

Execution-Verified Natural Language to PostgreSQL: A Fine-Tuned Small Language Model that Returns the Fastest Query, Equivalent Alternatives and an Explanation

SR Shyamm¹, Jithesh Gopalan Srinivasan²

^{1,2}B. Tech CSE (AIML), SRM Institute of Science and Technology, Kattankulathur, India

Abstract—A generated SQL query can be valid and still be wrong, slow, or correct only by coincidence on today’s data. We present a system that translates questions into PostgreSQL and checks every answer by running it. Qwen3.5 models with 4 and 9 billion parameters are fine-tuned with low-rank adaptation on 95,661 examples from a synthetic multi-dialect corpus and the Spider and BIRD benchmarks, all transpiled to PostgreSQL with sqlglot and verified by execution; prompts include example values and, for BIRD, column descriptions. At inference time, an assistant looks up the values a question mentions in the database, samples nine candidate queries, repairs failing ones, votes on their results, and returns the fastest equivalent query, up to three alternatives of similar speed and an explanation. On 500-question samples, the 9B model reaches 81.6% execution accuracy on synthetic test questions, 79.6% on Spider dev and 55.6% on BIRD dev, 7–10 points above the untuned model.

On BIRD’s official Mini-Dev benchmark in PostgreSQL, scored with its official scripts, the 9B model alone reaches 53.2% and the full assistant 55.6%, against 36.0% for the best published baseline. Listing looked-up values among the example values adds 1.2 points on BIRD dev ($p = 0.017$), whereas naming them in the hint costs 3.0. Returning the fastest equivalent query rarely speeds BIRD answers up (geometric mean 1.05 $\times$), as they already talk about a millisecond, and can favour coincidental matches. The 4B model scores 68.6% exact match on the DBGorilla function-calling benchmark. Both models run on an 8 GB laptop GPU.

Index Terms—Text-to-SQL, PostgreSQL, large language models, low-rank adaptation, execution-guided decoding, query optimization, natural language interfaces to databases

I. INTRODUCTION

Relational databases hold the operational data of hospitals, retailers and banks, yet querying them requires SQL, so people without that skill depend on analysts. Text-to-SQL systems translate a natural language question into an executable query. Large language models (LLMs) have made them far more capable, but three practical problems remain. First, *correctness is hard to judge*: a query can run without error and still answer a different question. Second, *dialects differ*: much public training data mixes MySQL, SQL Server and SQLite syntax, while production databases use one specific engine, in our case PostgreSQL. Third, *equivalent queries differ in cost*: a NOT IN subquery, a LEFT JOIN ... IS NULL and a NOT EXISTS can return the same rows at different speeds, and users rarely see more than one of them.

This paper describes a system that addresses all three by treating the database as the judge. Every query, during training and at inference, is executed on PostgreSQL. The contributions are:

- A data pipeline that converts a synthetic multi-dialect corpus and the Spider and BIRD benchmarks into executable PostgreSQL with sqlglot, preserving SQLite semantics and porting the benchmark databases.
- Fine-tuned Qwen3.5-4B and 9B models trained with an answer-only objective that computes logits only at answer positions, which keeps long schema prompts affordable with a 248k-token vocabulary.
- An execution-verified assistant that looks up the values a question mentions, votes on and repairs

candidate queries, and returns the fastest equivalent query, alternatives of similar speed and an explanation, with guards against coincidental matches and time-dependent date filters.

- A local deployment of both models in Ollama on an 8 GB laptop GPU.
- An evaluation on three test sets, BIRD's official Mini-Dev benchmark in PostgreSQL and the DBGorilla function-calling benchmark, with an error analysis that motivated a second model iteration, a controlled comparison of question-aware prompts, and a measurement of what fastest-query selection gains and where it goes wrong.

II. RELATED WORK

Spider [1] established cross-domain Text-to-SQL evaluation over 200 multi-table databases, and BIRD [2] added large, noisy databases in which data values and external knowledge ("evidence") matter. Its Mini-Dev subset [3] offers 500 of the dev questions in SQLite, MySQL and PostgreSQL, with two further metrics: a reward-based valid efficiency score (R-VES), which weights each correct query by its speed relative to the reference, and a soft F1 score that gives partial credit for matching result cells. These benchmarks compare execution results, so a wrong query that happens to return the right rows counts as correct; our assistant faces the same issue at inference time and adds guards against such coincidental matches (Section V-D).

With LLMs, prompting methods such as DIN-SQL [4] decompose questions and self-correct, and DAIL-SQL [5] studies prompt and example selection; TA-SQL [6] reduces hallucinated schema links and logic by aligning each step with a familiar task. CodeS [7] instead trains open-source models of 1B–15B parameters specifically for Text-to-SQL. Our work follows the fine-tuning line with a small open model, using parameter-efficient LoRA [8] as popularized for single-GPU training by QLoRA [9].

Several outputs can also be combined at decoding time: self-consistency [10] samples them and keeps the majority answer. Our assistant applies this vote to the execution results of complete queries, repairs failing queries with the database's error message, and adds a third use of the database: measuring the run time of equivalent queries to return the fastest one.

Function calling offers an alternative to SQL generation: Gorilla [11] trained models to call APIs, and DBGorilla [12] benchmarks LLMs that query a vector database through a structured tool, which we use to test the fine-tuned model's tool calling.

