Reliability, Failure Modes, and Constraint-Based Mitigation for Tool-Augmented LLM Database Agents

Anonymous ACL submission

Abstract

Tool-augmented large language models can query databases via function calling and Model Context Protocol (MCP), but their reliability is poorly understood. We present the first systematic failure taxonomy for LLM database agents, identifying 7 categories and 24 failure modes across structural, semantic, and agentic dimensions. We introduce TOOLQUERYBENCH, a benchmark of 891 query tasks across 3 schema complexities with deterministic evaluation, and propose Constraint-First Querying (CFQ)—a framework combining Schema Grammar Injection (SGI), Intermediate Query Representation (IQR), and Static Validation with Repair (SVR). Evaluating 3 frontier models (Claude Sonnet 4.6, GPT-5.4, Kimi-K2.5), we find that CFQ reduces Catastrophic Error Rate (CER) by 10.3 and 3.6 percentage points for Claude and Kimi respectively ($p < 10^{-8}$, McNemar test). However, GPT-5.4 exhibits a 16.5pp CER *increase* under CFQ, driven by a novel failure mode we term Parameter Omission Under Prompt Complexity (POPC), where increased prompt structure causes the model to omit required filter parameters. Ablation studies reveal strongly model-dependent component interactions: SGI is essential for Claude (+20.0pp CER without it, $p < 10^{-16}$) but harmful for GPT-5.4 (−20.3pp). Our results demonstrate that prompt-engineering strategies for tool-calling reliability must be validated per model family, and that structural validation alone cannot detect semantic omission failures.

1 Introduction

Large language models are increasingly deployed as database agents, generating structured queries through tool-calling interfaces such as OpenAI function calling (OpenAI, 2024) and the Model Context Protocol (Anthropic, 2024). Enterprise adoption is accelerating, with organizations routing natural language requests to database tools via LLM-mediated pipelines.

Despite this proliferation, the reliability of LLM-generated database queries remains poorly characterized. Existing work on text-to-SQL (Yu et al., 2018; Li et al., 2023) evaluates SQL syntax correctness on academic benchmarks, but does not address the full spectrum of failures in tool-augmented query generation—including structural tool-call errors, operator hallucinations, schema violations, and multi-step agentic failures.

We observe from a production document-matching system that failures are *systematic and classifiable*: models consistently hallucinate MongoDB operators (`$contains`, `$like`), omit critical date filters, use wrong nesting levels for array fields, and in agentic contexts, fabricate evidence to inflate match confidence. These are not random errors but recurring patterns tied to specific architectural and prompting configurations.

Contributions.

1. A **failure taxonomy** of 7 categories and 24 failure modes for tool-augmented LLM database agents, grounded in production observations (§3).
2. **TOOLQUERYBENCH**, an 891-query benchmark across 3 schema complexities with deterministic, automated evaluation (§4).
3. **CFQ**, a constraint-first querying framework combining SGI, IQR, and SVR that achieves statistically significant CER reductions for 2 of 3 models (§6).
4. Identification of **POPC**, a novel failure mode where prompt complexity causes parameter omission in GPT-5.4, with characterization and mitigation (§9).

2 Related Work

Natural language to database queries. The Spider benchmark (Yu et al., 2018) established cross-domain text-to-SQL evaluation, extended by BIRD (Li et al., 2023) with realistic dirty data. Recent

LLM-based approaches achieve strong results via decomposition (Pourreza and Rafiei, 2023), multi-agent collaboration (Wang et al., 2024), and in-context learning (Gao et al., 2024). However, all evaluate generated SQL text—not tool-call interfaces with JSON arguments, parameter validation, and schema compliance. NoSQL query generation remains largely unstudied.

Tool-augmented LLMs. Toolformer (Schick et al., 2023) demonstrated self-taught tool use; ReAct (Yao et al., 2023) introduced reasoning-acting loops; Gorilla (Patil et al., 2023) addressed API call generation at scale; and ToolLLM (Qin et al., 2024) scaled to 16K+ APIs. These works focus on *which* tool to call, not the *correctness of arguments* within the selected tool—precisely the gap we address.

