Systematic Evaluation of Large Language Models in NL-to-SQL Translation

By Petavue Research

Background

In recent years, Large Language Models (LLMs) like GPT have significantly advanced Natural Language Understanding (NLU), showing remarkable ability in mimicking human text (Brown et al., 2020). A key NLU challenge is translating Natural Language (NL) into Code. This offers several significant benefits that streamline software development processes, enhance data accessibility, and democratize programming.

Within the realm of NL-to-Code, the NL-to-SQL translation has emerged as a critical field. This reflects the growing emphasis on refining LLMs to transform natural language into SQL queries. The key motivation behind this is to enhance human-computer interactions and also to facilitate direct, user-friendly access to vast databases without requiring specialized SQL knowledge (Iyer et al., 2018). This has spurred numerous research, leading to few benchmarking studies of comparing and fine tuning LLMs, aimed at optimizing NL-to-SQL tasks.

While substantial progress has been made in the NL-to-SQL domain, several pivotal gaps remain unaddressed. First of all, benchmarking studies in this field have been limited so far, offering scant insights beyond basic model evaluations. Secondly, an analysis of how Large Language Models (LLMs) fare across different hosting platforms—such as cloud platforms like AnyScale, Bedrock, Anthropic, OpenAI versus self-hosted environment—remains largely absent. This overlooks how environmental factors can significantly influence model performance, a gap that is crucial for informed deployment and resource optimization.

Moreover, most existing efforts have narrowly focused on accuracy as the sole metric for assessing LLM efficacy, sidelining other critical dimensions like throughput, latency, cost, etc. These aspects are vital for gauging the practicality of LLMs in real-world applications, suggesting a pressing need for studies that offer a holistic view by comparing all relevant metrics concurrently. Additionally, there's a lack of comprehensive analysis on the economic impact of deploying LLMs, including the cost-performance trade-offs of various LLM architectures and deployment strategies.

Aim of this report

In light of these gaps in the existing literature, our research aims to provide a comprehensive evaluation of LLMs' performance in NL-to-SQL tasks. Our work is aimed at providing profound insights, enabling informed decision making about usage of LLMs, deployment, strategic resource allocation and budget planning. Our findings will not only provide valuable insights for researchers and practitioners, but also contribute to the ongoing efforts to improve the practical utility and scalability of LLMs in real-world applications.

Methods

Selection of Dataset

We chose the BIRD dataset for our NL-to-SQL translation tasks due to its complexity and the rich challenges it presents, offering a more rigorous testing ground than alternatives like Spider (Li et al., 2024; Yu et al., 2018). Despite the potential for lower accuracy scores given BIRD's complexity, we believe it better tests our models' capabilities in handling challenging queries. The BIRD dataset's inclusion of an 'evidence' column further supports nuanced query generation and evaluation.

Sample Size Determination and Stratification

Our study used 360 question-query pairs from BIRD's dev dataset, categorized into three difficulty levels: simple, moderate and challenging. This sample size, determined through iterative testing with the GPT-3.5 model, ensures comprehensive coverage and a balanced representation of the dataset's challenges. This sample was stratified to include an equal distribution of difficulty levels, with 120 pairs selected from each category ensuring a balanced representation of the dataset's complexity.

Design

The design involves running the 360 question-query pairs for each model across platforms, conducting five trials per model. The five trials vary in the number of instructions provided to the models. The instructions are general rules that LLMs need to know to provide correct SQL responses. The aim here is to assess adding more information to the model incrementally impact SQL query generation capabilities. First trial will not have any instructions or guided prompts. From trial 2 to 5 we had instructions in the sets of 5, 7, 9 and 11, incrementing them by 2 in each trial.

Platforms and Models

We ran the 360 question-query pairs across several platforms: Anyscale, Bedrock, Anthropic, Google Gemini 1.0 Pro, and a self-hosted setup. We tested a variety of models including CodeLlama, Mistral, Claude 3 models, GPT family of models (Table 1). We have also included Google’s Gemini 1.0 Pro and

Databricks' DBRX Instruct. This allowed us to compare performances, costs, and speed across different hosting environments.

Platforms	Models
Anyscale	Llama2-70B Mistral 7B (Version 1) Mixtral8x7B CodeLlama-70B
Amazon Bedrock	Llama2-70B Mistral-7B (Version 2) Mixtral8x7B Claude3 - Sonnet Claude3 - Haiku
OpenAI	GPT3.5 Turbo GPT4 Turbo GPT4 Omni
Anthropic	Claude3 - Opus Claude3 - Sonnet Claude3 - Haiku
Google	Gemini 1.0 Pro
Self-Hosted	DBRX Instruct CodeLlma-70B CodeLlma-34B Mistral-7B (Version 1) Mistral-7B (Version 2) Mixtral8x7B Sql Code 7B2 Sql Code 70B Alpha WizardCoder-33B

Table 1. List of all models used and hosted in different platforms

Results

Our benchmark study focused on 3 metrics: (1) Execution Accuracy (2) Throughput, and (3) Total Cost of running the models (compared token sizes and inference time).

Execution Accuracy (EX)

We used Execution Accuracy (EX) as the metrics for measuring the accuracy of SQL queries generated by LLMs . This approach involves evaluating the result table of LLM generated queries (table 2) against the table of expected SQL queries given by the BIRD dataset (table 1). This means that the script will search for occurrences of each column of table 2 in table 1 with the same order of values. To evaluate this logic, we developed a custom evaluation script by optimizing BIRD's existing script. This optimization

aimed to address the limitations of BIRD's script by modifying it to rectify the scenarios where it underperformed.

Overall EX% averaged across 1800 inferences

To compare overall accuracy between models, we averaged the accuracy of each model across 5 instruction trials (360 inferences per trial), summing up to 1800 inferences run per model. We found that OpenAI's GPT-4 Omni, Anthropic's Claude3-Opus and OpenAI's GPT 4 Turbo are the top performing models with the highest accuracy of 41.44%, 39.56% and 39.39% respectively (Figure 1). Following these are the Claude3-Sonnet (35.95% in Bedrock; 34.67% in Anthropic), Claude3-Haiku (34.22% in Bedrock; 34.72% in Anthropic), and Open AI's GPT 3.5 Turbo (33.83%). Self-hosted models such as Wizardcoder 33B and SQLcoder-70B-alpha achieved accuracy of 31% and 30%.

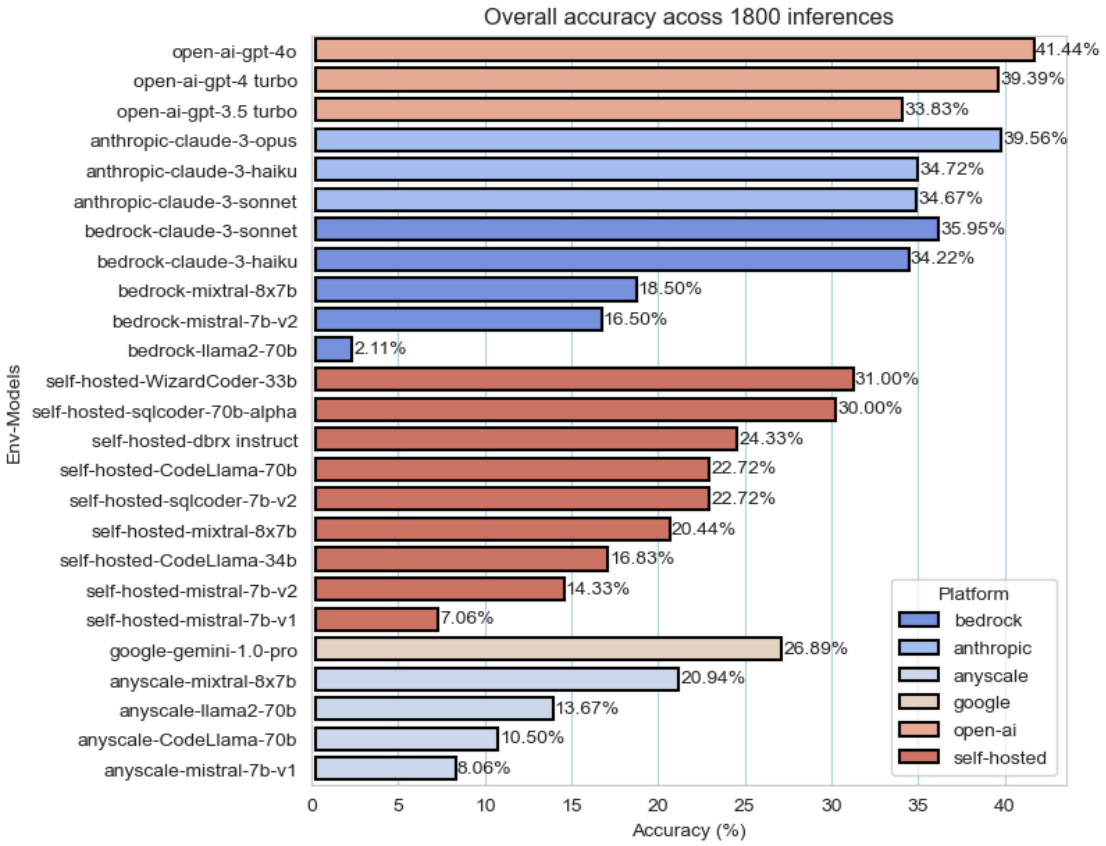

Figure 1. Average EX of each model compared to others.

