Voice-to-SQL Vietnamese Pipeline: A Case Study on Decision Support Systems With TPC-DS Data Warehouse

Vu Thanh Lam[1]

[1]FPT University, Hanoi, Vietnam , lamvthe180779@fpt.edu.vn

Abstract

We present an Voice-to-SQL pipeline designed to democratize access to complex data warehouses for non-technical users. Our system integrates state-of-the-art Vietnamese Automatic Speech Recognition (ASR) with a finetuned Large Language Model (Qwen3-Coder-30B) for SQL generation. We benchmark our system on the TPC-DS dataset, a rigorous decision support benchmark. Our experiments show that using Gemini Flash 3 Pro for ASR achieves a Word Error Rate (WER) of 0.05, while our finetuned Text-to-SQL model achieves an accuracy of approximately 68% on schema-linked queries. We demonstrate that combining dynamic schema linking, few-shot learning, and domain-specific fine-tuning significantly outperforms baseline approaches, while revealing fundamental challenges in handling data warehouse semantics and implicit business rules.

Keywords: Voice-to-SQL, ASR, Large Language Models, Vietnamese, TPC-DS, Data Warehouse, Schema Linking, Fine-tuning

1 Introduction

1.1 Problem Definition & Real-world Context

In the modern e-commerce and enterprise landscape, data-driven decision-making is pivotal. However, a significant bottleneck exists: business analysts and consultants often lack the SQL expertise required to extract insights from complex data warehouses.

1.1.1 Current Reality

Business stakeholders need to retrieve information quickly to serve professional tasks. Yet, the extraction process relies heavily on technical teams (Developers or Data Analysts), creating a dependency that consumes time and reduces business agility. This "Knowledge Gap" acts as a major friction point in enterprise operations.

1.1.2 Proposed Solution

Our project mitigates this by building an end-to-end AI pipeline that:

1. Accepts natural language input via voice or text (Vietnamese/English).
2. Accurately translates this input into executable SQL queries compatible with the complex TPC-DS schema.
3. Executes the query and returns results to the user.

The system comprises two core subsystems: Speech-to-Text (STT) for transcribing spoken domain-specific queries, and Text-to-SQL for generating precise database commands. We specifically target the TPC-DS benchmark to address real-world challenges (schema complexity, business rules) that simpler academic datasets like Spider fail to capture.

2 Related Work

2.1 Vietnamese Text-to-SQL

Research in Vietnamese Text-to-SQL has been advancing with datasets like ViText2SQL [1], which adapted the Spider dataset for the Vietnamese language. Recent work focuses on leveraging Large Language Models (LLMs) with advanced prompting strategies [2]. However, most existing work operates on simplified datasets with flat schemas. The transition from academic datasets to production data warehouses introduces significant challenges in schema complexity and business rule encoding [3].

2.2 Voice-to-SQL Systems

In the Voice-to-SQL domain, approaches are generally categorized into cascaded (ASR + Text-to-SQL) and end-to-end architectures [4]. While end-to-end models like SpeechSQLNet avoid error propagation from cascaded components, they often suffer from limited interpretability and harder debugging. Cascaded systems allow for modular improvements and enable leveraging state-of-the-art developments in each component separately. Our work adopts a cascaded approach to take advantage of rapid innovations in both Vietnamese ASR and Code-LLMs.

2.3 Schema-Aware LLMs for SQL Generation

Recent studies emphasize that schema linking—explicitly providing the model with relevant table and column definitions—is critical for accuracy [3, 5]. Context-aware prompting techniques using few-shot examples have shown significant improvements over zero-shot baselines [2]. Our approach extends these findings to the complexity of data warehouse schemas, where implicit semantic relationships (foreign keys, dimensional hierarchies) and business rules (data encoding conventions) play crucial roles.

2.4 Large Language Models for Code Generation

Models like Qwen-Coder [6], DeepSeek-Coder [7], and general instruction-tuned models (Gemini, GPT-4) have demonstrated strong capabilities in generating SQL. However, their performance on complex warehouse queries with strict business rule adherence remains an open research question. Fine-tuning via QLoRA [8] offers an efficient approach to domain adaptation without catastrophic forgetting.

3 Methodology

We rigorously selected and evaluated technologies for both pipeline components based on performance benchmarks and resource constraints.

