc Viet Hung Nguyen, and Thanh Tam Nguyen. AV-SQL: Decomposing complex text-to-SQL queries with agentic views. *arXiv preprint arXiv:2604.07041*, 2026. doi: 10.48550/arXiv.2604.07041. URL https://arxiv.org/abs/2604.07041.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. *arXiv preprint arXiv:2304.11015*, 2023. doi: 10.48550/arXiv.2304.11015. URL https://arxiv.org/abs/2304.11015.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/974ff7b5bf08dbf9400b5d599a39c77f-Abstract-Conference.html.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0ae94013da7cd459402fd77874e09ee3-Abstract-Conference.html.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026. doi: 10.48550/arXiv.2605.06130. URL https://arxiv.org/abs/2605.06130.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv preprint arXiv:2405.16755*, 2024. doi: 10.48550/arXiv.2405.16755. URL https://arxiv.org/abs/2405.16755.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. doi: 10.48550/arXiv.2305.16291. URL https://arxiv.org/abs/2305.16291.

Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. Tool-assisted agent on SQL inspection and refinement in real-world scenarios. *arXiv preprint arXiv:2408.16991*, 2024. doi: 10.48550/arXiv.2408.16991. URL https://arxiv.org/abs/2408.16991.

Binyan Xu, Dong Fang, Haitao Li, and Kehuan Zhang. From multi-agent to single-agent: When is skill distillation beneficial? *arXiv preprint arXiv:2604.01608*, 2026. doi: 10.48550/arXiv.2604.01608. URL https://arxiv.org/abs/2604.01608.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. SPIDER: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. doi: 10.48550/arXiv.2406.07496. URL https://arxiv.org/abs/2406.07496.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024. doi: 10.48550/arXiv.2410.10762. URL https://arxiv.org/abs/2410.10762.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. doi: 10.48550/arXiv.2308.10144. URL https://arxiv.org/abs/2308.10144.

Jiongli Zhu, Haoquan Guan, Parjanya Prajakta Prashant, Nikki Lijing Kuang, Seyedeh Baharan Khatami, Canwen Xu, Xiaodong Yu, Yingyu Lin, Zhewei Yao, Yuxiong He, and Babak Salimi. Residual skill optimization for text-to-SQL ensembles. *arXiv preprint arXiv:2605.21792*, 2026. doi: 10.48550/arXiv.2605.21792. URL https://arxiv.org/abs/2605.21792.

A Formal Definitions and Proofs

A.1 Execution Accuracy and the Comparison Rule

For a question x and its reference execution result y_x, let $\mathrm{Exec}(S, x)$ denote the result of executing the skill's final SQL under the benchmark evaluator. On a finite set D,

$$A_D(S) = \frac{1}{|D|} \sum_{x \in D} \mathbf{1}[\mathrm{Exec}(S, x) = y_x]. \tag{4}$$

Let $R_D(S)$ be the ordered resource and complexity vector specified in Appendix B. The accuracy-first objective is the lexicographic ordering

$$\Phi_D(S) = \langle A_D(S), -R_D(S) \rangle. \tag{5}$$

The default accuracy tolerance is zero: resource measurements break exact accuracy ties. This ordering requires no weighted trade-off between accuracy and computation.

A.2 Trace-Level Behavioral Preservation

Proposition 1 (Trace-level behavioral fidelity)**.** *Fix a finite source execution trace, its initial state, and its external observations. Suppose each compiled module reproduces its source region's output boundary state and branch decision for every reached boundary input, and execution preserves the dependency order. Then the compiled execution produces the same terminal SQL as the source trace.*

Proof. Unroll the finite trace into a directed acyclic graph of semantic regions, including individual loop iterations when present. Choose a topological ordering $r_1, \ldots, r_T$. Both executions begin with the same initial boundary state. If the predecessors of r_t produce the same boundary outputs, then r_t receives the same inputs. By the premise, its compiled module produces the same output state and branch decision. Thus the same successors are activated with the same inputs. Induction gives equality at every reached terminal state, including the final SQL. This result holds for the fixed observations in the trace. □

A.3 Complementary Coverage and the Selection Oracle

Proposition 2 (Exclusive correctness increases the source-output oracle)**.** *For two fixed source systems with correct sets C_a, C_b on the same nonempty set D, if both $C_a \setminus C_b$ and $C_b \setminus C_a$ are nonempty, then*

$$\frac{|C_a \cup C_b|}{|D|} > \max\left(\frac{|C_a|}{|D|}, \frac{|C_b|}{|D|}\right). \tag{6}$$

Proof. The nonempty set $C_b \setminus C_a$ makes the union strictly larger than C_a, and $C_a \setminus C_b$ makes it strictly larger than C_b. Division by $|D| > 0$ preserves both inequalities. □

This oracle is the accuracy obtainable by a selector that knows which fixed source outputs are correct. It bounds selection from those outputs. A composed generator changes the available outputs and can solve questions outside their union. For a common evaluation set and fixed source checkpoints, we define net oracle-gap recovery as

$$R_{\mathrm{gap}} = \frac{A_D(S) - A_{\text{best source}}}{A_{\text{oracle}} - A_{\text{best source}}}, \tag{7}$$

when the denominator is positive. Reporting which source answers are retained, which new questions are solved, and which previously correct answers become wrong explains how the net gain arises.

A.4 Bounded Search and Accepted Updates

Proposition 3 (Finite search termination)**.** *With a finite evaluation budget and bounded proposal attempts, composition and evolution terminate. If the legal candidate space is finite and states are memoized, exhaustive evaluation additionally terminates after at most one evaluation per legal state.*

Proof. Every evaluation consumes one unit of the finite budget, and invalid proposals consume a bounded proposal allowance. Thus neither an evaluation loop nor an invalid-proposal loop can continue indefinitely. In the finite-space case, memoization limits evaluations to the number of legal states. □

Proposition 4 (Nondecreasing dev accuracy under the acceptance rule)**.** *Fix an independent dev set G, its evaluator and comparison measurements. If each accepted update satisfies $\Phi_G(S_{t+1}) \succeq_{\text{lex}} \Phi_G(S_t)$ with accuracy as its first coordinate, then dev accuracy does not decrease after an accepted update.*

Proof. Lexicographic acceptance cannot decrease the first coordinate. Transitivity gives $A_G(S_u) \geq A_G(S_t)$ for every later accepted state $u > t$. Rejected proposals retain S_t and preserve equality. The conclusion concerns the fixed dev set; accuracy on a new sample is an empirical quantity. □

An efficiency phase with a fixed minimum accuracy requirement follows a different rule: it guarantees that threshold, but does not require every update to improve accuracy.

