ISSN: 2633-4828 Vol. 5 No.4, December, 2023

International Journal of Applied Engineering & Technology

ENHANCING STRUCTURED QUERY GENERATION THROUGH FINE-TUNED SMALL LANGUAGE MODELS: TOWARDS ROBUST AND EFFICIENT SQL SYNTHESIS

Ravi Kiran Vadlamani[1] and Dharmateja Priyadarshi Uddandarao[2]
[1]Carnegie Mellon University, Pittsburgh, USA
[2]Northeastern University, Boston, USA
[1]Kiranvadlamani94@gmail.com and [2]dharmateja.h21@gmail.com

ABSTRACT

We explore the fine-tuning of **open-source small language models** (with fewer than 12B parameters) for text-to-SQL generation in the financial domain. Using a filtered variant of the Spider benchmark – focusing on financial database queries – we fine-tune models including LLaMA 7B, CodeGen2 (6B), GPT-NeoX 6.7B, and Falcon 7B. We evaluate these models on SQL synthesis tasks with financial-domain questions, comparing their performance against prior state-of-the-art (SOTA) text-to-SQL systems. Our experiments measure exact match accuracy, execution accuracy, BLEU score, and other relevant metrics to assess both the correctness and robustness of generated SQL. The results demonstrate that fine-tuned small models can achieve substantial accuracy gains over zero-shot baselines and approach the performance of larger models on domain-specific queries. In particular, a CodeGen2-6B model fine-tuned on financial data achieves the highest exact-match and execution accuracy among the tested models, narrowing the gap to prior SOTA systems. We discuss how domain-focused fine-tuning improves SQL synthesis reliability, and we highlight the efficiency benefits of these smaller models – which require significantly less computational resources – making them attractive for practical deployment in financial analytics. Our findings suggest that carefully fine-tuned small LMs can serve as robust and efficient alternatives for structured query generation in specialized domains.

INTRODUCTION

Natural language interfaces to databases – the task of translating a user's question into a SQL query – have gained widespread attention in NLP and data management. This text-to-SQL problem is a form of semantic parsing that requires understanding a user’s query and mapping it to an executable SQL statement [1]. Recent benchmarks like **Spider[1] and WikiSQL[2]** have catalyzed research in this area**.** Spider is a cross-domain text-to-SQL dataset with 10,181 questions and 5,693 SQL queries across 138 different domains[1], designed to require generalization to new database schemas and complex queries. WikiSQL, conversely, provides 87,726 pairs of natural questions and SQL on single-table Wikipedia datasets, covering a broad range of topics but simpler query structures. These benchmarks have driven progress, with advanced neural techniques achieving high accuracies (e.g., >70% exact-match on Spider) in recent years [9].

However, **financial and business domains** remain challenging for text-to-SQL systems. Existing broad benchmarks **fall short on coverage of finance** and accounting use-cases. For example, a state-of-the-art model that reached 80.5% exact-match on Spider’s general test set achieved only 10.8% on a new financial dataset[10], highlighting poor domain generalization. Even large proprietary models like GPT-4 have shown significant performance gaps on specialized financial query tasks[10]. This motivates developing **focused models** tailored to the financial domain, which contains unique terminology and data patterns (e.g. revenue, expenses, stock prices, accounting tables). By fine-tuning models on financial-domain text-to-SQL data, we aim to improve their understanding of such queries and the correct generation of SQL involving financial schemas.

Crucially, we focus on **open-source small language models (<12B parameters)** for this task. While massive models (like GPT-3/4) have demonstrated remarkable text-to-SQL abilities, their size and proprietary nature pose practical limitations. We investigate whether *small, publicly available LMs* can be fine-tuned to achieve robust SQL synthesis, offering a lightweight and accessible solution. We select a representative set of models: **LLaMA 7B[3]**, **CodeGen2 6B [4]**, **GPT-NeoX 6.7B [5]**, and **Falcon 7B [7]**. These models are all recent open-source, known for strong performance in general or code generation tasks. LLaMA, for instance, is a foundation model family ranging from 7B to 65B parameters that showed even a 13B model can outperform GPT-3 (175B) on many benchmarks[3]. CodeGen2 models are specifically tuned for code synthesis [4]. GPT-NeoX models provide open alternatives with performance on par with GPT-3 Curie [6]. Falcon-7B, trained on 1.5 trillion tokens of refined web text, has shown best-in-class performance among 7B models [7].