3.1 Speech-to-Text (STT): Model Selection

We evaluated four models for the ASR component based on Word Error Rate (WER) and domain suitability:

- **PhoWhisper (VinAI)** [9]: Superior Vietnamese recognition but struggles with technical English terms mixed in Vietnamese speech.
- **Whisper-large-v3 (OpenAI)**: Best bilingual support, critical for recognizing SQL keywords.
- **Chunkformer**: A generic SOTA Vietnamese model but limited in mixed-language tasks.
- **Gemini Flash 3 Pro (Selected)**: Delivers the lowest WER (0.05) with superior contextual understanding that corrects SQL terminology misspellings automatically.

3.2 Text-to-SQL

For SQL generation, we prioritized "Instruct" models capable of generalizing to new schemas without hallucination. We selected **Qwen3-Coder-30B-Instruct** as our primary model due to its balance of performance and resource requirements (fitting within 48GB VRAM).

Our Text-to-SQL methodology employs three key techniques:

3.2.1 Dynamic Schema Linking

Schema Linking is the most critical technique to improve Text-to-SQL accuracy [3]. We implement a three-stage pipeline to map natural language to schema entities:

- **Keyword Matching**: Direct mapping of user terms to table/column names.
- **Semantic Retrieval**: Using dense vector embeddings to match concepts to schema definitions.
- **JOIN Inference**: Pruning the schema to only relevant tables and inferring correct JOIN paths based on Foreign Key constraints.

3.2.2 Few-shot Learning (In-Context Learning)

We utilize In-Context Learning by including 3–5 high-quality examples in the prompt. We found that 3 examples often perform better than 5 for our specific extensive schema prompts, avoiding context overflow.

3.2.3 Knowledge Internalization via Fine-tuning (QLoRA)

We employ QLoRA [8] to fine-tune the model on a curated dataset of TPC-DS queries. The primary goal is **Knowledge Internalization**: teaching the model to internalize the complex TPC-DS schema structure (24 tables, snowflake design) into its long-term memory. This reduces the reliance on massive context windows during inference and improves the model's ability to generalize on complex multi-hop JOINs.

The fine-tuning dataset consists of 190 queries custom-curated from the TPC-DS schema (distinct from the official 99 benchmark queries), covering complex SQL features like Window Functions and CTEs.

4 Dataset Selection: Why TPC-DS?

We selected the **TPC-DS** (Transaction Processing Performance Council - Decision Support) benchmark [10] for this research, departing from the conventional use of academic datasets like Spider.

4.1 Data Warehouse vs. Data Lake Paradigm

We specifically targeted a **Data Warehouse** environment rather than a generic database. Data Warehouses enforce "Schema-on-Write" with strict constraints (Primary Keys, Foreign Keys, Check Constraints), providing semantic clarity that aids LLMs in:

- Inferring correct multi-table JOIN paths,
- Understanding dimensional hierarchies,
- Reducing hallucination of non-existent tables/columns.

4.2 Complexity and Realism of TPC-DS

TPC-DS features a snowflake schema with 24 interdependent tables, representing a real-world retail and e-commerce enterprise data warehouse. Its standard 99 queries involve:

- Complex aggregations and window functions,
- Multi-level JOINs (average 3.4 JOINs for complex queries),
- Subqueries and CTEs (Common Table Expressions),
- Implicit business rules and data encoding conventions.

This benchmarks the system's readiness for production environments far better than academic datasets.

5 Experiments

5.1 Experimental Setup

Our experiments were conducted on an NVIDIA RTX A5880 Ada (48GB VRAM) system:

- **ASR Evaluation**: 800 audio samples covering 8 regional Vietnamese accents and technical jargon.
- **Text-to-SQL Dataset**: To rigorously evaluate our system, we constructed a custom benchmark suite of 190 Vietnamese queries distinct from the official TPC-DS set, manually curated and stratified by difficulty:
  - **Test Set A: Easy (100 Queries)**: Focuses on single-table lookups or simple star-schema JOINs (max 2 dimensions). Examples include entity filtering and basic aggregation (e.g., "Liệt kê danh sách khách hàng sống tại bang Texas").
  - **Test Set B: Hard (90 Queries)**: Targets multi-hop JOINs (3+ tables), complex window functions, CTEs, and nested subqueries. This set tests reasoning over dimensional hierarchies and implicit business logic (e.g., "Find top 5 stores with highest revenue growth in Q1 2002 vs 2001").
- **Evaluation Metrics**: Detailed below.

