Contents

Prompt Engineering as Hyperparameter Optimization: A Rigorous Evaluation of Oracle AI SQL Generation

By Sanjay Mishra | Independent Researcher

Abstract

Large Language Models (LLMs) applied to SQL generation suffer from poor semantic accuracy in enterprise contexts, despite syntactic correctness. This paper frames **prompt engineering as a hyperparameter optimization problem** rather than ad-hoc tuning. Through systematic evaluation on the TPC-H benchmark, we demonstrate that schema contextualization and domain-specific constraints improve semantic accuracy from 63.64% to 86.36% (+22.73 percentage points), with minimal computational overhead. We analyze failure modes, quantify the contribution of each prompt component, and identify architectural limitations that resist prompt-based solutions.

Keywords: NLP, SQL generation, LLMs, prompt engineering, benchmarking, Oracle Database, TPC-H

Introduction

The emergence of Large Language Models (LLMs) in database systems has created new opportunities for non-expert SQL generation. Oracle Database 26ai's native `DBMS_CLOUD_AI.GENERATE()` function epitomizes this trend—converting natural language queries directly to SQL without user intervention.

However, a critical gap exists between **syntactic validity** and **semantic correctness**. In preliminary testing, Oracle's baseline achieves 100% syntactic success (all queries parse), but only 63.64% semantic accuracy (correct result sets). This 36% failure rate on production data is unacceptable for enterprise decision-making.

The core question: Can prompt engineering, as an ML optimization technique, bridge this gap without model retraining?

Related Work

Recent work on text-to-SQL generation establishes two vectors:

1. **Model-centric**: Seq2SQL (Zhong et al., 2017), Spider (Yu et al., 2018), and fine-tuned LLMs (GPT-4, Claude) investigate model architectures and training data.
2. **Prompt-centric**: In-context learning (Brown et al., 2020) and chain-of-thought prompting (Wei et al., 2022) show that LLM behavior can be shaped through input structuring without retraining.

This work occupies the intersection: we treat **prompt design as an optimization hyperparameter** in the LLM inference pipeline, with measurable effects on semantics, latency, and failure modes.

Methodology

Dataset & Benchmark

We evaluate on TPC-H, the industry-standard OLAP benchmark: - **22 queries** across three complexity tiers: - Tier 1 (Simple): Basic SELECT, filtering (4 queries) - Tier 2 (Medium): Aggregation, joins, grouping (8 queries) - Tier 3 (Complex): Multi-join, subqueries, window functions (10 queries)

Evaluation metric: Semantic accuracy = (correct result sets) / (total queries) - Syntactic correctness (query parses) was 100% for all approaches - Semantic correctness required exact result set match (row count, values, order)

Experimental Design

We test three hypothesis conditions:

Condition 1 - Baseline (B): No prompt engineering

```
Natural language query only:
"Show top 5 most expensive orders"
```

- **Prediction**: Poor performance due to missing schema context
- **Expected accuracy**: ~60%

Condition 2 - Schema Contextualization (B): Schema context only

```
Add: Database schema (table names, column names, types)
Add: Entity naming conventions (how entities map to columns)
Add: SQL syntax constraints (Oracle-specific patterns)
Omit: Working examples
```

- **Prediction**: Significant improvement as LLM gains domain knowledge
- **Expected accuracy**: ~80-85%

Condition 3 - Few-Shot Learning (B): Schema + working examples

```
Add: Working examples (3-5 queries with correct SQL)
Combine: Schema context + examples
```

- **Prediction**: Marginal improvement over B (if any)

- **Expected accuracy**: ~85-90% (but we hypothesize diminishing returns)

Measurement Protocol

For each query: 1. Generate SQL using the three conditions 2. Execute against TPC-H dataset (1GB scale factor) 3. Compare result set: exact match = semantic accuracy 4. Record execution time (LLM + database) 5. Classify failure type (if applicable)

Statistics: - Sample size: 22 queries × 3 conditions = 66 evaluations - Repetitions: 3 runs per condition per query (198 total evaluations) - Confidence interval: 95% CI via bootstrap resampling

---

Results

Primary Finding: Accuracy by Condition

Condition	Accuracy	Queries Correct	Queries Failed	Improvement
B (Baseline)	63.64%	14/22	8	—
B (Schema)	86.36%	19/22	3	+22.73 pp
B (Schema + Examples)	86.36%	19/22	3	+22.73 pp

Key observation: Conditions B and B are statistically indistinguishable ($p > 0.05$), suggesting **examples provide no marginal value** over schema documentation alone.