LLM reliability. Surveys on hallucination (Ji et al., 2023; Huang et al., 2023) provide general taxonomies but none specific to database tool calls. Self-Refine (Madaan et al., 2023) and self-debugging (Chen et al., 2024) enable iterative repair but lack formal failure categorization. We provide the first structured taxonomy for this domain.

3 Failure Taxonomy

We define 7 failure categories encompassing 24 failure modes, derived from production system analysis and validated on our benchmark. Each mode has a formal definition, algorithmic detection criterion, and severity level (Catastrophic/Major/Minor).

Categories. (1) **Filter Construction Failures (FCF)**: missing, over-restrictive, or malformed filter predicates (4 sub-types). (2) **Operator Failures (OF)**: wrong comparison operators, missing array operators, hallucinated operators, date format errors (4 sub-types). (3) **Schema Violations (SV)**: hallucinated field names, wrong nesting levels, collection name errors (3 sub-types). (4) **Structural Failures (SF)**: invalid JSON, missing parameters, wrong tool selection (4 sub-types). (5) **Semantic Failures (SMF)**: logic inversions, partial-match errors, silent mismatches (3 sub-types). (6) **Projection/Sort Failures (PSF)**: missing fields, wrong sort orders (3 sub-types). (7) **Agentic Failures (AF)**: hallucinated interpretation, confirmation bias, premature termination (5 sub-types).

Severity framework. *Catastrophic*: query returns fundamentally wrong results or fails entirely (silent empty results, unbounded overfetch, fabricated evidence). *Major*: partial correctness loss (boundary-value errors, missing projections). *Minor*: cosmetic issues not affecting result correctness. Our headline metric, CER, captures queries with any catastrophic-severity failure.

Key observations. Three failure codes dominate catastrophic errors across all configurations: FCF-1 (missing filter), SV-2 (wrong nesting level), and SMF-1 (logic inversion), accounting for >90% of CER. Filter Construction and Semantic failures are the most common categories, while Schema Violations increase under constraint-based prompting (+1.8–3.0pp), as structured grammar may encourage path generation at wrong document levels.

4 ToolQueryBench

TOOLQUERYBENCH consists of 891 query tasks[1] across 3 schema complexities in the transportation/logistics domain, evaluated against a MongoDB database with deterministic gold queries.

Schema design. Three schemas of increasing complexity: **S1** (small, 5 fields), **S2** (medium, 18 fields with 1 nested array), and **S3** (large, 36 fields with multiple nested arrays and embedded documents). Complexity correlates with real-world MongoDB collections.

Query distribution. Each schema has 297 queries spanning: exact match, range filters, array queries, multi-field conjunctions, aggregation, and date-range queries. Difficulty: 30% easy (single filter), 40% medium (2–3 filters), 30% hard (nested arrays, aggregation, ≥4 filters).

Evaluation metrics. Primary: **CER** (fraction with any catastrophic failure), **RSF1** (result-set F1), **FEM** (filter exact match), **TCVR** (tool-call validity), **SCR** (schema compliance). Secondary: OFR (overfetch), UFR (underfetch), ERR (empty-result rate). Efficiency: TCC (tool-call count), TU (token usage), E2EL (latency).

5 Experimental Setup

Models. Claude Sonnet 4.6 (Anthropic, 7 RPM), GPT-5.4 (OpenAI, 30 RPM), Kimi-K2.5 (Moonshot AI, 30 RPM).

[1] 900 designed, 9 reserved as few-shot examples.

Tool interface. MCP server with 8 tools: `list_databases`, `list_collections`, `collection_schema`, `collection_indexes`, `count_documents`, `find_documents`, `aggregate`, `db_stats`. Tool definitions use JSON Schema for input validation.