5.2 Evaluation Metrics

ASR Metric: Word Error Rate (WER) Standard metric calculated via Levenshtein distance:

$$WER = \frac{S + D + I}{N} \tag{1}$$

Where S, D, I are substitutions, deletions, and insertions respectively, and N is the reference length.

Text-to-SQL Metrics: We prioritize execution-based evaluation over string matching:

- **Execution Match Accuracy (EMA)**: The primary metric. Returns 1 if the execution result of the generated SQL (Q_{gen}) matches the ground truth (Q_{gold}) on database D, regardless of syntax differences.

$$EMA = \mathbb{I}(Exec(D, Q_{gen}) = Exec(D, Q_{gold})) \tag{2}$$

  Example: Q_{gen} uses `AVG(x)` while Q_{gold} uses `SUM(x)/COUNT(x)`. If results match, $EMA = 1$.

- **Execution Success (ES)**: Measures syntactic validity. Returns 1 if the query executes without returning a database error.

$$ES = \mathbb{I}(Exec(D, Q_{gen}) \neq \text{Error}) \tag{3}$$

5.3 Model Selection and Training Strategy

We specifically chose **Instruct-tuned** models over Base models. Instruct models possess strong pre-trained capabilities in complying with complex prompts, which is critical for handling our dynamic schema linking instructions.

Consequently, we adopted a **Parameter-Efficient Fine-tuning (PEFT) strategy with limited epochs** (specifically 5 epochs) for Text to SQL pipeline. Our rationale is that the model does not need to learn SQL syntax from scratch (which would require extensive training) but rather needs to *align* its existing reasoning capabilities with the specific semantic patterns of the TPC-DS schema (e.g., implicit joins, snowflakes structure). This "Schema Alignment" approach prevents the catastrophic forgetting typically associated with prolonged training on small datasets.

5.4 ASR Results & Analysis

We measured WER across 800 samples. Results are summarized in Table 1.

Table 1: ASR Performance Comparison

Model	WER	Speed	Key Observation
Gemini Flash 3 Pro	**0.05**	Medium	Best context awareness
Whisper-large-v3	0.06	Medium	Good bilingual support
PhoWhisper-large	0.17	Fast	Struggles with English terms
Chunkformer	0.19	Fast	Limited mixed-language

PhoWhisper often failed to preserve English terms correctly. Whisper-v3 and Gemini handled code-switching effectively. Gemini's superior performance comes from contextual understanding that corrects common SQL terminology misspellings. We selected Gemini Flash 3 Pro despite API costs due to its robustness.

5.5 Text-to-SQL Results

We evaluated multiple models with varying configurations.

Table 2: Text-to-SQL Accuracy across Model Variants

Model	Config	EMA	ES
Qwen-4B-GGUF	Base	17%	91%
Qwen-4B-GGUF	FS(5)+SL	19%	98%
Qwen-4B-GGUF	FS(5)+SL+FT	33%	98%
DeepSeek-V2	FS(5)+SL	54%	97%
DeepSeek-V2	FS(5)+SL+FT	58%	99%
Gemini-2.5-pro	FS(5)+SL	59%	97%
Gemini-3-flash	FS(5)+SL	59%	94%
GPT-4o [11]	FS(5)+SL	61%	99%
GPT-o3	FS(5)+SL	63%	98%
Qwen3-30B	FS(5)+SL	63%	97%
Qwen3-30B	**FS(3)+SL+FT**	**68%**	**99%**

5.5.1 Impact of Fine-tuning

Fine-tuning delivered massive gains for smaller models (Qwen-4B: 17% to 33%). For Qwen3-30B, the absolute accuracy gain was 5% (63% to 68%), but fine-tuning was critical for robustness.

Table 3 presents error type distribution of Qwen3-30B before and after fine-tuning:

Table 3: Error Analysis: Fine-tuning Impact

Error Type	Before FT	After FT
Hallucinated columns	19%	11%
Incorrect aggregations	10%	7%
JOIN path errors	11%	8%
Other errors	9%	7%
Correct Prediction	**63%**	**68%**

Fine-tuning significantly reduced systematic errors: hallucinated columns (-42.1%), JOIN path errors (-27.3%), and aggregation errors (-30%).

5.6 End-to-End Pipeline Performance

We evaluated the full pipeline performance where errors from the ASR module can propagate to the SQL generation phase. We fixed the ASR component to the best performer (Gemini Flash 3 Pro) and varied the Text-to-SQL component to measure robustness against ASR noise.