III. SYSTEM ARCHITECTURE

The system has two parts (Fig. 1 and Fig. 2). An offline pipeline, run in Google Colab on an NVIDIA A100, builds the training data, fine-tunes the model and evaluates it. An online assistant answers questions about a live PostgreSQL database, with the model loaded either as a LoRA adapter in PyTorch or as a merged GGUF file served by Ollama (8-bit for the 4B model, 5-bit for the 9B).

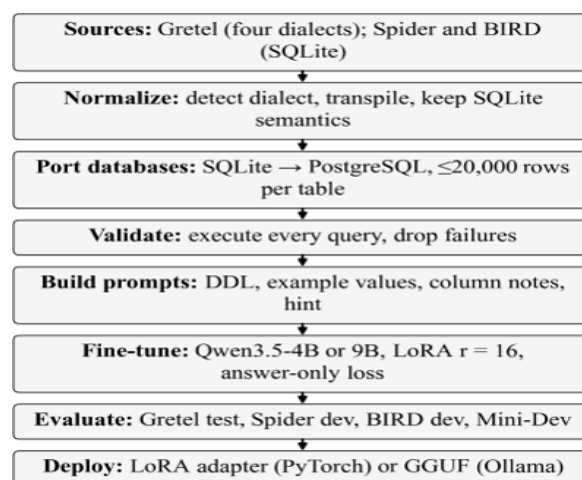


Fig. 1. Offline pipeline: data preparation, fine-tuning, evaluation and export.

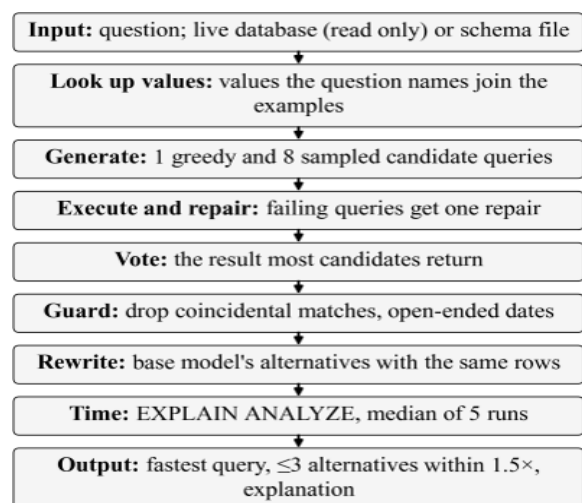


Fig. 2. Online assistant: from a question to a verified, timed answer.

All queries run inside transactions that are always rolled back; on an existing database the session is also READ ONLY, so the assistant cannot change data. Each statement has a time limit (10 s by default).

IV. METHODOLOGY

A. Data Sources

Table I lists the three sources. The Gretel corpus [13] supplies volume and breadth (100 domains, including DML and DDL), but its SQL mixes dialects. Spider [1] and BIRD [2] supply human-written questions over real multi-table SQLite databases; BIRD questions also come with a hint. In total, 95,661 training examples were kept, and 500 more held out for validation.

Table I Training and Evaluation Data After Cleaning

Source	Split	Kept / original
Gretel synthetic text to sql	train	83,415 / 100,000
Gretel synthetic text to sql	test	4,886 / 5,851
Spider	train	6,519 / 7,000
Spider	dev	921 / 1,034
BIRD (filtered train)	train	6,227 / 6,601
BIRD	dev	1,268 / 1,534

B. Dialect Normalization with sqlglot

Each Gretel query is first assigned a likely source dialect from syntax hints, e.g. TOP n or GETDATE () for SQL Server, DATE_SUB or backquotes for MySQL, and STRFTIME for SQLite. It is then parsed with sqlglot [14] in that dialect (falling back to the others), written out as PostgreSQL and made idiomatic, for example by making NULLS ordering explicit. Typical rewrites are:

```
DATE_SUB(CURDATE(), INTERVAL 1 YEAR) ->
CURRENT_DATE - INTERVAL '1 YEAR'
SELECT TOP 5 name ... -> SELECT
name ... LIMIT 5
YEAR(sale_date) ->
EXTRACT(YEAR FROM CAST(sale_date AS DATE))
```

Spider and BIRD queries are written for SQLite, whose semantics differ in ways that change results. Double-quoted strings such as "France" are literals in SQLite but identifiers in PostgreSQL, so they are rewritten; SQLite's case-insensitive LIKE becomes ILIKE; and identifiers are lower-cased to match the ported schemas.

C. Porting the Benchmark Databases

Every Spider and BIRD SQLite database is copied into its own PostgreSQL schema, taking at most 20,000

rows per table. A column receives a numeric or date type only if every copied value fits, and a text column holding only numbers becomes numeric, because SQLite compares such columns with numbers where PostgreSQL would raise a type error. A primary key is added when the data satisfies it, so that PostgreSQL accepts SQLite-style GROUP BY t.id.

D. Execution Validation

Every converted query is executed on PostgreSQL inside a rolled-back transaction: Gretel queries in a scratch schema created from their own CREATE TABLE and INSERT statements, Spider and BIRD queries on their ported databases. Queries that fail are dropped (about 17% of Gretel; Table I). Evaluation sets also drop questions whose reference answer is empty, since any query returning nothing would match them.