B Algorithmic and Implementation Details

Algorithm 1 MechaSkill compilation, composition, and local evolution

Require: Workflows $\mathcal{W}$; code, documentation, and training traces E; training set D_{train}; independent dev set G; finite budgets B_c, B_e
Ensure: Unified skill S and versioned decision history $\mathcal{H}$

$\mathcal{M} \leftarrow \varnothing, \mathcal{H} \leftarrow \varnothing$
for all $W_k \in \mathcal{W}$ **do**
  $G_k \leftarrow \textsc{BuildEvidenceGraph}(W_k, E)$
  $C_k \leftarrow \textsc{DiscoverBoundaries}(G_k)$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \textsc{Compile}(C_k)$
end for
$\mathcal{R} \leftarrow \textsc{AlignInterfaces}(\mathcal{M})$
$S \leftarrow \textsc{CoverageGuidedCompose}(\mathcal{M}, \mathcal{R}, D_{\text{train}}, G, B_c)$
while $B_e > 0$ and proposal budget remains **do**
  $F \leftarrow \textsc{CollectFailures}(S, D_{\text{train}})$
  $(m, e) \leftarrow \textsc{CounterfactualAttribute}(F, S)$
  if no supported revision target remains **then**
    break
  end if
  $\widehat{S} \leftarrow \textsc{ApplyBoundedPatch}(S, m, e)$
  $B_e \leftarrow B_e - 1$
  if $\textsc{Valid}(\widehat{S})$ and $\text{Pass}_G(\widehat{S}, S)$ **then**
    Record patch, complete-system scores, and acceptance in $\mathcal{H}$
    $S \leftarrow \widehat{S}$
  else
    Record rejection in $\mathcal{H}$; retain S
  end if
end while
return $S, \mathcal{H}$

B.1 Discovery and Alignment

The compiler retains code locations and training-trace references for each candidate, including inputs, outputs, and later steps that use them. Operations are grouped by signature; recurrence is

checked across at least two questions and two databases. Stored boundary states and stable interfaces identify candidates that can be replayed. The review request allows keep, merge, split, or reject decisions. In the archived MVP1 loader, a split requires a new replayable subgraph to be extracted before review; it is not applied directly to an existing card. Code matching ranks candidate implementations by how closely they match the recorded operations. Documentation helps explain what each recovered operation does.

Alignment compares input/output requirements, state flow, operation type, and position in the task sequence. Structural equivalence denotes compatible interfaces; containment identifies a supported sub-operation; dependency requires an output to be available before its consumer. Behavioral interchangeability and interactions are evaluated by substitution or joint execution. We check both whether modules can be connected and whether their combination improves the complete skill.

B.2 Search State, Scoring, and Fair Comparison

Supported edits are `ADD`, `DROP`, `SWAP`, `MERGE`, trigger refinement, usage refinement, tool-parameter tuning, and reordering within valid timeline constraints. A content hash identifies each candidate state. The log records its parent, operator, implementation diff, evaluation record, and decision; a previously evaluated state can reuse its stored record.

By default, the comparison treats accuracy as tied only when scores are equal. After accuracy, it compares recorded cost (USD when available, otherwise total tokens), wall time, LLM calls, SQL executions, external calls, module count, `SKILL.md` character count, and seed standard deviation, in that order. The reported GPU-hour profile is an evaluation measurement. A dedicated efficiency phase sets a minimum accuracy requirement and requires an improvement in the ordered resource measurements.

Candidate comparisons use the same dev examples, model configuration, temperature, seed, evaluator version, concurrency, turn and token limits, SQL row limits, and timeouts. Training examples drive composition proposals and module revisions; complete-skill accuracy on the independent dev set determines acceptance. Dataset usage is documented in Appendix B.5.

B.3 Testing Module Outputs and Applying Edits

Attribution records an unmet module requirement or a controlled change to its output, such as substituting a cached output, a previous version, or a compatible alternative; removing the module; or rerunning later steps. The controller requires the same effect on later steps in at least two repetitions, tested on both failing and previously solved training questions. A patch targets one or two declared modules and no more than two distinct edit locations.

Allowed files include `SKILL.md`, the tool manifest, module graph, composition configuration, and Python tools. Each edit matches a 1–4096-character fragment exactly once and replaces it with up to 4096 characters; an empty replacement deletes the fragment. The patch is applied to a copied candidate tree and checked against its parent hash; source artifacts and accepted history remain available for rollback. The system records the patched version as awaiting evaluation. It becomes an accepted version only after the dev-set comparison succeeds.

B.4 Versioning and Resource Accounting

Each candidate state is identified by its content hash and linked to its parent, implementation diff, evaluation record, and decision log. Comparisons use the fixed data split and evaluation configuration described above. Offline compilation, index construction, search, and evolution are tracked separately from online inference.

B.5 Development Data and Evaluation Separation

We develop skills using 3,488 examples from the three sources in Table 2, partitioned into training and internal development sets at a 4:1 ratio. Training examples support module discovery, composition, and revision; the internal development set determines which complete-skill updates are accepted. Skills are frozen before final evaluation.

Table 2: Data used for skill development before the internal 4:1 split.

Source	Selected examples
Cleaned BIRD train and official dev, combined (Li et al., 2023)	1,188
Selected SPIDER 1.0 train examples (Yu et al., 2018)	2,000
Private business examples	300
Total	3,488

The BIRD evaluation subset follows AdaSkill (Xu et al., 2026). We exclude its 147 questions and associated error-analysis records before constructing the 1,188-example BIRD development pool. Official BIRD dev denotes the source collection; internal dev denotes the held-out portion used to select skills. SPIDER 1.0 supplies the selected development examples, while SPIDER2-135 is reserved for final evaluation and is not used for skill development. The private data comprise 300 development examples and the 64-question BIZ evaluation set.

B.6 Prompt Templates and Module Schemas

This subsection describes the prompt roles, review requirements, and runtime parameters used by the compiler and inference modules. The structural-planning prompt is reproduced below. Appendix C gives the module schema and composition details, followed by the complete BIRD-147 skill instructions and the three distilled source-skill specifications.

Table 3: Prompt roles and module requirements.

