ROBOPHD: SELF-IMPROVING TEXT-TO-SQL THROUGH AUTONOMOUS AGENT EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present RoboPhD, a system where AI agents autonomously conduct research to improve Text-to-SQL performance. RoboPhD implements a closed-loop evolution cycle with three coordinated components: a Database Analysis agent that profiles schemas without seeing test questions, a SQL Evaluation agent that generates executable queries conditioned on analysis and evidence, and an Evolution agent that designs new versions of the other two agents based on performance feedback. The Evolution agent employs weighted random sampling from multiple evolutionary strategies—including refinement, research-driven adaptation, and error-focused improvement—ensuring diverse exploration paths. Central to the framework is an ELO-based selection mechanism, which enables survival-of-the-fittest dynamics across competing agents while handling non-transitivity in performance. Starting from a simple baseline, RoboPhD evolves agents through iterative refinement, discovering how to analyze databases to facilitate Text-to-SQL, while also evolving a sophisticated prompt for SQL Evaluation that covers numerous common pitfalls such as proper ordering of columns, when to use COUNT() vs SUM(), etc.-—without being explicitly programmed for them. The full system achieves 68.6% execution accuracy on the BIRD test benchmark, demonstrating that AI can autonomously generate hypotheses, run experiments, and refine techniques.

1 INTRODUCTION

Text-to-SQL, the task of translating natural language questions into executable SQL queries, remains a fundamental challenge in natural language processing with significant real-world applications. Recent advances have shown that large language models (LLMs) can achieve impressive performance on benchmarks like Spider (Yu et al., 2018) and BIRD (Li et al., 2023), but these approaches typically require extensive manual prompt engineering, careful model selection, and domain expertise.

The ability of AI systems to conduct AI research autonomously has been identified as a crucial capability milestone with profound implications for the pace of AI progress (Steinhardt, 2022; Cotra, 2022). Several researchers have argued that AI systems capable of improving themselves or conducting research represent a potential discontinuity in technological development (Bostrom, 2014; Yudkowsky, 2008). Recent work has begun to explore this frontier, with systems demonstrating increasing autonomy in scientific discovery (Romera-Paredes et al., 2024), machine learning research (Lu et al., 2024), and algorithm development (Fawzi et al., 2022).

RoboPhD represents a concrete step toward this milestone by demonstrating that AI systems can autonomously improve their performance on complex tasks through systematic experimentation and iteration. While focused on the specific domain of text-to-SQL, our approach shows that AI agents can effectively conduct the research cycle of hypothesis generation, experimentation, and iterative refinement—core components of the scientific process with minimal inductive bias introduced by humans. We propose a paradigm shift: instead of humans manually optimizing prompts and architectures, we create AI systems that autonomously conduct research to improve themselves.

RoboPhD's architecture consists of three specialized agents that implement domain-independent learning constraints—a design that enables the framework to generalize beyond text-to-SQL to any task decomposable into offline analysis and online inference.

1. **Evolution AI**: Critic agent that analyzes experimental results, identifies patterns in failures and successes, and creates new agent designs for the Offline + Online Agents, combining successful strategies
2. **Offline Analysis AI**: Responsible for gathering useful information to support the online task in a specific problem instance (e.g. a specific database in text-to-SQL); for text-to-SQL this offline analysis examines the database schema and generates text summaries and insights without seeing test questions, bridging the gap between raw schemas and online SQL generation
3. **Online Evaluation AI**: Responsible for implementing the core 'online' inference task. In text-to-SQL, this generates SQL queries based on the offline database analysis and the specific input question, and this evaluates the result/failures to potentially self-correct

This creates a closed-loop research system where AI agents learn from their own experiments, discovering optimization strategies without human intervention in the research process.

This work makes the following key contributions:

- **Autonomous AI Research Framework**: We demonstrate AI agents conducting systematic research on text-to-SQL through a closed-loop evolution cycle, autonomously discovering effective optimization strategies without human intervention
- **ELO-Based Evolutionary Selection**: First application of ELO ratings for evolutionary prompt/agent optimization, effectively handling non-transitivity and asynchronous agent entry
- **Stochastic Multi-Strategy Evolution**: We introduce an evolutionary framework employing weighted random sampling from a portfolio of complementary strategies—including refinement, literature adaptation, error analysis, and pure selection—ensuring diverse exploration paths and robustness to local optima
- **Evolution of a complete Text-to-SQL system**: The framework simultaneously evolves the tool-using analysis agent (its prompt and tools) and the SQL-generation instructions to improve end-to-end performance.
- **Minimal External Knowledge, Domain-Independent Improvement**: The system improves with only minimal author-provided Text-to-SQL knowledge; gains arise from latent LLM capabilities, iterative error-driven learning across iterations, and occasional autonomous literature adaptation

2 Related Work

2.1 Text-to-SQL Systems

The BIRD benchmark (Li et al., 2023) has driven significant advances in Text-to-SQL, with top systems achieving 71-77% accuracy through diverse approaches. AskData (Shkapenyuk et al., 2025) achieves 77.14% using GPT-4o with data analysis agents. CHASE-SQL (Pourreza & Rafiei, 2024) reaches 76.02% through multi-path reasoning and preference optimization. XiYan-SQL (Liu et al., 2024) employs multi-scale few-shot learning for 75.63% accuracy. CSC-SQL (Sheng et al., 2025) uses community-driven schema construction achieving 73.67%. Reasoning-SQL (Pourreza et al., 2025) distills capabilities to smaller models reaching 72.78%. OpenSearch-SQL (Xie et al., 2025), OmniSQL (Li et al., 2025), and GenaSQL (Dönder et al., 2025) all achieve approximately 72% through various architectures. Distillery (Maamari et al., 2024) introduces compositional approaches at 71.83%. CHESS (Talaei et al., 2024) achieves 71.1% using contextual harnessing.