E. Prompt Construction

The model receives a fixed system prompt, then the schema as PostgreSQL DDL, an optional hint and the question. After each CREATE TABLE, the prompt lists up to three of the most common short values of every text column, so that the model learns the values the data actually contains (system prompt omitted):

```
Database schema:
CREATE TABLE departments (department_id INT
PRIMARY KEY, name TEXT, city TEXT);
-- example values: name: 'Engineering', 'Sales',
'Support'; city: 'London', 'Berlin'
CREATE TABLE employees (employee_id INT PRIMARY
KEY, department_id INT REFERENCES departments
(department_id), status TEXT);
-- example values: status: 'active', 'left'

Question: Which city has the most active
employees?
```

On a demo database whose country column stores ISO codes, a model trained without example values wrote country = 'Germany' and returned no rows, while Model 2 writes country = 'DE'.

BIRD also describes every column in CSV files. For Model 3, each description is condensed to what the column name does not say: an expanded abbreviation (admlname1: administrator's last name) or how values are coded (charter school (y/n): 0: N; 1: Y). The notes are stored as PostgreSQL column comments and appear as a column notes line after each table's example values, at most 120 characters per column and about 700 tokens per database. Spider and Gretel have no descriptions.

F. Model and Fine-Tuning

The base models are Qwen3.5-4B and Qwen3.5-9B [15], hybrid architectures of 32 layers in which three of every four use gated delta-rule linear attention [16] and every fourth uses full softmax attention. Their reasoning mode is switched off with an empty thinking block in the prompt, in training and at inference. LoRA adapters [8] ($r = 16$, $\alpha = 32$, dropout 0.05) are attached to all linear layers, including those of the linear-attention blocks. Table II compares the three training runs; Model 3 changes the model size and adds column notes at once, in response to the error analysis of Model 2 (Section VIII-B).

Table II Training Configurations

Setting	Model 1	Model 2	Model 3
Base model	Qwen3.5-4B	Qwen3.5-4B	Qwen3.5-9B
Spider, BIRD data	no	yes	yes
Example values	no	3 per column	3 per column
Column notes (BIRD)	no	no	yes
Max. tokens	512	3,072	4,096
Examples per update	16	16 (8×2)	16 (4×4)
Steps (1 epoch)	5,183	5,924	5,940
Learning rate	2×10^{-4} , 3% warm-up		
Precision, LoRA	bf16; $r = 16$, $\alpha = 32$		

V. ALGORITHMIC FRAMEWORK AND MATHEMATICAL MODEL

A. Answer-Only Training Objective

A training example consists of a prompt x (system prompt, schema with example values, hint and question) and an answer $y = (y_1, \dots, y_T)$, the SQL query. Only answer tokens contribute to the loss:

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{A}|} \sum_{t \in \mathcal{A}} \log p_\theta(y_t | x, y_{<t}), \quad (1)$$

where $\mathcal{A}$ is the set of answer positions in the batch. A standard implementation computes logits for every position and masks the prompt afterwards; with a vocabulary of $V = 248,320$ tokens and prompts of up to 3,072 tokens, the logits alone would hold $8 \times 3,072 \times V \approx 6.1 \times 10^9$ values per batch. Our trainer applies the output layer only at positions in $\mathcal{A}$, so the cost scales with the length of the SQL rather than of the schema, with the same loss to within 10^{-6} .

LoRA keeps each pre-trained weight $W_0 \in \mathbb{R}^{d \times k}$ frozen and learns a low-rank update:

$$W = W_0 + \frac{\alpha}{r} BA, \quad B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}. \quad (2)$$

B. Question-Aware Value Lookup

Example values show only the three most common values of a column, whereas a question often names a rare one: an administrator called Avetik Atoian, or charter number 40, stored as '0040'. At the first question, the assistant indexes, for every text column c , the set V_c of distinct values of 1–60 characters among its first 50,000 rows (5 s for the 11 BIRD dev databases). Values and questions are compared in a normal form $n(\cdot)$ that is case-folded and turns every run of characters other than letters and digits into a space. For a question Q , let $P(Q)$ be the word n -grams of $n(Q)$ with $n \leq 6$, longest first and without overlaps, together with strings quoted in the question or hint. The matches are

$$M(Q) = \{ (p, c, v) : p \in P(Q), v \in V_c, n(v) = p \}. \quad (3)$$

Single words are skipped if they are stop words, shorter than three characters or part of a table or column name (“user” usually means the users table); the hint contributes only quoted strings, because its other words often name SQL functions (“max” matched a driver called Max); and a number matches only a zero-padded value, when one of the three preceding words belongs to the column’s name. Each matched value is listed first in its column’s example values line, so the prompt stays in the training format.

C. Execution-Guided Voting

For a question, the fine-tuned model produces a greedy candidate c_0 and $K = 8$ sampled candidates $c_1, \dots, c_K$ (temperature 0.8, nucleus 0.95). Each is trans piled to PostgreSQL, duplicates are removed, and each remaining candidate is executed, giving a result $E(c)$ (a multiset of rows) or an error. Candidates are grouped by result, and the answer is the largest group:

$$r^* = \arg\max_r |\{ c : E(c) = r \}|, \quad (4)$$

with ties resolved in favour of the group that contains the earliest candidate, so the greedy answer wins ties.

