DSBC : Data Science task Benchmarking with Context engineering

Anonymous ACL submission

Abstract

Recent advances in large language models (LLMs) have significantly impacted data science workflows, giving rise to specialized data science agents designed to automate analytical tasks. Despite rapid adoption, systematic benchmarks evaluating the efficacy and limitations of these agents remain scarce. In this paper, we introduce a comprehensive benchmark specifically crafted to reflect real-world user interactions with data science agents by observing usage of our commercial applications. We evaluate three LLMs: Claude-4.0-Sonnet, Gemini-2.5-Flash, and OpenAI-o4-Mini across three approaches: zero-shot with context engineering, multi-step with context engineering, and with SmolAgent. Our benchmark assesses performance across a diverse set of eight data science task categories, additionally exploring the sensitivity of models to common prompting issues, such as data leakage and slightly ambiguous instructions. We further investigate the influence of temperature parameters on overall and task-specific outcomes for each model and approach. Our findings reveal distinct performance disparities among the evaluated models and methodologies, highlighting critical factors that affect practical deployment. The benchmark dataset and evaluation framework introduced herein aim to provide a foundation for future research of more robust and effective data science agents.

1 Introduction

Large Language Models (LLMs) have recently gained prominence due to their capability to automate and enhance various data science tasks. This growing capability has led to increased adoption of specialized data science agents across multiple domains. Despite widespread usage, there is a clear gap in comprehensive evaluations that accurately reflect practical user interactions and realistic task scenarios. This gap makes it challenging for practitioners and researchers to understand the true efficacy and limitations of these agents in applied settings.

In response to this, we introduce a detailed benchmark tailored to reflect actual usage patterns of data science agents, derived from observations of end-user behavior. We evaluate three leading LLMs, Claude-4.0-Sonnet (Anthropic, 2025), Gemini 2.5 Flash (Team, 2025), and OpenAI-o4-Mini (OpenAI, 2025), using three distinct methodologies: zero-shot with context engineering, multi-step with context engineering, and using SmolAgent (Roucher et al., 2025). Our benchmark covers diverse and practical data science tasks while also examining how sensitive these models are to common prompting issues like data leakage and slight ambiguity.

The performance of agents or Large Language Models (LLMs) critically depends on the context provided during inference, including both the maximum context length available and the efficiency of its utilization. (Mei et al., 2025). While most benchmarks come with a manually written or synthetically generated summary or a description of the datasets used, we use a standardized context through context engineering.

Additionally, we analyze the impact of varying temperature settings on both overall performance and task-specific effectiveness. Our findings underscore significant differences in the capabilities of evaluated models and strategies, identifying crucial considerations for practical deployment. Through this work, we aim to provide a foundational resource for furthering the development of more reliable and effective data science agents.

Prior Work	Domain of Benchmark	Context added about data files	Avg. row count In Data files	Avg. col count In Data files	Span Multiple task types	Prompt Variations	Temp. Variations	Sample Count
Other Domain								
Spider (Yu et al., 2018a)	Text-to-SQL	N/A	-	-	-	No	No	1,034
MLAgentBench (Huang et al., 2024a)	Machine learning	N/A	-	-	-	No	No	13
SWE-Bench (Jimenez et al., 2024)	Software engineering	N/A	-	-	-	No	No	2,294
Same Domain								
DS-1000 (Lai et al., 2023)	Data Science	Manually	N/A	N/A	No	Yes	No	1,000
QRData (Liu et al., 2024)	Data Science	Manually	15,186	46	Yes	No	No	411
Arcade (Yin et al., 2023)	Data Science	Notebook Cells	N/A	N/A	-	No	No	1,078
Spider2V (Cao et al., 2024)	Data Science	Manually	N/A	N/A	-	-	Yes	494
DSEval (Zhang et al., 2024)	Data Science	Manually	1,544	12	-	No	Partial	827
DSBench (Jing et al., 2025)	Data Science	Data Files	N/A	N/A	-	No	No	466
DA-Code (Huang et al., 2024b)	Data Science	Manually	9,639	11	No	No	No	500
DataSciBench (Zhang et al., 2025)	Data Science	Manually	N/A	N/A	-	No	No	222
DABstep (Egg et al., 2025)	Data Science	Manually	N/A	N/A	-	No	No	450
Ours (DSBC)	**Data Science**	Structured	7,793	10	Yes	Yes	Yes	303

Table 1: Overview of some similar prior works in other domains and other existing Data Science benchmarks.

2 Related Works

An overview of key differences between our benchmark and other benchmarks of the same domain can be seen in Table 1. Examples of samples for each of the benchmarks can be seen in Figure 1.

Other similar domain benchmarks: Several prior works exist that benchmark code generation and other closely related tasks. One such prominent one is HumanEval (Chen et al., 2021) for code generation from text descriptions, along with Spider (Yu et al., 2018b), MBPP (Austin et al., 2021), and APPS (Hendrycks et al., 2025). Other similar works include software engineering task benchmarks like automating code/PR review (Yang et al., 2016; Li et al., 2022; Tufano, 2023), bug localization (Kim et al., 2019; Chakraborty et al., 2018), testing (Kang et al., 2023; Xia et al., 2024; Wang et al., 2024), program repair (Xia and Zhang, 2022; Fan et al., 2023; Sobania et al., 2023), and guiding code-editing (Chakraborty and Ray, 2021; Zhang et al., 2022; Fakhoury et al., 2023).

Data-Science Benchmarks : DS-1000 (Lai et al., 2023) is one of the earliest works for data science task evaluation with simple data analysis tasks sourced from StackOverflow; however, these questions lack usage of data files and require simpler 1-2 line code for many samples. DABstep (Egg et al., 2025) contains both unstructured and structured data files for the benchmark with most queries that can be solved in under 5 lines of code beyond loading of data files. DataSciBench (Zhang et al., 2025) consists of prompts that resemble text-to-code instructions, with the query itself providing context on necessary data file information. DA-Code (Huang et al., 2024b) also consists of queries that resemble text-to-code tasks, as the queries themselves provide step-by-step instructions on what steps to take to solve the given query. DS-Bench (Jing et al., 2025) contains tasks that are primarily designed for Excel-based workflows sourced from Modeloff competitions. DSEval (Zhang et al., 2024) covers queries whose solutions mostly range from 1 to 3 lines of code, with a subset of questions being Leetcode problems. Arcade (Yin et al., 2023) consists of tasks in a notebook environment, while the previous cell of code provides the required context. QRData (Liu et al., 2024) is a benchmark that closely resembles our benchmark with manually written data file descriptions appended to input prompts to provide the models with context. However, a majority of questions were MCQs.

No.of Task types	Sample Count
One task categories	63
Two task categories	154
Three task categories	86

Table 2: Multi-label task category statistics of the queries (303) in our benchmark

3 Our Benchmark

Most prior benchmarks either have sourced or manually created descriptions of the data files used for the task, which are passed along with the query to the evaluation models. Few other works use data files of limited size that can be directly added in the context. This introduces randomness in the results based on how the data files' descriptions are written, what information is added, and what is withheld.

The benchmark consists of 303 questions, which

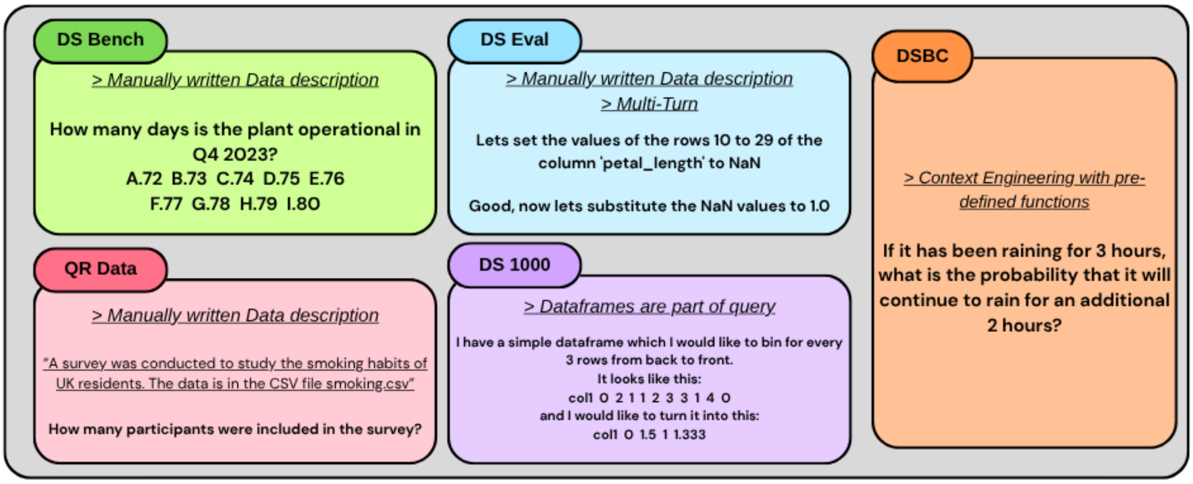
DS Bench
> Manually written Data description
How many days is the plant operational in Q4 2023?
A.72 B.73 C.74 D.75 E.76
F.77 G.78 H.79 I.80
DS Eval
> Manually written Data description
> Multi-Turn
Lets set the values of the rows 10 to 29 of the column 'petal_length' to NaN
Good, now lets substitute the NaN values to 1.0
DSBC
> Context Engineering with pre-defined functions
If it has been raining for 3 hours, what is the probability that it will continue to rain for an additional 2 hours?
QR Data
> Manually written Data description
"A survey was conducted to study the smoking habits of UK residents. The data is in the CSV file smoking.csv"
How many participants were included in the survey?
DS 1000
> Dataframes are part of query
I have a simple dataframe which I would like to bin for every 3 rows from back to front.
It looks like this:
col1 0 2 1 1 2 3 3 1 4 0
and I would like to turn it into this:
col1 0 1.5 1 1.333

Figure 1: An example each from our benchmark along with some works cited above which were published less than 2 years ago.

span 8 categories of tasks, with most questions covering more than one type of task. The categories and their descriptions, as well as their sample counts in the benchmark, can be seen below. Table 2 shows the number of task categories each sample in the benchmark spans.

- **Correlation Analysis (44) :** Computing and interpreting relationships between variables using correlation coefficients, covariance matrices, and statistical significance testing.
- **Statistics (172) :** Performing descriptive and inferential statistical operations including confidence intervals, probability calculations, and computing mean, min, max, median etc.
- **Data Parsing & Understanding (113) :** Interpreting data structure, content patterns, and contextual meaning to infer data origins, identify column semantics, and extract implicit information from datasets.
- **Data Pre-processing (69) :** Cleaning and preparing raw data through handling missing values, removing duplicates, removing empty rows and columns, removal of redundant features, outlier detection, and basic data quality assurance.
- **Feature Engineering (91) :** Creating new variables and features from existing data through mathematical transformations, aggregations, binning, cumulative and rolling features, and domain-specific feature construction.

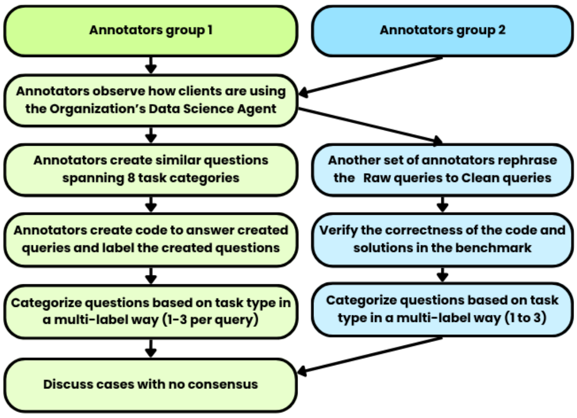
Annotators group 1
Annotators group 2
Annotators observe how clients are using the Organization's Data Science Agent
Annotators create similar questions spanning 8 task categories
Another set of annotators rephrase the Raw queries to Clean queries
Annotators create code to answer created queries and label the created questions
Verify the correctness of the code and solutions in the benchmark
Categorize questions based on task type in a multi-label way (1-3 per query)
Categorize questions based on task type in a multi-label way (1 to 3)
Discuss cases with no consensus

Figure 2: An overview of the annotation process used in our benchmark construction

- **Feature Transformation (85) :** Applying scaling, normalization, encoding categorical variables, dimensionality reduction, rounding and mathematical transformations.
- **Distribution Analysis (33) :** Examining data distributions through descriptive statistics, probability density functions, normality tests, and distributional property assessment.
- **Data Visualization (22) :** Creating charts, plots, and visual representations to explore patterns, communicate insights, and present analytical findings effectively.