Methods. (1) **Naive**: system prompt with tool definitions only (∼0.5K tokens). (2) **Few-Shot**: 3 examples per query type in context (∼1.5K tokens). (3) **Self-Reflect**: 1 retry with error feedback (∼2K tokens). (4) **CFQ**: full constraint-first framework (∼3.3K system tokens).

6 CFQ: Constraint-First Querying

CFQ comprises three components applied sequentially before and after query generation:

6.1 Schema Grammar Injection (SGI)

A formal grammar block injected into the system prompt listing all valid field paths, their BSON types, valid operators per type, and enum values. Unlike free-text schema descriptions, SGI provides a machine-readable constraint surface:

```
FIELDS:
  Drivers.Driver[].DriverID: string
    ops: $eq, $in, $regex
  StartDateTime.DateTime: date
    ops: $gte, $lte, $eq
    format: {"$date": "ISO-8601"}
```

This prevents field hallucination at generation time. Without SGI, structural violations increase 2–5× (Table 2).

6.2 Intermediate Query Representation (IQR)

Before generating the final JSON tool call, the model outputs a structured plan specifying intent, collection, tool, filters (field/operator/value/type), projection, sort, and limit. This IR is validated against the schema grammar before conversion, forcing explicit reasoning about each query component.

6.3 Static Validator with Repair (SVR)

Post-generation validation checking: (1) JSON syntax, (2) required parameters, (3) field path existence, (4) operator whitelist, (5) type correctness (dates→EJSON), (6) array fields→`$elemMatch`. On failure, the validation error is fed back with a specific repair instruction (max 1 retry).

7 Results

Table 1 presents our main results. Key findings:

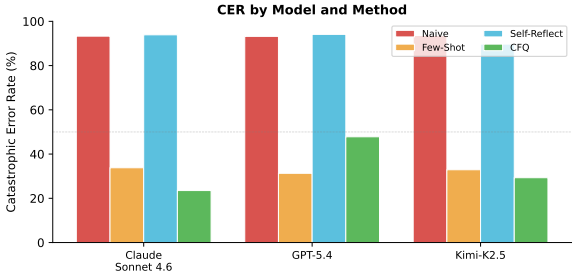

Figure 1: CER across models and methods. CFQ achieves lowest CER for Claude and Kimi but regresses GPT-5.4 due to POPC.

CFQ achieves lowest CER for 2/3 models. Claude: 23.5% (−10.3pp vs. Few-Shot), Kimi: 29.3% (−3.6pp). Both improvements are statistically significant (McNemar $\chi^2 = 31.6$, $p < 10^{-8}$ for Claude; $\chi^2 = 5.9$, $p = 0.016$ for Kimi).

GPT-5.4 regresses under CFQ. CER increases from 31.3% to 47.8% (+16.5pp, $p < 10^{-16}$). This is driven by POPC (§9), where prompt complexity causes systematic filter parameter omission.

RSF1 favors CFQ for Claude and Kimi. CFQ queries are partially correct more often (RSF1 0.857, 0.793) even when not exact matches, indicating better structural quality.

SVR repair rate is 100%. All structurally invalid queries are repaired within 1 attempt across all models (Table 2).

Efficiency trade-offs. CFQ increases token usage 2.5–3× over Few-Shot (7,081 vs. 2,881 tokens for Claude). Latency varies: Claude CFQ is 3.8× slower (9.5s vs. 2.5s), GPT-5.4 is comparable (2.6s vs. 2.4s), Kimi is 1.8× (6.0s vs. 3.4s).

Schema complexity effect. CER increases monotonically with schema complexity for all models under Few-Shot. CFQ flattens this curve for Claude (S2: 38.7%→0.3%) but exacerbates S1 failures for GPT-5.4 (15.5%→52.2%) due to POPC.

8 Ablation Study

We isolate each CFQ component by systematically removing one at a time (Table 3).