Contributions: In this work, we (1) curate a financial-domain text-to-SQL dataset derived from the Spider benchmark, focusing on databases and queries relevant to finance (e.g. corporate budgets, sales records, banking transactions); (2) fine-tune four open-source language models (7B-scale or smaller) on this dataset, using efficient adaptation techniques to handle the limited model sizes; (3) evaluate the models on a hold- out test set of financial-domain queries using multiple metrics – exact match, execution accuracy, BLEU, and component-wise accuracy – to thoroughly assess SQL generation quality; and (4) compare our fine- tuned models against prior state-of-the-art results and larger models, analyzing the trade-offs in accuracy versus model size and highlighting the robustness and efficiency gains for domain-specific SQL synthesis.

Our experiments show that all fine-tuned small LMs dramatically outperform zero-shot performance and approach the accuracy of prior larger models on financial queries. For example, the best model (CodeGen2-6B) achieves around 63–65% exact match accuracy on our financial test set – a substantial improvement over <5–10% without fine-tuning, and only ~8–10 points behind much larger T5-based models reported on the general Spider task. We also demonstrate that these models generate syntactically correct and executable SQL in most cases (execution accuracy closely tracks exact match), and maintain strong domain fidelity. Notably, the fine-tuned models are efficient: Falcon-7B and LLaMA-7B can run on a single GPU or even high-end consumer hardware , enabling low-cost deployment of an NLP-powered financial query interface.

The rest of this paper is organized as follows. Section II describes the dataset and task setup, as well as the small LMs and our fine-tuning approach. Section III presents the experimental results on SQL generation metrics, including comparisons with baselines and analysis of errors. Section IV discusses implications, including model efficiency and robustness considerations. We conclude in Section V with a summary of findings and future directions for enhancing text-to-SQL generation in specialized domains.

METHODOLOGY

Financial Domain Text-to-SQL Dataset

To train and evaluate our models, we derived a financial-domain text-to-SQL benchmark based on the Spider dataset [1]. Spider's broad coverage (200 databases across numerous domains) includes some databases relevant to finance (e.g., college financials, payment records), but these are limited. We therefore constructed a focused dataset by filtering Spider for any databases and questions pertaining to financial topics (such as budgets, salaries, sales, accounts, etc.), and supplementing with similar structured queries from other public sources. The resulting dataset, which we call "Spider-Financial", maintains the original Spider format: each example consists of a natural language question, a database schema (tables with columns and types, plus foreign keys), and the ground-truth SQL query. By using Spider as the base, we inherit its train/dev split methodology requiring generalization to unseen databases . In other words, the training set questions come from a distinct set of financial databases than those in the test set, to ensure models must generalize to new schemas. . This setup tests robustness – a model must synthesize SQL for novel tables using only the question and provided schema. We reserved ~20% of the collected examples for validation and another ~20% for final testing, ensuring a variety of SQL complexity (simple selections, aggregate queries, JOINs, nested queries).

The training set contains approximately 5,000 questions after augmentation, covering topics such as corporate budget ledgers, stock trading records, loan databases, and sales transactions. Each question is paired with a gold SQL query written for the corresponding database. We ensured a broad range of difficulty: some are straightforward single-table queries ("What is the total revenue for 2022?"), while others require multi-table joins or nested subqueries typical in financial analytics ("Find all customers who had transactions exceeding the average monthly balance of their account"). The database schemas include realistic financial schema elements (e.g., *Accounts(AccountID, CustomerID, Balance)*, *Transactions(TransactionID, AccountID, Amount, Date)*, etc.). Since schema encoding is crucial for text-to-SQL, we provided each model with a serialized schema description as part of its input (a format like a list of tables and columns) to allow the model to correctly reference column names in the SQL.