Comparison of models run in multiple platforms

One of the key goals of this study is to find out how the same model when hosted in different platforms varies in terms of accuracy. From this interesting results have emerged (Figure 2). First, for Claude3 (Sonnet and Haiku) models we noticed only marginal differences in accuracy when hosted between

Anthropic and Bedrock. Sonnet's accuracy is marginally higher in Bedrock (35.95%) than in Anthropic (34.67%). In Haiku performances are nearly identical in Bedrock (34.22%) and Anthropic (34.72%).

But the biggest difference between platforms was observed for Llama2-70B. In Bedrock, we got an accuracy of 2.11% versus Anyscale at 13.67%, revealing a nearly 6 times difference. Similarly, a big difference was also observed for Codellama-70B when hosted on Anyscale (10.5%) versus self-hosted (22.72%), more than a 2x times difference. Also, the models of Mistral including Mistal-7B versions and Mixtral 8x7B, exhibited a significant difference mainly between Bedrock and when self-hosted. For example, Mistral 7B-V2 at Bedrock showed a higher accuracy (16.5%) compared to self-hosted (14.33%). On the other hand Mixtal 8x7B showed a higher accuracy (20.44%) in self-hosted compared to Bedrock (18.5%).

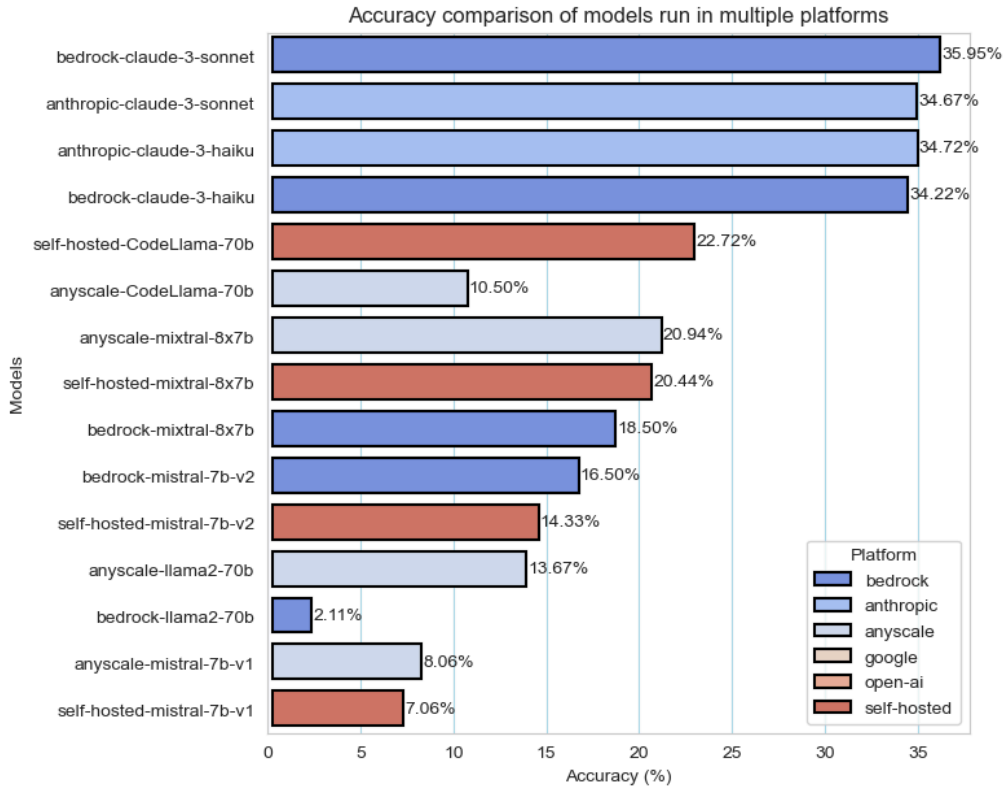

Figure 2. EX % comparison of models between platforms.

Impact of query difficulty on accuracy

The sample chosen from the BIRD dataset has questions with three levels of difficulties: simple, moderate and challenging. We compared the accuracy in relation with these levels. We found that across models accuracy dropped when query hardness level increased from simple to moderate, and from moderate to challenging (Figure 3).

In simple level NL-to-SQL tasks, GPT-4 Omni was observed to be the top performing model at 64.33%. This is followed by GPT-4 Turbo (62%) and Claude3-Opus (60.84%). Interestingly, we observed that Claude3-Haiku has achieved a higher accuracy (57.5%) compared to Sonnet (56%) for simpler tasks, even though Sonnet is a superior model. Amongst those self-hosted, Wizardcoder-33B achieved the highest accuracy of 52.67%, followed by SQLcoder-70B Alpha (42%). However, none of the self-hosted models were able to achieve the level of GPT-4 models or Opus.

When moved to moderate level tasks, we observed accuracy drops for all models. Claude3 Opus (36.33%) stood at par with GPT-4 Omni (36%), followed by GPT-4 Turbo (33.67%). Claude3-Sonnet (32.17% in Bedrock & 29.83% in Anthropic) outperformed Haiku (27.17% in Bedrock & 27.67% in Anthropic), contrary to what was observed in simpler tasks. Amongst self-hosted models, Sqlcoder-70B alpha (26.33%) achieved highest accuracy followed by Wizardcoder-33B (24.33%). We noticed that the drop in accuracy from simple to moderate difficulty is also least in Sqlcoder-70B alpha, Claude3-Opus and Sonnet compared to other models.

In challenging tasks, GPT-4 Omni achieved the highest accuracy of 24% compared to any model. This is followed by GPT-4 Turbo (22.5%), Opus (21.5%) and SQLcoder-70B alpha in self-hosted (21.67%). The accuracy drop is observed across all the models. Amongst all three levels of hardness, GPT-4 Omni consistently proved to be the best performing model, followed by GPT-4 Turbo and Claude3 Opus.

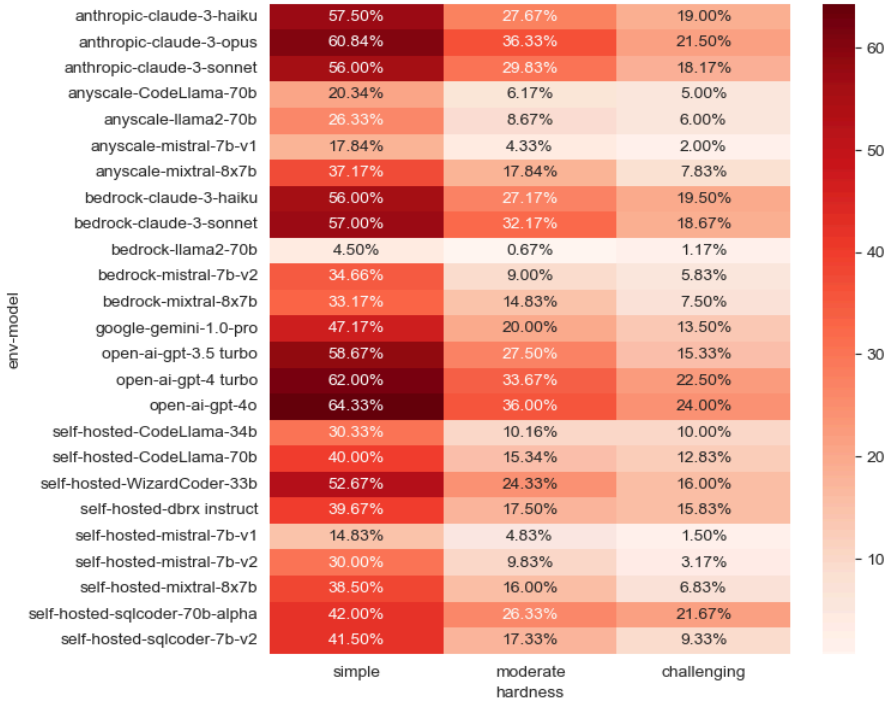

Figure 3. EX % across 3 levels of Question Hardness.

We also analyzed the relationship between size of the models with respect to hardness levels of the queries (Figure 4). We noticed the existence of clear relationships between them that is in line with a logical presumption. Higher sized models across all tasks performed the best followed by medium and smaller models. Also as the hardness of queries increased, model performances dropped for models of all sizes.

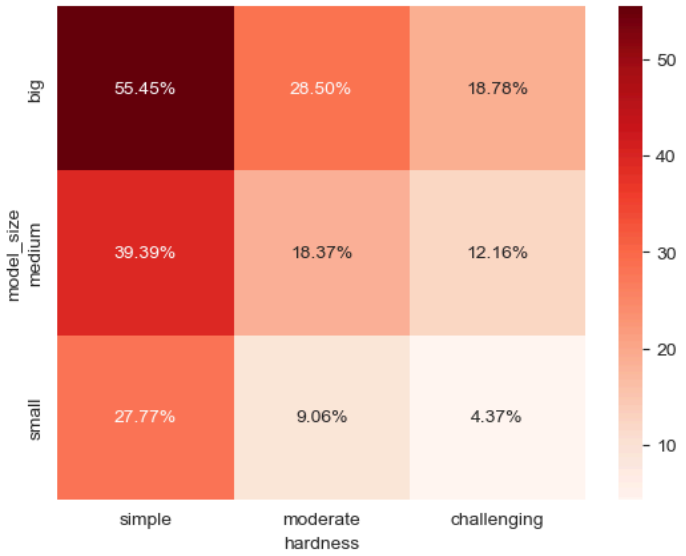

Figure 4. EX % shown in relation between model sizes vs hardness level of queries.