Our work uniquely focuses on *autonomous discovery* of these strategies rather than manual design. While prior systems require human experts to engineer prompts and architectures, RoboPhD discovers optimization techniques autonomously. Note that, with the exception of the BIRD benchmark paper (Li et al., 2023), all of the above papers served as inputs to the evolution agent in the *research-driven* evolution strategy discussed in Section 3.2.4.

Prior work on prompt optimization includes APE (Zhou et al., 2022), OPRO (Yang et al., 2023), DSPy (Khattab et al., 2023), and TextGrad (Yuksekgonul et al., 2025). While these systems optimize

prompts, our approach evolves entire agent architectures including tools, strategies, and multi-phase workflows. Crucially, our system conducts research autonomously without human intervention in the optimization loop. One key aspect of our approach is allowing the Evolutionary agent access to the errors made from previous candidate agents. This is similar to the idea of *text gradients* explored in the TextGrad paper (Yuksekgonul et al., 2025), where the LLM produces a (metaphorical) gradient to describe how to address a particular error. In our system, the error gradients are available for the agent to choose how to incorporate them into the next candidate agent.

Evolutionary algorithms have been applied to neural architecture search (Pham et al., 2019) and hyperparameter optimization (Loshchilov & Hutter, 2016; Jaderberg et al., 2017). We extend this to evolving complete AI agents with natural language specifications and tool usage, creating a fully autonomous research system.

The ELO rating system (Elo, 1978), originally developed for chess, has found recent applications in AI evaluation. The Chatbot Arena (Zheng et al., 2023) uses ELO ratings to create a dynamic leaderboard of LLMs based on human preferences, demonstrating ELO's effectiveness for model comparison. While these systems use ELO for passive evaluation and ranking, we introduce ELO as an active selection mechanism for evolutionary optimization (see §3.2.5 for our implementation).

3 METHOD

3.1 CONCEPTUAL OVERVIEW: THE ACADEMIC RESEARCH METAPHOR

RoboPhD operates like an autonomous doctoral research program, with two key actors and the systems they build:

The Advisor (Evolution Strategies) provides high-level research directions—"focus on fixing error patterns," "see if there are any good ideas in this paper," or "focus on refining your strongest candidate"—without prescribing specific solutions. Like a PhD advisor, it sets strategic direction while granting autonomy to explore and implement (see Appendix C for example prompts).

The Graduate Student (Claude Code[1], wrapping Claude Opus-4.1[2]) interprets this guidance to build complete AI-powered text-to-SQL systems, analogous to prior efforts on the BIRD benchmark. It reads academic papers when directed, analyzes experimental failures to identify patterns, forms hypotheses about improvements, and implements these ideas as new system designs. Each system it creates consists of a Database Analysis Agent with supporting tools and an SQL Evaluation Agent with generation instructions. Like a doctoral student, it maintains context across research iterations, accumulating expertise about which architectural choices and techniques work in which contexts.

This academic structure enables truly autonomous research: the system reads papers from top performers on the BIRD benchmark, learns from its failures, develops novel techniques, and iteratively improves without human intervention. Each iteration represents a complete research cycle: review results → read literature (when applicable) → form hypotheses → build new system → test across databases → analyze outcomes → repeat.

Note that a core design goal of RoboPhD is *domain independence*: we intentionally avoid injecting author-crafted Text-to-SQL techniques. The evolution strategies provide only meta-level research directions (e.g. "focus on prompts," "refine a starting agent," "analyze errors," or "read and adapt a paper"), rather than prescribing scientific content about Text-to-SQL itself. Consequently, measured gains arise primarily from two sources: (i) the latent capabilities of the underlying LLMs and (ii) experience accumulated over many iterations as the Evolution AI learns from outcomes and refines agent designs. External domain information is introduced only when the *research-driven* strategy is sampled, at which point the system autonomously reads a high-performing paper.

Algorithm 1 presents the complete evolution cycle, showing how strategy selection, agent evolution, evaluation, and ELO updates work together to drive continuous improvement.

[1] `https://claude.com/product/claude-code`

[2] `https://www.anthropic.com/claude-opus-4-1-system-card`

Algorithm 1 RoboPhD Evolution Cycle

Input: Initial agent pool $\mathcal{A}$, evolution weights W, iterations N
    // Example: W = {refine: 30%, research: 30%, error: 30%, none: 10%}
Output: Final agent rankings and performance history

// Initial agent pool typically has 1 - 3 agents
$competitors \leftarrow \mathcal{A}$

for $i = 1$ to N **do**
  // Evaluation
  $databases \leftarrow$ RandomSample(BIRD.train, 8)
  for each db in $databases$ **do**
    $questions[db] \leftarrow$ RandomSample(db.questions, 30)
  end for
  for each $agent$ in $competitors$ **do**
    $accuracy[agent] \leftarrow$ EvaluateOnQuestions($agent$, $databases$, $questions$)
  end for