Stage	Prompt or module requirement
Module review	Review graph-derived candidates, respect forbidden inputs, and return a decision with evidence for each candidate.
Relation planning	Produce a five-field structural plan before SQL generation.
Checker-based revision	Revise SQL from a checker suggestion.
Execution-based revision	Revise SQL from execution feedback.
Pairwise selection	Compare two SQL candidates and their execution results.
Structural probe	Generate provisional SQL hypotheses with different plausible join paths or aggregation grains.
Patch validation	Check edit metadata, boundaries, and supporting evidence before evaluation.
Skill execution	Follow the compiled routing, tool-use, and output instructions described in Appendix C.

Discovery groups trace operations by their recorded signature, checks recurrence across at least two questions and two databases, and ranks implementation matches by token overlap and operation-type compatibility. The matcher retains at most 20 code candidates. Its review request asks for a decision and evidence for every candidate; a split requires new replayable boundaries before the loader can apply it.

For evolution, the controller validates the patch identifier, parent-state hash, operation, target modules, exact edits, support and protection cases, expected effect, possible side effects, affected tools or parameters, and attribution. It limits a patch to one or two modules and at most two locations, and requires repeated downstream-effect evidence.

Exact structural-planning prompt. The following prompt is extracted from the final BIRD skill, with line wrapping added for display. The planner's function defaults use temperature 0, a 768-token response budget, and a supplied seed.

```
You are a database query-planning expert. Do NOT write SQL.
Given the complete SQLite schema, question, and evidence, produce a short structural
plan that another model must follow when writing SQL.
Resolve entity roles and join direction explicitly; distinguish row grain from output
grain. Preserve evidence exactly.

SCHEMA:
{database_schema}

QUESTION:
{question}

EVIDENCE:
{evidence}

Return exactly these five labelled lines and no other text:
GRAIN: <one row represents ...; final output represents ...>
TABLE_ROLES: <table=role; ...>
JOINS: <ordered table.column = table.column path, or NONE>
PREDICATES: <filters and value ownership, or NONE>
REDUCE_OUTPUT: <group/aggregate/order/limit/distinct and requested columns>
```

The generator accepts caller-provided messages. Its implementation defaults are eight candidates, temperature 0.8, an 8,192-token response budget, and seed 42. Checker and pairwise-selection defaults are temperature 0. These parameters are exposed by the runtime interfaces and can be set by the caller.

C Prompts and Skill Specifications

C.1 Package Contents and Editable Boundaries

The unified `SKILL.md` contains instructions for each stage, triggers, output requirements, and links to the relevant tools. The tool manifest describes callable interfaces; the module graph and composition record identify module dependencies and selected configurations. A module can use a defined section of a shared specification and refer to shared tools. Source references link that section to code, documentation, and execution evidence.

Module field	Content used by composition and evolution
Inputs and outputs	Required state, produced observations, and fields consumed downstream.
Trigger and procedure	Activation conditions, actions, interpretation, skip conditions, and stopping rules.
Tools and dependencies	Executable functions, resources, and required upstream observations.
Source evidence	Code locations, document references, and recorded boundary states.
Edit boundary	Owned instruction region or tool implementation, parent version, and declared dependent edits.

C.2 Provenance of the Composed Skills

The SPIDER2-135 contract skill combines AV-SQL's validated agentic-view procedure with deterministic guidance derived from APEX-SQL's `select_guidance` stage. The two modules connect through the agent's logical plan:

- **Validated context (AV-SQL).** Inputs are grounded schema and proposed task-specific views. The procedure calls `validate_views`, retains views that pass read-only, execution, and identifier checks, and exposes them with the connecting keys needed for planning.
- **Contract guidance (APEX-SQL).** Inputs are the public question and the agent's logical plan. The deterministic tool `select_contract_guidance` returns rule identifiers and

> instructions for applicable risks, including dependent ranking, ties, comparison domains, and join grain.

After grounding and view validation, the agent forms a logical plan. When the request contains a supported risk, the procedure invokes the guidance tool once before generation. The existing generator uses the validated context, logical plan, and selected rules. The guidance tool requires no additional model call or SQL execution.

The following excerpts are taken from the compiled skill specification:

> For nested Top/Bottom-N tasks, compute the ranking measure at the requested entity/group grain, materialize the boundary, and only then compute downstream averages, totals, attributes, or nested extrema.
>
> When the request asks for a result in each/per group, retain that group key in the final projection and downstream `GROUP BY`; never collapse all partitions into one global result row.

BIRD-147: retained computation and adapted decisions. The BIRD-147 package exposes DeepEye's eight-candidate generation, tool-chain checking and revision, and confidence-gated selection through the retained base runtime. The consensus module implements an APEX-style execution-voting rule adapted to protect a unique leading cluster of at least four of eight executable candidates; the DeepEye selector handles the fallback. Conditional relation planning adapts planning support from the source workflows and uses a diagnostic disagreement trigger. In Figure 2, retained DeepEye modules appear in gold, the APEX-style guard in blue with a red adaptation cap, and conditional planning in red. Appendix C.4 reproduces the final instruction body.

C.3 Historical Selection-Rule Revisions

In an earlier BIRD-147 skill, three selection-rule changes address residual failures (Table 4). That version applies these rules without benchmark question identifiers.

Table 4: Selection-rule revisions for three residual failures in an earlier BIRD-147 skill.

Decision	Revised instruction
Temporal precision	Match the precision supplied by the request to the stored representation; use an appropriate prefix or interval when stored values contain finer precision.
Scalar output shape	Preserve the requested scalar projection without adding auxiliary output columns.
Arithmetic	Apply transformations supported by the question, evidence, or grounded schema semantics; require that support before changing a candidate's formula.

These revisions change selection guidance and narrow checks while retaining the existing inspection and execution interfaces. All three targeted reruns produce a correct final answer.