Impact of number of instructions provided in the prompt on accuracy

At the next level, we analyzed how model accuracy varies when the number of instructions given to the prompt are increased in the set of 0, 5, 7, 9 to 11 (Appendix A). Unlike what was observed with respect to hardness, varying the number of instructions did not provide a striking pattern in accuracy of models (Figure 5).

Also, we noticed that the extent of fluctuations in accuracy in models when the number of instructions are increased changes from model to model and platform to platform. For example, in Anyscale, we observed that for all 4 models accuracy increased when 5 instructions were provided compared to no instructions. Models such as Mixtral-8x7B, Llama2-70B and Codellama-70B attained peak performance in the trial of 11 instructions (23.06%, 15.28%, and 12.5%) and Mistral-7B-V1 peaked in the trial of 5 instructions (9.44%). Similarly fluctuations were observed in Bedrock, Open AI and Anthropic (Figure 6 & 7). However the extent of fluctuations observed in Mistral or Llama2 family models are more than what is observed in higher sized models such as GPT-4 Turbo, GPT-4 Omni and all Claude-3 models.

Figure 5. EX % across different sets of instructions provided to the prompt.

Similarly self-hosted models fluctuate a lot when we increase instructions from 0 to 11 (Figure 7). For example, for models such as Wizardcoder, Sqlcoder-70B, Sqlcoder-7B-2, Mixtral 8x7B, Mistral 7B-V1 and V2, the accuracy % increases drastically from no instruction to 5 instructions. Also for DBRX Instruct and Codellama-70B, we noticed a massive drop in accuracy for 0 to 5 instructions, from 26.67% to 22.5% and 29.17% to 23.33%, respectively.

Overall, we observe that each model attained its own peak accuracy irrespective of the number of instructions provided, exhibiting no pattern or consistency in accuracy in relation to number of instructions. However, the extent of fluctuation in accuracy also varied drastically in self-hosted compared to other platforms and higher sized models exhibited relatively lesser fluctuation in accuracy.

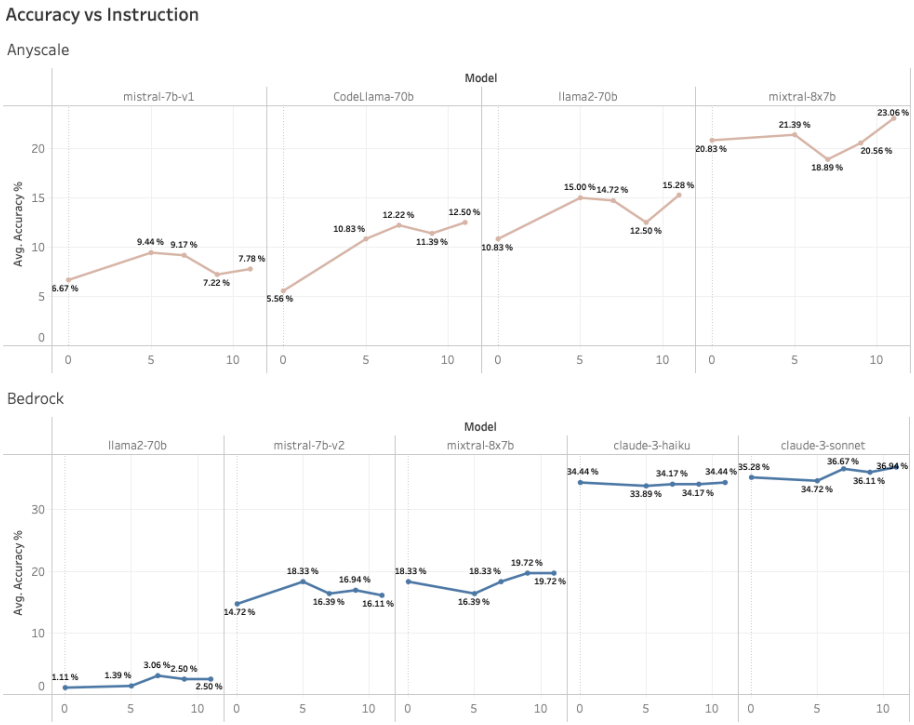

Figure 6. EX % comparison between models within Anyscale and Bedrock. The chart shows how EX changes from 0 to 11 instructions.

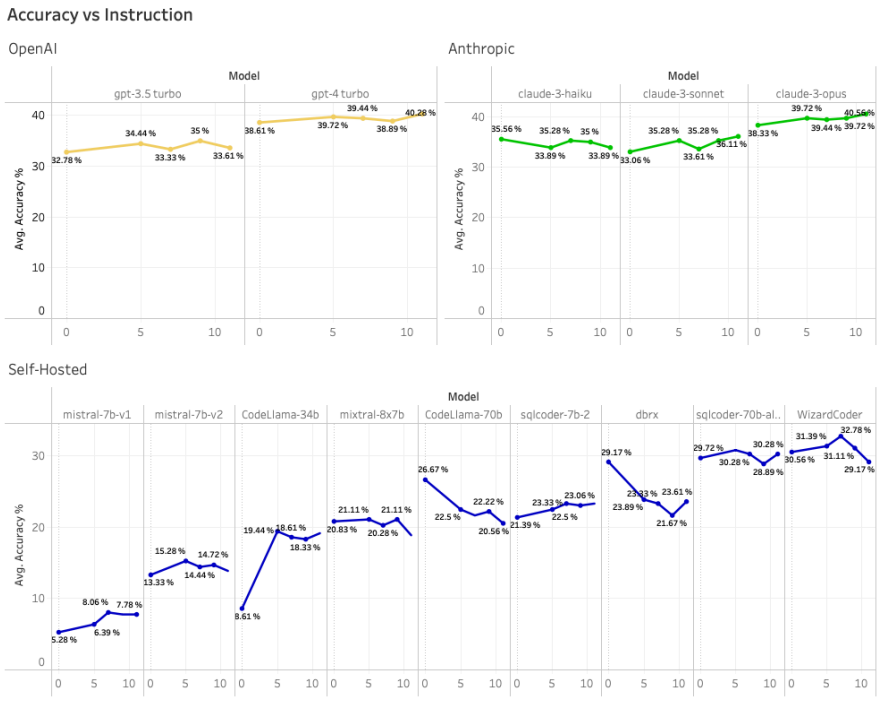

Figure 7. EX % comparison between models within OpenAI, Anthropic and Self-Hosted. The chart shows how EX changes from 0 to 11 instructions.

Cost

Total cost serves as the primary metric for evaluating the expenditure associated with operating various large language models (LLMs) across different hosting platforms. Our analysis also considered how costs fluctuate based on factors such as model size, question hardness, and the number of instructions provided.

Total cost incurred for running each model for 1800 inferences

Our findings indicate that self-hosted models incurred significantly higher costs compared to those hosted on other platforms. Notably, the models Codellama70b and DBRx Instruct lead in expenses, costing $94.82 and $88.81, respectively (Figure 8). Other models like Mixtral 8x7b and Coellama 34b also showed considerable costs. Following closely are anthropic claude 3 Opus at $47.3 and GPT 4 turbo at $22.61. The lowest expenses were observed for models operated on Anyscale.

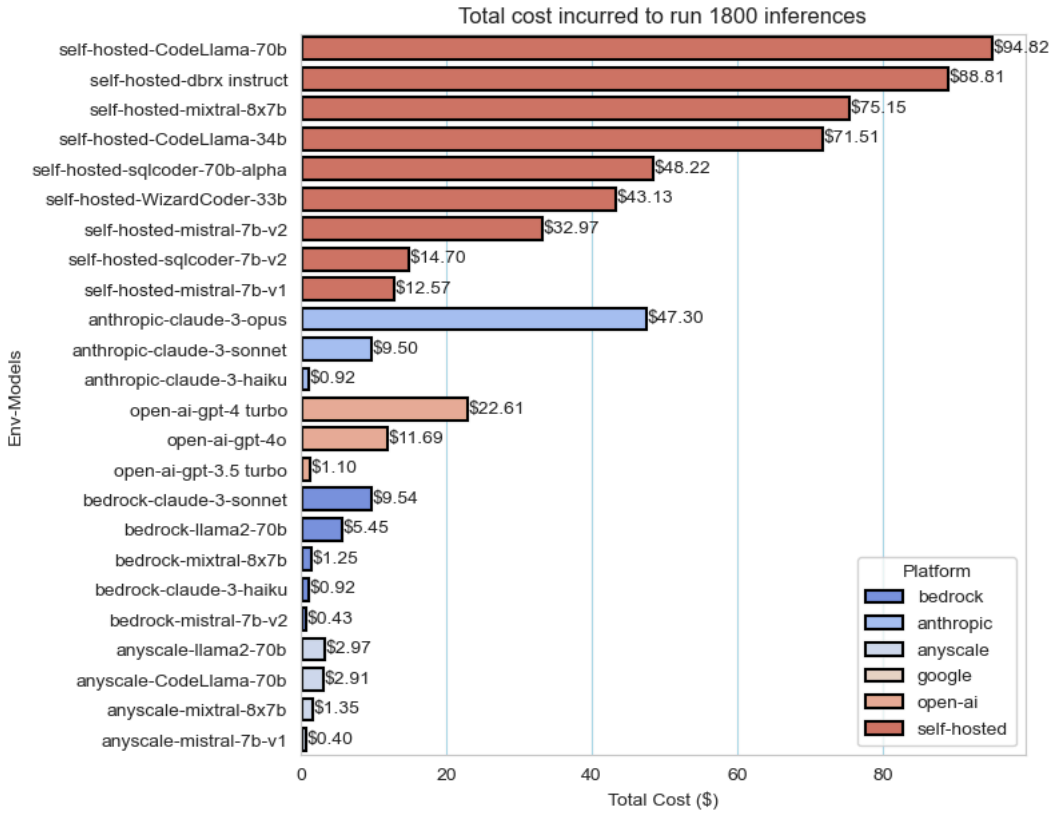

Figure 8. Total cost incurred to run all models for 1800 inferences.