  // ELO Update via Pairwise Decomposition
  for each pair (a, b) in $\{(1, 2), (1, 3), (2, 3)\}$ **do**
    $S \leftarrow 1$ if $accuracy[a] > accuracy[b]$, 0.5 if equal, else 0
    UpdateELO($competitors[a]$, $competitors[b]$, S, $K = 32$)
  end for

  $winner \leftarrow \arg\max_{agent \in competitors} accuracy[agent]$

  Prepare next iteration
  if $i < N$ **then**
    $strategy \leftarrow$ WeightedRandom(W)

    if $strategy \neq$ `none` **then**
      $agent_{new} \leftarrow$ EvolveAgent($winner$, $strategy$, ErrorAnalysis(i))
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{agent_{new}\}$
    end if

    // Tournament Selection for Next Round
    $competitors[1] \leftarrow winner$ {Current winner}
    $competitors[2] \leftarrow agent_{new}$ if exists, else RandomTop($\mathcal{A}$, $2k$)
    $competitors[3] \leftarrow$ RandomTop($\mathcal{A}$, $2k$) {ELO-based selection}
  end if
end for
return $\mathcal{A}$ with final ELO rankings

3.2 Technical Implementation

We now detail how RoboPhD orchestrates three distinct AI model calls in each iteration cycle to achieve autonomous research.

3.2.1 System Overview: The Three-Agent Architecture

RoboPhD's coordinates three specialized agents as described in Section 1, each playing a distinct role in the research cycle. This is illustrated in Figure 1.

Crucially, these agents operate in a carefully orchestrated sequence where the output of one becomes the input to another, creating a closed-loop research system. The SQL evaluation agent never sees the database schema directly—all understanding must flow through the analysis layer, forcing the system to develop effective communication strategies between its components.

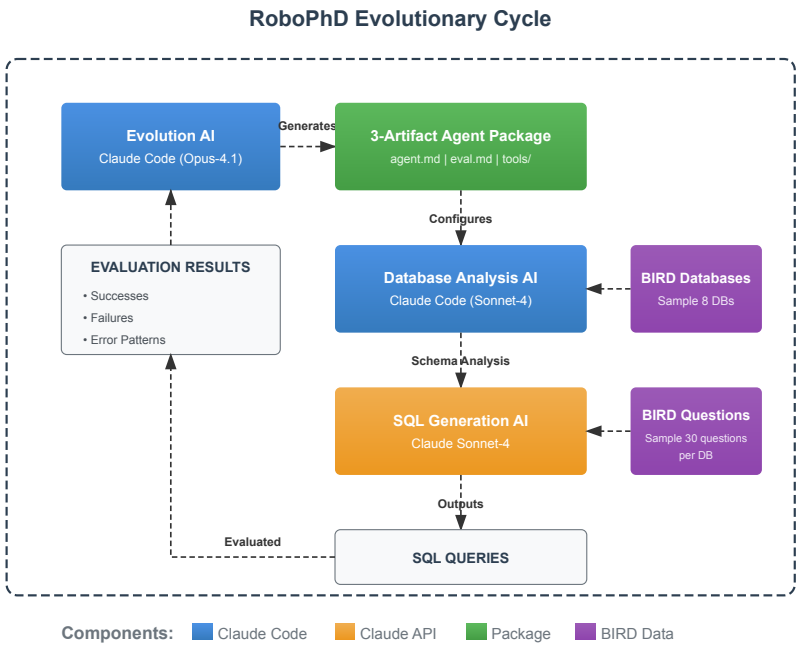

Figure 1: The RoboPhD evolutionary cycle

3.2.2 Offline Database Analysis Phase

The database analysis phase is conducted without seeing the questions that will be asked. This is typical of industrial requirements where it is acceptable to do extensive database analysis offline to facilitate SQL generation, which operates under tight time constraints. This pattern generalizes broadly: separating offline analysis from online inference enables thorough preprocessing to support rapid real-time responses

The Database Analysis agent receives 1) full access to a complete SQLite database, including schema and row data, 2) Analysis instructions from `agent.md`, which were evolved by the Evolution agent, and 3) (Optional) Python scripts for systematic database analysis, also evolved by the Evolution Agent.

The analysis process is executed by a Claude Code agent operating under the instructions of the `agent.md` file. Through evolution, we observe that some agents are designed to rely exclusively on the LLM, in some cases the agent simply orchestrates sophisticated Python tools that perform systematic database analysis, and in some cases the agent combines programmatic tool execution with natural language synthesis, using tools for systematic analysis while adding semantic interpretation and domain understanding.

3.2.3 Online SQL Generation Phase

The SQL generation phase transforms natural language questions into executable SQL through concatenation of the following prompts:

$$\text{Prompt} = \text{DatabaseAnalysis} \oplus \text{EvalInstructions} \oplus \text{Question} \oplus \text{Evidence}$$

where $\oplus$ denotes concatenation. The Question is the natural language question as provided by the user (here, BIRD benchmark). Evidence is provided by BIRD and supplies hints about the domain/schema and clarifications for the question; in a general system this is analogous to complementary semantic annotations about the task/data. DatabaseAnalysis was discussed in Section 3.2.2.

The Role of Eval Instructions: The `eval_instructions.md` file serves as the bridge between database analysis and SQL generation. While the analysis agent examines the database schema, the eval instructions guide how to transform natural language questions into precise SQL queries. These instructions operate at the final online inference step. Successful eval instructions balance comprehensiveness with precision.