D. Equivalence Guards

Returning the same rows on the current data does not make two queries equivalent: in our demo database, customers 18,001–20,000 never order, so WHERE customer_id > 18000 returns exactly the customers without orders, and an early version of the assistant reported it as the fastest answer. A candidate q is therefore compared with the reference q_{ref} (the greedy

answer when it is in the winning group). With $T(\cdot)$ the set of tables read and $K(\cdot)$ the set of constants (a date literal also stands for its year), q is kept only if

$$\begin{aligned} T(q) \subseteq T(q_{\text{ref}}) \vee T(q_{\text{ref}}) \subseteq T(q), \\ K(q) \setminus \{0,1\} \equiv K(q_{\text{ref}}) \setminus \{0,1\}. \end{aligned} \quad (5)$$

Two temporal patterns that pass this test but fail on future data are flagged: BETWEEN ... AND '2026-12-31' on a timestamp column, which misses the last day after midnight, and order_date >= '2026-01-01' without an end date when the question names a year. Flagged queries are set aside if a clean equivalent exists; otherwise, the model is asked for a fixed version, kept only if it returns the same rows.

E. Fastest-Query Selection and Alternatives

Each equivalent query q is timed with EXPLAIN (ANALYZE, BUFFERS). After one warm-up run, it is run five times, each inside a savepoint that is rolled back, and its time is the median of planning plus execution time:

$$\begin{aligned} \tau(q) &= \text{median}_{k=1..5} \left(t_{\text{plan}}^{(k)}(q) + t_{\text{exec}}^{(k)}(q) \right), \\ q^* &= \underset{q}{\operatorname{argmin}} \tau(q). \end{aligned} \quad (6)$$

More formulations come from the base model (the LoRA adapter switched off), asked for rewrites using joins, EXISTS, common table expressions or window functions. A rewrite becomes an alternative if it returns r^* , passes the guards, and satisfies

$$\begin{aligned} \tau(q) &\leq \max(\lambda \tau(q^*), \tau(q^*) + \delta), \\ \lambda &= 1.5, \delta = 0.05 \text{ ms}, \end{aligned} \quad (7)$$

and if its structure (the sequence of syntax-tree node types without identifiers, literals and select list) differs from the fastest query and from alternatives already chosen. Finally, the base model writes a two- to four-sentence explanation of the fastest query.

F. Error Repair

A candidate that fails to execute receives one repair attempt (at most two per question): the failed query and the database's error message are placed in the prompt's hint line, the format the model saw for BIRD hints during training, and the model writes a new query that joins the vote if it runs.

VI. DEPLOYMENT AND INTERFACES

The assistant is a Python module with a command-line interface and a companion Colab notebook. It accepts a live database, whose schema, example values and

column notes are read from the PostgreSQL catalog, or a schema file in any SQL dialect with optional INSERT rows, loaded into a scratch schema for every query. The report shows the fastest query with its time, the alternatives with their speed relative to it, and the explanation; on request it adds plans, costs and notes such as disagreement warnings, repairs and the values found in the database.

PyTorch. On a laptop with an 8 GB NVIDIA RTX 4060, the model runs in 4-bit NF4 precision (peak 4.3 GB) with the adapter on top of the base model, which also writes alternatives and explanations at no extra memory cost. A full answer takes 30–100 s, largely because the fast linear-attention kernels need Triton, which is unavailable on Windows.

Ollama. The merged models were converted with llama.cpp [17] to GGUF and served by Ollama [18]: Model 2 in 8 bits (4.9 GB of GPU memory, about 1 s per query) and Model 3 in 5 bits (6.8 GB, 1–2 s), since its 8-bit file (9.5 GB) does not fit. On 60 test questions, the 8-bit Model 2 wrote the same SQL as the bf16 model for 57, and the 5-bit Model 3 for 51 (46 in 4 bits); complete assistant answers took 10–38 s. Three deployment problems silently changed answers.

The GGUF file carried the base model's chat template, which Ollama preferred over ours and which enabled a thinking mode; a template that reproduces the training prompt without the literal thinking tags fixed it. Ollama's desktop app resends the whole conversation, so after a greeting answered with a degenerate query, every later answer copied it; the template now renders only the latest message, and it rebuilds the training prompt from a schema file attached in the app, so a user can attach the schema and type only the question. Finally, a Modelfile saved with Windows line endings put carriage returns into the template, which changed 3 of the 60 answers.

VII. EXPERIMENTAL SETUP AND EVALUATION

A. Benchmarks and Metrics

Each test set (Gretel test, Spider dev, BIRD dev) is shuffled with a fixed seed and its first 500 questions are scored, the same for every model. A prediction is valid if it passes as PostgreSQL, executes if it runs, is an exact match if it equals the reference after sqlglot normalization, and is correct under execution accuracy (EX) if it returns the same multiset of rows as the

reference (in order if the reference has a top-level ORDER BY). We also report EX with rows compared as sets, BIRD's official rule. Because the databases are capped copies and questions with empty answers were removed, these numbers are not directly comparable with leaderboards; Mini-Dev, run on its official database with its official scripts (Section VII-D), is. For Model 3 we also score execution-guided voting over the greedy query and eight sampled ones (temperature 0.7, nucleus 0.95), and the untuned Qwen3.5-9B with the same prompts.