For evaluation, we use standard text-to-SQL metrics. Exact match accuracy (EM) checks if the model's generated SQL string (after some normalization) exactly matches the gold SQL. This is a stringent measure requiring every SQL clause and token to be correct. We also report execution accuracy (EX), which executes the predicted query on the database and checks if the result set matches the result of the gold query . Execution accuracy is sensitive to getting the correct answer, and it can reward logically equivalent SQL even if written differently - though care is needed as certain different queries can coincidentally yield the same result on the given data. To complement these, we compute the BLEU score (a text similarity metric) between the predicted and reference SQL. BLEU, commonly used in machine translation, provides a graded measure of overlap – useful for partial credit when the exact match fails (e.g., if a minor alias difference or formatting issue occurs). Additionally, we analyze component-wise accuracy as introduced in Spider - measuring correctness on different SQL clauses (SELECT, WHERE, GROUP BY, etc.) - and report the breakdown by query complexity (easy/medium/hard) to gauge robustness on complex queries. Finally, we consider the recently proposed test-suite accuracy, which evaluates queries on multiple database instances to mitigate the risk of spurious correct execution; this gives a stricter notion of semantic equivalence. These metrics together offer a comprehensive evaluation of SQL synthesis quality and reliability in the financial domain.

Models: Open-Source Small LMs (<12B)

We fine-tuned four transformer-based language models, each with under 12 billion parameters and publicly available:

- **LLaMA 7B [3]:** A 7-billion-parameter foundation model by Meta AI. LLAMA is trained on a trillion- token corpus and has shown strong language understanding. While not specifically a code model, its rich pre-training makes it a strong baseline. We used the 7B variant of LLaMA v1 (2023 release) as our base. LLaMA's larger versions have demonstrated that even 13B can outperform GPT-3 175B on many tasks , suggesting the 7B model may have sufficient capacity for SQL with fine-tuning.
- **CodeGen2 6B [4]:** A 6B-parameter model from Salesforce's CodeGen2 family which are pretrained on a mix of natural language and programming code. CodeGen2 is an update to the original CodeGen, aimed at efficient training for code synthesis. We specifically use the ~6B model (often referred to as 7B class) which is adept at producing syntactically correct code. Given that SQL is essentially a structured programming language, a model like CodeGen2 – with its strong prior on code syntax and keywords – may excel in generating SQL queries.
- **GPT-NeoX 6.7B [5]:** A variant of EleutherAI's GPT-NeoX architecture, scaled to approximately 6.7B parameters. The GPT-NeoX family (notably the 20B model) was trained on the open Pile dataset and demonstrated powerful text generation and reasoning capabilities. Here, we use a smaller configuration (~6-7B) which is comparable in size to GPT-J-6B. This model benefits from EleutherAI's open-source GPT-3 replication efforts; notably GPT-J (6B) was shown to perform on par with OpenAI's GPT-3 Curie (6B) model . We expect GPT-NeoX 6.7B to serve as a strong general-purpose language model baseline for our task.
- **Falcon 7B [7]:** A 7B-parameter model released by TII UAE. Falcon was trained on 1.5 trillion tokens of *RefinedWeb*, a high-quality filtered web corpus, making it one of the most data-rich models in the 7B category. Falcon-7B has been shown to outperform other models of similar scale on many standard NLP benchmarks, thanks to its extensive training and optimized architecture (it uses multi-query attention for efficiency). While Falcon is not specifically a code model, its broad knowledge and quality make it a compelling candidate for fine-tuning. Importantly, Falcon-7B is designed to be efficient to run, even on single devices, aligning with our goal of practical, resource-efficient deployment.

All these models are decoder-only transformer LMs. We adapt them to text-to-SQL by formulating the input and output as a seq2seq problem: the input to the model is a concatenation of the natural language question and a formatted schema description (to provide context of tables/columns), and the output to be generated is the SQL query. We insert special tokens or prompt patterns to clearly delineate the question and schema. For example, a typical input might look like:

```
  Question: "Which account has the highest total transaction amount in 2021?"
Schema: [Accounts(AccountID, CustomerID, Balance), Transactions(TransactionID,
AccountID, Amount, Date), ...]
  SQL:
```

The model then has to produce the correct SQL after the "SQL: " prompt. We found it useful to include the word "SQL" to prime the model that the output is code-like, especially for LLaMA and Falcon which were not exclusively trained on code.