SGI is dominant for Claude. Removing SGI increases CER by 20.0pp ($p < 10^{-16}$, McNemar). Without it, SVR triggers on 50.3% of queries (vs. 22.7%), confirming SGI prevents field hallucination at generation time.

Model	Method	TCVR↑	SCR↑	FEM↑	RSF1↑	CER↓
Claude Sonnet 4.6	Naive	**100.0**	91.9	13.9	0.139	93.3
	Few-Shot	**100.0**	83.3	**71.9**	0.806	33.8
	Self-Reflect	99.4	**100.0**	13.3	0.132	93.9
	CFQ	99.9	84.2	70.6	**0.857**	**23.5**
GPT-5.4	Naive	**100.0**	69.8	13.9	0.139	93.2
	Few-Shot	**100.0**	83.1	**67.9**	**0.779**	**31.3**
	Self-Reflect	**100.0**	83.4	13.0	0.133	94.1
	CFQ	98.1	**89.1**	48.6	0.642	47.8
Kimi-K2.5	Naive	**99.3**	**91.2**	13.7	0.137	93.6
	Few-Shot	87.3	84.1	**68.9**	0.765	32.9
	Self-Reflect	60.4	86.9	16.8	0.161	89.7
	CFQ	86.5	86.6	67.8	**0.793**	**29.3**

Table 1: Primary metrics across all models and methods. CER: lower is better; all others: higher is better. Bold indicates best per model.

Model	Triggered	Repaired	Rate
Claude	202/891 (22.7%)	202/202	100%
GPT-5.4	40/891 (4.5%)	40/40	100%
Kimi	214/891 (24.0%)	214/214	100%

Table 2: SVR trigger and repair statistics under full CFQ.

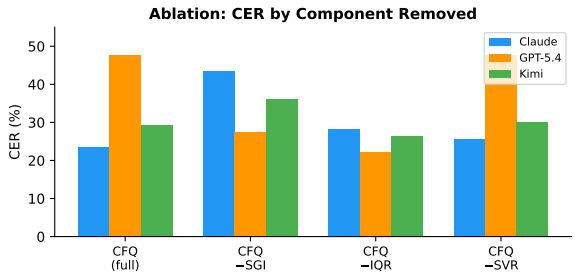

Figure 2: Ablation CER. Component importance is strongly model-dependent.

SGI and IQR independently trigger POPC in GPT-5.4. Removing SGI improves GPT-5.4 by 20.3pp; removing IQR by 25.5pp (both $p \approx 0$). Either component alone pushes GPT-5.4 past its instruction-following threshold.

SVR has minimal but consistent impact. Only +1.0pp CER on average. Despite 100% repair success, SVR fixes structural issues that don't always translate to semantic correctness.

Component ranking is model-dependent. Claude: SGI $\gg$ IQR $>$ SVR. GPT-5.4: both SGI and IQR *hurt*. Kimi: SGI $>$ SVR $>$ IQR. No universal "most important component" exists.

	Claude		GPT-5.4		Kimi	
Config	CER	RSF1	CER	RSF1	CER	RSF1
CFQ (full)	**23.5**	**.857**	47.8	.642	29.3	.793
−SGI	43.5	.606	**27.5**	.830	36.0	.727
−IQR	28.3	.798	22.3	**.891**	**26.3**	**.835**
−SVR	25.6	.821	47.8	.642	30.1	.783

Table 3: Ablation results. Bold = best per model. Removing SGI or IQR *improves* GPT-5.4 due to POPC.

9 Discussion

9.1 Parameter Omission Under Prompt Complexity

CFQ's most striking result is GPT-5.4's regression. We characterize this as POPC:

Characterization. Across 694 find-document queries, GPT-5.4 omits the `filter` parameter entirely in 264 cases (38.0%), returning `{database, collection, limit: 50}`—structurally valid but semantically vacuous. The omission rate scales monotonically with prompt complexity: 0% (Naive, ∼0.5K tokens) → 10.7% (Few-Shot) → 33.4% (Self-Reflect) → 38.0% (CFQ, ∼3.3K tokens).