C.4 COMPLETE MECHASKILL SPECIFICATION FOR BIRD-147

The following listing reproduces the BIRD-147 skill instructions and the referenced timeline contract. It specifies the routing conditions, candidate budget, checker and selection calls, final-output binding, and prohibited information sources.

```
# MECHASKILL

Run exactly two independently routed mechanisms at separate Timeline
stages:

1. Produce exactly three cheap provisional SQL hypotheses before
   main generation.
2. Compare only their table sets, grouping shapes, and aggregate
   sets; never score them with gold.
3. Discard all three probe hypotheses.
4. Call `generate_structural_relation_plan` once only when table
   relations or row grain disagree; skip aggregation-only
   disagreement.
5. Generate a fresh base pool of exactly eight candidates and run
   the frozen checker unchanged.
6. Call `protect_unique_four_of_eight_consensus` only when all eight
   revised candidates execute and one result cluster uniquely leads
   with at least four votes.
7. On a 4-4 tie, fewer than four votes, or incomplete execution, use
   the base selector fallback unchanged.
8. Hard-bind the selected SQL and record both T2 and T5 routing
   decisions separately.

Do not promote a probe SQL into the main pool. Do not use benchmark
IDs, evaluator output, known error lists, few-shot examples, tips,
views, rewrites, reward models, or any third mechanism. Never let
the T5 selector alter the T2 relation-plan trigger.

Read
[references/timeline-contract.md](references/timeline-contract.md)
for the exact ordering contract.

# Timeline contract

`T0/T1 raw context -> T2 provisional SQL x3 -> structural comparison
-> discard probe -> relation/grain-only route -> conditional
relation plan x1 -> T3 base generation x8 -> T4 frozen checker -> T5
unique 4-of-8 consensus protection -> T5b base selector fallback`

The probe is diagnostic only and always discarded. The route may
inspect table, GROUP BY, and aggregate signatures, but
aggregation-only disagreement must skip the planner. It must not
inspect execution correctness, gold SQL, benchmark IDs, or evaluator
results. Main candidate count, seeds, checker, frozen fallback
selector, and final evaluator remain fixed. Record relation-plan
routing and consensus-protection routing independently; neither
stage may override the other stage's trigger.
```

C.5 COMPLETE APEX-SQL DISTILLED SKILL

The following three listings reproduce the complete instruction bodies of the APEX-SQL, DeepEye-SQL, and AV-SQL distilled skills, including their tool-invocation examples. They specify each skill's request contract, stages, conditional calls, and final-answer requirements.

```
# APEX-SQL Compiled

Produce exactly one executable read-only SQLite query. Keep one shared state
through T0-T6; do not run separate agents or submit several answers.

## Output contract

Return one JSON object with `selected_tables`, `selected_columns`, and `sql`
when the harness requests JSON. Otherwise return only the SQL statement. Never
return Gold data, benchmark identifiers, reasoning prose, or multiple
alternatives.

## T0 - Freeze the request contract

Before tools or SQL, record internally:

- requested expressions and their order;
- one-row grain;
- eligible population and filters;
- aggregation and grouping semantics;
- ties, ordering, and limit behavior;
- evidence-specified mappings or formulas.

Do not infer `DISTINCT`, fuzzy matching, a latest snapshot, extrema, or
arbitrary tie-breaking unless the request requires it.

## T1 - Ground schema and values

Call `inspect_schema` when exact columns, keys, types, or join paths are
uncertain. Inspect only likely tables when the supplied schema is large.

Call `search_values` only for high-information literals whose stored value or
owning column is uncertain. Prefer exact matches; use substring evidence only
when the wording actually implies partial matching.

Treat successful inspection as identifier/value evidence, not proof of the
intended semantics. Stop if required identifiers or the database are
unavailable; do not invent them.

## T2 - Build a logical plan and retrieve guidance

Write a short schema-grounded plan that covers projection, joins, predicates,
aggregation, grouping, ties, and ordering. Preserve uncertainty explicitly.

Call `select_guidance(question, evidence, logical_plan, db_schema)`. Apply
selected tips only when their conditions match the frozen request contract.
The tool is deterministic; it does not call a model.

Skip plan elaboration for a direct single-table projection, but still preserve
the request contract and exact identifiers.

## T3 - Generate one candidate set

Generate 2-8 complete SQL candidates inside this same Agent. Candidate 0 must
be the most direct evidence-faithful reading. Add a candidate only when it
tests a real ambiguity in grain, join path, literal form, aggregation, ties,
or decomposition. Do not create cosmetic rewrites.
```

APEX-SQL (continued)

````
Each candidate must:

- use exact grounded identifiers and literals;
- match the requested output count and order;
- preserve entity/event grain and aggregate population;
- avoid unjustified `DISTINCT`, `LIKE`, extrema, extra joins, or formulas;
- be a single read-only SQLite statement.

## T4/T5 - Execute, cluster, and select

Call `execute_and_cluster` with the complete candidate set. It executes
read-only and groups candidates by set-of-rows equivalence while preserving
column count.

- If `status=ok`, use `selected_sql` exactly.
- If `status=needs_semantic_tiebreak`, compare only `tied_representatives`
  against the frozen contract, question, evidence, schema, and grounding. Call
  the tool again with `tiebreak_index` set to the chosen representative. Never
  use a random fallback.
- If `status=no_executable_candidate`, repair the smallest evidenced error and
  make one bounded retry. Stop after that retry if none executes.

Execution success or majority agreement is not semantic correctness. Before
accepting, confirm projection, row grain, filter ownership, aggregate
population, tie behavior, and temporal assumptions.

## T6 - Submit

Submit the tool-selected SQL without rewriting it. Stop once one executable
candidate satisfies the frozen contract. Do not add an unvalidated final edit.

## Tool invocation

The skill root is on `PYTHONPATH`. Invoke tools directly without filesystem
searches:

```bash
python3 - <<'PYEOF'
import json
from tools.apex_runtime import (
    execute_and_cluster, inspect_schema, search_values, select_guidance,
)

db = "/path/to/database.sqlite"
print(json.dumps(
    inspect_schema(db, ["likely_table"]),
    ensure_ascii=False, default=str,
))
print(json.dumps(
    search_values(db, ["uncertain literal"], table_names=["likely_table"]),
    ensure_ascii=False, default=str,
))
guidance = select_guidance(
    "question", "evidence", "logical plan", "schema text",
)
print(json.dumps(guidance, ensure_ascii=False, default=str))
vote = execute_and_cluster(db, ["SELECT ...", "SELECT ..."])
print(json.dumps(vote, ensure_ascii=False, default=str))
PYEOF
```

All tools use Python's standard library, open SQLite in read-only/query-only
mode, enforce time limits, expose errors, and report execution counts and
elapsed time.

Read [references/module-provenance.md](references/module-provenance.md) only
when auditing source evidence, fidelity, or model/data provenance.
````