Also, the cost incurred by the latest GPT-4 Omni model, at $11.69, was approximately half of the $22.61 cost associated with the GPT-4 Turbo model.

Cost comparison of models run in multiple platforms

From both Figure 8 and 9, we can observe that there exists a significant disparity in costs between self-hosted and other hosting platforms. For instance, the codelama70b model on a self-hosted platform incurred a cost of $94.82 compared to just $2.91 on Anyscale. Similarly, the Mixtral 8x7b model costs approximately the same on Anyscale and Bedrock; however, it required $75.15 to run on a self-hosted platform. While the cost differences among Anyscale, Bedrock, and Anthropic platforms are minimal, self-hosting significantly escalated the costs.

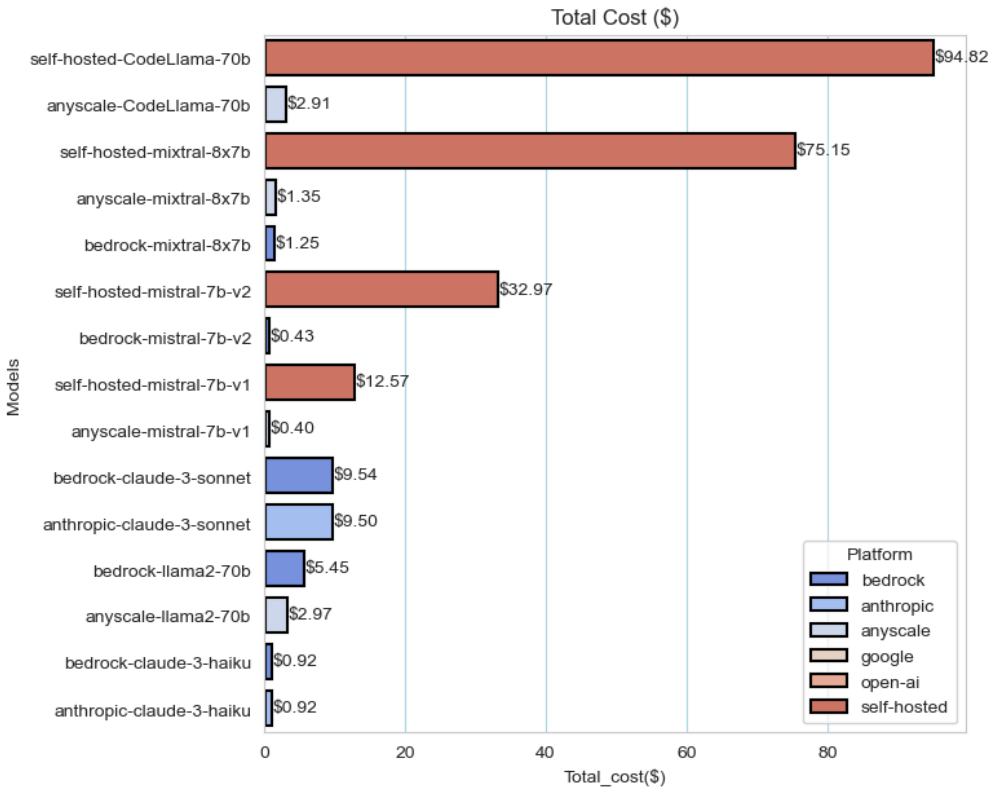

Figure 9. Total cost spent to run the models between platforms (self hosted vs other platforms)

The stark cost disparity between self-hosted and other platforms can be attributed to the differing basis for cost calculation. For non-self-hosted platforms, costs are determined by the number of input and output tokens used to run the models. Conversely, for self-hosted models, costs are calculated based on inference time or network latency - the time taken to process inferences. Because of this, it is important to look into the cost comparison separately between self-hosted and non-self hosted platforms.

Total cost incurred for non self-hosted platforms

From Figure 9, we can see that Anthropic's Claude3 Opus turned out to be the most expensive model ($47.3) followed by GPT-4 turbo ($22.61) and GPT-4 Omni ($11.69). For these models hosted on different platforms, the cost is calculated by number of input tokens and number of output tokens. Below is the formula for cost of a model:

Total Model Costs = {(Total number of input tokens x Cost per input token) **+** (Total number of output tokens x Cost per output token) }

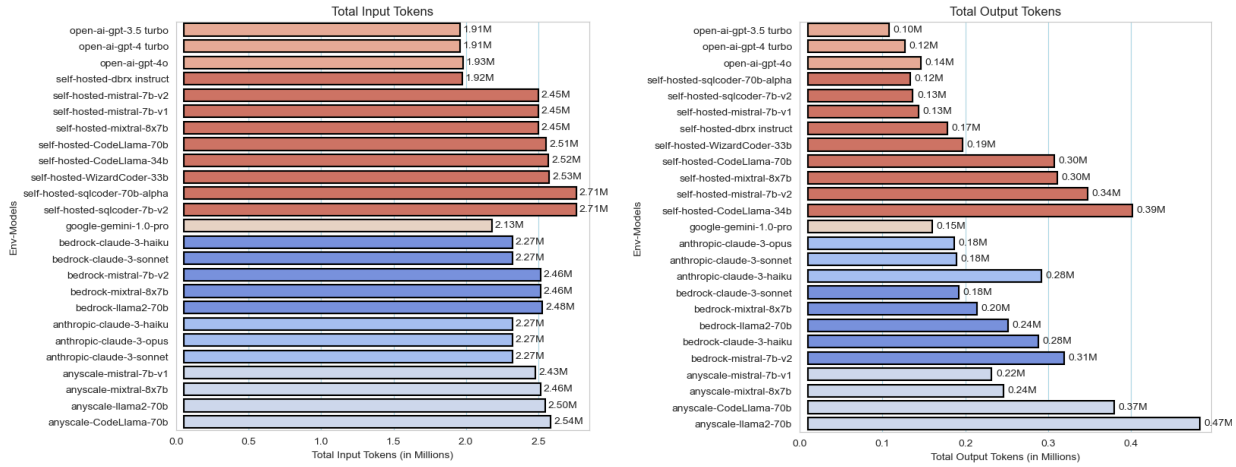

Figure 10. Total number of input tokens and total output tokens used for running all models across 1800 inferences.

From Figure 10, we can note that for non self-hosted models, the number of input tokens vary from one another as each model uses its own tokenization method. Similarly, models also vary from one another in terms of output tokens generated. As a result there will also be a difference in cost incurred. In terms of output tokens, we can observe that Open AI models have the least number of output tokens compared to other platforms. In Appendix B, observe the cost per million tokens for each model in various platforms is provided.

Total cost incurred for self-hosted models

Unlike other platforms, costs of self-hosted models are calculated based on the amount of time taken to run inferences, otherwise called network latency. From figure 9, we can note that the total cost of running a model self-hosted is higher than when they are run on Bedrock or Anyscale. This is primarily due to higher latency of running the model on Bedrock or Anyscale versus self-hosted as shown in Figure 11.

Observe that CodeLlama-70b and DBRX Instruct have the highest network latency (21.11 sec and 17.78 sec, respectively). As a consequence, the total cost spent is also highest for them as shown on Figure 8.

But note that the latency of CodeLlama70b on Anyscale is only 7.87 sec compared to 21.11 sec on self-hosted. This could be attributed to the fact that while self-hosting, we only used the inherent versions of the models which come with base transformers and did not apply any inference engines above them. On the other hand, a platform such as Anyscale or Bedrock do perform certain optimizations and use inference engines to reduce the network latency.