Annotation : The overview of the annotation process can be seen in Figure 2. The annotators have created the dataset samples by observing how the clients were using the organization's data science agents. The distribution of task types was made to be close to the observed task type usage distribution. Queries were annotated, with their solutions manually coded and verified. Additionally, a copy of the raw queries was made to avoid data leakage, i.e., the clean queries. During annotation of task types for each query, the annotators were instructed to assign 1-3 task category tags to each query. In case the query appeared to be spanning more than 3 categories, they were instructed to assign the closest three. Samples where there was no consensus among the 2 sets of annotators were later manually verified after further discussion. The samples with no consensus were 26 out of 303, i.e., approximately 8.5%. More details about annotator guidelines can be found in Appendix C.

4 Context Engineering :

Organizations handling sensitive data face significant challenges when working with externally deployed or proprietary models. Direct file sharing raises privacy and security concerns, making it impractical to include raw data files in model contexts (even if they fit within the context limit). A structured approach that extracts only essential metadata, such as column counts, data types, categorical values, and temporal ranges, provides necessary context while maintaining data confidentiality and compliance requirements.

Further, manual dataset documentation becomes increasingly burdensome as organizations scale their data operations. Writing detailed descriptions for hundreds or thousands of data files is time-consuming and resource-intensive. While automated description generation using LLMs offers a potential solution, it often produces incomplete or inaccurate characterizations that can mislead downstream analysis and compromise benchmark reliability.

We hence use context engineering to provide the models with the data files' description in a structured format covering several features like row and column count, column names, and data types, among other features. This can be seen in detail in Table 3, which describes the features used in the context as a nested JSON dictionary.

5 Datasets used

The benchmarks utilize 11 different data files sourced from Kaggle, each of which covers a different domain: Farm produce data (Hirapara, 2023), Walmart sales (Mikhail, 2024), COVID-19 mortality data (Nizri, 2022), weather datasets (Biswas, 2024), insurance claims dataset (Choi, 2021), stock price datasets (Crow, 2020), food inflation data (Tanwar, 2023), world population stats (DS, 2024), air quality data (Jha, 2024), electricity load data (Shahane, 2023), and life expectancy datasets (Pedersen, 2023). The statistics about the data files can be seen in Table 4.

key Differences : The data files chosen for the benchmarks have tricky features, which would make the benchmarks' samples more challenging compared to the rest. For instance, 8 of the 11 data files have a date or time column but not in a date-time datatype. The models/agents were provided with the first 5 rows of the data file through context engineering and are required to comprehend and figure out whether they need to perform a datatype conversion to successfully solve a query. One such tricky feature is the change of data frequency. The frequency of data for the population dataset changes midway from once every 5 years to yearly. Using the provided unique values list, the model/agent is required to figure out the necessary changes in the approach towards solving a query. Issues like these make the current benchmark more challenging than prior works.

6 Query Types

The raw (original) queries were rewritten (referred to as clean queries) to ensure no data leakage occurs even though the impact is negligible. Though the effect could be negligible, this was done to compare the change in results with both sets of queries. Most queries of usage of our commercial agents resembled the format of raw queries. Table 5 shows an example for the raw and clean query of one of the samples. Another difference between Raw and Clean queries is that Clean prompts require an SQL-like approach, as demonstrated in Table 6.

7 Evaluation

We evaluate the samples using 3 models (Claude-4-Sonnet, Gemini-2.5-Flash, and OpenAI-o4-Mini) in 3 settings (directly with context engineering with

Item	Description
Rows	Number of Rows whether index is correctly ordered
Columns	Numbers of Columns and names of columns
Data Types	Data types of each of the columns
Null Counts	Null value counts of each of the columns
Numeric Summary	If the column is Numeric then the Minimum, Maximum, Mean, Median, 25th percentile value, 75th percentile value
Categorical Summary	If the column is Categorical or has low unique count (i.e <20), then the unique value counts
DateTime Summary	If the column is already DateTime type, then the start and end dates and whether the column values' frequency is uniform
Sample rows **	First 5 rows' data of each data file

Table 3: Features and description of info added through Context engineering

** If the data file is of private nature, this is excluded

Dataset	Rows	Cols	Null.cols	DT	STR	CAT	BOOL	INT	FLT	AVG.ACC
Insurance	1,200	7	0	0	3	5	2	2	2	71.78
Weather	8,616	8	0	0	2	3	0	2	4	53.02
Power	3,624	17	0	0	1	3	2	3	13	48.06
COVID	10,000	21	0	0	1	19	3	20	0	45.31
AQI	8,737	23	21	0	2	3	0	0	21	41.60
Inflation	4,548	8	5	0	3	2	0	0	5	40.33
Sales	409,695	5	0	0	1	2	1	2	1	36.92
Health	13,942	5	0	0	2	1	0	1	1	34.78
Stocks	4,932	7	0	0	2	1	0	1	4	33.08
Population	3,290	14	3	0	5	2	0	6	3	30.91
Production	9,464	9	6	0	1	3	0	2	5	27.34

Table 4: Features of datasets used for our benchmark : Row Count (ROW), Column Count (COLS), No.of columns with at least one null value (NULL.COLS), No.of Datetime (DT),String (STR), Categorical (CAT), Boolean (BOOL), Integer (INT) and float (FLT) columns respectively and the average score obtained from all attempts and temperature values over the queries using that dataset (AVG)

Queries Example 1	
Raw	Among those who have died, how many deaths were not attributed to COVID-19 ?
Clean	Did any deaths occur not due to COVID-19? If so, how many?

Table 5: An example of Raw and Clean prompts : The raw prompt assumes deaths occurred (data leakage), while the clean prompt requires the model to check if deaths happened first.

Queries Example 2	
Raw	Which three countries have had the most stable fertility rates ?
Clean	Which countries have had the most stable fertility rates? List the top 3.

Table 6: An example of Raw and Clean prompts : The raw prompt directly requests the top 3 countries, while the clean prompt follows a SQL-like approach of first identifying all stable countries, then selecting the top 3.

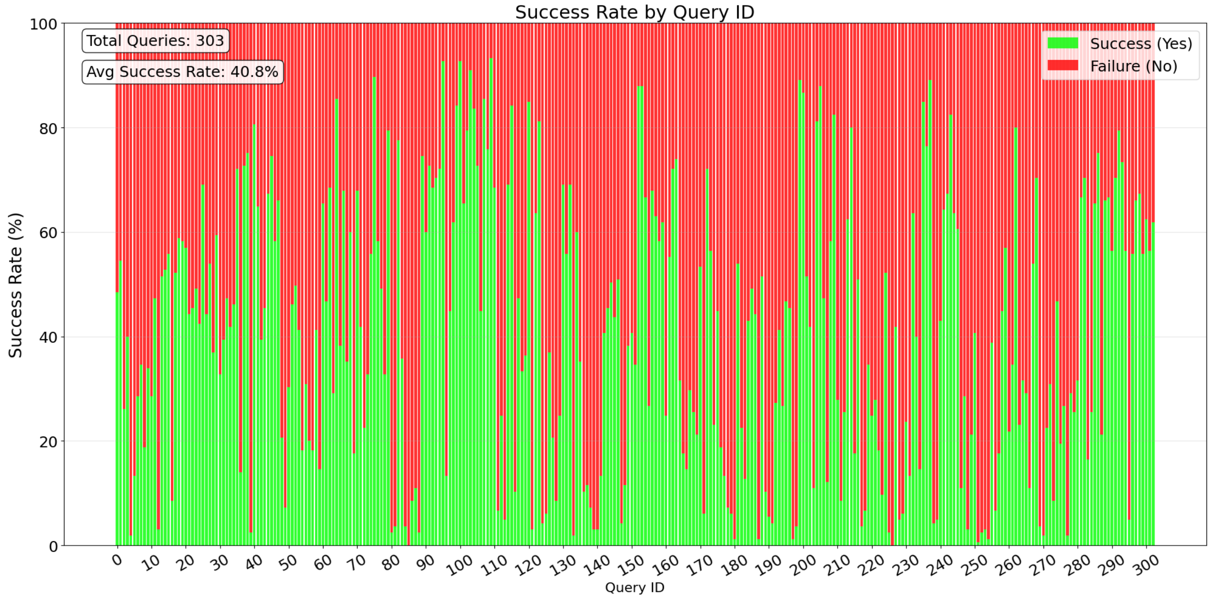

Figure 3: Success rates for each sample (303) of our benchmark over several (165) attempts

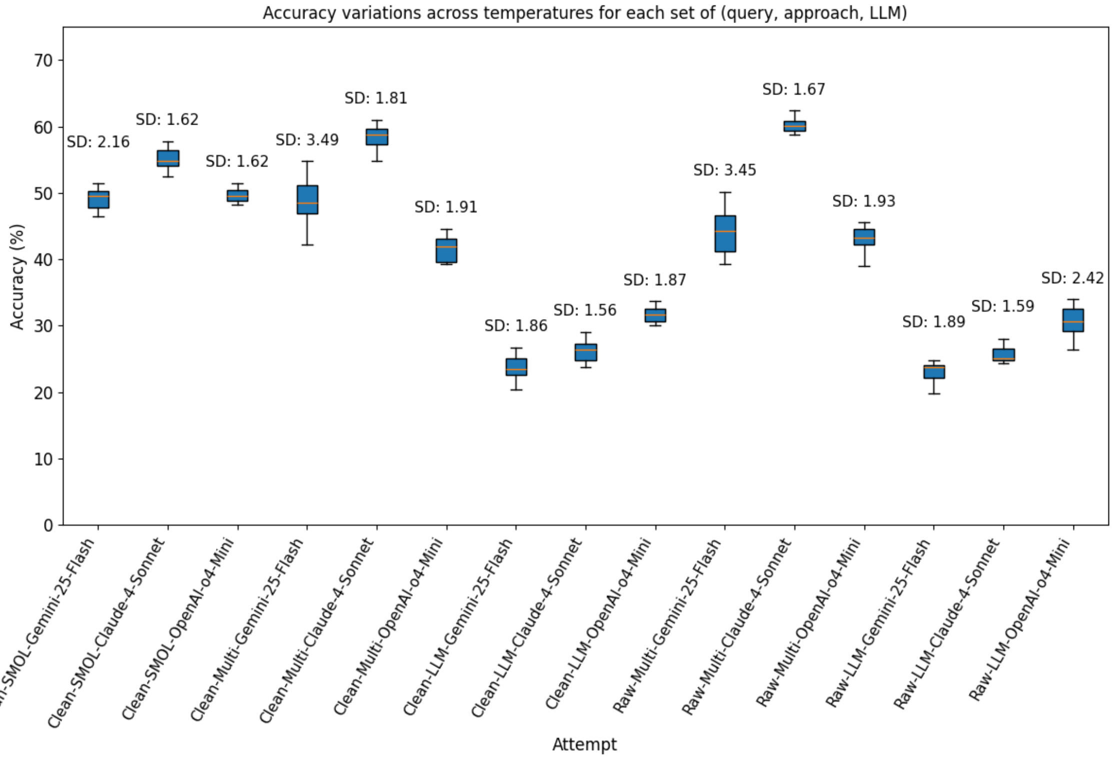

Figure 4: Variation in Accuracies through each set of (LLM, query type and approach)

LLMs (ReACt), multi-step with context engineering using LLMs (ReAct), and SmolAgent) and through 2 query types (raw and clean). These 15 unique setups [1] were tested over 11 temperature values ranging from 0.0 to 1.0 in 0.1 increments. This resulted in a total of 165 attempts over each of the samples whose results are described below.