B. Main Results

Table III shows the results. Adding Spider and BIRD data and example values raised execution accuracy on the same 500 Gretel questions from 75.6% to 81.4% (39 questions fixed, 10 broken). Moving to the 9B model with column notes (Model 3) gained 4.0 points on BIRD but only 0.2 on Gretel and Spider, where fixed and broken questions nearly cancel. Fine-tuning adds 6.8 to 9.8 points over the untuned 9B model, which also writes valid PostgreSQL but rarely in the references' style. Voting helps most where single answers fail to run: on BIRD it raises the share of queries that execute from 95.2% to 99.2% and accuracy by 2.2 points (15 fixed, 4 broken), on Spider by 1.6 and on Gretel by 0.2. Model 2's set-rule score on BIRD comes from the local re-run of Section VIII-B. Nearly every prediction is valid PostgreSQL; the remaining gap is semantic.

Table III Results on 500-Question Samples (%)

Model	Valid	Executes	Exact	EX	EX (set)
Gretel test					
Model 1	99.8	98.2	35.8	75.6	–
Model 2	100.0	99.6	39.8	81.4	–
Model 3	99.8	99.0	40.6	81.6	82.0
Model 3 + voting	100.0	100.0	40.0	81.8	82.4
Untuned 9B	99.8	94.4	22.4	74.8	74.8
Spider dev					
Model 2	100.0	97.0	59.4	79.4	–
Model 3	100.0	97.4	61.0	79.6	81.2
Model 3 + voting	100.0	99.6	61.0	81.2	82.8
Untuned 9B	100.0	96.0	30.0	69.8	71.8
BIRD dev					
Model 2	99.6	93.8	15.8	51.6	59.0
Model 3	99.6	95.2	18.8	55.6	62.2
Model 3 + voting	100.0	99.2	19.2	57.8	64.4
Untuned 9B	98.2	89.2	5.0	46.8	51.8

By Gretel query type, Model 1 to Model 2 gained most on basic queries (78.2% to 86.1%), aggregation and subqueries, while window functions remain the weakest type (4 of 13 correct for Model 3).

C. BIRD by Difficulty and Scoring Rule

Scoring rows as sets, as BIRD does, raises Model 2's BIRD accuracy from 51.6% to 59.0% (Table IV); 32 of the 37 answers that become correct differ from the reference only by repeated rows from a missing DISTINCT. Model 3's gains grow with difficulty: 2.1 points on simple, 5.1 on moderate and 11.5 on challenging questions (6 fixed, none broken). The untuned 9B model already answers 32.7% of challenging questions, more than Model 2, so on the hardest questions model size appears to matter most.

Table IV. BIRD Dev Execution Accuracy (%), Multiset Rule / Set Rule (BIRD)

Difficulty	n	Model 2	Model 3	Model 3 + voting
Simple	292	58.9 / 65.4	61.0 / 67.5	62.3 / 68.8
Moderate	156	46.8 / 53.8	51.9 / 57.7	55.8 / 61.5
Challenging	52	25.0 / 38.5	36.5 / 46.2	38.5 / 48.1
All	500	51.6 / 59.0	55.6 / 62.2	57.8 / 64.4

D. Official Benchmark: BIRD Mini-Dev in PostgreSQL

Mini-Dev [3] consists of 500 BIRD dev questions chosen by the benchmark's authors (148 simple, 250 moderate, 102 challenging, a harder mix than the sample of Table IV) over the full-size dev databases in an official PostgreSQL version. We used the July 2025 release with the reference queries of its canonical Hugging Face copy and BIRD's evaluation scripts at their defaults (16 processes, 30 s per query), changing only the database connection; EX here follows the set rule. As a check, the scripts score BIRD's published PostgreSQL predictions of GPT-4 and GPT-4-turbo at 35.4% and 36.2%, against the reported 35.8% and 36.0%, computed before the references were revised. Prompts were built by the assistant's live-database code from the official database, with Model 3's column notes matched to its columns by name. The official database keeps names such as "Examination Date" in mixed case, which training had lower-cased;

the model copied them correctly in all but one query. No Mini-Dev question occurs in the training data, and Model 3 ran in Ollama with 5-bit weights.

Table V. BIRD Mini-Dev in PostgreSQL, Official Scripts (%)

System	EX	R-VES	Soft F1
Ours: Model 3, 5-bit, Ollama			
Model alone ^a	53.2	50.7	51.9
Model + value lookup	53.2	50.7	53.2
Full assistant	55.6	53.3	55.3
Published baselines [3]			
GPT-4-turbo	36.0	35.2	38.4
GPT-4	35.8	35.2	39.0
GPT-4-32k	35.0	34.6	39.6
Llama-3-70B-Instruct	29.4	28.8	31.4
GPT-3.5-turbo	27.4	26.8	30.2
^a Mean of two runs and repeated scorings; EX ranged from 53.0 to 53.4.			

The model alone answers 53.2% of the questions correctly (70.3% of simple, 50.0% of moderate and 36.3% of challenging ones), 17 points above the best published baseline (Table V). Those baselines are prompted models without fine-tuning, value lookup or voting; methods with schema and value processing, such as TA-SQL [6], have been reported on Mini-Dev only for SQLite (63.0% with GPT-4o). The full assistant reaches 55.6% (72.3%, 52.8%, 38.2%), the same 2.4-point gain as on BIRD dev: queries that fail to run fall from 31 to 4, and against the model alone it fixes 24 answers and breaks 11 or 12 (exact McNemar $p = 0.04$ and 0.07 for the two runs), a borderline result. The value lookup alone made no difference (8 fixed, 8 broken), so its gain on BIRD dev did not carry over to this harder selection. R-VES per correct answer barely changes (0.953 for the model alone, 0.959 for the assistant's fastest queries), in line with Section VII-H. Greedy decoding was not bit-exact on the GPU: a second run changed 31 of the 500 queries, mostly into equivalent or equally wrong ones, and scored 53.0%, so differences below about half a point are noise. The bfl6 model, 1.2 points above the 5-bit build on BIRD dev, may score somewhat higher.