Through observing many evolutionary runs, RoboPhD typically evolves instructions that include components such as the following: (1) **Output Constraints**: Instructions regarding clean SQL generation—no markdown, comments, or explanations that could corrupt execution. (2) **Pattern Recognition**: mappings from question types to SQL templates (e.g. "How many X?" → `SELECT COUNT(*)`, "Which Y has the most Z?" → `ORDER BY COUNT(Z) DESC LIMIT 1`). (3) **Column Selection Discipline**: Explicit rules to return only requested columns—a critical requirement where returning extra 'helpful' columns counts as failure in BIRD evaluation. (4) **"Evidence" Supremacy**: when domain-specific evidence is provided by the BIRD benchmark, it overrides all other rules.

Agentic Query Result Validation: text-to-SQL systems face two critical failure modes: syntax errors and semantic errors. Syntax errors prevent execution, while semantic errors produce incorrect results. Prior work typically addresses these challenges through multiple generation attempts. For instance, OpenSearch-SQL (Xie et al., 2025) generates multiple candidate queries and uses self-consistency voting. Our system employs an agentic answer evaluation approach:

Universal Verification: Every generated SQL query undergoes verification where the model reviews its own query execution results in the context of the question and the system prompt with database analysis information. The model then responds with either `CORRECT` to accept those results as the answer or an improved SQL query in light of this new information from the candidate execution result. This self-verification and correction process can iterate up to $k = 2$ times, using progressive temperature ($0.0 \rightarrow 0.2 \rightarrow 0.3$) to encourage exploration on retries. Additionally, if iteration k produces errors or an empty (null) result, the system employs an additional targeted retry with an alert highlighting the SQL error or null result. Note that while our current verification and error recovery mechanisms are fixed, we see opportunity to evolve these components as discussed in Section 5.

3.2.4 Evolution: Creating New Agents

The Evolution AI creates new agents based on comprehensive performance feedback from previous iterations (see Algorithm 1). It receives ELO rankings, accuracy metrics, detailed error analysis, and historical context, then generates three artifacts that define a candidate solution instance: 1) offline analysis agent instructions (`agent.md`); 2) online agent instructions (`eval_instructions.md`); 3) tools as python code. In this system, the authors hand-crafted the evolution strategies themselves. It is future work to consider another meta-learning layer as an agentic critic that improves the evolution strategies themselves (Section 5).

Evolution Strategies: At each epoch of evolution, the system randomly draws from five core strategies with configurable weightings (detailed in Section 4.3; see Appendix C for example prompts):

- **Refinement**: Surgical improvements to an agent from the previous round
- **Research-Driven**: Reads papers from top BIRD leaderboard performers and adapts their techniques
- **Error-Focused**: The evolver is prompted to do extensive error analysis before building an agent
- **Prompt-Focused**: The evolver is prompted to extensively study the generated system prompts of prior agents before building its new agent
- **None**: Skips evolution to gather more data on existing agents

3.2.5 Evaluation and ELO Ranking

The system evaluates three agents simultaneously on identical sets of databases and questions, computing overall accuracy using BIRD's set-based comparison (row order ignored, exact match re-

quired). Each iteration tests agents on randomly selected databases with their associated questions, providing head-to-head comparisons on the exact same tasks.

These three-way competitions are decomposed into three pairwise comparisons. For example, if agents achieve 65%, 62%, and 62% accuracy, we process three head-to-head results: A beats B (1-0), A beats C (1-0), and B ties C (0.5-0.5).

Maintaining an accurate picture of relative agent strength is critical for the evolutionary algorithm. This work introduces ELO as a ranking mechanism for evolutionary algorithms, updating scores using the standard formula ($\Delta_{ELO} = K(S - E)$, $K = 32$). ELO offers key advantages for our setting:

(1) **Asynchronous Competition**: Like chess players who play their first tournament match at different times, agents can join the population at any point while maintaining fair comparisons through persistent ratings. ELO naturally weights unexpected outcomes more strongly than expected outcomes—when a low-rated (or newly-created) agent defeats a high-rated one, both scores change much more than if the higher-rated agent defeats the lower-rated.

(2) **Non-transitivity Handling**: ELO naturally accommodates rock-paper-scissors dynamics where Agent A beats B, B beats C, but C beats A—common when different agents excel on different databases or question types.

(3) **Task Normalization**: Win/loss treatment normalizes across varying difficulty—although accuracy commonly swings from 60% to 75% on different database samples, relative rankings on the same task remain stable.

This competitive framework enables survival-of-the-fittest dynamics while maintaining fairness across agents with different entry points.

3.2.6 Experimental Protocol: Selection and Sampling

Agent Selection: Each iteration tests three agents selected via a priority system: (1) previous winner for continuity, (2) newly evolved agent if created, (3) previously untested agents (mostly relevant at the beginning of a run), and (4) a random selection from the top 2-4 top performers by ELO (we select 1 of 2 normally, but 2 of 4 if using the **None** evolution strategy). This balance ensures thorough testing while maintaining evolutionary pressure.