The generated output and explanation/rationale by the model/agent were evaluated using VLM-as-Judge using Gemini-2.5-Flash. Three of the 165 attempts (one from each LLM) were checked manually to see whether the VLM-as-a-judge is

[1] SmolAgent was initially tested with 1 randomly chosen temperature value over all 3 models; no change was observed in results between the Raw and Clean queries. Hence, due to a limited compute budget, SmolAgent was only tested on clean queries for all temperature values.

LLM used	Approach Used	Query Type	Highest Accuracy	Highest at Temperature	Lowest Accuracy	Highest at Temperature	Overall Accuracy	Standard Deviation
Claude-4-Sonnet	SmolAgent	Clean	57.756	0.0	52.475	0.5	55.236	1.625
OpenAI-o4-Mini	SmolAgent	Clean	51.485	0.7	45.215	1.0	49.385	1.618
Gemini-2.5-Flash	SmolAgent	Clean	54.455	0.0	46.535	0.1	49.415	2.162
Claude-4-Sonnet	Multi-Code	Clean	61.056	0.9	54.785	0.1	58.356	1.812
Claude-4-Sonnet	Multi-Code	Raw	62.376	0.2	55.776	0.5	59.916	1.673
OpenAI-o4-Mini	Multi-Code	Clean	44.554	0.4	39.274	0.8	41.524	1.913
OpenAI-o4-Mini	Multi-Code	Raw	45.545	0.9	38.944	0.7	43.054	1.931
Gemini-2.5-Flash	Multi-Code	Clean	54.785	0.9	42.244	0.7	49.055	3.488
Gemini-2.5-Flash	Multi-Code	Raw	50.165	0.7	39.274	0.5	44.164	3.452
Claude-4-Sonnet	Single-Code	Clean	29.043	0.3	23.762	0.4	26.163	1.561
Claude-4-Sonnet	Single-Code	Raw	29.703	0.3	24.422	0.8	25.953	1.594
OpenAI-o4-Mini	Single-Code	Clean	33.663	0.1	26.733	0.4	31.443	1.873
OpenAI-o4-Mini	Single-Code	Raw	33.993	0.1	26.403	0.2	30.633	2.420
Gemini-2.5-Flash	Single-Code	Clean	26.733	0.0	20.462	1.0	23.792	1.859
Gemini-2.5-Flash	Single-Code	Raw	27.393	0.7	19.802	0.5	23.342	1.889

Table 7: Results from each of the 15 attempts : Overall Accuracy and Standard Deviation with varying temperature

indeed evaluating the responses accurately. JSON schema was used for the VLM-as-a-judge setup, where the VLM responds with a single word, either 'Yes' or 'No,' based on whether the response is accurate. To account for noise, these responses were parsed through regex as a final step.

The success rate (the percentage of attempts (165) that resulted in an accurate response) had a mean of 40.8% and a median of 41.21%. No question was answered correctly in all attempts. Only two questions were answered incorrectly in all attempts, both of which are reasonably hard questions. These samples and why the models/agents always resulted in an incorrect response were elaborated in detail in Appendix D. The maximum success rate was 93.33%, and only 23 samples had a success rate > 80%. More can be seen in Figure 3.

7.1 Using LLMs Directly with Context Engineering

In this approach, the execution was done in a single-step ReAct (Yao et al., 2023) loop with sandboxed code execution, where the LLM generates code that takes the query and added context as input; the code is then executed locally, which becomes the final output. When generating the code, the LLM is also instructed to add its rationale for the generated code. The rationale and code were returned as a JSON, which are then individually parsed through predefined functions.

7.2 Using LLMs in multiple steps with Context Engineering

This approach is similar to the previous approach, but with one minor change: the code and explanations generated were divided into 2-3 steps. Unlike the traditional ReAct workflow, in this approach the LLM generates 2-3 code snippets (one for each step) to solve the query in one go, which are then executed one by one. The rationale is generated separately for each snippet, which are then appended to one another in the end.

7.3 Using SmolAgent

In this approach, we use SmolAgent's CoderAgent out of the box with minor changes: adding the data files to the execution environment's runtime and adding the path to the files in the query. No additional context is provided to the agent. The agent then uses some of its multiple steps allowed to iteratively gain context on what the data files contain and answers the query in subsequent attempts. A computational and time limit was assigned at 8 steps and 90 seconds, respectively, for each sample. None of the samples' 33 attempts (3 LLMs * 11 temperature values) resulted in a timeout error.

8 Results

Effect of temperature on results : Results from each of the 15 attempts can be seen in Table 7, which combines all samples across each temperature value. No significant pattern was seen among the change in accuracies versus each temperature value for all sets of prompt, model, and approach, as seen in Figure 46. Claude-4-Sonnet, through multiple code snippets in a single step, clearly outperformed the rest, including SmolAgent with several steps, irrespective of temperature. Multi-step and SmolAgent performed significantly better than single-cell direct use of LLMs when looking at the overall results. However, when looking at the same results for each task category separately, there is a large variation in the same plot. This can be seen in Appendix F.

Variations in results with Model and approach : Gemini-2.5-Flash demonstrated greater sensitivity to temperature compared to the other two LLMs tested, especially with the multi-code-cell approach and through SmolAgent. For single code cell implementation, o4-mini clearly outperformed the other two LLMs, but the performance of o4-mini was closer to SmolAgent and lower than Gemini-2.5-Flash in multi-code-cell implementation, as seen in Figure 4.

Variations in results with Model and approach : Certain domains' samples had higher accuracy than the rest irrespective of model, approach, or query type despite the code complexity and task difficulty being the same. The temperature values that produced the better results in each of the domains also varied by a considerable extent.

9 Error Analysis

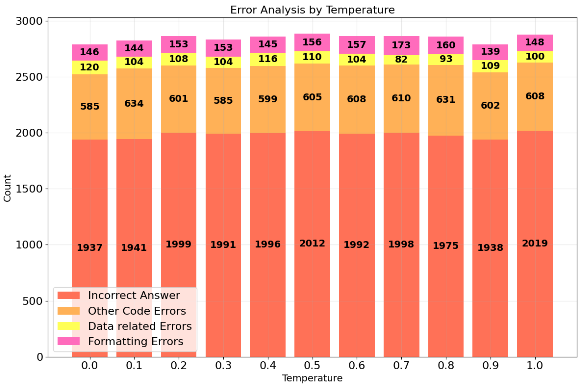

Figure 5: Error cause distribution overall : temperature wise

Among the unsuccessful cases, roughly 70% were due to incorrect final responses, another 20% were due to other errors in code, 6% were due to data error from not being able to access the data, and the rest were due to formatting errors by the LLM in returning the code. This can be seen in Figure 5 for all attempts combined for each temperature value used. The formatting errors did not occur through SmolAgent due to the use of ReAct and are from the rest of the approaches. Formatting errors disproportionately occurred while using o4-mini as the LLM as compared to the other two LLMs. In the single-code-cell approach, the count of code errors outnumbered the count of instances where an incorrect response was generated. More on this can be seen in Appendix H.

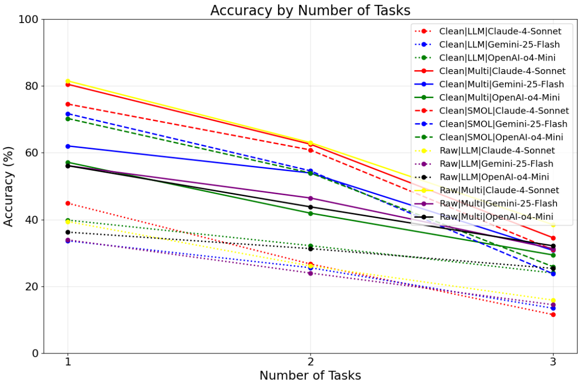

Figure 6: Average accuracy with each attempt VS number of task categories the query covers

Error rate on single and multi-task samples : The overall accuracy, including all attempts with all temperature values over samples that cover 1, 1,2 and 3 task categories, is 55.86%, 43.13%, and 25.46%, respectively; i.e., the accuracy drops significantly if the query requires code that spans more than one task type, as in section 3. The comparison of accuracies for each attempt over samples that span varying numbers of task types can be seen below in Figure 6. Reasoning models like OpenAI-o4-mini have observed smaller drops in performance with increasing complexity of queries, i.e., queries that span several task types, while a non-reasoning model might be an efficient choice for simpler queries. Additionally, we have also seen that certain tasks benefit from expensive approaches (SmolAgent or multi-cell code), while many other tasks do not, as seen in Appendix F. This hints that an efficient query-based model routing could be built that can reduce costs incurred while retaining similar performance by solving simpler queries with a less expensive approach and LLM.

10 Conclusion

Through this paper, we introduce DSBC, a data science agent benchmark with context engineering and additional parallel prompts. The benchmark closely resembles the type and way of usage of the organization' commercial data science agent. The benchmark is being released through the Apache 2.0 license to facilitate further research in the domain of data science agents.

Limitations

The benchmark does not include multilingual or multi-modal questions and is limited to text based queries. Multilingual queries can work through agents with additional steps for translation and back translation, however they haven't been tested.

References

Anthropic. 2025. Introducing the next generation of claude.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Bhanu Pratap Biswas. 2024. Weather data.

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Wenjing Hu, Yuchen Mao, et al. 2024. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? *Advances in Neural Information Processing Systems*, 37:107703–107744.

Saikat Chakraborty, Yujian Li, Matt Irvine, Ripon Saha, and Baishakhi Ray. 2018. Entropy guided spectrum based bug localization using statistical language model. *arXiv preprint arXiv:1802.06947*.

Saikat Chakraborty and Baishakhi Ray. 2021. On multimodal learning of editing source code. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 443–455. IEEE.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Miri Choi. 2021. Insurance.

Jackson Crow. 2020. Stock market dataset.

Anxo DS. 2024. World population and forecast dataset.

Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. 2025. Dabstep: Data agent benchmark for multi-step reasoning. *arXiv preprint arXiv:2506.23719*.

Sarah Fakhoury, Saikat Chakraborty, Madan Musuvathi, and Shuvendu K Lahiri. 2023. Towards generating functionally correct code edits from natural language issue descriptions. *arXiv preprint arXiv:2304.03816*.

Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1469–1481. IEEE.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2025. Measuring coding challenge competence with apps. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Aradhana Hirapara. 2023. Farm produce data 80 years.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024a. Mlagentbench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning*, pages 20271–20309. PMLR.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, et al. 2024b. Da-code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13487–13521.

Abhishek Jha. 2024. Time series air quality data of india (2010-2023).

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues?

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2025. Dsbench: How far are data science agents from becoming data science experts? In *The Thirteenth International Conference on Learning Representations*.

Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring llm-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2312–2323. IEEE.

Yunho Kim, Seokhyeon Mun, Shin Yoo, and Moonzoo Kim. 2019. Precise learn-to-rank fault localization using dynamic and static features of target programs. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(4):1–34.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.

Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pretraining. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1035–1047.

Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. 2024. Are LLMs capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 9215–9235, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A survey of context engineering for large language models.

Mikhail. 2024. Walmart sales.

Meir Nizri. 2022. Covid-19 dataset.

OpenAI. 2025. o4-mini system card.

Ulrik Thyge Pedersen. 2023. Life expectancy.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 2025. 'smolagents': a smol library to build great agentic systems.