E. Function Calling: DBGorilla

DBGorilla [12] poses 315 questions about five synthetic databases. A model answers by calling a `query_database` tool whose arguments (collection, search query, filters, aggregations, group-by) are compared with the ground truth; exact match requires

every compared component to agree. We reproduced the published protocol (tool definition, prompt, temperature 0), and our reimplementations of the scorer reproduces all 2,520 stored scores of the eight published models exactly.

Table VI. DBGorilla Results (Official Protocol)

Model	EM (%)	AST	Routing (%)	No call (%)
Claude 3.5 Sonnet	74.3	0.973	96.8	3.2
GPT-4o mini	73.7	0.952	97.5	1.0
GPT-4o	71.7	0.966	96.8	2.9
Gemini 1.5 Pro	70.2	0.956	93.0	5.4
Ours (4B, 8-bit)	68.6	0.959	98.1	1.9
Command R+	59.4	0.933	94.3	4.4
Command R7B	39.0	0.887	84.4	13.3
Gemini Flash	37.1	0.975	46.0	54.0
Llama 3.1 8B	32.1	0.893	76.8	21.9

The fine-tuned 4B model (Model 2) keeps its tool-calling ability (Table VI): it scores 68.6% exact match, within about three points of GPT-4o, with the highest collection routing accuracy of all models; most misses (58 of 93) disagree with the ground truth about whether to use a semantic search query. In a SQL variant, where the model queries the collections as tables, only the 155 questions without a search query are comparable; with two allowances for SQL, our model matches the ground truth for 85.2% of them, equal to Claude 3.5 Sonnet and below GPT-4o (87.7%) and GPT-4o mini (91.6%).

F. Error Repair

Every query of Model 2 that failed to execute in the Colab evaluation was given one repair attempt with its error message. On Spider, 8 of 15 then ran and 4 were correct; on BIRD, 11 of 31 ran and 3 were correct, about +0.8 and +0.6 points for single answers. A conversational follow-up instead of the hint line fixed no more, so the simpler format was kept.

G. Value Lookup

The value lookup was measured with Model 3 in Ollama (5 bits), one greedy query per question, on the local copies, with every prompt rebuilt from the training prompt so that only the tested element differs. Three changes were tried on the 500 analysed BIRD questions (Table VII): listing the matched values among the example values, naming them in a clause

added to the hint, and letting the notes of the columns a question is about claim the notes budget first. The best change was then confirmed on the other 768 BIRD dev questions, which played no part in designing or choosing it.

Table VII. Question-Aware Prompts on BIRD Dev: Model 3 in Ollama, One Query per Question (EX / Set Rule, %)

Prompt	500 analysed	Fixed/broken	768 held out
Training prompt	54.4 / 61.4	—	55.1 / 61.3
Values as examples	55.4 / 62.8	8 / 3	56.4 / 62.9
Values in the hint	51.4 / 58.4	9 / 24	—
Question's notes first	54.2 / 61.4	4 / 5	—

Listing the matched values among the example values helped on both sets. On all 1,268 questions, execution accuracy rose from 54.8% to 56.0% and set-rule accuracy from 61.4% to 62.9%, with 25 answers fixed and 10 broken (exact McNemar test: $p = 0.017$ and $p = 0.004$). The clearest fixes are the two error types the lookup targets: a value stored differently (charternum = '40' became '0040') and a value in an unexpected column (admfname2 = 'Atoian' became admfname1 = 'Avetik' AND admfname1 = 'Atoian'). Most broken answers changed in ways unrelated to the listed values, such as the order of output columns. The lookup listed a value for 68.6% of the questions, and 74.0% of the listed values occur in the reference query. On 500 Spider dev questions, which name fewer values, accuracy rose from 79.8% to 80.4%. The local baseline is 1.2 points below the bf16 model's 55.6% (Table III), which reflects the 5-bit build, since the local copies reproduce Colab's verdicts on 499 of 500 questions.

Naming the same values in the hint instead cost 3.0 points (9 fixed, 24 broken). The model applies every value in a hint as a filter, as BIRD's hints taught it, so loose matches became conditions: "in descending order of dates" matched a card face named Order and produced AND facename = 'Order'. Among the example values, which the model treats as candidates rather than instructions, the same matches did little harm. Letting the question's columns claim the notes budget first fixed 4 answers and broke 5, so note order is not a bottleneck.

H. What Fastest-Query Selection Gains

To measure the feature in the title, the assistant answered the 500 analysed BIRD dev questions as deployed (Model 3 in 5 bits). Its answers were correct for 57.6% (64.0% under the set rule), against 55.2% (62.4%) for its own greedy candidate. For 416 questions the greedy query was in the winning group, and for 287 of them a different formulation was returned as the fastest. Because the minimum of several noisy timings overstates speed-ups, the greedy and returned queries were re-timed in fresh interleaved rounds, on the capped copies and on full-size copies of the databases. The median speed-up is $1.00 \times$ and the geometric mean $1.05 \times$ on both; 2.6% of the questions became at least twice as fast, and for the 78 greedy queries taking at least 10 ms on full data the geometric mean is $1.01 \times$. BIRD's queries are simply fast: on full data they take a median of about 1 ms and at most 162 ms, and equivalent formulations usually share a plan. Under R-VES the assistant scores 61.7 against 60.2 for the greedy query; VES rose more, but one 18-second reference query accounts for 6.6 points of it.