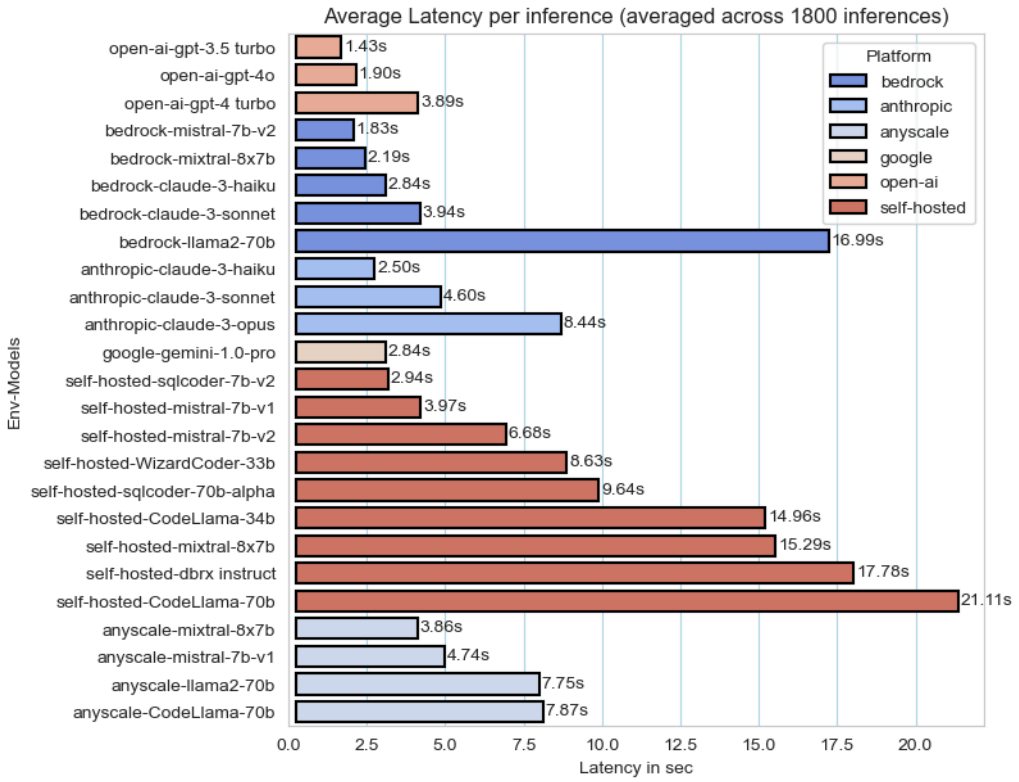

Figure 11. Average latency per inference.

Overall the above findings further emphasizes the economic impact of the hosting choice in deploying large language models.

Impact of query hardness and number of instructions on total cost spent

Query hardness is found to have a significant impact on cost in the self-hosted models as shown in Figure 12. This is primarily due to the fact that harder questions took more time to process with more latency. Higher latency impacted the cost directly. Also amongst non self-hosted models it is observed that queries of moderate level hardness cost marginally more than complex questions, as the total number of input tokens were found to be higher for moderate questions compared to the complex ones.

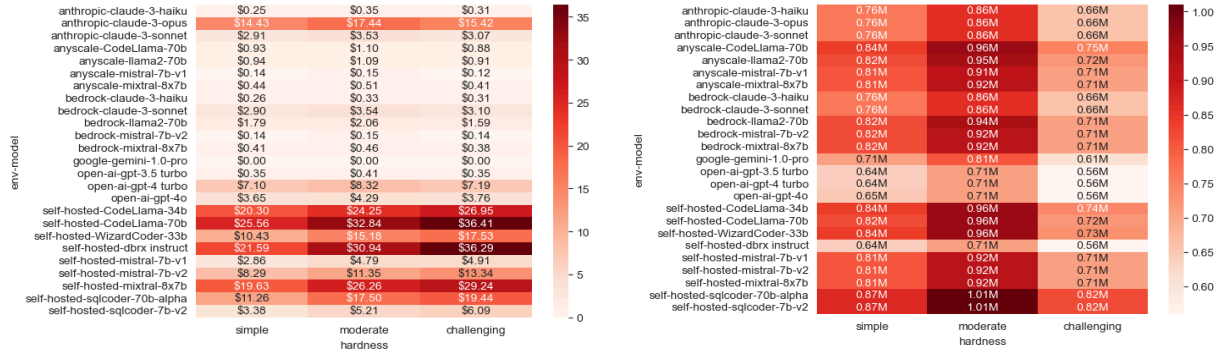

Figure 12. Total cost incurred for all models and total number of input tokens across different levels of query hardness.

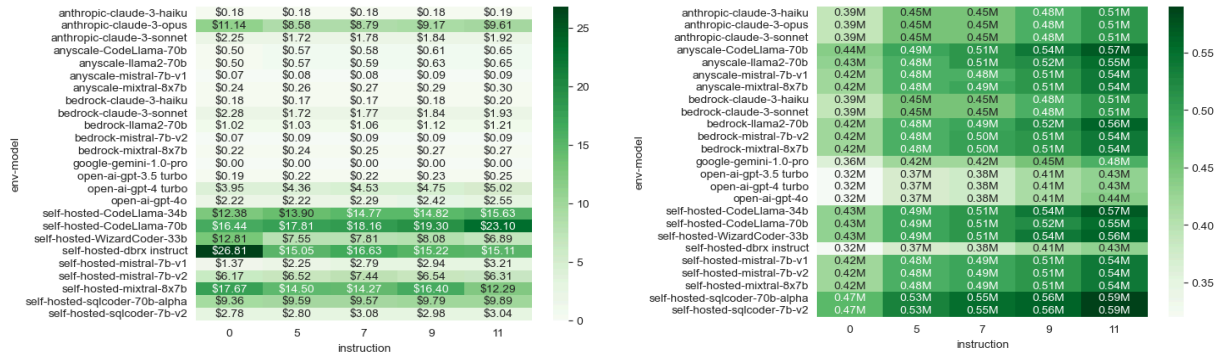

Figure 13. Total cost incurred for all models and total number of input tokens across different levels of instructions.

The number of instructions also had a marginal impact on the total cost across all models. For all non self-hosted models, increasing the number of instructions increased the cost spent. This is because the total number of input tokens is directly proportional to the number of instructions provided in the prompt as shown in Figure 13. However, in self-hosted models the total cost was not directly impacted by the size of the instructions and the extent of fluctuation in cost also varied from model to model.

Model accuracy versus total cost spent

When evaluating LLMs based on their accuracy relative to the total cost incurred, it becomes evident that a higher financial investment does not necessarily translate to superior accuracy. For instance, substantial sums were allocated to self-hosted models, yet their accuracy was lower in comparison to larger models from companies like OpenAI, Anthropic, or even Google's Gemini 1.0 pro. A striking example is the contrast between Codellama70b, which cost $94.82, and Claude3 Haiku, which cost a mere $0.92, but achieved significantly higher accuracy than Codellama70b. This observation underscores the pivotal role

played by the platform on which the models are run, as it can profoundly impact their performance. However note that, even while self-hosting the cost and latency could significantly get lowered when inference engines are applied. This is the scope of future research.

Furthermore, the language models situated in the top-left quadrant of the graph in Figure 14, such as Haiku, Sonnet, and GPT-4 Omni, are the ideal options while considering accuracy versus cost. These models offer an optimal combination of lower costs and higher accuracy when compared to models like GPT-4 Turbo and Opus. Among these top-performing models, GPT-4 Omni emerges as the most ideal choice, providing the highest accuracy at a lower cost, even surpassing the accuracy of Opus.

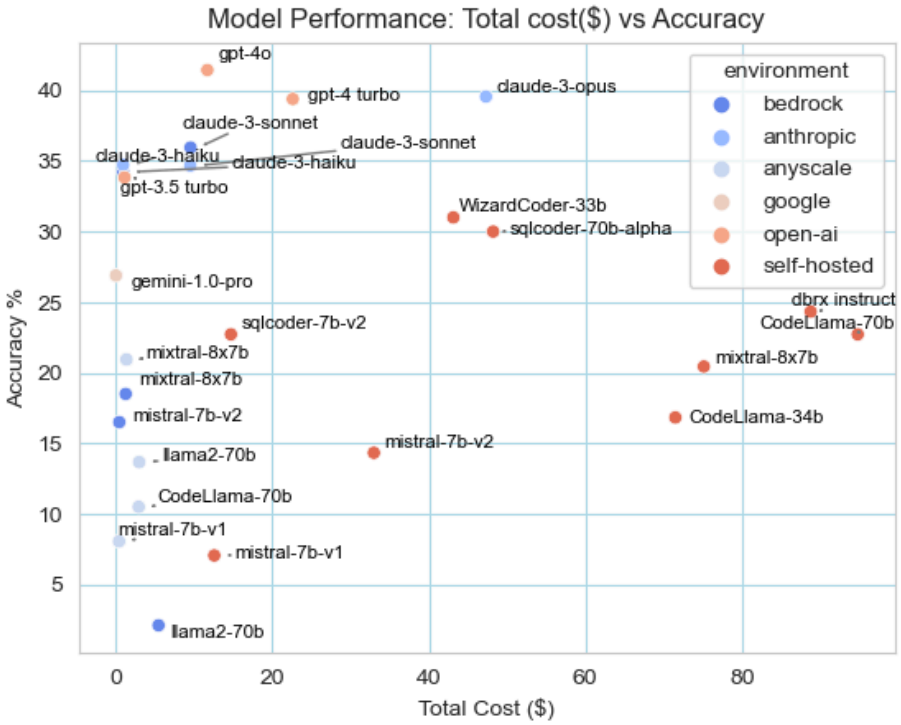

Figure 14. Total cost incurred for all models in relation to the model accuracy.