Saurabh Shahane. 2023. Electricity load forecasting.

Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An analysis of the automatic bug fixing performance of chatgpt. In *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*, pages 23–30. IEEE.

Ansh Tanwar. 2023. Monthly food price estimates.

Gemini Team. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities.

Rosalia Tufano. 2023. Automating code review. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 192–196. IEEE.

Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 50(4):911–936.

Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 959–971.

Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the modern code review repositories: A dataset of people, process and product. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 460–463.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, et al. 2023. Natural language to code generation in interactive data science notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 126–173.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018a. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. 2025. Datascibench: An llm agent benchmark for data science.

Jiyang Zhang, Sheena Panthaplackel, Pengyu Nie, Junyi Jessy Li, and Milos Gligoric. 2022. Coditt5: Pretraining for source code and natural language editing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12.

Yuge Zhang, Qiyang Jiang, XingyuHan XingyuHan, Nan Chen, Yuqing Yang, and Kan Ren. 2024. Benchmarking data science agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5677–5700.

A Prompts Used

Prompt 1 : SmolAgent

```
INSTRUCTIONS : the final answer must contain the answer mentioned explicitly whether
if it is a text, list of answers, something numeric or text in the desired format.

QUERY :
{question}.

DATASET PATH :
{dataset_path}
```

Prompt 2 : Multi-Code-Cell

```
You are a data analyst. The user asked: "{user_query}"
Dataset information:
- First 5 rows: {sample_rows}
- Dataset description: {dataset_description}

1. Generate COMPLETE, STANDALONE code snippets (2-3 steps) that
could each run independently. For each step, provide:
1. A text explanation of what this step does
2. The Python code snippet that implements this step

Format your response as JSON with this structure:
{{
  ”steps”: [
    {{
      ”explanation”: ”Detailed explanation of what this step accomplishes”,
      ”code”: ”Complete Python code with imports, analysis, and output”
    }}
  ]
}}

REQUIREMENTS:
- Use pandas to load and analyze the data: pd.read_csv('{filepath}')
- MUST include at least one visualization using matplotlib/seaborn
- Include necessary imports in each relevant snippet
- Complete analysis logic
- Clear visualization code when applicable
- Proper print() statements to show results
- Don't use markdown formatting, just pure JSON
```

Prompt 3 : Explanation prompt

```
A user submitted the following question about their dataset: "{user_question}"

Dataset Context:
{dataset_info}
Generated Analysis Code:
{code}
Execution Results:
{answer}

Please provide a clear, comprehensive explanation that:
1. Directly addresses the user's original question
2. Interprets the results in the context of the dataset
3. Explains what the findings mean in practical terms
4. Highlights any key insights or patterns discovered
5. Always use first-person when speaking.

Your explanation should be written in plain language that a business
stakeholder could easily understand.
```

Prompt 4 : Single-Code-Cell

```
You are a data analyst. Generate Python code to analyze a CSV file and answer
the following query: {user_query}

      Dataset information:
      {dataset_description}
      # # # # # # # # # # # # # # # # # # # # #
      Requirements:
      - Use pandas to load and analyze the data: pd.read_csv('data.csv')
      - Include necessary imports
      - Use print() statements to show results
      - For visualizations, use matplotlib/seaborn
      - Handle data type conversions if needed
      - Return ONLY executable Python code, no markdown formatting
    - The dataframe is already loaded in a variable called 'data'. Do not re-read it
      - Have the answer ready in a variable called 'answer'.
      Just declare and add your values there.
      - Do not have subplots, only main plots
      Generate clean, executable Python code
    For the Explination, it should describe why a step was taken and not whats done.
      Use this format for the response :

      {{
          "explanation": "...",
          "code": "..."
      }}
```

Prompt 5 : VLM-as-a-Judge

```
Respond in this exact JSON format:
{
  {
   ”Evaluation”: 'Yes' 'No' 'The Evaluation should be Yes only when the response is
    technically correct. Sometimes the answer might be of a different format but
   still correct (Ex : March , 3 when asked the month etc..). For numerical values,
     the rounded .2f values should match to be considered correct.'
  }
}

You are being used for LLM-as-judge. In numeric solutions an error beyond the 2nd
decimal point after rounding can be ignored.

####
The Query by the user is :
{Q}
----
The Ground Truth for the query is :
{A}
----
The Code Snippet to obtain the ground truth was :
{C}
----
The Response by the model is :
{R}
----
The Code Snippet in the submission was :
{S}
----
The Reasoning given with the submission was :
{N}
```

B Reproducibility

The hyperparameters not specified or tested with multiple values through the paper are all use through their default values. A max token limit of 8192 was used. All experiments were done on Google Cloud. All inference runs combined have cost approx. 1200$, 200$ and 700$ respectively for Claude-4-Sonnet, Gemini-2.5-Flash and OpenAI-o4-Mini respectively. For VLM-as-a-judge experiments, it cost us around 150$ using Gemini-2.5-Flash. For SmolAgent, we used a time limit of 90 seconds and step limit of 5 for each query during inference. Additional allowed imports were constrained to a few packages that should be enough to solve the queries of the benchmarks : pandas, numpy, scipy, scikitlearn, matplotlib, seaborn, re, math, datetime. For the same amount of inference samples, compared to single-code-cell, multi-code-cell was 1.8x expensive and SmolAgent was 3.2x expensive when averaging costs across all LLMs tested. All LLMs were used with a seed value of 1024.

C Annotation Guidelines

Annotator Guidelines : Creating Samples

When creating training samples, focus on generating simple, realistic queries that reflect actual user interactions you've encountered in client work or observed in system usage logs. mirror the natural language patterns, terminology, and problem types that real users typically present. Draw from common scenarios you've witnessed, such as troubleshooting requests, feature questions, or workflow clarifications, ensuring that each sample captures the authentic tone and complexity level of genuine user inquiries rather than overly polished or artificial examples.

Annotator Guidelines : Making Clean queries

When creating clean queries, remove language that leaks information or makes assumptions about the data. The query "Among those who have died, how many deaths were not attributed to COVID-19?" assumes non-COVID deaths exist in the dataset. A cleaner version asks "Did any deaths not occur due to COVID-19? If so, how many?" This removes the data leakage while keeping the same core question. Make sure that the exact meaning of both the Clean query and Raw query is the same.

Annotator Guidelines : categorizing Samples

Categorize the queries based on what tasks they require to be performed to be able to answer the queries. A query can be categorizied as atleast one category and at most 3 categories. If the sample seems close to more than three categories, assign the best matching three categores, use the below examples for reference : {examples}.

D Unsolved Samples

The two unsolved samples, its solution and the common errors made by the LLMs/agents were :

Example 1 : Query ID 8

If it rains today, what is the likelihood that it will rain tomorrow as well?

```
import pandas as pd
# Convert to datetime
df_AQI['From Date'] = pd.to_datetime(df_AQI['From Date'])
df_AQI['Date'] = df_AQI['From Date'].dt.date
# Daily rainfall sums
daily_rain = df_AQI.groupby('Date')['RF (mm)'].sum().reset_index()
daily_rain = daily_rain.sort_values('Date').reset_index(drop=True)
# Binary rain indicators (>0.1mm = rain)
daily_rain['Rain_Today'] = (daily_rain['RF (mm)'] > 0)
daily_rain['Rain_Tomorrow'] = daily_rain['Rain_Today'].shift(-1)
# Remove last row (no tomorrow data)
daily_rain = daily_rain[:-1]
# Calculate probability
rain_today_count = daily_rain['Rain_Today'].sum()
rain_both_days = ((daily_rain['Rain_Today'] == 1)
                 & (daily_rain['Rain_Tomorrow'] == 1)).sum()
probability = rain_both_days / rain_today_count if rain_today_count > 0 else 0
print(f”P(Tomorrow | Today) = {probability:.3f} ({probability*100:.1f}%)”)
```

Mistakes made by the LLMs and Agents include, A) blind trust in "From Date" column to be a date-time column without verifying, B) assuming "From Date" to cover daily data rather than hourly data without verifying, C) using rainfall >0.1 mm as rain rather than directly using 0.0, incorrectly assuming all rows are of .1f type

Example 2 : Query ID 226

Which of the calendar months typically experience the highest sales in an year ? List top 2

```
df_SALES['Date'] = pd.to_datetime(df_SALES['Date'])
df_SALES['Month'] = df_SALES['Date'].dt.month
monthly_sales = df_SALES.groupby('Month')['Weekly_Sales'].mean()
sorted_monthly_sales = monthly_sales.sort_values(ascending=False)
top_2_months = sorted_monthly_sales.head(2)
month_names = { 1: 'January', 2: 'February', 3: 'March', 4: 'April',
                5: 'May', 6: 'June', 7: 'July', 8: 'August',
               9: 'September', 10: 'October', 11: 'November', 12: 'December'}
top_2_month_names = [month_names[month] for month in top_2_months.index]
print(f”{top_2_month_names[0]} and {top_2_month_names[1]}”)
```

Mistakes made by the LLMs and Agents include, A) grouping 2 month periods from start rather than considering all possible rolling 2 month periods B) assuming dataset starts and ends in same month. i.e if it starts in May for example and ends in July of a different year. There is more data for June and hence considering sum() instead of mean() leads to a incorrect response. C) assuming the question meant which 2 months despite being asked for "calendar months"

E Task wise Success Rates

The task wise success rates of each query spanning 165 attempts used can be seen in Figure 15, Figure 16, Figure 17, Figure 18, Figure 19, Figure 20, Figure 21, Figure 22 respectively for each task category separately. The overall success rates for the several task categories ranged from a lowest value of of 29.1% for Feature engineering to a highest value of 46.9% for Correlation analysis. the distributions of each of the tasks' success rates are considerably skewed half each of either direction with differences between mean and median ranging from 4-15% among the samples of the same task category.

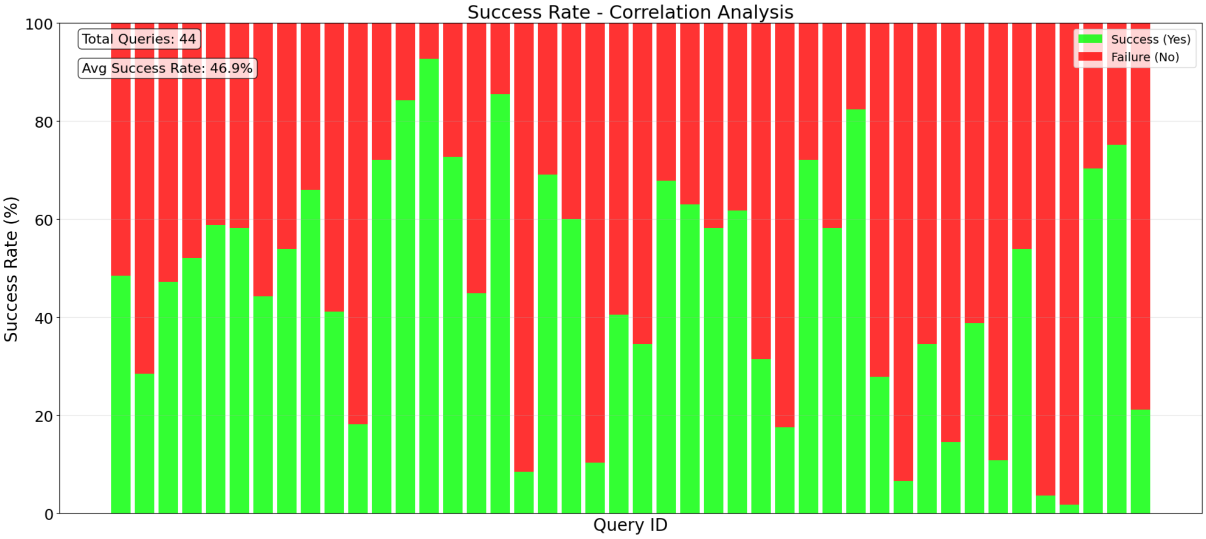

Figure 7: Success rates for each sample over several (165) attempts - Correlation Analysis