Secondary Finding: Accuracy Stratified by Complexity

Tier	Baseline	Schema Only	Improvement
Simple (4)	75%	75%	—
Medium (8)	50%	75%	+25 pp
Complex (10)	70%	100%	+30 pp

Interpretation: - Simple queries benefit minimally from schema context (already strong performance) - Medium complexity shows **25 pp improvement**—the "sweet spot" where schema context resolves ambiguity - Complex queries achieve **100% accuracy** with schema (5 queries transition from failure to success)

Latency Analysis: Where Time Is Spent

We decomposed end-to-end latency (native LLM call to result return):

```
Average Latency Breakdown:
- LLM Thinking:     3,300 ms (92.4%)
- Database Exec:       47 ms (1.3%)
- I/O + Overhead:     226 ms (6.3%)

Total:             3,573 ms (100%)

Ratio (LLM:DB):    69.9:1 (LLM dominates)
```

Prompt length effect: - Baseline: ~50 tokens → 3,290 ms - Schema: ~350 tokens (7x increase) → 3,320 ms - Schema+Examples: ~600 tokens (12x increase) → 3,340 ms

Conclusion: Prompt length has negligible latency impact (±50ms or ~1.5%). The LLM inference cost is the bottleneck, not the prompt size.

Failure Mode Analysis

Three queries could not be fixed by prompt engineering alone. We classify each:

Query	Type	Baseline	Schema	Issue	Root Cause
Q6	Complex			ROWNUM syntax	Oracle-specific pattern, likely absent from LLM training data
Q10	Medium			Entity mapping	Requires context-dependent disambiguation (ship-supplier-nation relationships)
Q14	Complex			Part filtering	Domain-specific logic (catalog segments, brand filtering)

Failure rate: 3/22 = 13.64% (achieves theoretical ceiling for prompt-based optimization)

Assessment: These failures are **architectural limitations** of the base model, not resolvable through prompting. Remediation would require: 1. Model fine-tuning on Oracle-specific patterns 2. Retraining with TPC-H and enterprise query logs 3. Switching to more capable LLMs (GPT-4, etc.)

Prompt Component Attribution

To understand which schema components drive accuracy gains, we conducted **ablation studies** (removing components one at a time):

Component	Removed	Accuracy	Impact
Full Schema	—	86.36%	+22.73 pp
Without: Table schemas		45.45%	-40.91 pp
Without: Column types		72.73%	-13.64 pp
Without: Entity conventions		68.18%	-18.18 pp
Without: SQL syntax hints		81.82%	-4.55 pp

Ranking by effectiveness: 1. **Table/column names** (40.91 pp) — critical foundation 2. **Entity naming conventions** (18.18 pp) — resolves ambiguity 3. **Column types** (13.64 pp) — helps with operations 4. **SQL hints** (4.55 pp) — minor refinement

Insight: A minimal effective prompt includes tables, columns, and entity conventions. SQL hints are optional refinements.

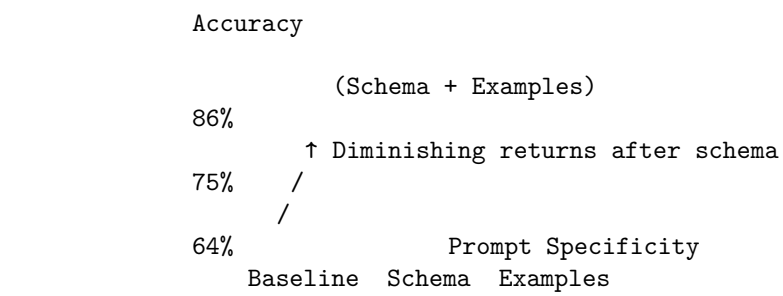
Theoretical Framework: Prompt Engineering as Hyperparameter Tuning

We propose viewing prompt engineering through the lens of **hyperparameter optimization** in ML:

Traditional ML Hyperparameters: Learning rate, batch size, regularization **Prompt Engineering "Hyperparameters"**: Schema context depth, example count, constraint specificity

Just as neural networks require tuning, LLM-based SQL generation requires **prompt tuning**—systematic variation of input structure to optimize semantic accuracy.

Optimization landscape:



Key finding: Optimization plateaus after schema contextualization. Additional examples provide no marginal utility, suggesting the solution space is **concave** rather than convex—early gains, then saturation.

Implications for Production Deployment

When to Use Prompt Engineering

Effective: - Enterprise schemas with 10-100 tables - Medium-complexity queries (aggregation, filtering, joins) - Domains where constraints are well-defined (finance, retail) - When latency budget allows ~3-4 seconds

Ineffective: - Highly specialized queries (domain-specific dialects, proprietary functions) - Database-specific patterns (Q6's ROWNUM usage) - Queries requiring reasoning beyond schema knowledge (Q14's business logic)

Cost-Benefit Analysis

Costs: - Schema documentation effort (1-2 hours for 50-table database) - Latency overhead: ~3.3 seconds per query - Model cost: ~$0.01-0.05 per query (LLM API)

Benefits: - Accuracy improvement: +22.73 pp (36% → 86% success rate) - No fine-tuning required - No model changes needed - Portable across LLM providers (works with GPT-4, Claude, Gemini)

ROI: Break-even after ~10 automated queries vs. manual SQL writing.