Following GPT-4 Omni, Claude3 Haiku stands out as an even more cost-effective option. Despite its lower cost compared to GPT-4 Omni, Claude3 Haiku achieves accuracy on par with Sonnet and even outperforms Sonnet in simpler query tasks.

Throughput

In the context of LLMs, throughput is a metric that refers to the amount of text or the number of tokens that a model can process within a given time frame. It measures a model's efficiency and performance in handling text input and generating output. It is measured by the unit called tokens per second

(tokens/sec). A higher throughput means that the model can process more tokens in a shorter amount of time, which is desirable for applications and end users.

To evaluate the throughput of the LLMs in our study, we employed the following formula:

Throughput = Total Output Tokens / Elapsed Time (Including Network Latency)

Specifically, we divided the total number of tokens generated as output by the LLM by the total time taken for the model to process the input, perform inference, and produce the output, including the time required for data transmission over the network. This approach allowed us to quantify the throughput in terms of tokens per second, taking into account the impact of network latency, which is a crucial factor when assessing the performance of LLMs deployed in a platform server.

Average throughput of all models across 1800 inferences

The study showed that the Mistral models on bedrock, specifically Mistral 7b v2 and Mixtral 8x7b, achieved the highest throughputs of 87.11 tok/sec and 55.31 tok/sec, respectively. Additionally, the Claude3-Haiku model on Anthropic exceeded the performance of Mixtral 8x22b with a throughput of 62.27 tok/sec (Figure 15). Amongst Open AI models, GPT-4 omni has higher throughput (40.32 tok/sec), even compared to GPT-3.5 turbo (36.24 tok/sec) and GPT-tubro (17.51 tok/sec).

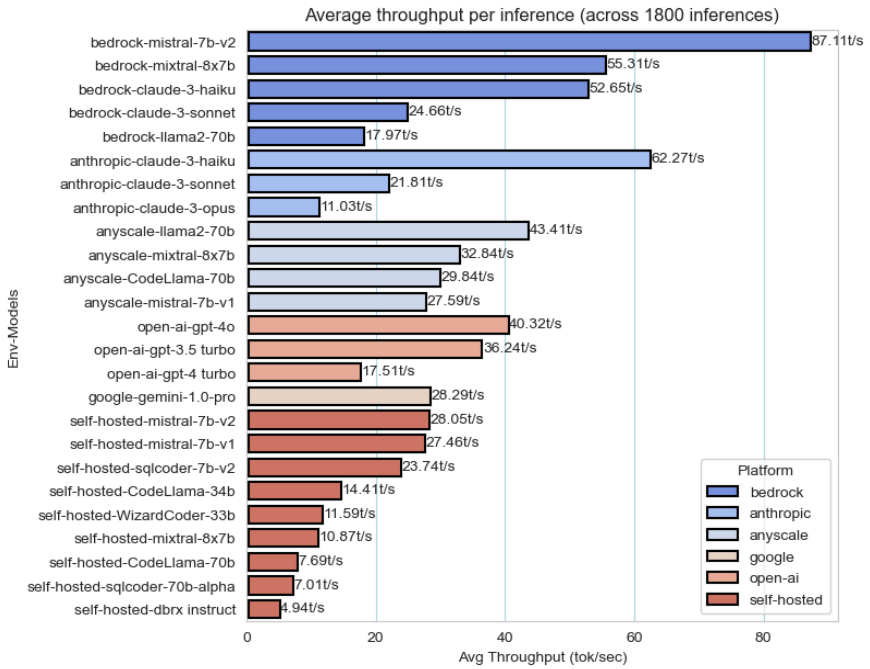

Figure 15. Throughput of all models across different platforms.

Throughput of models run in different platforms

We also observed variations in throughput for the same model across different platforms (Figure 16). For instance, the throughput of the Haiku model on Anthropic is 62.27 tok/sec compared to 52.65 tok/sec on Bedrock. Additionally, there is a significant difference in the throughput of the Mixtral 8x7b model, which stands at 55.31 tok/sec on bedrock, 32.84 tok/sec on Anyscale, and only 10.87 tok/sec when self-hosted. The lower throughput observed in self-hosted models can be attributed to the higher latency noted in previous sections, which considerably extends the processing and output production time.

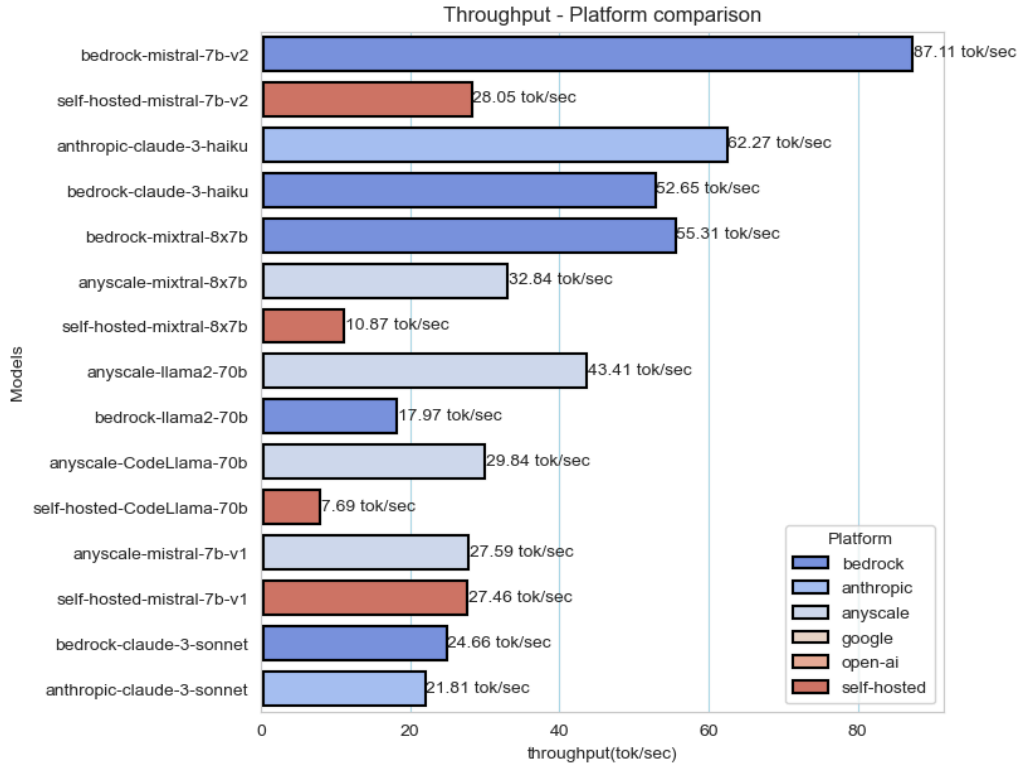

Figure 16. Throughput of all models across different platforms.

Variation of throughput across model sizes

From Figure 17, notice that throughput is lower for models of higher sizes. For example, this can be clearly observed in GPT-4 turbo (17.51 tok/sec), Claude3 Opus (11.03 tok/sec) and DBRX Instruct (4.94 tok/sec). Interestingly GPT-4 Omni despite being a higher model has throughput (40.32 tok/sec) that is at par with models of medium sizes. Amongst medium sizes, Haiku on Anthropic tops the list with 62.27 tok/sec.

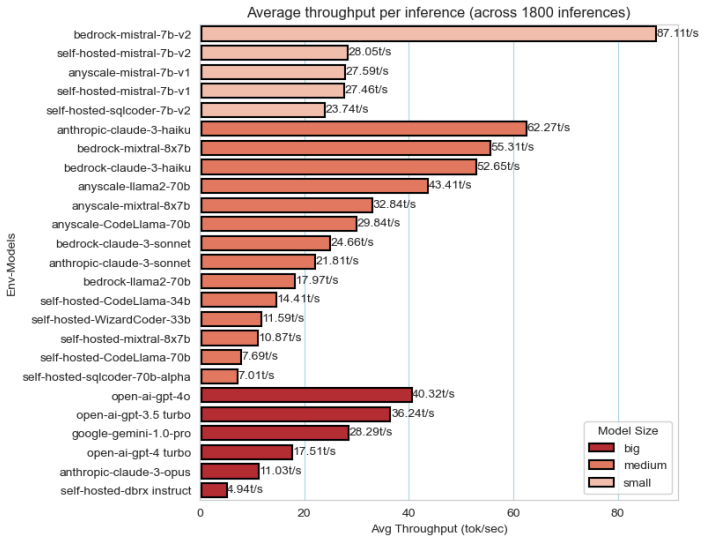

Figure 17. Throughput of all models across different platforms grouped by model size.

Impact of query hardness and number of instructions on throughput

Query hardness did have a marginal impact on the throughput of the models. Note that the throughput increases for all the models when the complexity of the query increases (Figure 18).