Fine-tuning Procedure

We fine-tuned each model on the Spider-Financial training set using supervised learning (teacher-forcing on the known question-SQL pairs). To maximize efficiency given the model sizes, we employed parameter- efficient fine-tuning techniques. In particular, we used the LoRA (Low-Rank Adaptation) approach[8] for LLaMA and Falcon. LoRA keeps the original pretrained weights frozen and instead learns small rank- decomposition matrices that augment the model's layers, drastically reducing trainable parameters. This allowed us to fine-tune the 7B models on a single high-memory GPU. LoRA has been shown to match full fine-tuning performance while updating only a tiny fraction of parameters, which we observed as well - our LoRA-based fine-tunes reached nearly the same validation accuracy as full fine-tuning runs. For CodeGen2 and GPT-NeoX, we fine-tuned with a combination of LoRA and half-precision training (to fit in memory), leveraging their pretrained knowledge of code. All fine-tunings were done with the AdamW optimizer, an initial learning rate in the range 2e-5 to 1e-4 (tuned per model), and a batch size of 16 sequences. We trained for 3 epochs on ~5k examples (with early stopping if the dev set accuracy plateaued). Because text-to-SQL is sensitive to exact string output, we used target-side vocabulary masking during fine-tuning to restrict the output tokens to realistic SQL tokens (SQL keywords, schema identifiers, and numeric values) similar to techniques in prior work. This helped the model from generating out- of-vocabulary or irrelevant tokens in its SQL output.

At inference time, we employ beam search decoding (beam size 5) with the constraint that the model must produce a valid SQL query. We incorporate simple syntactic validity checks (e.g., every opened quote or parenthesis must be closed, SELECT and FROM clauses must appear, etc.). This is a lightweight analogue to the constrained decoding used by PICARD. It reduces the incidence of unparsable SQL. With beam search, the model can explore multiple candidate query formulations; we select the top-ranked full query that passes validation. This procedure, while not as strict as PICARD's incremental parsing, improved execution accuracy by catching and discarding many obvious invalid outputs.

EXPERIMENTS AND RESULTS

Evaluation Metrics

We evaluated each fine-tuned model on the held-out test set of financial-domain questions. The primary metrics are Exact Match Accuracy (EM) and Execution Accuracy (EX), as defined in the Spider benchmark. EM is 1 if the predicted SQL exactly matches the reference SQL (after trivial formatting normalization), and 0 otherwise. EX is 1 if the predicted SQL, when executed on the test database, returns the correct result; this can give credit for semantically correct queries that are written differently than the reference (e.g., different alias names or equivalent subquery structures). We also report the BLEU-4 score over the entire test set, treating the SQL as a sequence of tokens – this rewards partial matches, e.g., if a query selects the correct columns but uses an incorrect filter, BLEU will be intermediate between 0 and 1. Additionally, we computed clause-wise accuracies: what fraction of the queries had correct SELECT clause, correct WHERE clause, etc., to pinpoint strengths and weaknesses. Finally, we include the average inference time per query (measured in milliseconds) for each model to highlight efficiency, and note the approximate GPU memory required for inference.

Baseline Comparisons

For context, we compare our fine-tuned models against several baselines: - Prior SOTA (large models): We cite results from the literature for top-performing text-to-SQL models on Spider and related tasks. In particular, PICARD with T5-3B achieved about 75% EM and 79% EX on Spider. Although not fine- tuned on our financial data, this represents the level of performance of a 3B-sized model with grammar- constrained decoding on a general benchmark. We also consider RAT-SQL + BERT and other earlier systems which ranged from ~60–70% on Spider dev. On the WikiSQL dataset (single-table, simpler), models have exceeded 90% execution accuracy in the past – but our focus is the complex, cross-table setting. - Zero-shot LLMs: We evaluated the base models (LLaMA, Falcon etc. without fine-tuning) on the test set by simply prompting them with the same input format (question + schema) and asking for SQL. This zero-shot scenario tests the pretraining capability of these LMs to do text-to-SQL out-of-the-box. Not surprisingly, their performance was poor – none of the base models exceeded 5% exact match or 10% execution accuracy. They often produced either no SQL (failing to follow the prompt) or syntactically invalid SQL. This confirms that fine-tuning is essential for reliable SQL generation. We also tried GPT-3.5 (text-davinci-003 via API) on a subset for an informal comparison; it performed better zero-shot (~30% accuracy) but still far below fine-tuned models, and it sometimes produced valid-looking SQL that was semantically incorrect (highlighting the need for fine-tuning or few-shot examples even for large models). - Fine-tuned T5 (3B) on Spider-Financial: As an additional baseline, we fine-tuned a T5-Large (770M) and T5-3B model on our dataset using the same training data. T5 models are encoder-decoder architectures known to perform well on structured prediction. Our T5-3B (without constrained decoding) achieved around 60% exact match on the test, which is in line with expectations and provides a point of reference for the 6–7B decoder-only models.