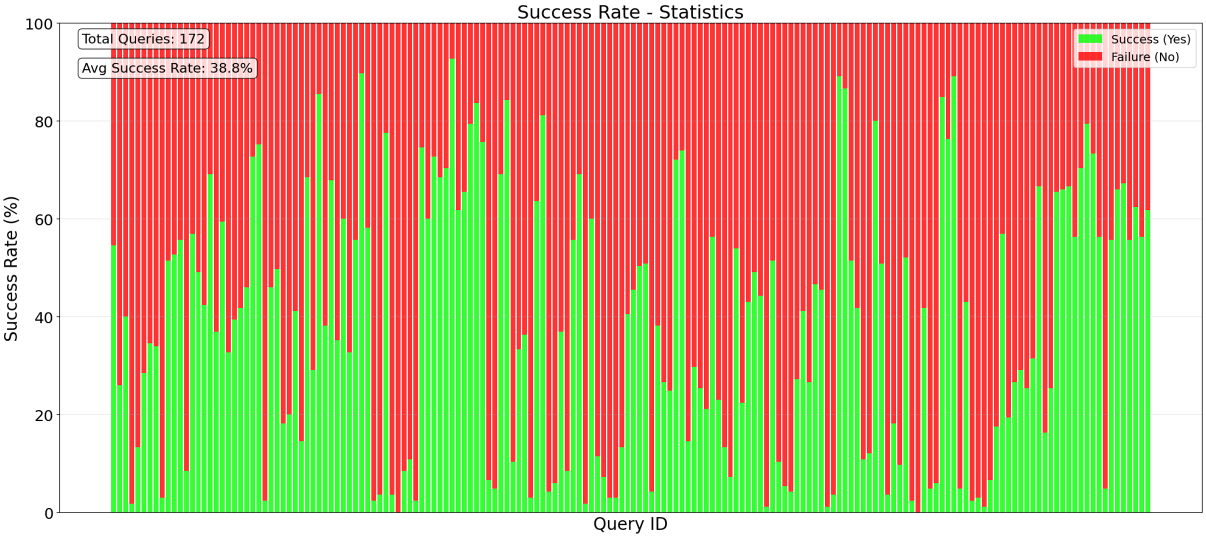

Figure 8: Success rates for each sample over several (165) attempts - Statistics

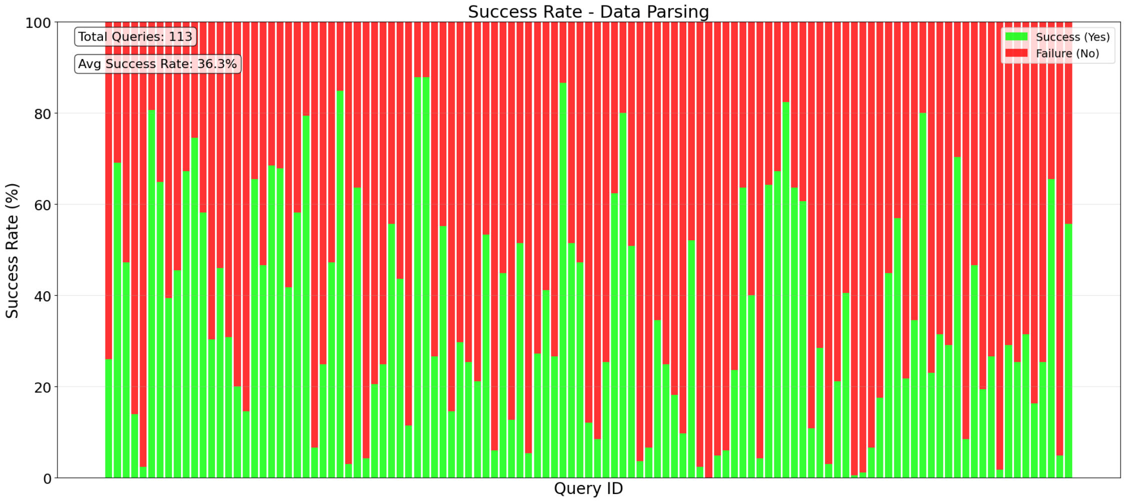

Figure 9: Success rates for each sample over several (165) attempts - Data Parsing

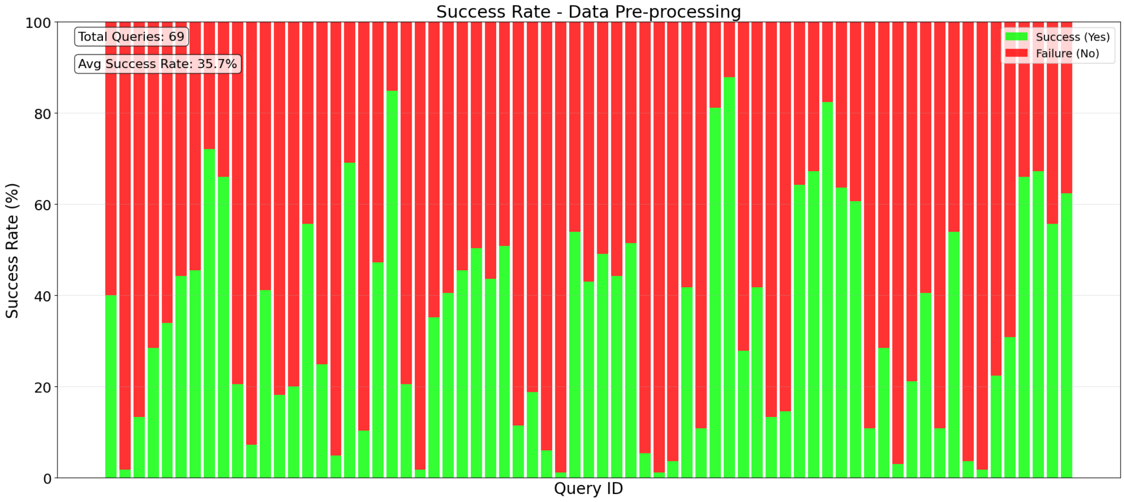

Figure 10: Success rates for each sample over several (165) attempts - Data Pre-processing

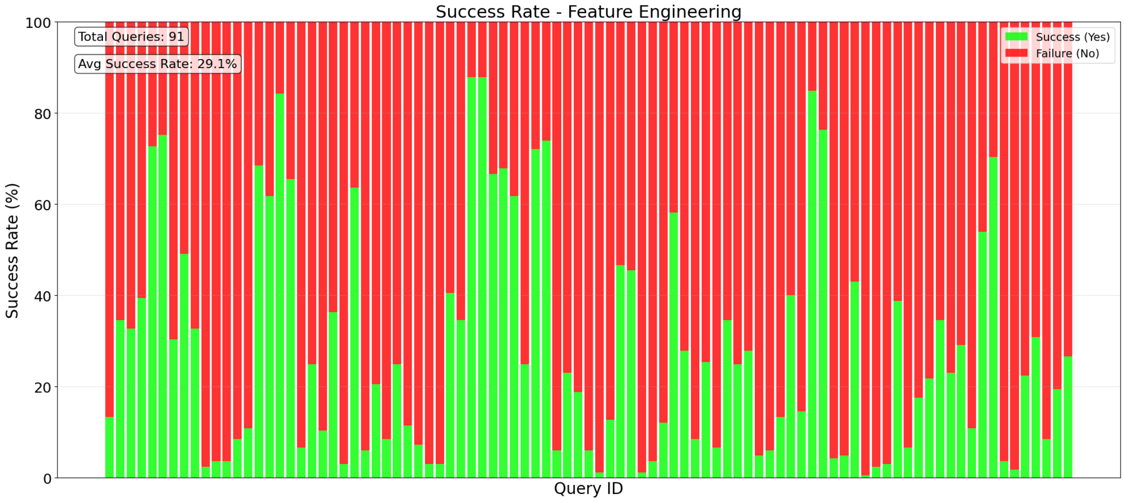

Figure 11: Success rates for each sample over several (165) attempts - Feature Engineering

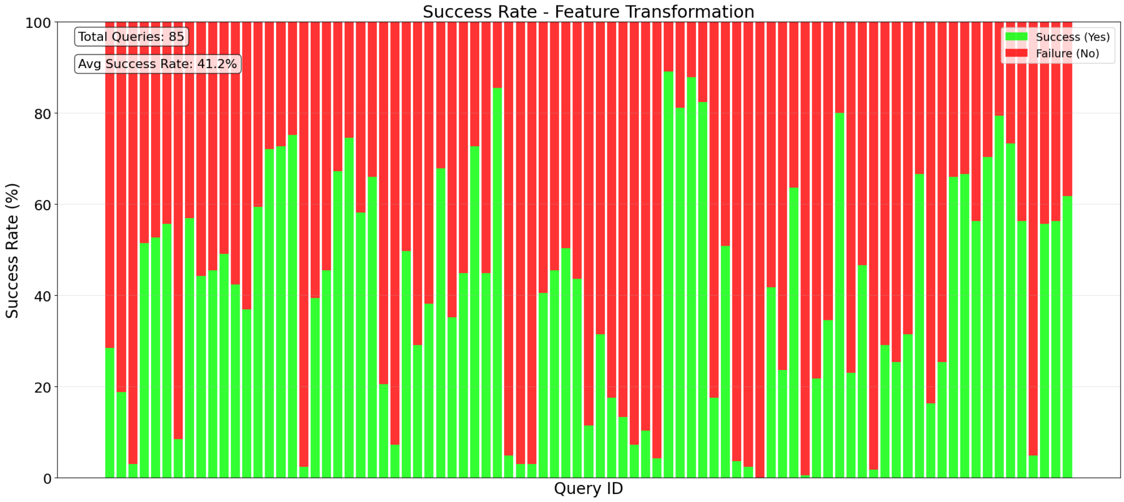

Figure 12: Success rates for each sample over several (165) attempts - Feature Transformation

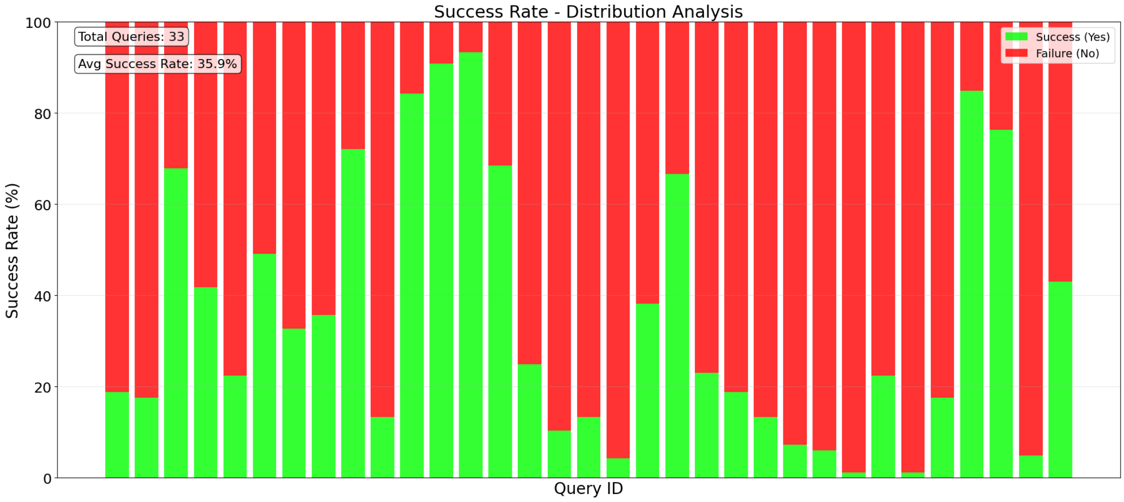

Figure 13: Success rates for each sample over several (165) attempts - Distribution Analysis