C.6 COMPLETE DEEPEYE-SQL DISTILLED SKILL

```
# DeepEye-SQL Compiled

Produce exactly one executable read-only SQLite query. Run the stages below as
one stateful Agent; do not invoke the original DeepEye workflow and do not
submit an ensemble.

## Output contract

When the harness requests JSON, return exactly one object with
`selected_tables`, `selected_columns`, and `sql`. Otherwise return only the
SQL. Never expose benchmark IDs, Gold data, hidden evaluator data, reasoning
prose, or multiple final alternatives.

## T0 - Freeze meaning before writing SQL

Record internally the requested expressions and order, one-row grain, eligible
population, filters, aggregation, grouping, tie behavior, ordering, limits,
and any evidence-defined mappings. Do not invent `DISTINCT`, fuzzy matching, a
latest snapshot, an extremum, or arbitrary tie-breaking.

## T1 - Bidirectional schema grounding

Call `inspect_schema` whenever exact names, types, keys, or join paths are
uncertain.

Ground in both directions:

1. Direct link: map every requested output, filter, grouping key, and evidence
   literal to candidate tables and columns.
2. Reverse link: for every selected identifier, explain internally which
   request element requires it; remove identifiers with no owner.
3. Closure: add only keys needed to connect the retained tables. Recheck that
   no request element is left ungrounded.

Stop with an explicit error if the database or a required identifier is
unavailable. Never fabricate a column.

## T2/T3 - Generate eight meaningful versions

Create up to eight complete candidates in one call. Candidate 0 is the most
direct evidence-faithful interpretation. Other candidates may vary only a real
uncertainty: grain, join path, literal form, aggregation population, tie
handling, or decomposition. Do not pad the set with cosmetic rewrites.

Every candidate must be one read-only statement, use exact identifiers,
preserve projection count/order, and avoid unsupported `DISTINCT`, `LIKE`,
extrema, extra joins, or formulas.
```

DeepEye-SQL (continued)

```
## T4 - Tool-chain testing and bounded revision

Call `execute_candidates` on the complete set. Inspect every structured error
and result cluster.

- Repair syntax, identifier, type, cardinality, or empty-result problems only
  when the observation and request justify the change.
- Recheck projection, row grain, filter ownership, aggregate population, null
  behavior, ties, and temporal assumptions.
- Make at most one revised candidate set and execute it again. Do not loop
  until something happens to run.

Execution success is necessary but not sufficient; an executable query can
still encode the wrong meaning.

## T5 - Confidence-aware selection

Use the largest execution-equivalent cluster only when it is unique and its
representative satisfies T0. If `execute_candidates` returns
`needs_semantic_tiebreak`, compare only the tied representatives against the
frozen contract and choose the semantically supported one; do not randomize.
If no candidate executes, stop after the one bounded revision.

## T6 - Submit

Submit one already executed candidate without rewriting it. List only tables
and columns actually used by that SQL.

## Tool invocation

The skill root is on `PYTHONPATH`:

'''bash
python3 - <<'PYEOF'
import json
from tools.runtime import execute_candidates, inspect_schema

db = "/path/to/database.sqlite"
print(json.dumps(
    inspect_schema(db), ensure_ascii=False, default=str,
))
result = execute_candidates(db, ["SELECT ...", "SELECT ..."])
print(json.dumps(result, ensure_ascii=False, default=str))
PYEOF
'''

The tools use only Python's standard library, open SQLite
read-only/query-only, reject writes, enforce per-query deadlines, expose
errors, and report execution counts and elapsed time.

Read [references/module-provenance.md](references/module-provenance.md) only
for fidelity, source, cost, or audit questions.
```

C.7 COMPLETE AV-SQL DISTILLED SKILL

```
# AV-SQL Compiled

Produce exactly one executable read-only SQLite query. Apply the capabilities
below inside one stateful Agent; do not call the AV-SQL workflow and do not
return multiple agents' answers.

## Output contract

When JSON is requested, return exactly one object with `selected_tables`,
`selected_columns`, and `sql`. Otherwise return only the SQL. Never reveal
benchmark identifiers, Gold data, hidden evaluator data, reasoning prose, or
several final alternatives.

## T0 - Freeze the request contract

Record internally the requested expressions and order, result-row grain,
eligible population, filters, aggregation, grouping, tie behavior, ordering,
limits, and evidence-defined formulas or mappings. Do not silently add
`DISTINCT`, fuzzy matching, extrema, latest-row assumptions, or arbitrary
tie-breaking.

## T1/T2 - Rewrite and construct validated agentic views

Rewrite the question and supplied evidence into a concise task statement
without deleting constraints or inventing facts. Call `inspect_schema` for
exact identifiers, types, keys, and join paths.

For a large schema, split likely tables into coherent partitions. For each
useful partition:

1. propose one task-specific CTE and a JSON-like list of the source
   tables/columns it uses;
2. validate the complete CTE query with `validate_views`;
3. keep it only if it is read-only, executes, and the declared identifiers
   agree with the SQL;
4. make at most one correction for an evidenced failure.

Aggregate only validated views and required connecting keys. A view is compact
context, not proof that its semantics answer the question.

## T2 - Plan on the validated context

Build a short plan covering projection, joins, filters, aggregate population,
grouping, ties, ordering, and limits. For every plan step, cite internally the
base identifier or validated view that supports it. If a required field is
absent, return to schema grounding instead of guessing.
```

AV-SQL (continued)

````
## T3 - Generate one primary SQL candidate

Generate one complete SQLite query from the frozen contract, rewritten
request, validated views, and plan. It may inline or reuse the validated CTEs.
Preserve output count/order and row grain; use only grounded identifiers and
literals.

## T4 - Execute and revise once

Call `execute_sql` on the candidate.

- If it fails, revise only the evidenced syntax, identifier, type, or
  cardinality issue.
- If it succeeds, still check projection, row grain, filter ownership,
  aggregate population, null behavior, ties, and temporal assumptions.
- An empty result is evidence to inspect, not permission to weaken filters.

Perform at most one revision and execute the revised SQL. Never hide an error
or loop until a query happens to run.

## T5/T6 - Select and submit

Prefer the revised SQL only when the revision fixed an evidenced defect
without changing the request contract. Submit one already executed query
without an unvalidated final edit. List only tables and columns actually used.

## Tool invocation

The skill root is on `PYTHONPATH`:

```bash
python3 - <<'PYEOF'
import json
from tools.runtime import execute_sql, inspect_schema, validate_views

db = "/path/to/database.sqlite"
print(json.dumps(
    inspect_schema(db), ensure_ascii=False, default=str,
))
print(json.dumps(
    validate_views(db, [{
        "name": "v",
        "sql": "WITH v AS (...) SELECT * FROM v LIMIT 20",
    }]),
    ensure_ascii=False, default=str,
))
print(json.dumps(
    execute_sql(db, "SELECT ..."), ensure_ascii=False, default=str,
))
PYEOF
```

All tools use Python's standard library, open SQLite read-only/query-only,
reject writes, enforce deadlines, expose errors, and report execution counts
and elapsed time.

Read [references/module-provenance.md](references/module-provenance.md) only
for fidelity, source, cost, or audit questions.
````

D Additional Experimental Results

D.1 Evaluation Records and Resource Accounting

From Table 1, MechaSkill achieves 71.4%, 55.6%, and 71.9% EX on BIRD-147, SPIDER2-135, and BIZ-64. Using the exact counts in Table 5, the gains over the strongest full workflow are 0.7, 3.7, and 3.1 percentage points, respectively; gains over the strongest distilled skill are 6.8, 4.4, and 1.6 points. Relative to the strongest workflow on each dataset, cost falls by 82.5%, 79.2%, and 97.4%, averaging 86.4%.