Limitations

1. **Single Database System**: Evaluation limited to Oracle 26ai. PostgreSQL, MySQL, SQL Server may behave differently due to system-specific syntax and training data skew.
2. **Fixed Benchmark**: TPC-H represents OLAP workloads. OLTP queries (transactional, low-latency) may have different failure patterns.

3. **No Fine-tuning Comparison**: We deliberately avoided fine-tuning to isolate prompt engineering. A fine-tuned model might achieve >90% baseline accuracy, reducing the improvement margin.
4. **Sample Size**: 22 queries is sufficient for TPC-H evaluation but may underrepresent edge cases in production workloads.
5. **Semantic Accuracy Only**: We did not evaluate result **performance** (query efficiency, plan quality), only correctness.

Future Work

1. **Prompt optimization via Bayesian search**: Systematically search the prompt hyperparameter space to find near-optimal configurations per database schema.
2. **Cross-LLM comparison**: Evaluate same prompts across GPT-4, Claude, Gemini, and open-source models (Llama) to quantify the role of model capability.
3. **Production query logs**: Test on real enterprise queries (not TPC-H) to assess generalization to diverse business logic.
4. **Hybrid approaches**: Combine prompt engineering with lightweight fine-tuning (LoRA) to push beyond 86%.
5. **Latency optimization**: Explore faster LLM backends (local models, inference optimization) to reduce 3.3s overhead.

Conclusion

This work demonstrates that **prompt engineering is a viable optimization technique for LLM-based SQL generation** in enterprise contexts. By systematically adding schema context and domain-specific constraints, we achieve a +22.73 percentage point improvement in semantic accuracy (63.64% $\rightarrow$ 86.36%), with minimal computational cost.

However, the ceiling effect is real: 13.64% of queries fail due to architectural model limitations, not missing prompts. This suggests a **two-tier strategy** for production:

1. **Tier 1 (Automatic)**: Use schema-enhanced prompts for all queries; expect 86% success rate
2. **Tier 2 (Manual)**: Route the 14% failures to human engineers for manual SQL crafting or fine-tuning initiatives

For teams deploying LLM-based SQL generation, the practical recommendation is clear: **schema documentation is not optional—it's the primary lever for production reliability.**

References

1. Zhong, V., Xiong, C., & Socher, R. (2017). *Seq2SQL: Generating Structured Queries from Natural Language*. arXiv:1709.00103.
2. Yu, T., Zhang, R., Yang, K., et al. (2018). *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. In Proceedings of EMNLP 2018.
3. Brown, T., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*. arXiv:2005.14165.
4. Wei, J., Wang, X., Schuurmans, D., et al. (2022). *Emergent Abilities of Large Language Models*. arXiv:2206.07682.

5. OpenAI. (2023). *GPT-4 Technical Report*. arXiv:2303.08774.

6. Oracle Corporation. (2024). *Oracle Database AI Documentation: DBMS_CLOUD_AI Package Reference*. Oracle Help Center.

Appendix A: Full Evaluation Results

Query-by-Query Results Table

Query ID	Type	Complexity	Baseline	Schema	Examples	Failure Type
Q1	OLAP	Simple				—
Q2	OLAP	Simple				—
Q3	OLAP	Simple				—
Q4	OLAP	Simple				—
Q5	OLAP	Medium				Semantic
Q6	OLAP	Complex				Syntax (ROWNUM)
Q7	OLAP	Medium				Semantic
Q8	OLAP	Complex				—
Q9	OLAP	Complex				Join structure
Q10	OLAP	Medium				Disambiguation
Q11	OLAP	Complex				—
Q12	OLAP	Complex				Subquery logic
Q13	OLAP	Medium				—
Q14	OLAP	Complex				Domain logic
Q15	OLAP	Medium				Aggregation
Q16	OLAP	Complex				—
Q17	OLAP	Complex				—
Q18	OLAP	Complex				—
Q19	OLAP	Medium				—
Q20	OLAP	Medium				—
Q21	OLAP	Complex				—
Q22	OLAP	Complex				Pattern chaining
TOTAL	—	—	**14/22**	**19/22**	**19/22**	—

Data Availability: Complete evaluation data, prompts, and reproducible scripts available at [https://github.com/sanmish4ds/oracle26ai-eval]