Database Sampling: To prevent overfitting and encourage generalizable improvements, each iteration evaluates agents on only 8 randomly selected databases from the 63-database training set, with 30 randomly sampled questions per database (BIRD supplies a median of 116 questions per database for a total of 9.4K training questions). This deliberate undersampling ensures that agents never see the complete dataset in any single iteration, forcing the evolution process to discover robust patterns rather than memorizing specific database quirks. The random sampling changes each iteration, exposing agents to different challenges and preventing specialization on a fixed subset.

Note that the Evolution AI maintains context across iterations, allowing it to learn from previous evolutions. We reset context after every fourth evolution so as to remain within the 200K token limit of Claude Opus 4.1.

4 Experiments

4.1 Experimental Setup

We evaluate on the BIRD benchmark (Li et al., 2023), a large-scale cross-domain dataset with 63 training databases and 11 development databases. Each evolution iteration costs approximately $4 in Claude API calls and runs for 23 minutes on average, enabling rapid experimentation cycles.

4.2 Main Results

We demonstrate RoboPhD's effectiveness through an 80- and a 30-iteration evolutionary run starting from different baseline, primordial agents. The `naive` baseline represents simple starting prompts (for online and offline agents) similar to what someone would type if they were new to the Text-to-

SQL problem (see Appendix D for the complete agent), while the `minimal_3a` baseline is slightly more sophisticated. Both represent starting points for evolution without advanced expertise in Text-to-SQL and rely heavily on the latent knowledge of the Claude Sonnet-4 LLM.

Table 1: Performance evolution from baseline agents to refined champions. Run 1 used weighted random strategy selection: (30% Research-driven, 30% Refinement, 15% Error-focused, 15% Prompt-focused, 10% None). Run 2 used: (30% each for Research-driven, Error-focused, and Refinement, 10% None).

Agent	Train	Dev	Test	ELO	Wins
Run 1: From minimal_3a baseline (80 iterations)					
minimal_3a (baseline)	66.7%	66.6%	N/A	1495	2/80
iter60_column_order_precision	68.0%	68.6%	68.6%	1652	8/80
Run 2: From naive baseline (30 iterations)					
naive (baseline)	64.9%	63.8%	N/A	1512	7/30
iter13_focused_context_optimizer	67.1%	66.4%	N/A	1666	10/30

The first run evolved from `minimal_3a` through a complex lineage detailed in Section 4.3, ultimately producing our champion agent. The second run refined the `naive` baseline through 30 iterations, producing `iter13_focused_context_optimizer` which dominated the later iterations. Note that in both cases, we are able to improve by 2 to 2.6% over the baseline without any intervention by the authors.

Our champion agent, `iter60_column_order_precision`, achieved 68.59% accuracy on the BIRD test set—our official result as verified by the BIRD benchmark organizers on their held-out test dataset. Note that train, dev and test accuracies are nearly identical for our champion agent, and train-dev gaps are consistently small across all agents, indicating that RoboPhD's evolution process resists overfitting.

4.2.1 Hardware and Costs

RoboPhD requires modest computational resources. Both evolutionary runs completed on a MacBook Pro (48GB RAM), with the 30-iteration run finishing in 11 hours. Total costs: $117 in Claude API fees for SQL generation (Sonnet-4). Evolution and Database Analysis usage came at no incremental cost beyond a $200/month Claude subscription[3].

4.3 Evolution Analysis

Case Study: Evolution of the Champion Agent. Our best-performing agent, `iter60_column_order_precision`, achieved 68.6% accuracy on both dev and test sets. This agent's evolutionary pathway revealed three key mechanisms of autonomous improvement:

(1) Research-Driven Foundation. On iteration 2, the "dynasty" began with `iter2_schema_value_miner`, created through the *research driven* strategy. The Evolution Agent autonomously studied OpenSearch-SQL (Xie et al., 2025), which achieved 72.28% on BIRD through sophisticated vector retrieval and three-phase processing. Lacking access to neural retrieval infrastructure, the Evolution Agent translated these concepts into practical Python tools: schema analyzers, value extractors, and relationship mappers. This adaptation demonstrates the system's ability to extract actionable insights from literature despite architectural constraints.

(2) Error-Driven Refinement. In iteration 12, after observing that `iter2` failed on complex aggregations and evidence parsing (30% of errors), the Evolution Agent created `iter12_evidence_precision_orchestrator` using the *error focused* strategy. The agent's reasoning explicitly referenced both successful (`iter2`: "strong on simple queries") and unsuccessful (`iter11`: "too much focus on reserved words") predecessors, synthesizing their lessons to emphasize evidence-first parsing—a capability that became foundational for all descendants.

[3] `https://claude.com/pricing/max`

(3) Emergent Discovery Through Refinement. After 48 additional iterations which clarified `iter12` as the current leader, `iter60` emerged through refinement of `iter12`, adding the critical innovation of column ordering precision. The Evolution Agent discovered that BIRD's set-based evaluation is sensitive to column order, leading to systematic rules for matching column sequence to question phrasing.

Emergent Capabilities. The final agent employs nine specialized tools executing in sequence, each addressing weaknesses identified across generations:

- Schema and relationship analysis (inherited from iter2's OpenSearch-SQL adaptation)
- Evidence parsing and pattern matching (iter12's contribution)
- Column ordering and selection logic (iter60's innovation)
- Aggregation disambiguation and reserved word detection (accumulated refinements)

The evolutionary chain shows progressive improvement: 65.5% (iter2) $\rightarrow$ 67.9% (iter12) $\rightarrow$ 68.6% (iter60) on dev set.