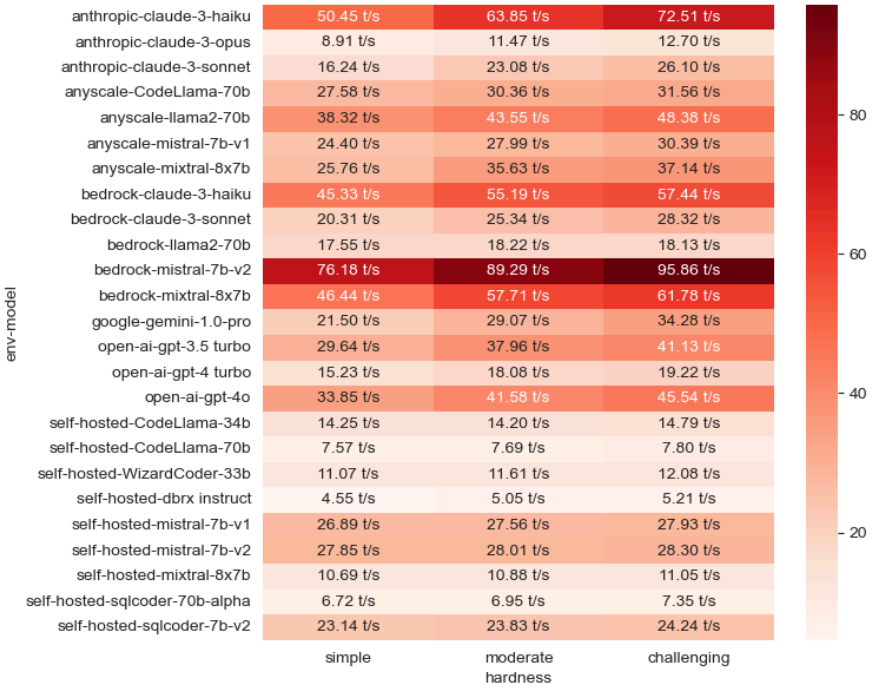

Figure 18. Throughput of all models across different query hardness.

This is primarily due to the increase in number of output tokens produced for more complex queries when we move from simple to harder questions.

However, it was observed that the number of instructions provided did not impact throughput as shown in Figure 19.

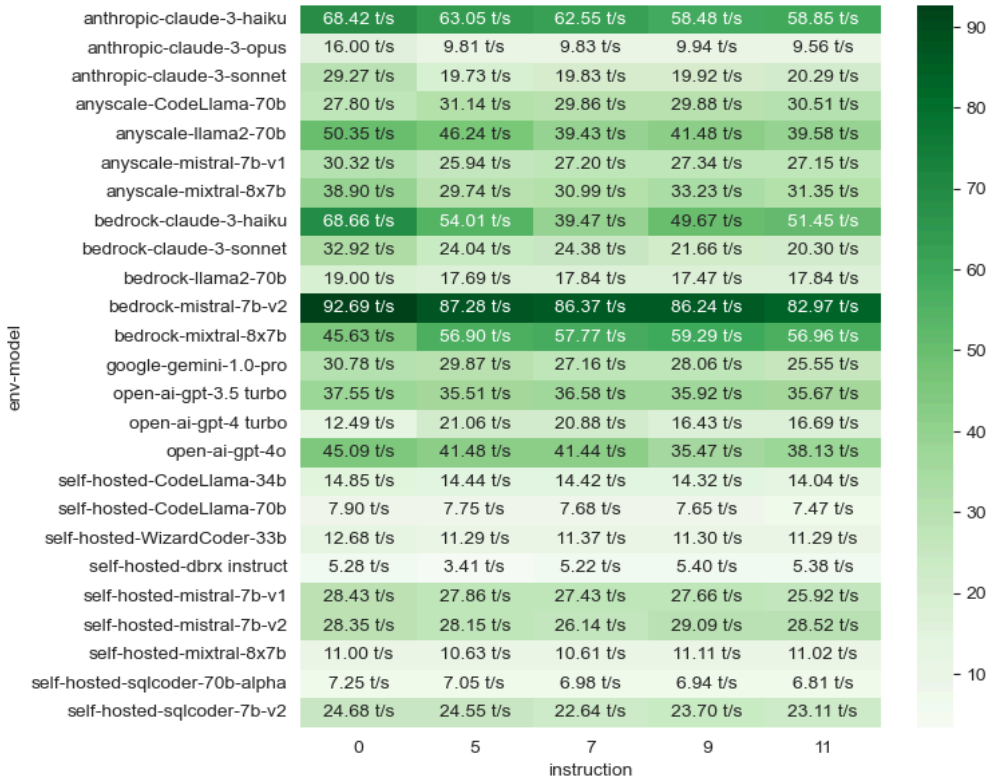

Figure 19. Throughput of all models across different levels of instructions.

Model accuracy versus throughput

Analyzing the relationship between throughput and accuracy provides crucial insights. Ideally, a model should exhibit both high throughput and accuracy. According to Figure 20, the Claude3 Haiku model on both Bedrock and Anthropic platforms stands out as an ideal choice, demonstrating superior throughput and accuracy compared to other models. However, if one is willing to accept a slight reduction in throughput for even greater accuracy, the GPT-4 Omni emerges as the optimal choice compared to any other model.

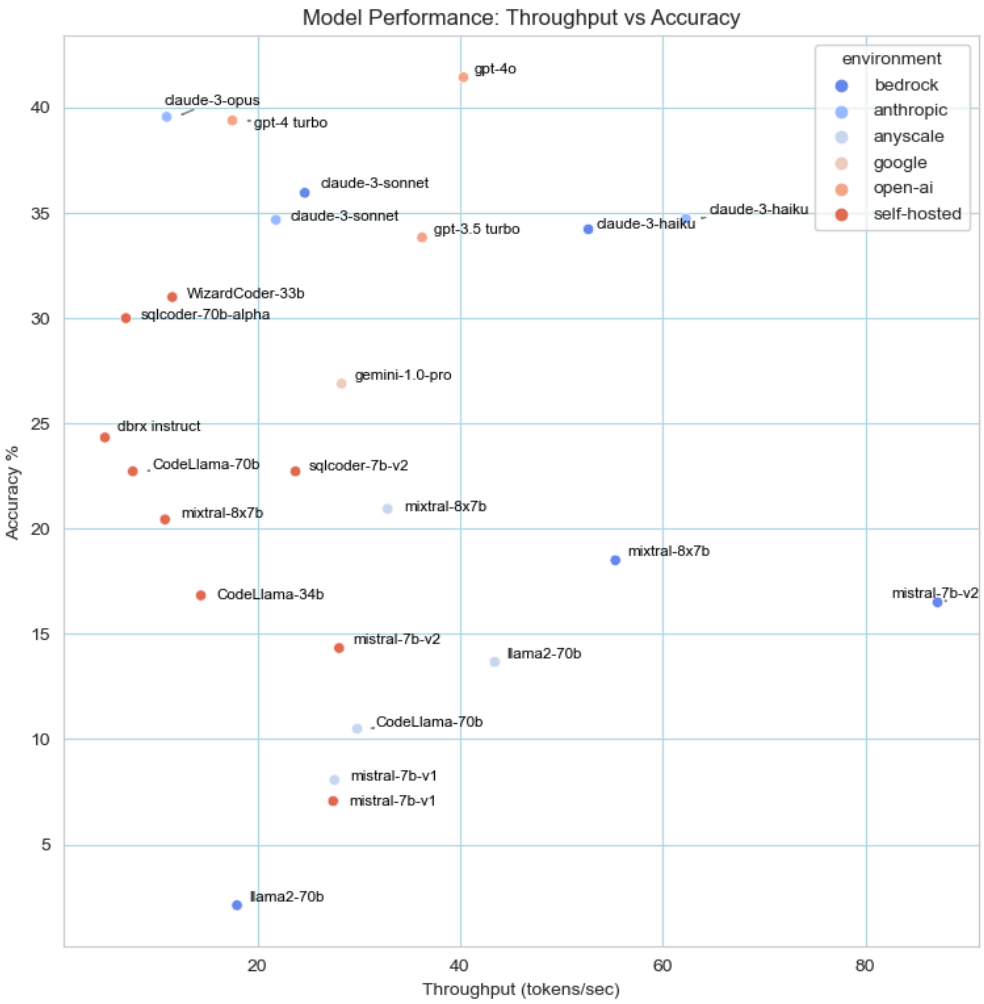

Figure 20. Average throughput per inference in relation to the model accuracy.

Total cost versus throughput

When selecting models based on higher throughput and cost-effectiveness, we observed that models from the Mixtral family, specifically Mistral 7b v2 and Mixtral 8x7b, offer the highest throughput at a lower cost. Additionally, the Claude3-Haiku model also presents itself as an ideal option, combining high throughput with affordability (Figure 21). Among the offerings from OpenAI, the GPT-4 Omni stands out as it matches the top models in performance while being more economical. This makes it a particularly attractive choice for those looking to balance cost with computational efficiency, making it suitable for a variety of applications where budget constraints are a consideration without sacrificing performance.