The reported comparison uses the relation-consensus skill on BIRD-147 and the unified contract skill on SPIDER2-135. The BIZ-64 skill is private. Table 5 gives the exact counts underlying the main comparison.

The gains over the strongest full workflow correspond to one, five, and two questions on BIRD-147, SPIDER2-135, and BIZ-64. On BIRD-147, MechaSkill also costs $61.50/5.52 = 11.14$ times as much as the strongest distilled baseline. The comparison therefore measures a tradeoff between accuracy and computation; the small aggregate gains do not alone establish statistical significance.

The BIRD-147 main-table EX and resource measurements come from the accuracy report and separate profiling runs, respectively. The accuracy–cost comparison also includes SPIDER2-135 and BIZ-64 (Figure 3). Resource profiles use local vLLM and four B200 GPUs with method-specific concurrency and candidate budgets. Offline index construction is excluded from inference measurements. For a batch of N questions, elapsed time T, and g allocated GPUs, cost is $gT/(3600N)$ GPU-hours per question and Time is T/N. This time measurement describes throughput; per-request latency requires individual request timestamps. Dollar costs require a deployment-specific GPU price. For BIZ-64, Time is converted from the reported Cost under the same four-GPU allocation: $\mathrm{Time} = 0.9\,\mathrm{Cost}$, with Time in seconds/question and Cost in 10^{-3} GPU-hours/question. Values are rounded to two decimals.

Table 5: Correct counts in the reported 27B comparison.

Method	BIRD-147	SPIDER2-135	BIZ-64
Bare model	88	43	30
Base agent	93	40	33
APEX-SQL	95	70	42
DeepEye-SQL	104	67	44
AV-SQL	91	69	40
APEX distilled skill	94	60	39
DeepEye distilled skill	95	69	45
AV-SQL distilled skill	94	68	41
MechaSkill	105	75	46

The expanded BIRD-147 resource profile counts recorded model requests, including nested calls. Offline compilation, index construction, composition, and evolution are separate from online inference.

Table 6: BIRD-147 resource profiles per question. Input and output are thousands of tokens; calls are recorded model requests.

Method	Calls	Input (k)	Output (k)	Time (s)	GPUh (10^{-3})
Bare model	1.00	3.09	0.38	1.91	2.12
Base agent	5.99	13.38	0.67	3.66	4.07
APEX skill	5.89	41.20	1.83	4.15	4.61
DeepEye skill	5.71	52.00	2.00	4.97	5.52
AV-SQL skill	6.04	50.28	1.34	4.16	4.62
MechaSkill	14.27	69.81	3.95	55.35	61.50
APEX-SQL	57.18	164.93	15.42	82.63	91.81
DeepEye-SQL	78.94	175.24	109.88	317.06	352.29
AV-SQL	6.35	28.55	4.45	44.94	49.93

D.2 TRANSFER FROM 27B TO QWEN3.5-35B-A3B

The MECHASKILL skill is constructed, composed, and evolved using 27B, then frozen for evaluation with 35B-A3B. The target-model evaluation reuses the same skill instructions, module composition, and trigger rules. No additional composition search or skill evolution is performed using 35B-A3B feedback. The results therefore measure direct cross-model transfer of a 27B-developed skill.

We compare the transferred skill with all eight baselines on BIRD-147, SPIDER2-135, and BIZ-64, yielding 27 completed scores (Table 7). The public-benchmark suite uses temperature 0.2 and seed 42. BIRD-147 methods and SPIDER2-135 lightweight methods use four B200 GPUs; SPIDER2-135 full workflows use two.

Table 7: Cross-model transfer to 35B-A3B. MECHASKILL directly reuses the skill developed with 27B, with no target-model iteration. Cells show Correct/Total and EX (%); bold indicates the best result on each dataset.

Method	BIRD-147	SPIDER2-135	BIZ-64
Bare model	81/147 (55.10)	26/135 (19.26)	25/64 (39.06)
Base agent	85/147 (57.82)	33/135 (24.44)	31/64 (48.44)
APEX-SQL	95/147 (64.63)	49/135 (36.30)	40/64 (62.50)
DeepEye-SQL	**99/147 (67.35)**	**66/135 (48.89)**	**42/64 (65.63)**
AV-SQL	83/147 (56.46)	60/135 (44.44)	39/64 (60.94)
APEX distilled skill	92/147 (62.59)	55/135 (40.74)	39/64 (60.94)
DeepEye distilled skill	90/147 (61.22)	49/135 (36.30)	**42/64 (65.63)**
AV-SQL distilled skill	90/147 (61.22)	52/135 (38.52)	40/64 (62.50)
MECHASKILL	97/147 (65.99)	61/135 (45.19)	41/64 (64.06)

On BIRD-147, the transferred MECHASKILL skill improves on the best distilled skill by five questions and on the base agent by 12. On SPIDER2-135, the corresponding gains are six and 28. DeepEye leads MECHASKILL by two and five questions. In the BIRD-147 resource records, MECHASKILL uses 2,133 model requests and 18.994 GPU-hours, compared with DeepEye's 12,029 requests and 58.152 GPU-hours. On SPIDER2-135, AV-SQL scores 60/135 (44.44%), compared with MECHASKILL at 61/135. Its complete resource record totals 1,214 model requests and 60.2439 GPU-hours, including the original run and recovery rounds. On both public benchmarks, composition gains over individual source skills transfer to a model that was not used to develop the skill. On BIZ-64, the transferred skill solves 41/64 questions (64.06%), while DeepEye-SQL and its distilled skill tie at 42/64 (65.63%).

D.3 RECOVERED MODULE INVENTORY

Discovery produces nine, eight, and 13 raw candidates from APEX, DeepEye, and AV-SQL. Review consolidates them into six, four, and six capabilities, respectively. The stored cached checks cover 9/9, 8/8, and 13/13 raw candidates. These records describe compiler inventory and cached boundary checks. The reviewed capabilities are listed by source in Table 8.