5 Discussion and Future Work

We are currently conducting systematic ablation studies across evolution strategies, model configurations, and architectural components, which will be reported in a forthcoming extended version. In addition, we see several directions that could enhance RoboPhD's capabilities:

Meta-evolution: A meta-agent could critique and evolve the evolution strategies themselves, adjusting both the strategy pool and their weights based on long-term performance trends. This would address local maxima where a single moderately strong agent dominates for extended periods.

Structured error analysis: A dedicated error analysis agent could provide comprehensive failure reports to all evolution strategies, ensuring consistent learning from mistakes across different evolutionary paths—analogous to gradient accumulation in optimization.

Multi-iteration refinement: Allowing the Evolution Agent to refine their previous creation across 2-3 "mini-iterations" in the same agentic context could enable deeper exploration and "aha" moments through sustained focus.

Challenger dynamics: Inspired by chess rankings, a "challenger" strategy could periodically introduce promising but under-tested agents (moderately high ELO, few tests) to disrupt stagnant hierarchies.

Tool evolution: The Evolution Agent could expand beyond database analysis and SQL generation to also evolve verification and error recovery mechanisms—learning optimal retry strategies, error interpretation patterns, and when to trust versus challenge initial results, creating a fully evolved three-phase system.

6 Conclusion

We presented RoboPhD, a system where AI agents autonomously conduct research to improve Text-to-SQL performance. Our system achieves 68.6% accuracy on the BIRD Test and Dev sets through systematic evolution, discovering effective strategies without human intervention in the research loop. Like a tireless PhD student, RoboPhD runs experiments continuously, learning from failures and evolving better approaches.

This work demonstrates a fundamental shift in AI development: rather than humans manually engineering AI systems, we can create AI researchers that improve themselves. While our system operates within the bounded domain of database queries, it demonstrates key components of autonomous research: hypothesis generation, systematic experimentation, and iterative refinement. As these capabilities generalize to broader domains, we may witness AI systems that can tackle increasingly complex research problems with minimal human oversight (Lu et al., 2024). We open-source our framework to enable the community to build upon this autonomous research paradigm, potentially transforming how we develop AI capabilities across domains beyond Text-to-SQL.

ETHICS STATEMENT

This work involves autonomous AI systems with significant operational freedom, raising important safety and security considerations. We acknowledge several critical risks:

Execution Safety: RoboPhD operates with Claude Code in an automated mode where generated code is executed without human review. While model alignment has reduced risks significantly, autonomous code execution remains inherently dangerous—the system could theoretically execute destructive commands. We mitigate this through process isolation and filesystem permissions, but acknowledge this remains an open challenge for autonomous AI systems.

Security Vulnerabilities: Autonomous agents in the inference pipeline introduce novel attack surfaces. Adversarial inputs could potentially manipulate the database analysis agent to exfiltrate data or execute unintended operations. While these considerations are not a concern for BIRD's open-source databases, production deployments would require strict sandboxing, network isolation, and comprehensive security auditing.

Autonomy and Safety Implications: Echoing longstanding warnings about recursive self-improvement and rapid capability escalation (Yudkowsky, 2008; Bostrom, 2014; Steinhardt, 2022; Cotra, 2022), while our system advances beneficial AI research capabilities, the techniques for autonomous agent evolution could potentially be misused. We therefore release our code openly to enable scrutiny and collaborative safety research, believing that transparency can accelerate both capability and safety progress.

We view this work as part of the broader conversation about AI safety and autonomy. The risks we identify are not unique to ROBOPHD but are fundamental to any system granting AI agents autonomous execution capabilities (Yudkowsky, 2008; Bostrom, 2014). In line with recent calls to proactively shape research practices and governance as autonomy increases (Steinhardt, 2022; Cotra, 2022), we encourage the community to develop stronger safety frameworks (e.g., sandboxing, least-privilege execution, red-teaming) as such systems become more capable.

REFERENCES

Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

Ajeya Cotra. Without specific countermeasures, the easiest path to transformative ai likely leads to ai takeover. *AI Alignment Forum*, 2022.

Ahmet Dönder et al. Genasql: A generative agent for advanced sql generation. *arXiv preprint arXiv:2505.14174*, 2025.

Arpad E Elo. *The Rating of Chessplayers, Past and Present*. Arco Publishing, 1978.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Max Jaderberg, Victor Dalibard, Oriol Osindero, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Hao Li et al. Omnisql: Database systems enhancement with large language models. *arXiv preprint arXiv:2503.02240*, 2025.

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 C.C. Chan, Leyang Fan, Yongji Du, Shaodong Qu, Hongkun Yao, Ming An, Annan Cheng, Gang Zhao, Ziwei Zou, and Fei Yang. BIRD: Big bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=qJVWOgJT5H.

Yichen Liu et al. Xiyan-sql: A multi-scale few-shot learning method for text-to-sql. *arXiv preprint arXiv:2507.04701*, 2024.

Ilya Loshchilov and Frank Hutter. Cma-es for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Daniel Maamari et al. Distillery: A compositional framework for text-to-sql. *arXiv preprint arXiv:2408.07702*, 2024.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2019.

Mohammadreza Pourreza and Davood Rafiei. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*, 2024.