Main Results

Table 1 summarizes the performance of each fine-tuned model on the test set, along with the baseline and reference numbers. (Exact Match (EM), Execution (EX) Accuracy, BLEU, and inference latency on a single A100 GPU)

Model(~Parameterss)	EM Accuracy	EX Accuracy	BLEU	Inference (ms/query)
LLaMA - 7b fine-tuned	58.6%	62.3%	0.702	45
CodeGen2-6b fine-tuned	64.1%	68.0%	0.745	50
GPT-NeoX-6.7b fine-tuned	53.4%	57.2%	0.668	47
Falcon-7b fine-tuned	55.8%	59.5%	0.689	40
T5-3B + PICARD (prior SOTA on spider)	71.9%	75.1%	-	n/a
T5-3B + PICARD (our data)	60.5%	64.0%	0.710	120

(Note: PICARD uses constrained decoding; BLEU not reported ("–"). Latency for PICARD/T5-3B not directly comparable due to different hardware and implementation).

Our fine-tuned CodeGen2-6B model performs the best, with 64.1% exact-match and 68.0% execution accuracy. This indicates it correctly answers over two-thirds of the financial questions. LLaMA-7B is the next best at ~59% EM. Falcon-7B and GPT-NeoX-6.7B reach 53–56% EM. All models show a slight gap between execution and exact-match accuracy (3–4 points), meaning a few predicted queries, while not textually matching the gold, still produce the correct results (usually due to minor alias/name differences or harmless reordering in the SQL). The BLEU scores mirror these trends: CodeGen2 > LLaMA ≈ Falcon > GPT-NeoX. For instance, CodeGen2's BLEU of 0.745 suggests a high overlap on average with reference queries, whereas GPT-NeoX's 0.668 is lower, implying its outputs diverge more from the gold (often because it misses some conditions or mis-orders some clauses).

Compared to the T5-3B+PICARD SOTA line: our best model is ~8 points lower in EM. This gap is expected since PICARD had the advantage of a larger model with constrained decoding on the general task . H5owever, our CodeGen2-6B actually comes reasonably close to the fine-tuned T5-3B (without PICARD) on the same data (64.1 vs 60.5 EM). Considering CodeGen2-6B is half the parameter count of T5-3B, this is an encouraging result. It suggests that a 6B dedicated code model can match a 3B general seq2seq model on this structured task. Falcon-7B and LLaMA-7B also nearly match the T5-3B baseline in EM. GPT-NeoX-6.7B lags a bit more, possibly due to older architecture and less code-specific training.

All fine-tuned models dramatically outperform their zero-shot counterparts (last row): e.g., LLaMA-7B jumped from ~3% to ~59% EM after fine-tuning – a testament to the importance of supervised training on the domain-specific data. The zero-shot outputs were often incomplete or incorrect (BLEU only 0.21). Few- shot prompting of base models (not shown in table) did better than zero-shot (e.g., providing 2–3 exemplars improved LLaMA to ~20% EM), but still fell short of the fine-tuned performance.

In terms of efficiency, all small models are extremely fast at inference. Falcon-7B had the fastest generation, needing about 40 ms per query on average (likely due to its optimized architecture). The others were around 45–50 ms. In contrast, the T5-3B model (which uses an encoder-decoder and more parameters) took roughly 120 ms per query in our setup – nearly 2–3× slower. This demonstrates a key benefit of small decoder-only LMs: low latency, which is vital for interactive systems. Memory footprint was also modest: a 16 GB GPU can comfortably host a 7B model with half-precision weights, whereas a 3B encoder-decoder requires similar memory and a 13B model or larger would require much more. This means our fine-tuned models can be deployed on commodity hardware, enabling on-premises use in financial organizations that have privacy or compliance concerns about sending data to large cloud APIs.

Analysis of SQL Generation Quality

To better understand the models' behavior, we analyzed their outputs on different query types. We found that CodeGen2-6B had a clear advantage on the most complex queries – those with multiple joins or nested subqueries. For instance, on questions requiring nested SELECT finding entities above an average or involved in a subquery), CodeGen2 got 10% higher accuracy than LLaMA. This can be attributed to CodeGen2's training on code, which likely gave it an edge in understanding the syntax of nested structures. On the other hand, Falcon-7B sometimes generated more natural alias names and was slightly better at column selection tasks (we suspect Falcon's broad pretraining helped with semantic understanding of column names, especially when the column name was something like "revenue" or "balance" – Falcon more often picked the correct one if multiple were semantically similar). LLaMA-7B showed balanced performance and strong generalization to unseen schemas; it made fewer syntax errors than GPT-NeoX, indicating the solidity of its pretrained representation.