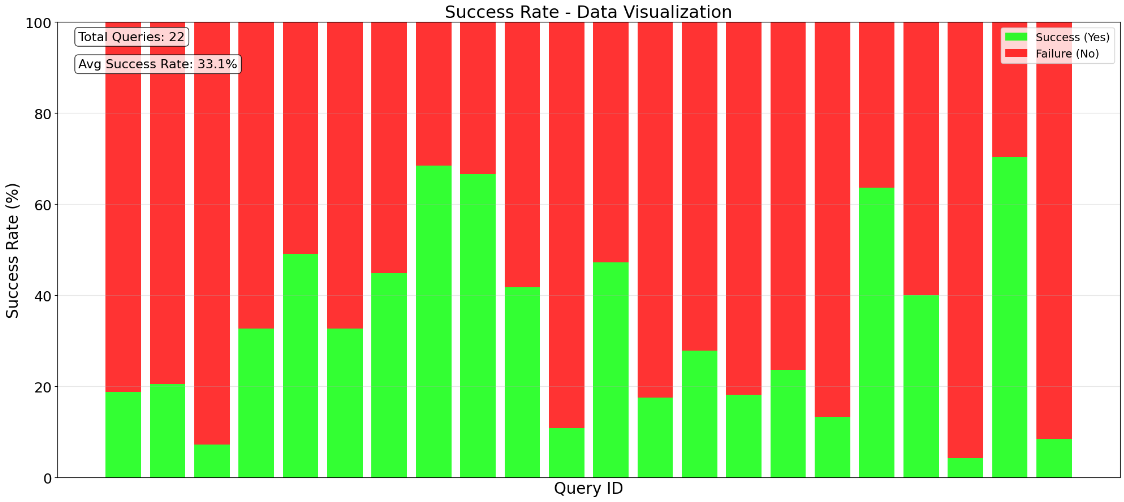

Figure 14: Success rates for each sample over several (165) attempts - Data Visualization

F Accuracies over Each Task

The task wise accuracies of each attempt compared to the temperature used can be seen in Figure 15, Figure 16, Figure 17, Figure 18, Figure 19, Figure 20, Figure 21, Figure 22 respectively for each task category separately. Some of the tasks had less variance between the accuracies obtained through each attempt while the rest had very high variance. **This hints that certain tasks might perform more or less the same with any set of (temperature, LLM, approach, query type) while others perform clearly better with a certain set of the same features.** This is especially important as costs incurred can vary a lot between using multi-code-cell approach or by SmolAgent compared to generating them directly by an LLM in a single step. Though the difference in costs is only a few cents per sample, it can make a difference when dealing with a large number of samples.

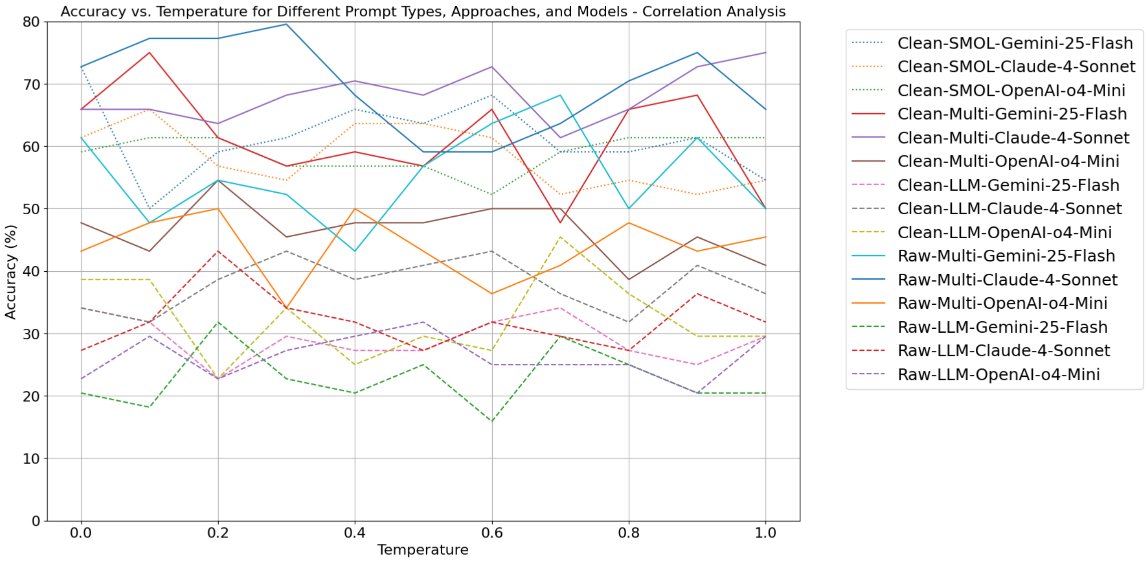

Figure 15: Results from each attempt over each temperature value used - Correlation Analysis

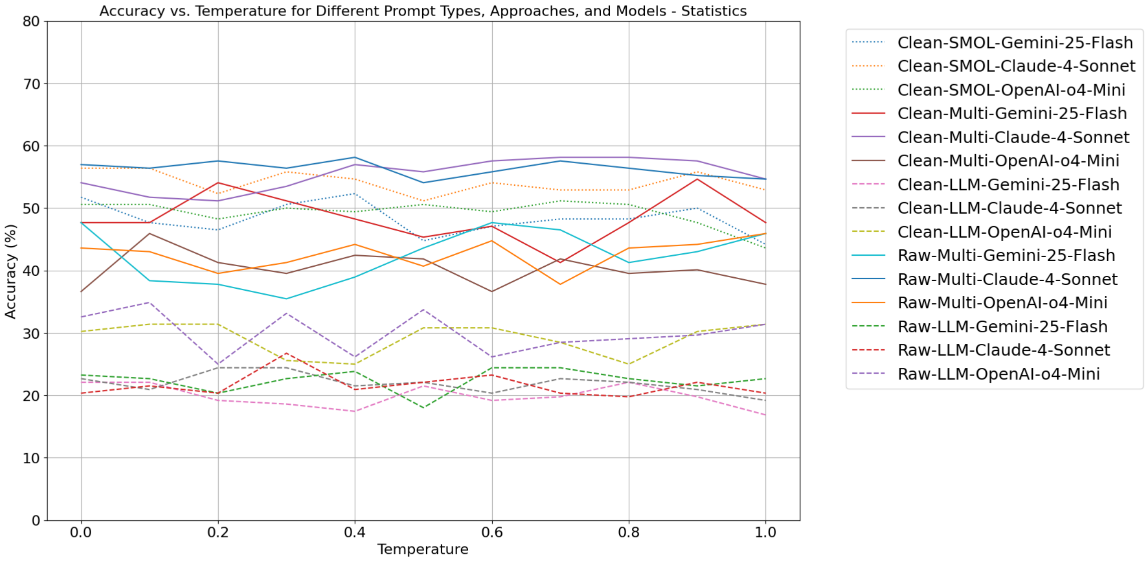

Figure 16: Results from each attempt over each temperature value used - Statistics

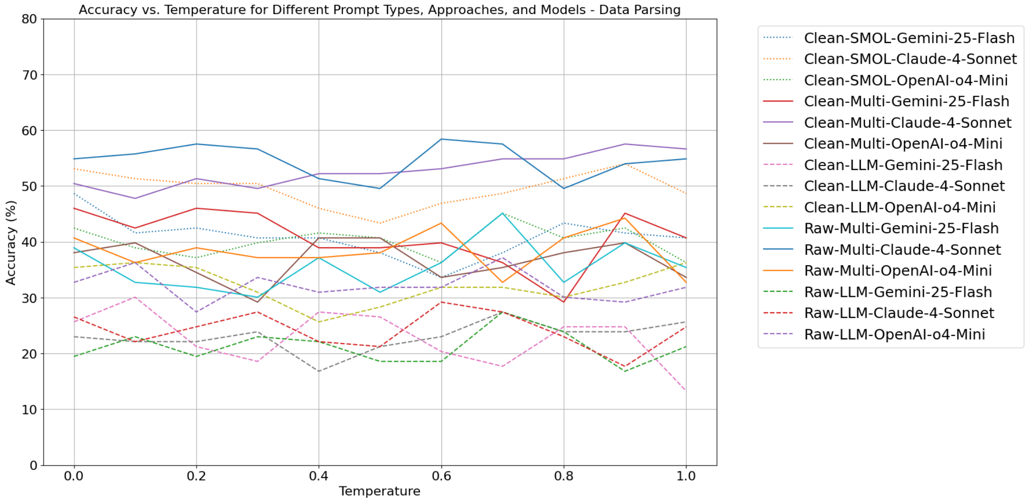

Figure 17: Results from each attempt over each temperature value used - Data Parsing

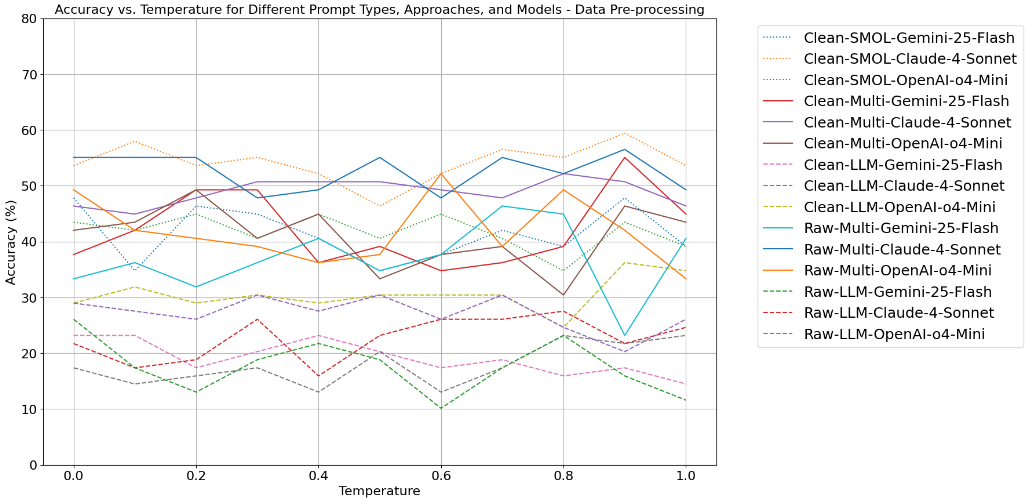

Figure 18: Results from each attempt over each temperature value used - Data Pre-processing

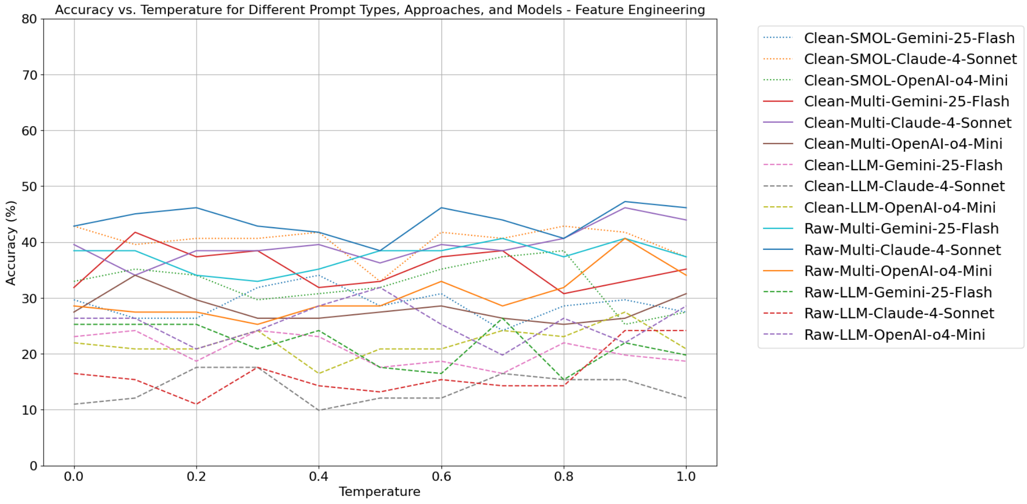

Figure 19: Results from each attempt over each temperature value used - Feature Engineering