Mohammadreza Pourreza et al. Reasoning-sql: Distilling reasoning capabilities from llms to smaller models for text-to-sql. *arXiv preprint arXiv:2503.23157*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Yunnan Sheng et al. Csc-sql: Community-driven schema construction for text-to-sql. *arXiv preprint arXiv:2505.13271*, 2025.

Vladislav Shkapenyuk et al. Askdata: Naturally leveraging data analysis agents for complex text-to-sql generation. *arXiv preprint arXiv:2505.19988*, 2025.

Jacob Steinhardt. Ai capabilities and alignment research. *AI Alignment Forum*, 2022.

Shayan Talaei, Mohammadreza Pourreza, and Davood Rafiei. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

Jianmin Xie et al. Opensearch-sql: Context-aware sql generation with an open-source suite. *arXiv preprint arXiv:2502.14913*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *EMNLP*, 2018.

Eliezer Yudkowsky. Artificial intelligence as a positive and negative factor in global risk. In *Global catastrophic risks*, pp. 308–345, 2008.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *NeurIPS*, 2023.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

A LLM USAGE

Scope and philosophy. In keeping with the spirit of this project, we made *extensive* use of large language models (LLMs) throughout. Our goal was to study whether LLMs can function as autonomous research agents with minimal domain-specific guidance from the authors. In addition to using Claude Code to help write, enhance, and refactor the code, we used Claude to help write design documents to brainstorm and clarify new ideas, and Claude helped author parts of the manuscript. In each case there was human oversight and revision applied, especially for the manuscript itself.

A.1 ROLES WITHIN THE ROBOPHD SYSTEM

Evolution (Section 3.2.6). As discussed in Section 3.1, a core contribution is showing that an LLM can serve in the role of a graduate student. Concretely, we used *Claude Code* running *Opus 4.1* for the Evolution Agent. This agent generated new agent packages (prompts, tools, and analysis plans) across iterations.

Database Analysis (Section 3.2.2). The Evolution Agent constructed *Claude Code* agents powered by *Sonnet 4* to perform schema profiling and produce database-specific analyses. These analyses were consumed downstream at inference time.

SQL Generation (Section 3.2.3). The SQL Generation phase used *Sonnet 4* via the Claude API (not via Claude Code) to translate questions (and evidence) into executable SQL, conditioned on the database analysis and evaluation instructions.

Minimal domain guidance (Sections 3.1, 3.2). By design, author-provided Text-to-SQL know-how was kept to an absolute minimum. Evolution strategies specify *process* (e.g. refine, analyze errors, optionally read and adapt a paper) rather than *scientific content*; improvements therefore arise primarily from the models' latent knowledge, error-driven iteration, and occasional literature adaptation.

A.2 LLMS USED AS SOFTWARE-ENGINEERING TOOLS

In addition, and consistent with the project's philosophy, we used *Claude Code* (primarily with *Opus 4.1*) as a development assistant for the RoboPhD infrastructure. The RoboPhD codebase was almost entirely authored through Claude Code sessions, with the human authors providing specifications, reviewing diffs, enforcing style/tests, and making final design decisions. In this context, Claude Code functioned as a coding tool under close human supervision. Very little of the final codebase was the result of direct manual edits by the authors, with one exception: the evolution strategy prompts discussed in Section 3.2.4 and shown in Appendix C were primarily hand-written. We see this as consistent with the Advisor/Graduate Student metaphor laid out in Section 3.1: we, the authors, provide thoughtful strategic guidance, while leaving autonomy to the graduate student to iterate towards a solution. As discussed in Section 5, evolving this strategic guidance is a direction for future research.

A.3 LLMS USED IN MANUSCRIPT PREPARATION

In contrast to the codebase, most of the manuscript text was written by the authors. However, portions of this paper were drafted and edited by the authors with assistance from *Claude Code (Opus 4.1)* and *ChatGPT 5*. In these cases, the authors determined the structure and arguments. Where LLM-suggested text was used, it was reviewed and, as needed, rewritten by the authors for accuracy and clarity.

A.4 ATTRIBUTION, AUTHORSHIP, AND ACCOUNTABILITY

Although LLMs played a significant role in ideation (via the Evolution Agent), engineering assistance, and writing support, all scientific claims, experiments, analyses, and conclusions are the responsibility of the human authors. The models did not have independent access to private test data or to submission systems, and they did not make publication decisions.

A.5 Data governance and safeguards

LLM prompts and outputs were constrained to benchmark-permitted artifacts (schemas, evidence, and allowed literature). We avoided leaking ground-truth SQL or test questions to the agents; evaluation used standard execution-accuracy protocols. Logs of prompts/outputs for key experiments are retained for audit and reproducibility.

B Reproducibility Statement

The results in this paper are reproducible using the open source code bundle provided as supplementary material to this manuscript. The entire framework can be run from scratch following instructions in the README, which will execute all of the iterations of evolution, reproducing agents similar to the ones featured in the results and discussion earlier in the paper. In the main body of the paper, we note the model families and modes (*Claude Code Opus 4.1* for evolution and *Claude Sonnet 4* for evaluation), interface (Claude Code vs. Claude API), and where nondeterminism is relevant, we note seeds and sampling parameters in the released configs. Despite seeds in our code for the randomness that we control, there is some non-determinism that we cannot control in Claude Opus/Sonnet API responses that may lead to minor differences in the results when run from scratch. Given that this is an evolutionary approach over many iterations, small variations in one iteration, may lead to amplified differences in subsequent iterations. While we haven't presented a thorough sensitivity analysis of this, we anecdotally note that most replication runs produce highly similar results on the development set, within 0.2 accuracy points.

