Text-to-SQL Comparison of Llama 3.1 vs ChatGPT 4

Glenn DeSouza
University of California, Berkeley
gdesouza@berkeley.edu

Abstract

The latest version of Meta's open-source LLM, Llama 3.1, shows some promise as a substitute for ChatGPT in various code generation tasks. This paper focuses on comparing text-to-SQL capabilities of both LLMs and finds that Llama 3.1 can match the performance of ChatGPT 4 on simple use-cases, but fails on more difficult SQL tasks.

1 Introduction

Text-to-SQL is a long-researched NLP task that seeks to translate natural language questions into executable database queries via SQL. A plethora of papers have been published on the subject in recent years. In the last ten years alone, SOTA paradigms have shifted several times: from rules-based methods such as parsing trees in the mid-2010s, which were followed by early encoder-decoder deep neural network frameworks such as SQL Sketch in 2019, and finally to more recent innovations taking advantage of the rapid advancements in LLM natural language understanding. These latest techniques include pre-training and schema-linking techniques in pre-trained language models, as well as a more recent focus on combining in-context learning (ICL) and fine-tuning of LLMs.

In this paper, I focus on comparing one of the top open-source LLMs, Llama 3.1, vs the top overall model for code generation, ChatGPT 4, as measured by its accuracy scores on the HumanEval code generation benchmark. We use the HumanEval Pass@1 (correct code execution on first pass) introduced in 2021 as a common industry benchmark for LLM code evaluation (Chen et al).

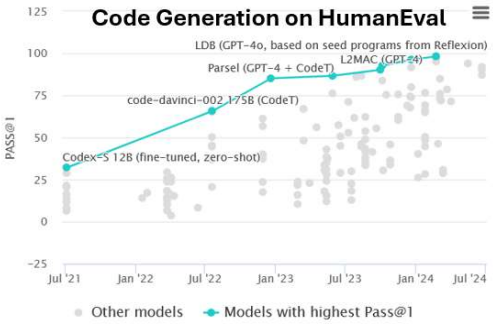


Table 1: Highest Scoring LLMs on Pass@1 benchmark. Only GPT-4 models exceed 90%.

I find that Llama 3 has closed the gap on closed-source LLMs on simple code tasks. This has near-term implications for corporate use, since open-source models can be privately hosted for inference with no external data leakage, a key corporate concern.

2 ChatGPT 4: Current Dominance

LLMs have made great strides in code generation accuracy in the last 3 years, with the top performing models going from a Pass@1 accuracy score of below 50% on the HumanEval Pass@1 benchmark in September 2021, to several GPT 4-based models exceeding scores of 90% in the last 12 months (Table 1). As scores move closer to 100% accuracy, these models are now approaching a level where they may soon become viable for corporate production use, particularly in domains where perfect or near-perfect accuracy is typically required. Accordingly, ChatGPT 4 will be used as the baseline to measure against.

3 Fine-Tuning An Open-Source Competitor: Llama 3.1

We select Llama 3.1 8B Instruct (8 billion parameter model trained on 15 trillion tokens – nearly 7x the training amount of Llama 2) as our model to optimize for text-to-SQL training. We choose the Instruct version of the Llama family as it is a pre-trained model specifically for instruction, as opposed to the Chat version, which is designed for conversational use (note that Llama also has an LLM trained for code generation, Code Llama, but it has not been released for Llama 3 yet).

On the HumanEval code generation benchmark, the highest Pass@1 score that any submitted Llama model has been able to achieve was 62% in September 2023.

4 Text-to-SQL Dataset

The text-to-SQL dataset we use for fine-tuning Llama 3.1 is Gretel AI's synthetic text-to-SQL dataset, released in April 2024 under an Apache 2.0 license. It contains over 100k records spanning 100 distinct industry domains and is claimed to be the largest open-source text-to-SQL dataset at the time of publication.

4.1 Dataset Format

The dataset contains 11 fields, including 3 fields that we will inject directly into our model: an SQL natural language prompt and its related context as inputs, and the corresponding SQL statement as our labels.