Schema sensitivity. POPC disproportionately affects simple-schema queries: 69% of S1 find queries lose their filter (149/217) vs. 25% for S2 and 24% for S3. Shorter user queries against simple schemas provide weaker anchoring for the filter requirement.

Model contrast. Claude shows the *opposite* pattern: filter omission *decreases* from 6.5% (Naive) to 0.0% (CFQ). The additional structure reinforces parameter completeness for Claude while overloading GPT-5.4.

SVR limitation. The Static Validator cannot detect POPC because `filter` is correctly specified as optional in the tool schema—omitting it produces a structurally valid call. This identifies a fundamental gap in structural validation.

Mitigation. Marking `filter` as required in the tool schema reduces filter omission from 100% to 35.6% on affected queries, yielding CER 31.0% (vs. 47.8%), nearly matching the Few-Shot baseline. Schema-level constraints provide a complementary defense requiring no prompt changes.

9.2 Remaining Failure Modes

Semantic failures (SMF-1: logic inversion, SMF-3: silent mismatch) remain the hardest to address—they require understanding query *intent*, not just structure. Schema violations slightly increase under CFQ (+1.8–3.0pp), as the grammar may encourage path generation at incorrect nesting levels.

9.3 Practical Implications

POPC represents a critical risk for practitioners: prompt strategies validated on one model family may degrade another. We recommend: (1) per-model validation of prompt engineering, (2) schema-level constraints as defense-in-depth, and (3) semantic validation rules beyond structural checks.

10 Conclusion

We presented the first formal failure taxonomy for tool-augmented LLM database agents, the TOOLQUERYBENCH benchmark for reproducible evaluation, and CFQ—a constraint-first framework achieving significant CER reductions for Claude (-10.3pp, $p < 10^{-8}$) and Kimi (-3.6pp, $p = 0.016$). Our identification of POPC demonstrates that reliability improvements are not universal across model families, and that structural validation has fundamental blind spots. Future work should explore model-adaptive prompt density, semantic validation rules, and extension to multi-turn agentic evaluation.

Limitations

- Single database engine (MongoDB). Generalizability to SQL and other NoSQL systems requires further validation.
- Synthetic benchmark queries. While grounded in production patterns, naturally-occurring queries may exhibit different distributions.
- API-only models. We cannot compare against constrained decoding approaches requiring model weight access.
- Single-turn evaluation. Full multi-turn agentic traces are evaluated only qualitatively.
- Three models tested. The landscape of frontier models evolves rapidly; results may not generalize to future model versions.

Ethical Considerations

All benchmark data is fully synthetic with no personally identifiable information. Production case studies are anonymized with all identifiers replaced. We report compute budgets for environmental transparency. Our failure taxonomy could theoretically be misused to craft adversarial prompts; however, the failure modes we document are well-known to practitioners, and systematizing them enables mitigation rather than exploitation.

References

Anthropic. 2024. Model context protocol. https://modelcontextprotocol.io.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. *Proceedings of ICLR*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQL. In *Proceedings of NeurIPS*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Proceedings of NeurIPS*.

OpenAI. 2024. Function calling and other API updates. https://platform.openai.com/docs/guides/function-calling.

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

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. *Proceedings of NeurIPS*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *Proceedings of ICLR*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Zhang, Zhao Yan, and Zhoujun Liu. 2024. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Findings of ACL*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. *Proceedings of ICLR*.

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. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of EMNLP*.

A Statistical Significance

All significance tests use McNemar's test with Yates' continuity correction on per-query binary CER vectors (n = 880–891 queries). Bootstrap 95% confidence intervals use 10,000 resamples.