C Evolution Strategy Prompts

This appendix provides excerpts from the evolution strategy prompts that guide the Evolution Agent. These prompts illustrate how we provide high-level research directions without injecting domain-specific Text-to-SQL knowledge.

C.1 Research-Driven Strategy

The research-driven strategy directs the Evolution Agent to learn from academic literature:

> *"You MUST read exactly ONE paper from the available research in papers/... The papers are from top BIRD methods achieving 71-77% accuracy."*
> *"Your Mission: Create an evolved three-artifact agent package that improves SQL generation accuracy by incorporating proven techniques from academic research. You have complete creative freedom in adapting research insights to our architecture."*
> *"How to think about the research: The key focus of a paper is on describing novel techniques they have invented that have enabled them to achieve high scores on the BIRD benchmark... Not everything in a paper is going to be practical for your system. As an example, some papers may be fine-tuning an LLM. In our work, we can't do that... Use your judgment on this."*

The papers provided to the research-driven strategy are selected from the top ten papers on the BIRD leaderboard as of 9/4/2025 (specifically, the top 10 results for which a PDF describing the method was provided): AskData (Shkapenyuk et al., 2025), CHASE-SQL (Pourreza & Rafiei, 2024), XiYan-SQL (Liu et al., 2024), CSC-SQL (Sheng et al., 2025), Reasoning-SQL (Pourreza et al., 2025), OpenSearch-SQL (Xie et al., 2025), OmniSQL (Li et al., 2025), GenaSQL (Dönder et al., 2025), Distillery (Maamari et al., 2024), and CHESS (Talaei et al., 2024).

C.2 Refinement Strategy

The refinement strategy focuses on incremental improvement of successful agents:

> *"The strategy you will be using is to adopt one agent as a starting point and to make targeted changes on top of that agent."*

> *"Pick an agent to be your starting point. This is your call. Maybe this is a current top-performer or maybe is an agent where you see an opportunity to fix it so that it would be the top performer."*
> *"Having developed ideas for improvement from your analysis of errors and system prompts produced by the agent you selected and its competitors, please refine your starting-point agent to improve it so that your new, refined agent will become the most accurate agent going forward."*

C.3 Error-Focused Strategy

The error-focused strategy emphasizes deep analysis of failure patterns:

> *"Use your judgment, but this time, try to be especially thorough in your examination of error cases. Look at more errors than you ordinarily would and work hard to study them for opportunities for significant improvements in accuracy."*
> *"Remember: Think harder than you normally would about this. Gather the information you need and use your knowledge and experience to improve accuracy... try to be guided by your thorough examination of error cases to push accuracy higher."*

C.4 Common Elements Across Strategies

All strategies share certain meta-level guidance:

> *"Your overall goal: Push accuracy higher. You are an expert in the field of Text2SQL. Use your knowledge of the field, your analysis of what is bringing accuracy down with current agents, and your analysis of the system prompts that prior systems are generating to build an agent package that will achieve higher accuracy than previous agents on a set of databases that you haven't seen before."*

These prompts demonstrate how we provide process-oriented guidance (analyze errors, read papers, refine agents) rather than domain-specific Text-to-SQL techniques, allowing the Evolution Agent to autonomously discover optimization strategies.

D The Naive Baseline Agent

This appendix presents the complete `naive` baseline agent referenced in Section 4.2. This agent represents the simplest possible starting point—prompts that someone new to the Text-to-SQL problem might write without any domain expertise.

D.1 Database Analysis Agent (agent.md)

The database analysis component simply extracts the raw database schema:

```
---
name: baseline
description: Baseline agent that outputs raw DDL schema
---

# Baseline DDL Extractor

Extract the database schema from `./database.sqlite` and save it to
`./output/agent_output.txt`.

## Instructions

1. Connect to the SQLite database at `./database.sqlite`
2. Extract the complete database schema using `.schema` command
```

```
3. Save the raw DDL output to `./output/agent_output.txt`

## Output Requirements

The output should contain:
- All CREATE TABLE statements
- All CREATE INDEX statements
- All CREATE VIEW statements (if any)
- Any other DDL statements present in the database

The output should be the raw SQL DDL with no additional formatting,
comments, or explanations.

## Example Output Format

The output file should contain statements like:
```
CREATE TABLE table_name (
    column1 type,
    column2 type,
    ...
);
CREATE INDEX index_name ON table_name(column);
```

Save the complete schema dump to `./output/agent_output.txt`.
```

D.2 SQL GENERATION INSTRUCTIONS (EVAL_INSTRUCTIONS.MD)

The SQL generation instructions are equally minimal:

```
# SQL Generation Instructions

## Core SQL Requirements

Generate clean, executable SQL:
- No markdown formatting or code blocks
- No comments or explanatory text
- Only the SQL statement

## Evidence Handling

Evidence is important!  When evidence is provided, be sure to follow
it very carefully

## SQLite Specifics

- This is a SQLite database.  Be sure to use SQLite syntax

## Remember

Keep it simple. Return exactly what's requested. Follow evidence literally.
```

D.3 TOOLS

The naive baseline contains no tools—the `tools/` directory is absent. All database analysis is performed directly by the LLM following the simple instructions above.