We also use its "sql_complexity" column, which classifies the dataset into 8 types of SQL queries. We take these groups and map them into a higher-level classifier column with categorical values of "Easy", "Medium", and "Hard" depending on the difficulty level of the SQL complexity category:

Raw Dataset SQL Complexity Type	Mapped Difficulty Level	% of Dataset
basic SQL	Easy	48.5%
aggregation	Medium	22.0%
single join	Medium	14.9%
subqueries	Hard	6.7%
window functions	Hard	3.6%
multiple_joins	Hard	2.9%
set operations	Hard	1.0%
CTEs	Hard	0.3%

Figure 1. SQL Complexity Mapping

Additional details on the dataset, including field layout, column descriptions, sample records, and distribution of domains are listed in Appendix A.

4.2 Examples of Easy and Hard Questions

We classify 'basic SQL' questions, which comprise nearly half of the dataset as 'Easy' questions. Questions in this category typically involve just one table and primarily measure the ability of the LLM to match semantically similar words in the user prompt to the given context. 'Easy' questions such as Figure x comprise nearly half (48%) of the dataset.

sql_prompt:	Update the end date of carbon offset initiative 'Initiative 1' in Australia to '2025-12-31'.
sql_context:	CREATE TABLE carbon_offsets (initiative_id INT, initiative_name VARCHAR(255), country VARCHAR(255), start_date DATE, end_date DATE);
sql:	UPDATE carbon_offsets SET end_date = '2025-12-31' WHERE initiative_name = 'Initiative 1' AND country = 'Australia';

Figure 2. Example Easy Question

In contrast, 'Medium' and 'Hard' questions more rigorously test the model's text-to-SQL capabilities for queries involving one or more joins, aggregation, window functions, subqueries, and other advanced SQL implementations.

sql_prompt:	List the top 3 marine species with the highest population growth rate.
sql_context:	CREATE TABLE marine_species (id INT, name VARCHAR(255), population_size INT, growth_rate DECIMAL(4,2)); INSERT INTO marine_species (id, name, population_size, growth_rate) VALUES (1, 'Clownfish', 10000, 0.05), (2, 'Sea Turtle', 5000, 0.10), (3, 'Dolphin', 20000, 0.02);
sql:	SELECT name, growth_rate FROM (SELECT name, growth_rate, ROW_NUMBER() OVER (ORDER BY growth_rate DESC) rn FROM marine_species) t WHERE rn <= 3;

Figure 3. Example Hard Question

Thus, for training purposes, we train our model via two different methods: first on the entire dataset, and then on just medium/hard questions, since there is a skew in the original dataset towards 'easy' (basic SQL) questions and we want to examine for possible imbalanced class issues and measure accuracy relative to SQL difficulty.

5 Experiment 1: Exact Match Results

Exact Match (EM) measures the percentage of dataset records where the predicted SQL is identical to the ground-truth SQL. We use this metric to evaluate our first experiment's results for both models referenced above.

5.1 Model Configuration

We use the free version of Unsloth, which is a 3rd party wrapper over Hugging Face's (HF)

supervised fine-tuner, for our model training. It contains similar parameters to HF, but claims faster training speed and reduced memory usage.

We set quantization to 4 bits to allow the Llama model to fit into the allocated VRAM in our training environment.

We also configure a LoRA adapter with a high (128) alpha scaling factor and low dropout rate to maximize task-specific performance, though this could risk some overfitting to the dataset.

5.2 Training and Validation Results

The Llama model is trained twice: first on a randomized 5k record subset of the overall dataset, and then on another randomized 4k record subset of just medium/hard questions. Training is conducted for one epoch. Training and validation loss are logged at 50 step intervals. Both the training and validation datasets track closely throughout the training cycle, with a gradual decline that plateaus at about 0.38 loss after 2000 steps for both models (Figure 4).