Table 8: Reviewed compiler inventory by source.

Source	Recovered capabilities
APEX-SQL	Schema grounding and linking; logical plan synthesis; deterministic guidance retrieval; eight-way agentic candidate generation; read-only SQL execution observation; execution-cluster vote and tie-break selection.
DeepEye-SQL	Bidirectional schema grounding; N-version SQL candidate generation; tool-chain SQL testing and revision; confidence-aware execution-cluster selection.
AV-SQL	Question and knowledge rewriting; validated agentic-view grounding; view-grounded query planning; view-grounded SQL generation; read-only SQL execution feedback; execution-guided SQL revision.

D.4 Diagnostic Variants and Local Evolution

The diagnostic analyses in this subsection and Section 5.4 were conducted after the main-table skills were frozen and evaluated; they did not guide the selection of those skills. The 16-question panel was selected for disagreement between sources. The archived evaluations of V4, V5, V6, V7, and V15 cover all 16 panel questions, recording both candidate coverage and final correctness to identify when selection discards a correct SQL candidate (Table 9). The variants alter candidate construction and selection rules; the best score among these configurations is 14/16 for V5.

Table 9: Candidate coverage and final selection on the same 16-question disagreement panel. Each row evaluates a complete configuration.

Configuration	Candidate coverage	Final correct
Second free-form verifier	16	10
Single critic with narrow guards	16	14
Role-based low-temperature generation	10	6
Eight high-temperature candidates to critic	13	7
Layered selector with evidence checks	16	13

The configurations in Table 9 correspond, in row order, to V4, V5, V6, V7, and V15. Candidate coverage counts questions with at least one correct candidate; final correctness counts the submitted answers.

Eight high-temperature candidates contain a correct answer for 13 of the 16 questions but produce only seven correct final answers. Configurations with complete candidate coverage range from ten to 14 correct final answers. This comparison holds the question set fixed while varying candidate construction and selection.

V15 has three remaining errors addressed by the local rules in Table 4. The V17 targeted rerun succeeds on all three. The stored V18 panel summary combines those reruns with 13 previous successes; it is an aggregate of those records. A separate complete BIRD-147 run in the diagnostic archive scores 105/147.

Corrected answers and new errors. We compare 18 BIRD-147 and 12 SPIDER2-135 historical runs against fixed references of 100/147 and 73/135 correct answers (Figure 6). These historical comparisons analyze component and instruction changes; the final evaluation is reported in Table 1. Model requests are counted over each full run and divided by its dataset size. The BIRD-147 records use direct timeline execution, and the SPIDER2-135 records use agent execution. Among the 17 BIRD-147 single-module candidates, four improve net EX without new system errors, three are neutral, and ten decline. The initial planner and consensus-only variants share the coordinate of five corrected answers and three new errors, shown as a split marker.

Even an update that improves overall accuracy can cause errors on previously solved questions (Figure 6): R14 repairs six questions but loses four, whereas the more selective R18 repairs four and loses two. Both score 102/147. The R21 composition combines relation planning and consensus protection; compared with R18, it adds three successes without losing any of R18's successes. Appendix D.8 shows the corresponding trigger change.

The SPIDER2-135 comparisons retain repeated runs of value grounding, percentage guidance, calendar-year handling, and session-endpoint handling (Table 10). Each pair uses the corresponding recorded skill package; the two calendar runs have the same package hash. The historical unified skill solves 76/135, separately from the 75/135 final evaluation in Table 1. A further full-name rerun has an aggregate score of 69/135 but no recovered paired outcomes, so its counts of corrected answers and new errors are left unreported. System errors are counted as failures in all paired comparisons.

The calendar-year contract yields net changes of $+3$ and -2 questions across two runs. The session-endpoint contract records 13 corrected answers against nine new errors, followed by 11 against nine. The historical unified skill repairs 13 questions and loses ten. On BIRD-147, the planning–consensus composition records seven corrected answers and two new errors, gaining five questions over the reference.

Table 10: SPIDER2-135 historical skill edits with 27B, relative to the 73/135 AV reference. LLM/q counts model requests per question; dashes mark unavailable paired outcomes.

Skill content	Run	Corrected	New errors	Correct	LLM/q
Explicit value grounding	First	7	10	70/135	9.33
Explicit value grounding	Repeat	6	12	67/135	9.44
Contract guidance	–	6	6	73/135	9.44
Set-union guidance	–	7	11	69/135	9.52
Percentage contract	First	10	8	75/135	9.53
Percentage contract	Repeat	7	9	71/135	9.53
Full-name projection	First	8	6	75/135	9.79
Full-name projection	Repeat	–	–	69/135	9.73
Calendar-year contract	First	12	9	76/135	9.80
Calendar-year contract	Repeat	9	11	71/135	9.70
Session-endpoint contract	First	13	9	77/135	9.56
Session-endpoint contract	Repeat	11	9	75/135	9.44
Unified skill	–	13	10	76/135	9.81

D.5 BIZ-64: Private Business Evaluation Set

BIZ-64 is a private evaluation set derived from real-world business scenarios and annotated by business domain experts. Both the dataset and its associated skill are proprietary company assets and cannot be publicly released. It contains 64 unique Chinese business questions, each paired with an expert-annotated reference SQL query. It covers advertising delivery, user profiling, and traffic analytics, including exposure and click counts, page-view and unique-user metrics, and comparisons across time windows. The set contains 51 questions from the delivery/profiling evaluation set and 13 delivery case studies. All 64 records have unique identifiers, distinct question text, and nonempty reference SQL.

With 27B, MechaSkill solves 46/64 questions (71.9%), compared with 44/64 (68.8%) for the strongest workflow and 45/64 (70.3%) for the strongest distilled skill (Table 1). Its cost is 17.53 units, compared with 676.56 for DeepEye-SQL, a 97.4% reduction. This combination of accuracy and cost lies on the BIZ-64 Pareto frontier (Figure 3). Time follows the four-GPU conversion described above; the 35B-A3B results are reported in Table 7.

D.6 Tool-Use Trajectories and Model Requests

We measure interaction length and total model computation separately (Figure 4). A *tool-calling turn* is an assistant turn containing at least one top-level tool invocation; the final text-only answer does not count. A *tool call* is one invocation recorded in that turn. Executable operations inside a `bash` call remain inside that invocation. An *agent turn* includes all recorded assistant turns, including the final answer.