Of the 20 questions sped up by $1.5 \times$ or more, 11 are genuine rewrites, such as a redundant join removed (6.8 to 1.5 ms) or a cheap predicate written first (18.7 to 9.3 ms). The other 9 return the same rows only on these data and pass the guards because their tables and constants match. A contradictory filter (ssb = 'negative' AND ssb = '0') matched only because the correct count is zero; PostgreSQL recognised the condition as never true, skipped the scan, and the query looked 36 times faster. Others read a denormalised copy of a joined column, add a redundant filter or compute a different formula, and the full-size data exposed only 2 of the 9. Selecting for speed thus favours the queries that do least work, including those that skip it by accident. On BIRD, the feature's value lies in the verified, timed alternatives; a speed benefit needs heavier analytical queries, such as those of the case study (Section VII-I).

I. Case Study

The demo database has 20,000 customers, 500 products, 200,000 orders and 400,000 order items, with indexes on the foreign keys and the order date. For the question "List customers who have never placed an order.", Model 2 in 4-bit PyTorch returned, in 30 s:

```

1. Fastest query (18.601 ms, cost 4,850.1, 374
pages)
  SELECT c.name FROM customers AS c
  WHERE NOT c.customer_id IN (SELECT DISTINCT
o.customer_id FROM orders AS o);
2. Alternatives (same result, similar speed)
  a) 1.26x ... FROM customers LEFT JOIN orders
ON ... WHERE orders.customer_id IS NULL;
  b) 1.28x ... WHERE NOT EXISTS(SELECT 1 FROM
orders AS o WHERE o.customer_id = c.customer_id);
  c) 1.48x ... NOT IN (SELECT o.customer_id FROM
orders AS o WHERE o.status IS NOT NULL);
3. Explanation: This query reads data from the
customers and orders tables by using a subquery
to identify all customer IDs that have placed
at least one order...

```

The DISTINCT lets PostgreSQL answer the subquery from the index on orders.customer_id (a Unique over an Index Only Scan), which makes this formulation the fastest. Alternative (c) is equivalent only because status is declared NOT NULL. For “How many orders did each customer from Germany place in 2026?”, one run chose order_date >= '2026-01-01' with no end date, which returned the right rows only because no 2027 orders existed yet; this case led to the open-ended date check, which now sets such a query aside and has it repaired to EXTRACT(YEAR FROM ...) = 2026.

VIII. RESULTS AND DISCUSSION

A. Interpretive Analysis

Five results stand out. First, training data mattered: with the model size unchanged, real-database examples and example values gained 5.8 points on Gretel, while the larger model with column notes helped only where questions are hard (11.5 points on challenging BIRD questions, 0.2 on Gretel and Spider). Second, verification by execution is cheap compared with generation and catches failures that prompt design does not prevent; voting added 2.2 points on BIRD dev, and the full assistant 2.4 on Mini-Dev, on top of a fine-tuned 9B model that already outscores every published PostgreSQL baseline. Third, where information enters the prompt matters as much as the information itself: the same database values gained 1.0 point as example values and cost 3.0 as a hint clause, consistent with the model rarely ignoring a hint’s values (Section VIII-B). Fourth, the fine-tuned 4B model still calls tools well, approaching GPT-4o on DBGorilla, although the untuned model was not evaluated. Fifth, choosing the fastest equivalent query is only as safe as the equivalence check: on BIRD it gained little speed, and nearly half of its largest speed-ups came from queries that matched by coincidence.

B. Error Analysis on BIRD

The wrong BIRD answers of both models were re-run on local copies (agreeing with Colab’s verdict on 497 and 499 of 500 questions) and each was assigned to one cause by fixed rules applied in order (Table VIII). The analysis of Model 2 shaped Model 3. Model 2’s largest group was schema linking: the wrong table, a column attached to the wrong table (21 queries fail outright this way) or joins that leave no rows. Repeated rows formed another large group; most of it is correct under the set rule, but 19 answers count rows inflated by a join. Hints were rarely ignored: only 16 of the 212 wrong answers with a hint (10 of 194 for Model 3) left out a value it gave.

Model 3 makes 20 fewer errors (44 fixed, 24 broken). The two groups that column notes target shrink most: schema-linking errors fall from 66 to 57, with failures on a misplaced column down from 21 to 13, and value errors from 27 to 21, with filters on a value absent from the data down from 12 to 4. Repeated rows, which neither change addresses, now form the largest group. Because model size and prompts changed together, these differences cannot be attributed to either alone.