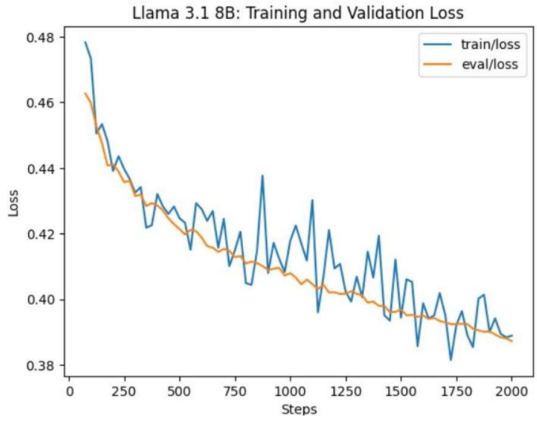


Figure 4. Experiment Training/Validation Loss

The absence of any steep drop in the training loss relative to the validation loss for both models (particularly in the early training steps) gives some confidence that the models are not exhibiting any significant overfitting at this stage. The final training and validation figures for both models are quite similar (Figure 5):

	Full Dataset Model	Medium/Hard Only Model
Training Loss:	0.396	0.402
Validation Loss:	0.383	0.386

Figure 5. Loss Metrics for Both Trained Models

For inference, the test dataset is segregated into Easy, Medium, and Hard questions. Inference is then run individually on each category. Interestingly, the model trained only on Medium and Hard questions performs worse on inference than the full dataset model, while it performed better than the full dataset model on the medium/hard questions it was fine-tuned for (Figure 6):

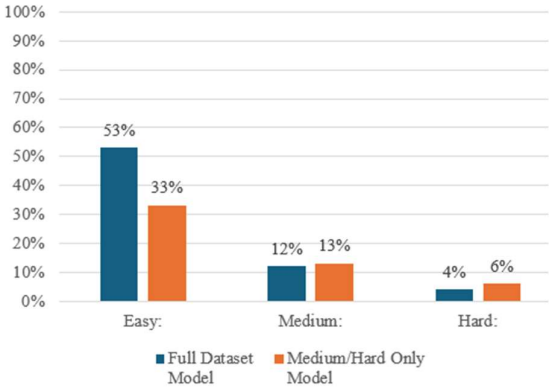


Figure 6. Exact Match (EM) Accuracy by Difficulty Category.

The Medium/Hard question model was significantly worse on matching ground-truth SQL in the Easy questions category (33% vs 53% accuracy for the full dataset model), while only modestly better on medium and hard questions.

5.3 Examination of Errors

Exact Matching on SQL is a popular evaluation metric for text-to-SQL but has its pitfalls – particularly since the returned SQL can be correct but structured with slightly different syntax. A manual review of matching exceptions yielded that a plurality of SQL differences (particularly in the easy questions subset) are due to immaterial differences – such as different ordering of non-impactful fields or clauses, or syntactic sugar such as column name aliases (Figure 7).

Question: *What is the total production of iron mines in Russia?*

Context: CREATE TABLE mine (id INT, name TEXT, location TEXT, mineral TEXT, production INT); INSERT INTO mine (id, name, location, mineral, production) VALUES (1, 'Mikhailovsky GOK', 'Russia', 'Iron', 12000), (2, 'Lebedinsky GOK', 'Russia', 'Iron', 15000);
Ground-truth SQL: SELECT SUM(production) FROM mine WHERE mineral = 'Iron' AND location = 'Russia';
Generated SQL: SUM(production) FROM mine WHERE location = 'Russia' AND mineral = 'Iron';

Question: *What is the maximum loan amount for socially responsible loans in the Asia-Pacific region?*

Context: CREATE TABLE socially_responsible_loans (loan_id INT, region VARCHAR(20), loan_amount DECIMAL(10,2)); INSERT INTO socially_responsible_loans (loan_id, region, loan_amount) VALUES (101, 'Asia-Pacific', 50000), (102, 'Europe', 30000), (103, 'Asia-Pacific', 70000);
SELECT MAX(srl.loan_amount) FROM socially_responsible_loans srl WHERE srl.region = 'Asia-Pacific';
SELECT MAX(loan_amount) FROM socially_responsible_loans WHERE region = 'Asia-Pacific';

Figure 7. Sample errors.

Some of these issues, such as the arbitrary injection of table aliases in the second example above, arise randomly in the ground-truth SQL and could be a possible drawback of the dataset's synthetic creation. Due to their randomness, these differences cannot be easily accounted for in model adjustment. Accordingly, we proceed to a second stage experiment, where the models will instead be measured on Execution Accuracy (EA) instead of EM.