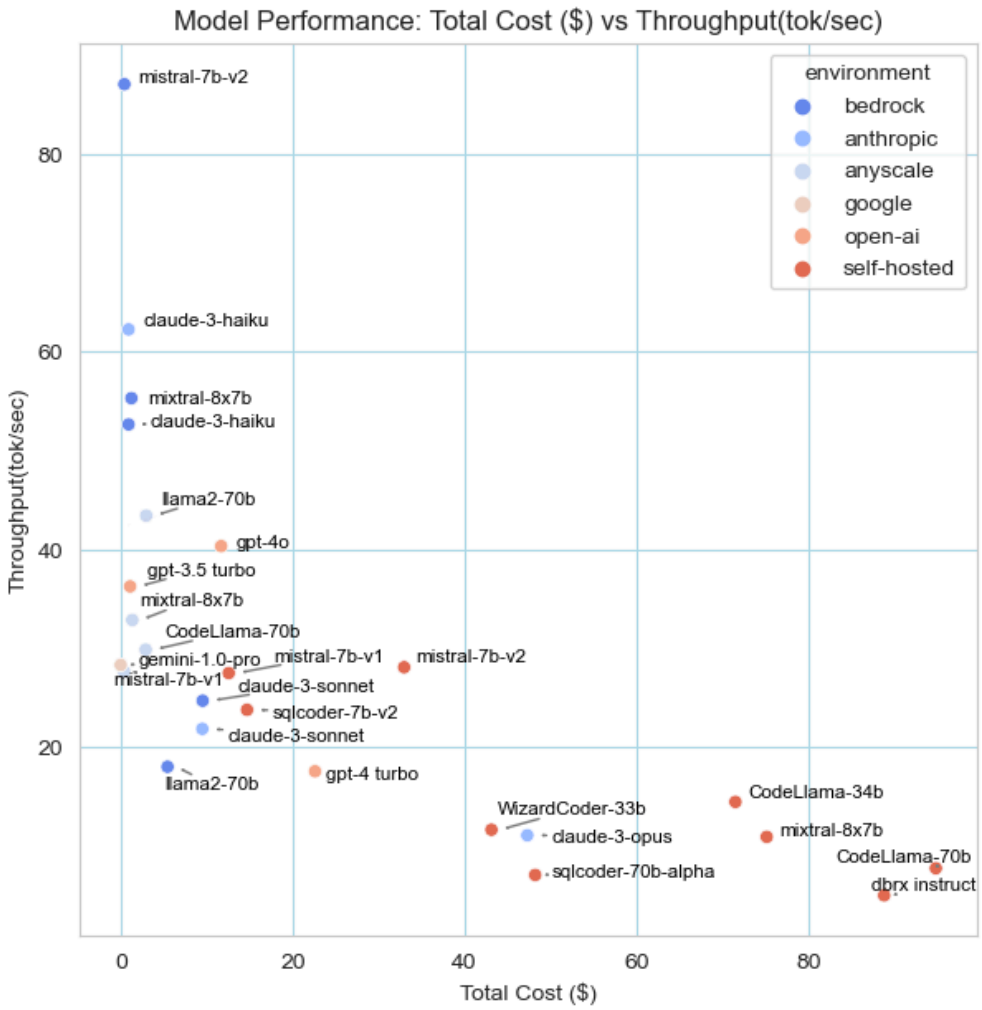

Figure 21. Average throughput per inference in relation to the total cost.

Discussion and Key Takeaways

This study reveals detailed results on performances of LLMs in the context of NL-to-SQL translation tasks, focusing on three crucial performance metrics: execution accuracy, throughput, and total cost. The study evaluates various LLMs across different hosting platforms, offering insights into the interplay between model performance and hosting platform factors. Here, we elucidate the nuanced findings from our comprehensive evaluation, discuss the implications of these results, and identify potential areas for future research and optimization.

Accuracy and Platform Influence

The findings clearly demonstrate that model performance in terms of accuracy is not solely dependent on the model's architecture but is significantly influenced by the choice of hosting platform. OpenAI's GPT-4 Omni and the Claude3 models from Anthropic consistently outperformed other models in terms of accuracy. This high level of performance across diverse hosting environments makes GPT-4 Omni particularly suitable for complex tasks requiring high fidelity in query translation.

However, platform variability plays a critical role in model performance. For example Claude3 models showed minimal variability in accuracy when hosted on either the Bedrock or Anthropic platforms, suggesting that these models are robust to different hosting conditions. Llama2-70B, in contrast, exhibited substantial performance differences across platforms. This variability could be due to platform-specific optimizations or differences in how models interact with the underlying hardware and software configurations.

Throughput Efficiency

Throughput is a measure of how quickly a model can process inputs and generate outputs, is another critical factor. Our study highlights that Mistral family models (Mistral 7b v2 and Mixtral 8x7b) had the highest throughput, particularly when hosted on Bedrock platforms, underscoring the potential for platform-specific optimizations to enhance model efficiency. Self-hosted platforms generally showed lower throughput, primarily due to higher latencies that extended processing and output times. For example, throughput for Mixtral 8x7b was notably higher on Bedrock compared to self-hosted setups, which can impact decision-making on whether to opt for self-hosting versus managed cloud services.

Cost Considerations

Cost is a decisive factor for many organizations, influencing the scalability and accessibility of LLM technologies. The study identified significant cost implications associated with different hosting arrangements. Self-hosted models incurred higher costs, largely due to increased network latency and longer inference times when it is not optimized and used with any inference engines. Cloud-hosted

solutions like Anyscale provided more cost-effective alternatives, minimizing the economic barriers to leveraging advanced LLMs for a broader range of applications.

Impact of Latency

Latency not only affects throughput but also has a significant impact on cost. Models hosted in self-hosted environments often faced higher latency, which directly translated into increased costs and reduced throughput. This emphasizes the importance of considering hosting environments carefully and suggests a potential area for optimization. Future research could focus on reducing latency through better hardware configurations or more efficient use of networking resources.

Ideal Model Selection

Choosing the right model involves balancing accuracy, throughput, and cost. From the findings we understood that:

- GPT-4 Omni emerges as an exceptionally balanced option, providing high accuracy with relatively low costs compared to any other model across all hosting platforms.
- Claude3-Haiku offers a cost-effective alternative, particularly suitable for scenarios where high throughput is needed without compromising significantly on accuracy.

Conclusions and Future Directions

This benchmark study of LLMs for NL-to-SQL tasks highlights the critical interdependencies between model selection, hosting platform, and performance metrics. As LLM technologies continue to evolve, future research should focus on:

- Exploring optimization techniques for self-hosted models to enhance their cost-effectiveness and performance. This could involve investigating advanced inference engines and other hardware optimizations to reduce latency and enhance throughput without significant cost increases.
- Further investigations into platform-specific performances to understand and mitigate the impacts of environmental variability. This would help in developing more robust models that can deliver consistent performance regardless of the hosting conditions.
- Expanding the scope of Retrieval-Augmented Generation (RAG) and fine-tuning techniques is also critica. This is a major area for future research involving examining how RAG and various fine-tuning methods can influence the metrics of accuracy, throughput, and cost. These techniques could potentially improve model responsiveness and efficiency, especially in complex query environments where precision and speed are paramount. RAG could be particularly

beneficial in enriching the model's context before generating SQL queries, potentially increasing accuracy without a substantial cost increase.

By continuously refining these models and their deployment strategies, we can enhance their practical utility, making sophisticated language understanding technologies more accessible and effective across a variety of applications. Also, future research in these areas will not only help in optimizing existing models but also pave the way for newer approaches to deploying LLMs in real-world settings.

Appendix A

List of instructions provided in the prompt

1. The answer generated must only be an SQL query ending with delimiter ";"
2. Dedicate time to understand the database schema fully, identifying the relevant tables and columns that align with the query's objectives.
3. Utilize only the data from the specified tables in the provided database schema.
4. Pay attention to case sensitivity in data and ensure the extraction of required information aligns precisely with the specified columns and tables.
5. Analyze the query's requirements to determine the appropriate use of GROUP BY, HAVING, and UNION clauses, ensuring they contribute to the accurate aggregation and segmentation of data.
6. Pay careful attention to the primary keys, foreign keys present in the database schema to determine appropriate columns for JOIN operations.
7. Apply WHERE clause conditions accurately to filter the dataset based on specified criteria.
8. Apply WHERE clause conditions accurately to filter the dataset and use ASC or DESC in sorting results where specified.
9. Assign meaningful aliases to tables and columns where necessary, especially in cases of grouping or joining, to enhance the clarity and maintainability of the SQL query.
10. When multiple tables are involved, prioritize selecting appropriate columns based on context and handle null values properly in columns.
11. Avoid any write operations (like modify, update, delete, or drop). Should the task demand such actions, respond with a polite refusal, stating, "I'm sorry, but I can't assist with that."

Appendix B

Cost per million input and output tokens for non self-hosted models

Platform	Model	Input cost per million tokens	Output cost per million tokens
Open AI	GPT-4 Turbo	$10	$30
	GPT-4 Omni	$5	$15
	GPT-3.5 Tubo	$0.5	$1.5
Anthropic	Claude3 Opus	$15	$75
	Claude3 Sonnet	$3	$15
	Claude3 Haiku	$0.25	$1.25
Bedrock	Claude3 Sonnet	$3	$15
	Claude3 Haiku	$0.25	$1.25
	Llama2-70b	$1.95	$2.56
	Mixtral-8x7b	$0.45	$0.7
	Mistral-7b	$0.15	$0.2
Anyscale	Llama-2-70b-	$1	$1
	CodeLlama-70b	$1	$1
	Mixtral-8x7B	$0.5	$0.5
	Mistral-7B	$0.15	$0.15

References

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

Iyer, S., Konstas, I., Cheung, A., & Zettlemoyer, L. (2018). Mapping language to code in programmatic context. arXiv preprint arXiv:1808.09588.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., ... & Li, Y. (2024). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Advances in Neural Information Processing Systems, 36.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., ... & Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887.