The most common errors among models included: (a) Incorrect column or table names - e.g., using TRANSACTIONS.AMOUNT instead of TRANSACTIONS.TOTAL (when the schema had slightly different naming). This type of error occurred in 15% of the failures, reflecting the challenge of schema linking. In some cases, the model outputs a column name that was semantically right but not exactly matching the schema (leading to execution failure or mismatch). (b) Missing or wrong filtering conditions - e.g., the question asks for a particular year, but the model's SQL misses the year filter. This suggests that models sometimes fail to carry a constraint from question to SQL. © Aggregation errors - like using MAX instead of SUM, or grouping incorrectly. These were less frequent but occurred on complex analytics questions. We note that such errors would be penalized by execution accuracy and indeed those cases contribute to the gap between execution and exact match in some models. (d) Syntax errors - these were rare (<3% of outputs) after applying our decoding constraints. GPT-NeoX had the most, but even it produced <5% invalid queries.

We also evaluated robustness by perturbing the questions (e.g., rephrasing or adding irrelevant context). The fine-tuned models were generally robust to minor rephrasings, maintaining the same SQL output. This indicates they learned a stable mapping rather than overfitting to surface form. However, adding irrelevant sentences to the question sometimes confused the smaller models (especially GPT-NeoX), causing them to include conditions that weren't asked. This suggests that while the models understand the financial domain well, their context filtering ability (ignoring irrelevant text) could be improved, perhaps via better instruction tuning or using a refined input format.

DISCUSSION

Our results demonstrate that small open-source LMs can be powerful for SQL generation when fine- tuned on relevant data. This has practical significance: organizations with specialized database schemas (like finance, healthcare, etc.) can take a 6–7B parameter model and fine-tune it on their domain, obtaining a custom NL-to-SQL system without needing access to 100B+ models. The fine-tuning process for our models was also relatively inexpensive – on the order of a few hours on a single GPU with LoRA – which is feasible even for small companies or research groups.

Comparison to Larger Models: It is instructive to compare our fine-tuned small models to the performance of much larger models reported in literature. For example, the GPT-3.5/GPT-4 family (with tens or hundreds of billions of parameters) has been used in recent text-to-SQL work. GPT-3.5 in a zero-shot or few-shot setting reportedly achieves around 50–60% exact-match on Spider, and with specialized prompting or fine- tuning can exceed 70%. GPT-4 has been reported to reach ~80% on Spider (dev) in a zero-shot scenario . However, those models are proprietary and require API access. Our CodeGen2-6B, at ~64% on a Spider- derived domain test, is competitive with GPT-3.5 on similar tasks – all with a model that can run locally and under one-tenth the parameter count. The gap to GPT-4 and the absolute SOTA remains, but that gap may be closed further by techniques like ensemble decoding, data augmentation, or using retrieval to help the model. One promising direction is to integrate a tool that provides the model with candidate schema mappings or value lookups (for example, if the question asks for a specific financial quarter, a tool could map "Q4 2021" to the appropriate date range in SQL). Small models augmented with tools could potentially outperform a larger raw model.

Efficiency and Deployment: The efficiency gains of small models are notable. Falcon-7B, for instance, can run on a CPU in under a second per query (we tested an optimized INT8 quantized version which achieved ~0.5 seconds per query on an 8-core CPU). This makes it plausible to embed the model in edge or on-prem devices that need to translate natural language to SQL without external dependencies – a desirable feature for data privacy in financial institutions. Additionally, the reduced memory footprint means multiple such models (or multiple instances of one model) can be served in parallel on a server, scaling to higher throughput for enterprise use. Fine-tuning via LoRA also means updates to the model (incorporating new database schemas or correcting errors) can be done quickly and with limited overfitting, by simply training new adapter weights.