The source-skill audit contains 441 complete trajectories: 147 each for APEX, DeepEye, and AV-SQL on identical question IDs. Every trajectory length matches its evaluation checkpoint. Each tool-calling turn in these records contains one tool call, so tool-turn and tool-call counts coincide. The mean, median, and 90th-percentile counts are reported in Table 11. The cumulative distributions in Figure 4(a) show the fraction of questions whose tool-turn count is at or below the horizontal-axis value.

Table 11: Source-skill trajectories on BIRD-147 (27B), 147 questions per method. P90 is the observed tool-turn count covering at least 90% of questions.

	Tool-calling turns			Tool calls	Agent turns
Method	Mean	Median	P90	Mean	Mean
APEX skill	8.11	8	12	8.11	9.11
DeepEye skill	5.92	5	10	5.92	6.92
AV-SQL skill	5.38	4	10	5.38	6.38

We compare all nine systems on both datasets using total model requests (Figure 4b). Model requests include calls made inside tools, with totals divided by 147 for BIRD-147 and 135 for SPIDER2-135. MECHASKILL uses 14.27 and 9.90 requests per question, respectively, compared with 57.18 and 97.62 for APEX and 78.94 and 25.25 for DeepEye. The BIRD-147 values also appear in Table 6. Panel (a) uses the recorded source-skill trajectory cohort; panel (b) uses the resource profiles associated with the main comparison.

A separate MECHASKILL run uses direct timeline execution, recording 147 top-level skill calls, 2,143 internal model calls, and 2,967 SQL executions. Its `total_turns=0` field belongs to that execution mode; it is not a measured zero-turn conversational trajectory. We therefore measure tool interaction from the three BIRD-147 source-skill traces and compare total model requests across all nine systems on both datasets.

D.7 INFERENCE RESOURCES AND CONDITIONAL MODULE USE

We also measure token use and how often each module is called (Figure 7). The MECHASKILL resource profile uses 18.1% of DeepEye's model requests, 3.6% of its generated tokens, and 17.5% of its GPU time. Input and output tokens are counted separately rather than inferred from execution accuracy. These measurements include each system's actual candidate budget and calls made inside tools.

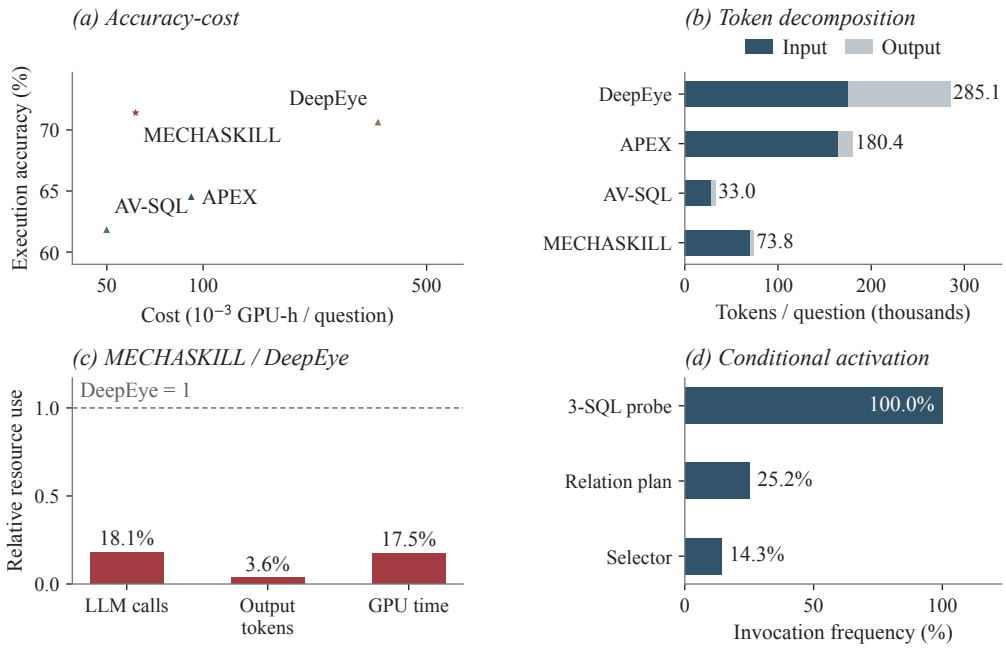


Figure 7: Efficiency on BIRD-147 with 27B. (a) Reported EX and profiled cost. (b) Input/output tokens per question. (c) MECHASKILL resource use relative to DeepEye. (d) Module invocation frequencies in the final-composition audit: the probe, planner, and selector run on 147, 37, and 21 questions, respectively.

Per question, input tokens fall from DeepEye's 175.24k to 69.81k and output tokens from 109.88k to 3.95k. The complete BIRD-147 resource measurements are listed in Table 6.

The final-composition audit invokes the three-candidate probe on every question, the relation planner on 37/147 (25.2%), and the selector on 21/147 (14.3%). Planning and selection are conditional on the evidence produced by earlier modules. These module counters describe the final-composition audit and are not combined with invocation counts from earlier skill variants.

D.8 Local Skill Edits and Their SQL Effects

In the archived diagnostic variants, the planner returns row grain, table roles, joins, predicates, and output reduction. Its revised trigger requires relation or grain disagreement; aggregation-only disagreement no longer activates planning. With the interface unchanged, calls fall from 46 to 35 at 102/147 correct answers (Section 5.4). Adding the consensus rule then corrects three more answers without introducing new errors. In `bird147_0625`, the four-vote execution cluster replaces the incorrect three-vote alternative.

Historical selection-rule repairs. Selection rules interpret shared schema and execution observations. A local guard changes how candidates are judged while preserving the candidate interface and database observations.

We replay the earlier skill on the same 11 executed candidates for BIRD-147 question 1527 when comparing its selection rules (Figure 5). The earlier selector chooses candidate 0, ranking stations by `SUM(Amount * Price)`. The revised rule requires support for multiplication, permits direct aggregates whose column names are absent from the request, and preserves output shape and predicates. Agreement among direct measures leads it to candidate 10, ranked by `SUM(t.Price)`. The returned station changes from 3250 to 4347, matching the reference query.

For the temporal-precision error, the guard selects an existing valid candidate with a normalized time prefix, matching whole seconds to stored fractional seconds without inventing a fractional value. Two accompanying rules preserve the requested scalar output and require evidence for arithmetic transformations. All three targeted corrections succeed on rerun. Table 4 gives the three rules; Appendix C.2 describes guidance shared across source workflows.