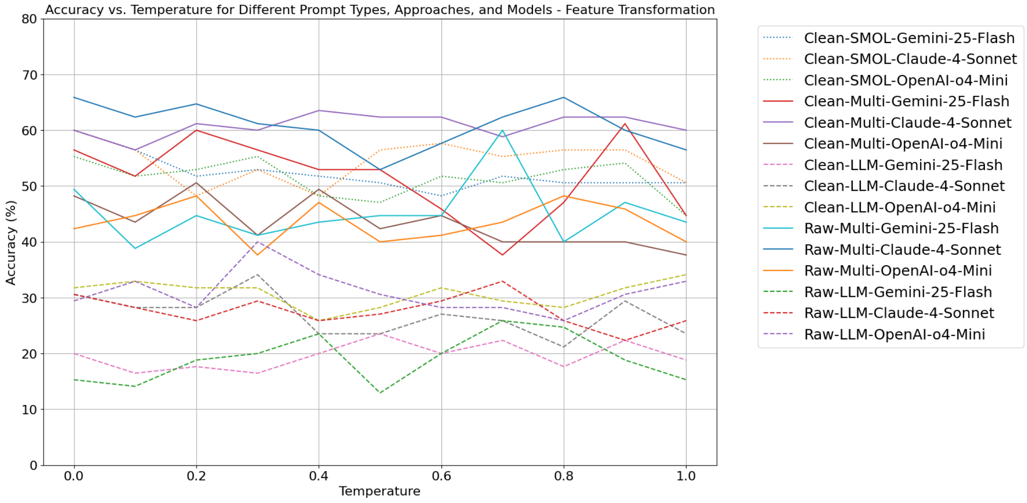

Figure 20: Results from each attempt over each temperature value used - Feature Transformation

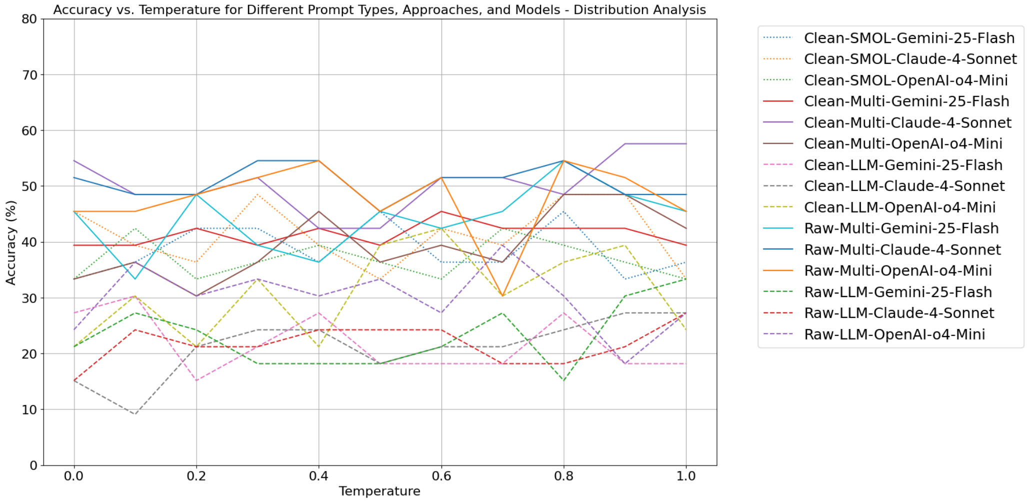

Figure 21: Results from each attempt over each temperature value used - Distribution Analysis

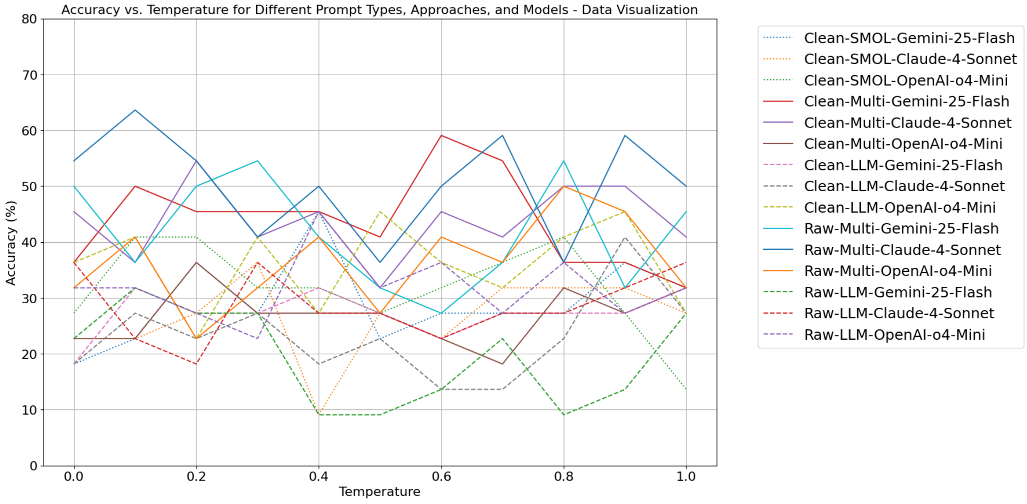

Figure 22: Results from each attempt over each temperature value used - Data Visualization

G Variation in Accuracies : task wise

The task wise variations in accuracies of each attempt can be seen in Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28, Figure 29, Figure 30 respectively for each task category separately. **Some task categories are less sensitive to temperature, while other tasks are more sensitive to changes in temperature.** The accuracy ranges for each task types can be seen to vary between each task category i.e some LLMs and approaches perform better in some task categories while some other LLM and approach might be better at other task categories. Hence, **Model and approach routing could be implemented by identifying the task category based on the input query**.