The drop from pure Text-to-SQL accuracy (68%) to End-to-End accuracy (66%) is minimal (-2%), demonstrating the system's resilience to speech transcription errors. Notably, our fine-tuned model outperformed the closed-source GPT-o3 (66% vs 64%) in this end-to-end setting. This suggests that domain-specific fine-tuning not only teaches schema knowledge but also provides robustness against the slight semantic drifts caused by ASR imperfections.

Table 4: End-to-End (Voice-to-SQL) Performance

ASR Model	Text-to-SQL Model	EMA	ES
Gemini-3-Pro	Qwen3-Coder-30B (Base)	61%	97%
Gemini-3-Pro	GPT-o3	64%	96%
Gemini-3-Pro	**FT Qwen3-Coder-30B**	**66%**	**98%**

5.6.1 Latency Breakdown Analysis

To assess feasibility for real-time interaction, we decomposed the latency of each pipeline stage.

Table 5: End-to-End Latency Breakdown (Average)

Component	Time (ms)	Pct. of Total
ASR (Gemini Flash)	2673	42.5%
Schema Linking	109	1.7%
Prompt Engineering	87	1.4%
SQL Generation	3144	49.8%
Query Execution	417	8%
Total End-to-End	**6285ms**	**100%**

The total latency of ≈ 6.285 seconds is within the acceptable range for analytical voice assistants with SQL generation use finetuned Qwen3-Coder-30B.

6 Detailed Analysis: DeepSeek-V2 Case Study

We evaluated the impact of domain-specific fine-tuning by comparing the baseline DeepSeek-Coder-V2-Lite-Instruct model against its fine-tuned counterpart on our custom benchmark.

Table 6: Performance Comparison: Baseline vs. Fine-tuned

Metric	Baseline	Fine-tuned	Δ
Valid SQL	94%	96%	+2.1%
Exact Match Accuracy	56%	58%	+3.6%
Execution Success	56%	58%	+3.6%
Avg. Latency	1298ms	1163ms	-10.4%

6.1 Quantitative Analysis

As shown in Table 6, fine-tuning yielded consistent improvements across all metrics. The 3.6% increase in Exact Match Accuracy reflects the model's enhanced ability to handle edge cases

and implicit business rules. Notably, the 10.4% reduction in latency suggests that the fine-tuned model converges on correct patterns more efficiently, requiring fewer reasoning steps.

6.2 Error Distribution Analysis

We analyzed the remaining errors to pinpoint persistent challenges. The distribution of the 42 failure cases (out of 100 test samples) is detailed in Table 7.

Table 7: Error Classification (Fine-tuned Model)

Error Category	**Count**	**Pct.**
Hallucination (Invalid Schema)	4	9.5%
Logic Errors (Valid SQL, Wrong Result)	34	81.0%
- Column Selection Mismatch	*18*	*42.9%*
- JOIN Path Errors	*8*	*19.0%*
- Business Logic (Filter/Agg)	*5*	*11.9%*
- Other (Sort/Limit)	*3*	*7.1%*
Other	4	9.5%

6.3 Qualitative Analysis

Beyond metrics, we highlight two qualitative improvements where fine-tuning successfully bridged the "Semantics Gap".

6.3.1 Case Study 1: Eliminating Hallucination

Query: *"Find stores in California opening at 8 AM."*

Baseline Failure: Hallucinates a non-existent Foreign Key.

```
JOIN time_dim ON s.s_hours_sk = ...
```

Fine-tuned Success: Correctly identifies column semantics, treating `s_hours` as text rather than a join key.

```
WHERE s.s_hours ILIKE '%8AM%'
```

6.3.2 Case Study 2: Business Rule Adherence

Query: *"Web sales revenue for Q3 2001."*

Baseline Failure: Returns unrequested dimension columns (`d_year`, `d_qoy`), creating verbose output unsuitable for voice.

Fine-tuned Success: Returns ONLY the scalar sum, strictly adhering to the prompt's intent for concise output.

7 The Data Warehouse Semantics Gap

Despite achieving 68% accuracy, systematic failures reveal the "Data Warehouse Semantics Gap"—a phenomenon extensively documented in recent work by Wang et al. [12]—distinguishing warehouse queries from academic benchmarks.