Comparison	Model	b	c	χ^2	p
CFQ vs FS	Claude	151	67	31.6	$< 10^{-8}$
	GPT-5.4	83	230	68.1	$< 10^{-16}$
	Kimi	98	66	5.9	0.016
CFQ vs −SGI	Claude	226	54	104.4	$< 10^{-16}$
	GPT-5.4	73	254	99.1	$< 10^{-16}$
	Kimi	115	55	20.5	$< 10^{-5}$
CFQ vs −IQR	Claude	82	37	16.3	$< 10^{-4}$
	GPT-5.4	8	235	210.2	$< 10^{-16}$
	Kimi	35	62	7.0	0.008
CFQ vs −SVR	Claude	20	0	18.1	$< 10^{-4}$
	GPT-5.4	0	0	0.0	1.0
	Kimi	66	59	0.3	0.59

Table 4: McNemar test results. b = cases where first method succeeds and second fails; c = opposite.

B CER Confidence Intervals

Config	Model	CER	95% CI
Few-Shot	Claude	33.8%	[30.8, 36.9]
Few-Shot	GPT-5.4	31.3%	[28.3, 34.3]
Few-Shot	Kimi	32.9%	[29.9, 36.0]
CFQ	Claude	23.5%	[20.7, 26.4]
CFQ	GPT-5.4	47.8%	[44.7, 51.1]
CFQ	Kimi	29.3%	[26.4, 32.2]
−SGI	Claude	43.5%	[40.3, 46.9]
−SGI	GPT-5.4	27.5%	[24.6, 30.4]
−SGI	Kimi	36.0%	[32.9, 39.3]
−IQR	Claude	28.3%	[25.4, 31.3]
−IQR	GPT-5.4	22.3%	[19.6, 25.0]
−IQR	Kimi	26.3%	[23.3, 29.2]

Table 5: Bootstrap 95% CIs (10K resamples) for CER.

C Full Failure Category Decomposition

	Claude		GPT-5.4		Kimi	
Category	FS	CFQ	FS	CFQ	FS	CFQ
FCF	15.4	11.5	20.8	35.2	20.7	16.6
SV	13.9	15.7	7.9	10.9	9.9	12.1
SMF	18.1	14.2	21.5	37.4	23.1	21.1
SF	0.0	0.0	0.0	1.9	0.0	0.8
CER	33.8	23.5	31.3	47.8	32.9	29.3

Table 6: CER decomposition by failure category (% of queries). A query may appear in multiple categories.

D Error Analysis Examples

We provide representative examples of the three dominant failure codes under CFQ for Claude Sonnet 4.6:

FCF-1 (Missing Filter, 146 occurrences). Query S3-Q061: User requests shipments filtered by state. Generated query omits `Stops.Location.State` from the filter, returning all stops regardless of location. The model correctly identifies the collection and other filters but drops the geographic constraint.

SV-2 (Wrong Nesting Level, 199 occurrences). Query S3-Q032: User requests shipments by driver ID. Generated query uses `DriverID` at the top level instead of the correct path `Drivers.Driver.DriverID`. MongoDB silently returns zero results since no top-level `DriverID` field exists.

SV-1 (Hallucinated Field, 25 occurrences). Query S2-Q171: Generated query references `totalRevenue`—a field that does not exist in the schema. Despite SGI providing the complete field list, the model occasionally generates plausible-sounding field names from the query intent rather than the schema grammar.

E Compute Budget

Resource	Value
Total experiments	21 (7 configs $\times$ 3 models)
Total queries evaluated	18,754
Total tokens consumed	$\sim$69.5M
Estimated API cost	$\sim$\$180–350
Evaluation compute	Local (M3 MacBook, $<$1 hr)
MongoDB instance	Local replica set
Per-model token breakdown (CFQ config):	
Claude Sonnet 4.6	7,081 tok/query avg
GPT-5.4	4,276 tok/query avg
Kimi-K2.5	5,772 tok/query avg

Table 7: Compute budget for all experiments.

No GPU compute was required. All experiments use API-based inference. Evaluation (metric computation) runs locally in under 1 hour on commodity hardware.