6 Experiment 2: Using a Real World Database and Code Execution for Better Evaluation

For our second experiment, we set up a toy SQLite database and a code execution framework to test direct SQL execution. This should alleviate the issues of comparing SQL statements directly, and ideally lead to more reliable accuracy results, by using the Execution Accuracy metric.

The database is composed of 3 tables related to stock market trading:

1) A 'portfolios' table, containing names of portfolios and their associated portfolio managers
2) A 'prices' table, containing stock market prices for several stocks for the current year
3) A 'portfolio_holdings' table, which contains the stocks held in each portfolio, as of a certain date

No other information other than 'CREATE TABLE' schema information is provided to either our fine-tuned Llama model or ChatGPT. In addition, no primary keys or foreign keys have been set or passed to the LLM prompts (though relationships do exist between the 3 tables). The table relationships are intentionally omitted in order to test the ability of each model to make the correct SQL join intuitively.

A new test pool of 100 natural language queries and their corresponding numeric answers is manually prepared on the toy database. The questions are evenly split 50/50 between 'easy' and 'medium/hard' questions. The number of test questions is significantly reduced from the size of the first dataset due in part to the manual intervention needed to prepare and check the answer set, as well as to minimize the token spend from using the OpenAI API to query ChatGPT 4 programmatically.

6.1 Results: Comparable Performance on Easy Questions Only

Our fine-tuned Llama model performs much better on "Easy" questions in Execution Accuracy than what was seen in the first experiment, achieving the correct numeric response 76% of the time, comparable to ChatGPT 4's 78% accuracy.

However, the model experiences significant failure in the "Medium/Hard" question category, achieving only 16% accuracy, vs. chatGPT's 66% (Figure 8).

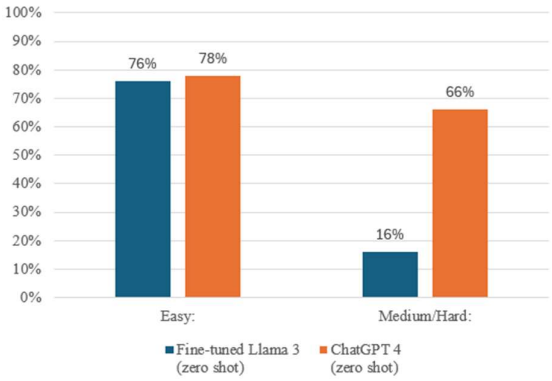


Figure 8. Execution Accuracy (EA) for fine-tuned Llama Model vs ChatGPT 4, by Category.

6.2 Examination of Errors

A closer inspection of the fine-tuned model's errors indicates that it had a higher propensity of invalid SQL, nearly double that of ChatGPT (Figure 9):

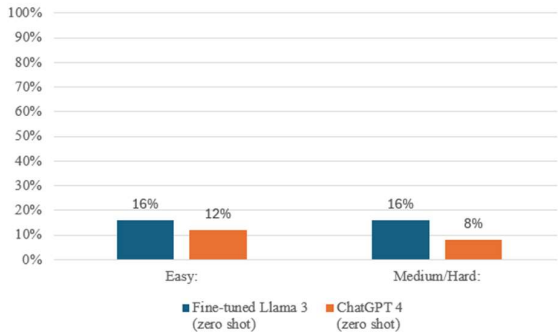


Figure 9. Percentage of Invalid SQL Errors

The majority of Llama-specific errors fell into two major categories:

1) Confusion between field names. Several queries were confused between ambiguous field names in the same table, such as portfolio name and portfolio manager name, which were both in the portfolios table
2) Invalid SQL dialect. A common mistake in the fine-tuned model is putting table aliases in positions where they are not accepted by

SQLite. It's possible that Llama is placing more emphasis on other, more common SQL dialects than SQLite.

ChatGPT was able to resolve the majority of these issues in the Medium/Hard SQL category more easily. It also processed results approximately 2x faster on the same machine.