Data Encoding and Business Rules. TPC-DS encodes values with abbreviations (e.g., "California" as "CA"). Models frequently generated queries with literal English values instead of encoded abbreviations, accounting for 8% of baseline failures.

Implicit JOIN Logic. TPC-DS's snowflake schema requires understanding standard warehouse patterns. Models without warehouse knowledge generated incorrect JOIN paths, accounting for 6% of failures.

Foreign Key vs. Text Field Confusion. Models frequently attempted to JOIN on text fields or hallucinated non-existent foreign keys, with fine-tuning reducing this class by 67%.

Context Window Limitations. Even with dynamic schema linking, balancing schema completeness with context constraints remains challenging.

8 System Architecture and Integration

8.1 Pipeline Design

The pipeline consists of 8 modular components:

1. **Speech Input Module**: Microphone or file upload.
2. **ASR Engine**: Gemini Flash 3 Pro transcription.
3. **Schema Linking Layer**: Keyword matching + semantic embeddings.
4. **Prompt Construction**: Assembly with few-shot examples and schema info.
5. **SQL Generation**: Fine-tuned Qwen3-Coder-30B inference.
6. **Query Validation**: Syntax and schema validation.
7. **Execution Engine**: DuckDB SQL execution.
8. **Result Presentation**: Chat interface output.

8.2 Implementation Details

- **Language**: Python 3.10+ with PyTorch.
- **LLM Inference**: vLLM for efficient batch inference.
- **Fine-tuning**: QLoRA for parameter-efficient adaptation.
- **Vector Store**: FAISS for semantic retrieval.
- **Database**: DuckDB for SQL execution.

9 Conclusion and Future Work

We presented the first end-to-end Vietnamese Voice-to-SQL pipeline benchmarked on TPC-DS. Our results show that domain-specific fine-tuning combined with dynamic schema linking allows open-weight models to compete with proprietary giants on specialized tasks. The Data Warehouse Semantics Gap remains a fundamental challenge.

9.1 Key Contributions

1. **First Vietnamese Voice-to-SQL on TPC-DS**: Bridging the gap between academic benchmarks and production warehouse complexity.
2. **Data Warehouse Semantics Gap Characterization**: Explicit documentation of persistent challenges in warehouse SQL generation.
3. **Fine-tuning Effectiveness**: Demonstrating 67% reduction in systematic error classes through domain adaptation.
4. **Practical System Design**: Reproducible modular pipeline enabling on-premise deployment.

9.2 Future Work

- **Latency Optimization**: Quantization to reach sub-500ms latency.
- **Interactive Refinement**: Multi-turn dialogue for iterative query refinement.
- **Explainability**: Natural language explanations of SQL generation decisions.
- **Cross-warehouse Evaluation**: Assessment on TPC-H and custom enterprise schemas.

References

[1] Anh Tuan Nguyen et al., "ViText2SQL: A Dataset for Vietnamese Text-to-SQL Generation," in *Proceedings of VinAI Research*, 2020.

[2] Dawei Gao et al., "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation," *arXiv preprint arXiv:2308.15363*, 2023.

[3] Wang et al., "Schema-Aware Prompting for Text-to-SQL," *arXiv preprint arXiv:2409.12172*, 2024.

[4] Li et al., "SpeechSQLNet: End-to-end Speech-to-SQL Generation," in *Proceedings of Interspeech*, 2020.

[5] Ki et al., "Restoring Vision in Blind ChatGPT: A Pixel is Worth One Word," *arXiv preprint arXiv:2305.09950*, 2023.

[6] Alibaba Qwen Team, "Qwen-Coder: Large Code Language Models," *Technical Report*, 2024. [Online]. Available: https://qwenlm.github.io/

[7] DeepSeek AI, "DeepSeek-Coder: Let the Code LLMs Go Further," *arXiv preprint arXiv:2401.14196*, 2024.

[8] Tim Dettmers et al., "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv preprint arXiv:2305.14314*, 2023.

[9] VinAI, "PhoWhisper: Automatic Speech Recognition for Vietnamese," 2023. [Online]. Available: https://github.com/VinAIResearch/PhoWhisper

[10] TPC Council, "TPC-DS Benchmark." [Online]. Available: https://www.tpc.org/tpcds/

[11] OpenAI, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[12] Wang et al., "The Generalist AI and the Data Warehouse Challenge," *arXiv preprint arXiv:2407.19517*, 2024.