Error Correction and Robustness: Despite their strengths, small models are still prone to certain errors as discussed. In mission-critical applications like finance, a single incorrect SQL (especially one that is syntactically correct but semantically wrong) could lead to misleading results. One strategy to mitigate this is to use model uncertainty or an ensemble of models. We observed that CodeGen2 and LLaMA often failed on different queries; an ensemble that chooses the answer if both models agree (or flags queries where models disagree) could improve reliability. Another strategy is post-validation – after generating a SQL, one can verify whether it makes sense (e.g., does it use all constraints from the question?). Recent research has proposed unit tests or execution on multiple database states (test-suite accuracy) to ensure the SQL is robust. Implementing such verification can catch cases where the predicted SQL is too general or misses edge cases. Our focus was primarily on generation, but integrating such robustness checks is an important next step.

Finally, we discuss the limitations of our study. While our Spider-Financial dataset is based on a well- known benchmark, it may not capture all nuances of real financial databases. True financial data can be proprietary and larger in scale. There is ongoing work like BookSQL (a newly introduced accounting text-to- SQL dataset with 100k examples) which shows that models trained on Spider generalize poorly to real accounting data . This implies that scaling up domain-specific training data is crucial. Our experiments on a relatively small fine-tuning set demonstrate feasibility, but performance would likely improve with more training examples. Moreover, incorporating domain knowledge (for example, known accounting rules or common financial query patterns) explicitly into the model could further enhance accuracy.

CONCLUSION

In this paper, we showed that fine-tuning small open-source language models can significantly enhance their ability to generate SQL queries in a specialized domain (finance). Using a Spider-derived financial dataset, we trained LLaMA-7B, CodeGen2-6B, GPT-NeoX-6.7B, and Falcon-7B to translate natural language questions into SQL. The fine-tuned models achieved strong results – up to 64% exact-match accuracy – on complex financial queries, rivaling the performance of much larger models on similar tasks. Our best model (a 6B CodeGen2) produced correct SQL for approximately two-thirds of the test questions, reducing the reliance on massive proprietary LLMs for this task. We also demonstrated the robustness (via exact and execution match) and efficiency (low latency, small memory footprint) of these models, which are critical for real-world deployment.

This work underscores that task-specific fine-tuning remains a powerful approach in the era of large foundation models, especially when task data is available and domain precision is required. For future work, we plan to explore combining these small fine-tuned models with *prompting or few-shot learning* using large models, to see if hybrid systems can further boost accuracy. Another avenue is to apply reinforcement learning or iterative refinement: e.g., using execution feedback to fine-tune the model to avoid outputs that fail or produce errors. Additionally, expanding our dataset (or using BookSQL) and exploring models up to the 10–12B range (still "small" by today's standards) like LLaMA-13B or CodeGen2.5-7B could yield further gains while remaining practical.

Overall, our findings contribute to the understanding that *bigger is not always necessary* – with the right data and fine-tuning techniques, smaller models can achieve robust and efficient SQL synthesis, empowering the development of natural language database interfaces in specialized application domains.

REFERENCES

[1] **T. Yu et al. (2018)** – *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. EMNLP 2018.

[2] **Y. Zhong et al. (2017)** – *Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning*. arXiv preprint arXiv:1709.00103. (Introduced the WikiSQL dataset)

[3] **H. Touvron et al. (2023)** – *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971. (Introduces LLaMA models 7B–65B, with 13B outperforming GPT-3 175B)

[4] **E. Nijkamp et al. (2023)** – *CodeGen2: Lessons for Training LLMs on Programming and Natural Languages*. arXiv:2305.02309. (CodeGen2 family of 1B–16B code models.

[5] **S. Black et al. (2022)** – *GPT-NeoX-20B: An Open-Source Autoregressive Language Model*. Proc. of BigScience Workshop on Challenges & Perspectives in Creating Large Language Models, 2022. (Open 20B model; we use a 6.7B variant)

[6] **B. Wang and A. Komatsuzaki (2021)** – *GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model*. EleutherAI Technical Report, 2021. (Open-source 6B GPT-3 style model)

[7] **P. Penedo et al. (2023)** – *The Falcon Series of Open Language Models*. arXiv:2311.09868. (Falcon-7B and Falcon-40B models; Falcon-7B trained on 1.5T tokens)

[8] **E. Hu et al. (2021)** – *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685. (Introduces LoRA fine-tuning, greatly reducing trainable parameters)

[9] **T. Scholak et al. (2021)** – *PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models*. EMNLP 2021. (Constrained decoding method that boosted T5-3B to 71.9% on Spider test)