Conclusion

While it appears the 8B Llama 3 model may still be lacking in achievement on difficult code generation tasks, several caveats should be noted:

1) Llama 8B to ChatGPT 4 is not an apples-to-apples comparison. Llama 8B has only 8 billion parameters, while chatGPT's parameter number is estimated to be orders of magnitude higher – perhaps near 100 trillion.
2) Training was limited by computing power. It's quite possible that the Llama 70B or 405B models could have performed much better on the hardest difficulty questions, but even the 70B model requires greater than 40 GB VRAM to run, which is beyond Google Colab Pro's A100 GPU capabilities. Training was also halted after one epoch and using only a portion of the 100k record dataset.
3) It's possible that synthetic data may have introduced errors of its own, instead of representing real-world SQL examples.
4) ChatGPT stores conversation history, and this may have improved its performance on the toy dataset as testing progressed.

In addition, it's possible that other enhancements such as multi-agent interactions might occur behind the scenes in order to give ChatGPT more accurate results. But as a closed-source model, their IP is not revealed.

Avenues for Future Research

Nevertheless, there are several avenues for future improvement, which other text-to-SQL researchers have already noted:

1) **Prompt Engineering.** Llama 3's new context window size of 128k tokens allows the potential for much more context to be input into a single prompt. This could lead to innovations in using RAG methods for injecting more useful context into the LLM, such as in the form of schema layouts/trees, natural language table and column descriptions, and other descriptive data that could aid in LLM understanding of table relationships.
2) **Multi-agent frameworks** such as AutoGen, LangChain, and CrewAI allow the training and interaction of task-specific agents to better achieve accurate results (though at the expense of increased latency). Some notably successful algorithms include ReAct, Reflection, and LATS.
3) Researchers at Berkeley have also proposed a combined fine-tuning + RAG implementation called RAFT (Retrieval Aware Fine-Tuning) – this could be an interesting area for future research on text-to-SQL and code generation tasks.

References

1. Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426

2. Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, Wojciech Zaremba. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

3. Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang. Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation. arXiv:2403.02951.

4. Tianjun Zhang, Shishir G. Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, Joseph E. Gonzalez. RAFT: Adapting Language Model to Domain Specific RAG. arXiv:2403.10131.

Appendix A: GretalAI synthetic-text-to-SQL dataset

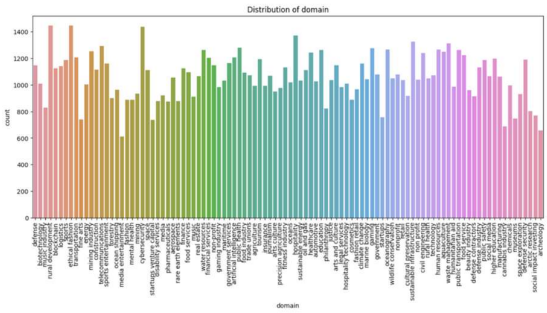
Gretal's synthetic-text-to-SQL dataset details and overview:

Data statistics

	column_name	Unique	Missing	Type	average token count	total token count
0	id	100000	0	int32	N/A	N/A
1	domain	100	0	object	2.205	233435
2	domain_description	100	0	object	18.229	1929598
3	sql_complexity	8	0	object	2.029	214809
4	sql_complexity_description	8	0	object	11.912	1260878
5	sql_task_type	4	0	object	2.882	305079
6	sql_task_type_description	4	0	object	9.737	1030683
7	sql_prompt	100338	0	object	17.098	1809835
8	sql_context	90034	0	object	82.136	8694143
9	sql	99605	0	object	30.578	3236692
10	sql_explanation	100113	0	object	44.868	4749329

A1. Distribution of industry domains

Dataset covers 100 unique industry domains, with well-balanced distribution:



Dataset field layout

Column Name	Data Type	Description
id	integer	Data record id
domain	string	Domain/vertical (e.g., aerospace)
domain_description	string	Domain/vertical data description
sql_complexity	string	SQL code complexity label
sql_complexity_description	string	SQL code complexity description
sql_task_type	string	Type of SQL task being performed
sql_task_type_description	string	Description of the SQL task type
sql_prompt	string	Natural language SQL prompt
sql_context	string	Database context for the SQL prompt
sql	string	SQL corresponding to the SQL prompt
sql_explanation	string	Explanation of the SQL query