Table VIII. Causes of Wrong BIRD Dev Answers

Cause	Model 2	Model 3
Wrong table, column or join (schema linking)	66	57
Repeated rows: missing DISTINCT (42; 43), COUNT inflated by a join (19; 19)	61	62
Wrong formula, aggregate or other logic	63	59
Wrong or missing filter value (e.g. '40' for '0040')	27	21
Extra or missing output columns, row order	15	12
Other execution errors (GROUP BY, truncated query, types)	10	11
Total	242	222

C. Limitations

Equivalence is established on the current data and schema, which is strong evidence but not proof, and the temporal checks cover two common patterns. The guards compare only tables and constants, so contradictory or redundant filters can pass them, and selecting for speed makes such queries more likely to win. Most evaluations use 500-question samples and capped copies, so differences of a few points are within noise. The value lookup’s matching is lexical:

a value the question names differently, such as “Czech Republic” for 'CZE', still needs a hint. Model 3 changes model size and prompts at once, so their effects are not separated. Mini-Dev was run with the 5-bit build and the full assistant only once. DBGorilla’s scorer ignores filters and aggregations that appear in only one of prediction and ground truth, which makes exact match lenient; we kept it for comparability. Latencies were measured on a single laptop.

IX. CONCLUSION

A. Summary of Contributions

We presented a Text-to-PostgreSQL system in which the database verifies every step. Fine-tuned on executable PostgreSQL data built with sqlglot, a 9B model reaches 81.6% execution accuracy on synthetic test questions, 79.6% on Spider dev and 62.2% on BIRD dev under BIRD’s rule (64.4% with voting), and a 4B model nearly matches it except on BIRD. On BIRD’s official Mini-Dev benchmark in PostgreSQL, the 9B model alone scores 53.2% and the full assistant 55.6%, against 36.0% for the best published baseline. The assistant returns the fastest equivalent query, alternatives of similar speed and an explanation, guards against coincidental and time-dependent matches, and runs on an 8 GB laptop GPU. Its gains come from verification rather than speed: voting and repair add 2.2–2.4 points on BIRD, whereas the fastest query is only 5% faster on average. Listing looked-up values among the example values adds 1.2 points on BIRD dev, whereas the same values in the hint cost 3.0 points.

B. Future Work

Training the 4B model with column notes would separate the effect of the notes from that of model size. The error analysis points to repeated rows, now the largest error group, which a duplicate-row check could address, and to values named differently from the data, which need semantic matching. Before speed can safely decide between queries, the equivalence check needs strengthening: plans in which PostgreSQL proves a filter never true can be rejected, and a faster query could be required to agree with the reference on perturbed or sampled data; measuring the speed benefit itself needs heavier analytical workloads than BIRD’s. Reinforcement learning with an execution-

based reward, for example group relative policy optimization [19], could teach the model directly from the database’s verdict.

ACKNOWLEDGMENT

We thank the Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning) at SRM Institute of Science and Technology, Kattankulathur, and our project guides and faculty for their support and feedback. We also thank the creators of the Spider, BIRD, Gretel synthetic Text-to-SQL and DBGorilla datasets, and the maintainers of sqlglot, PostgreSQL, Hugging Face Transformers and PEFT, llama.cpp and Ollama, whose open work made this project possible.

REFERENCES

- [1]. T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task,” in *Proceedings of EMNLP*, 2018, pp. 3911–3921.
- [2]. J. Li et al., “Can LLM Already Serve as a Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [3]. BIRD Team, “BIRD-SQL Mini-Dev,” GitHub repository and dataset, 2024, updated 2025. Available: https://github.com/bird-bench/mini_dev
- [4]. M. Pourreza and D. Rafiei, “DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [5]. D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, “Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation,” *Proceedings of the VLDB Endowment*, vol. 17, no. 5, 2024.
- [6]. G. Qu et al., “Before Generation, Align It! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation,” in

- Findings of the Association for Computational Linguistics (ACL), 2024, pp. 5456–5471.
- [7]. H. Li et al., “CodeS: Towards Building Open-Source Language Models for Text-to-SQL,” Proceedings of the ACM on Management of Data (SIGMOD), 2024.
 - [8]. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” in Proceedings of the International Conference on Learning Representations (ICLR), 2022.
 - [9]. T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “QLoRA: Efficient Finetuning of Quantized LLMs,” in Advances in Neural Information Processing Systems (NeurIPS), 2023.
 - [10]. X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” in Proceedings of the International Conference on Learning Representations (ICLR), 2023.
 - [11]. S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, “Gorilla: Large Language Model Connected with Massive APIs,” arXiv:2305.15334, 2023.
 - [12]. C. Shorten et al., “Querying Databases with Function Calling,” arXiv:2502.00032, 2025.
 - [13]. Y. Meyer et al., “Synthetic-Text-To-SQL: A Synthetic Dataset for Training Language Models to Generate SQL Queries from Natural Language Prompts,” Gretel, 2024. Available: https://huggingface.co/datasets/gretelai/synthetic_text_to_sql
 - [14]. T. Mao, “SQLGlott: Python SQL Parser, Transpiler, Optimizer, and Engine.” Available: <https://github.com/tobymao/sqlglot>
 - [15]. Qwen Team, “Qwen3.5-4B and Qwen3.5-9B,” model cards, 2026. Available: <https://huggingface.co/Qwen/Qwen3.5-4B>; <https://huggingface.co/Qwen/Qwen3.5-9B>
 - [16]. S. Yang, J. Kautz, and A. Hatamizadeh, “Gated Delta Networks: Improving Mamba2 with Delta Rule,” in Proceedings of the International Conference on Learning Representations (ICLR), 2025.
 - [17]. G. Gerganov et al., “llama.cpp: LLM Inference in C/C++.” Available: <https://github.com/ggml-org/llama.cpp>
 - [18]. Ollama, “Ollama.” Available: <https://ollama.com>
 - [19]. Z. Shao et al., “DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models,” arXiv:2402.03300, 2024.