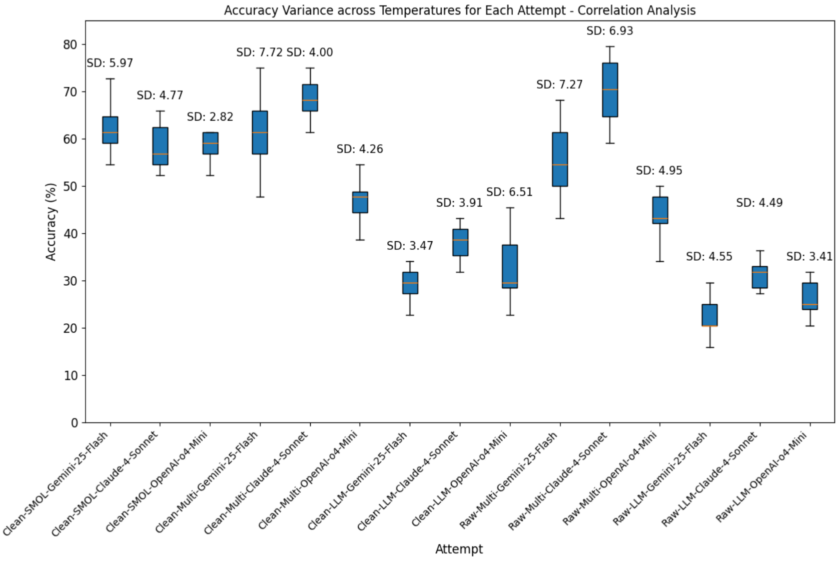

Figure 23: Variation in Accuracies through - Correlation Analysis

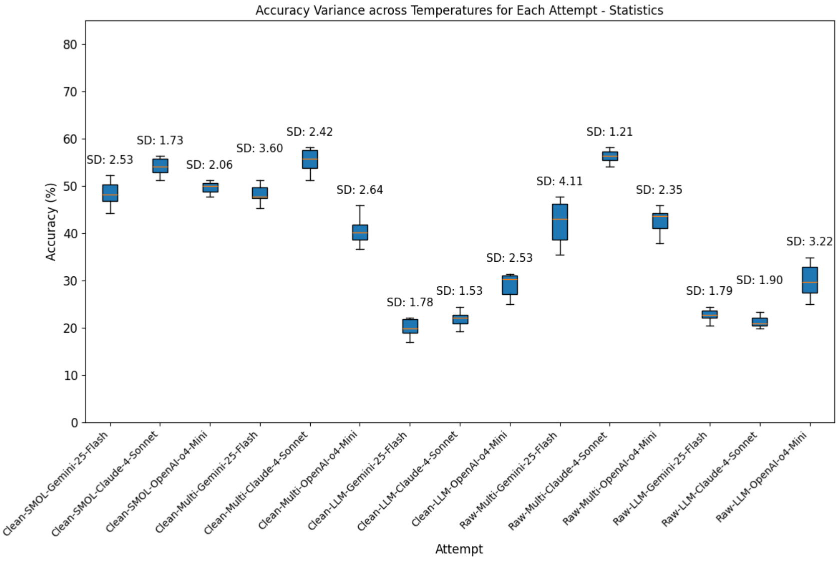

Figure 24: Variation in Accuracies through - Statistics

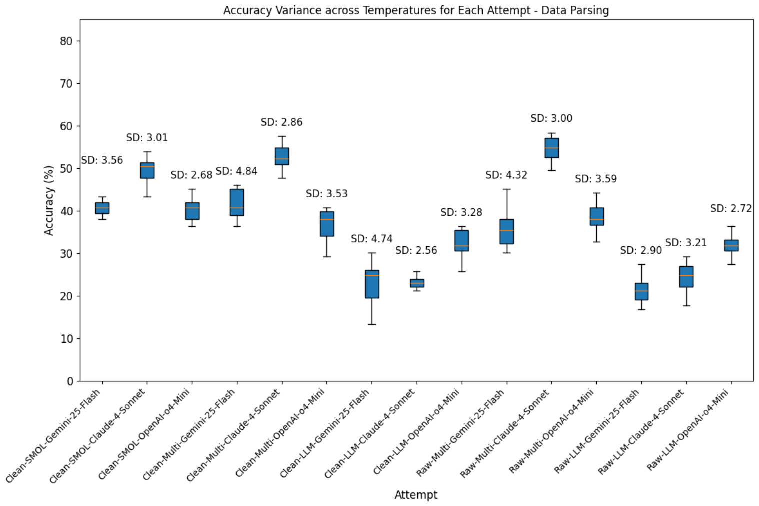

Figure 25: Variation in Accuracies through - Data Parsing

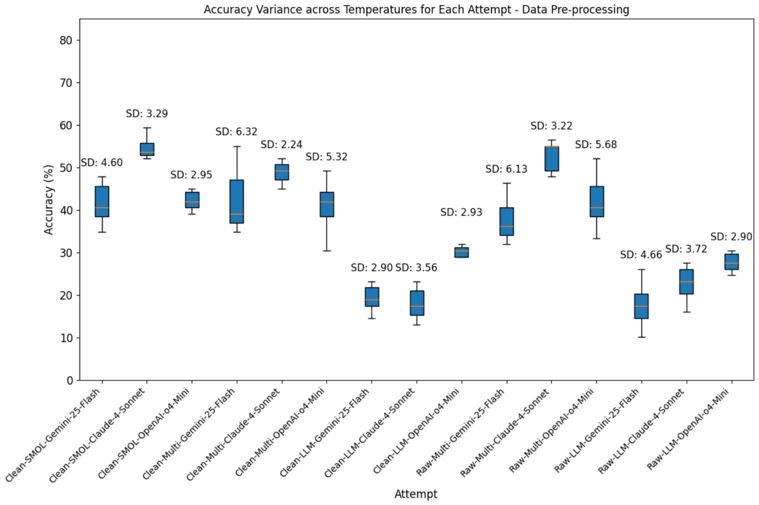

Figure 26: Variation in Accuracies through - Data Pre-processing

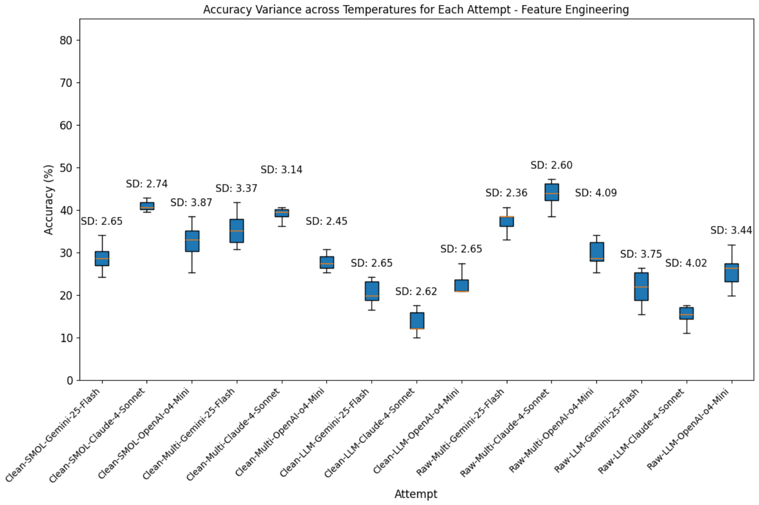

Figure 27: Variation in Accuracies through - Feature Engineering

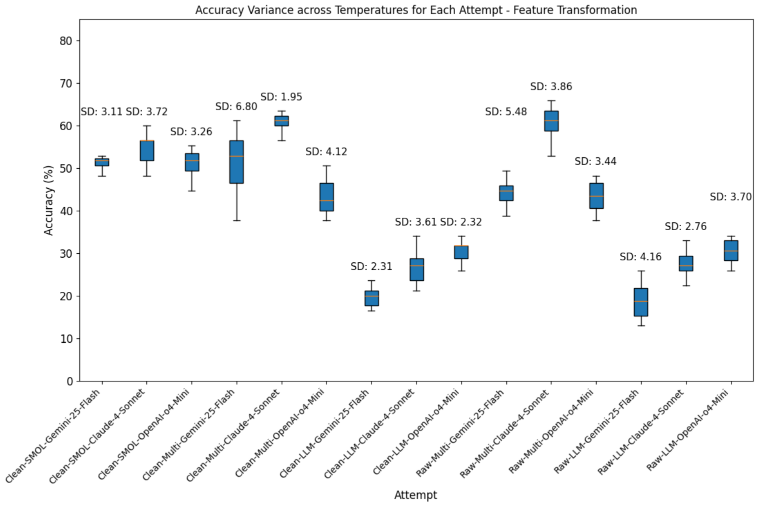

Figure 28: Variation in Accuracies through - Feature Transformation

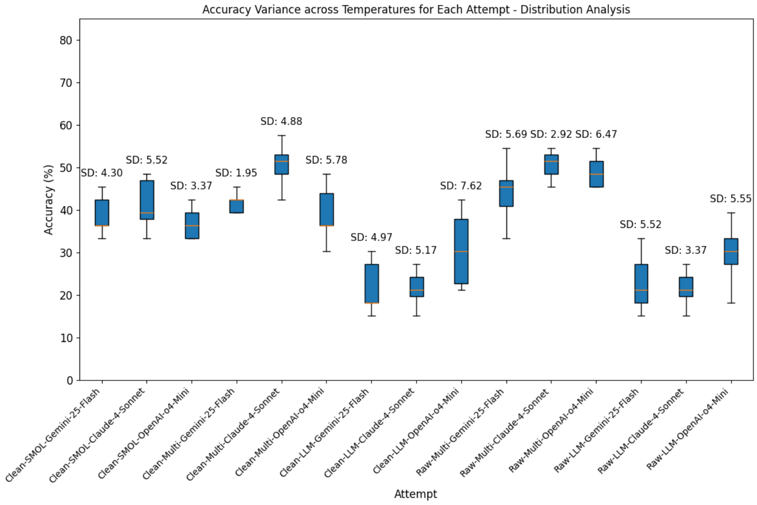

Figure 29: Variation in Accuracies through - Distribution Analysis

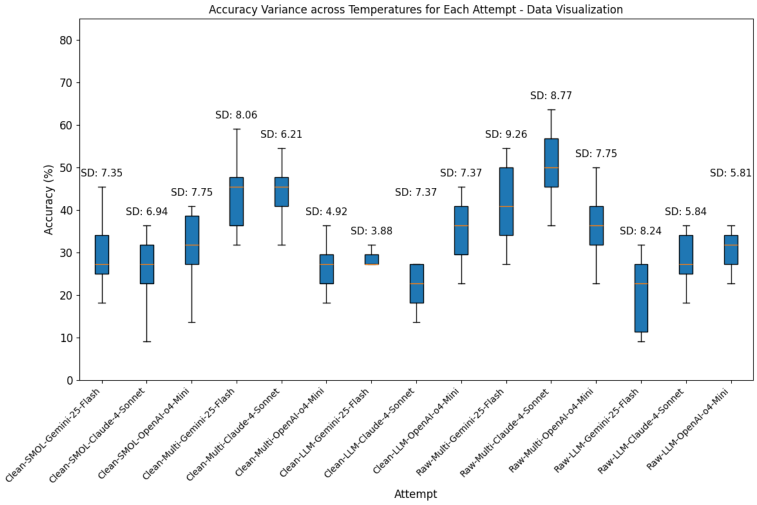

Figure 30: Variation in Accuracies through - Data Visualization

H Error Analysis : Approach wise

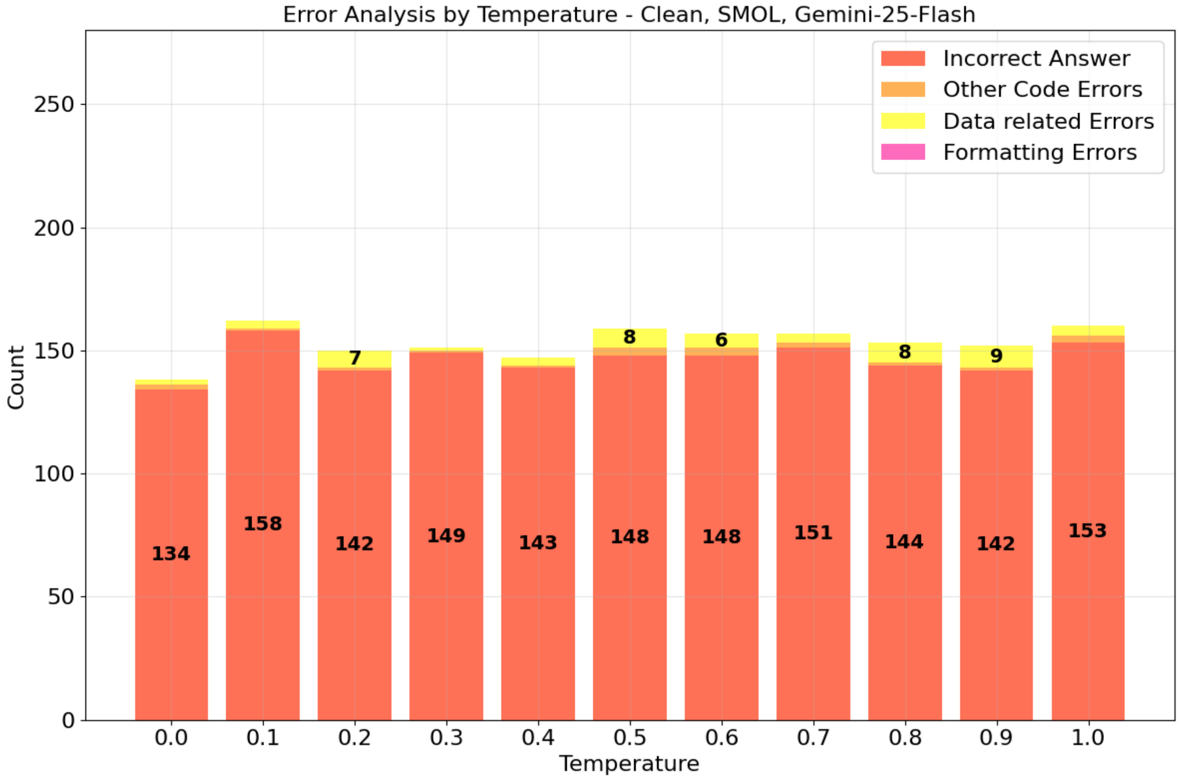

Figure 31: Incorrect response cause distribution -

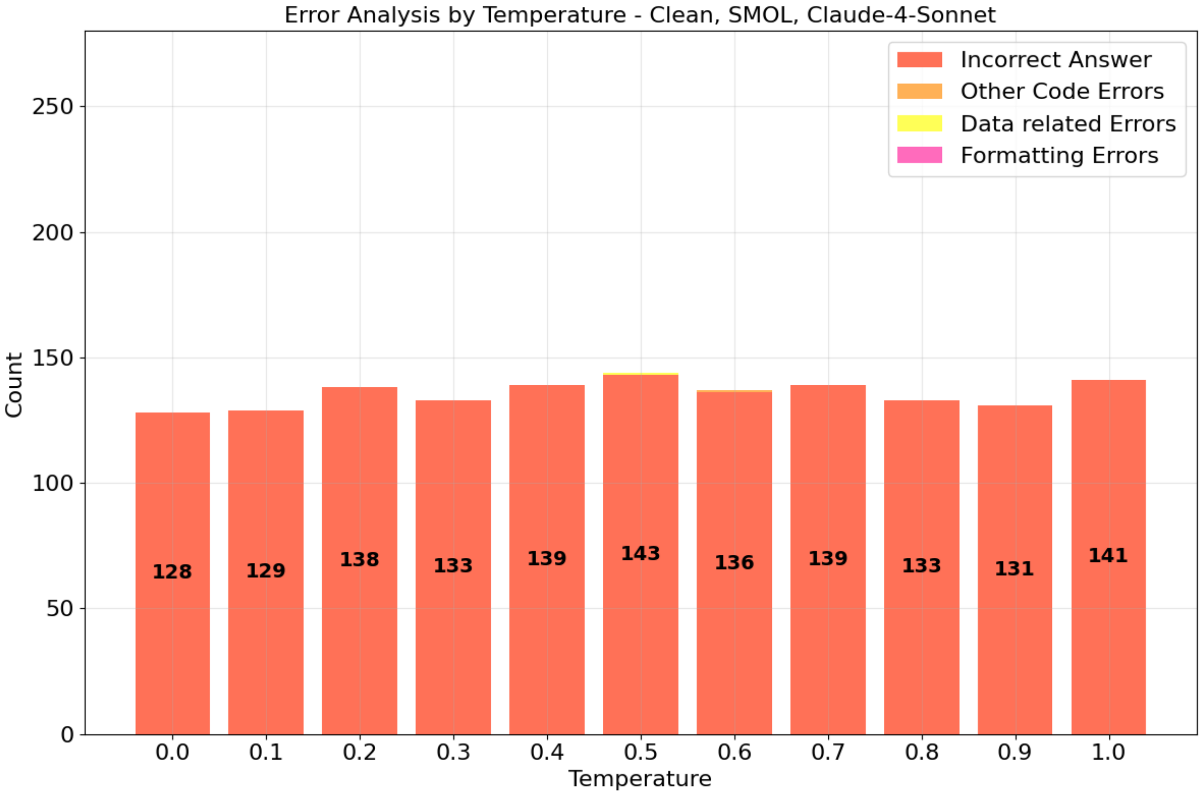

Figure 32: Incorrect response cause distribution -

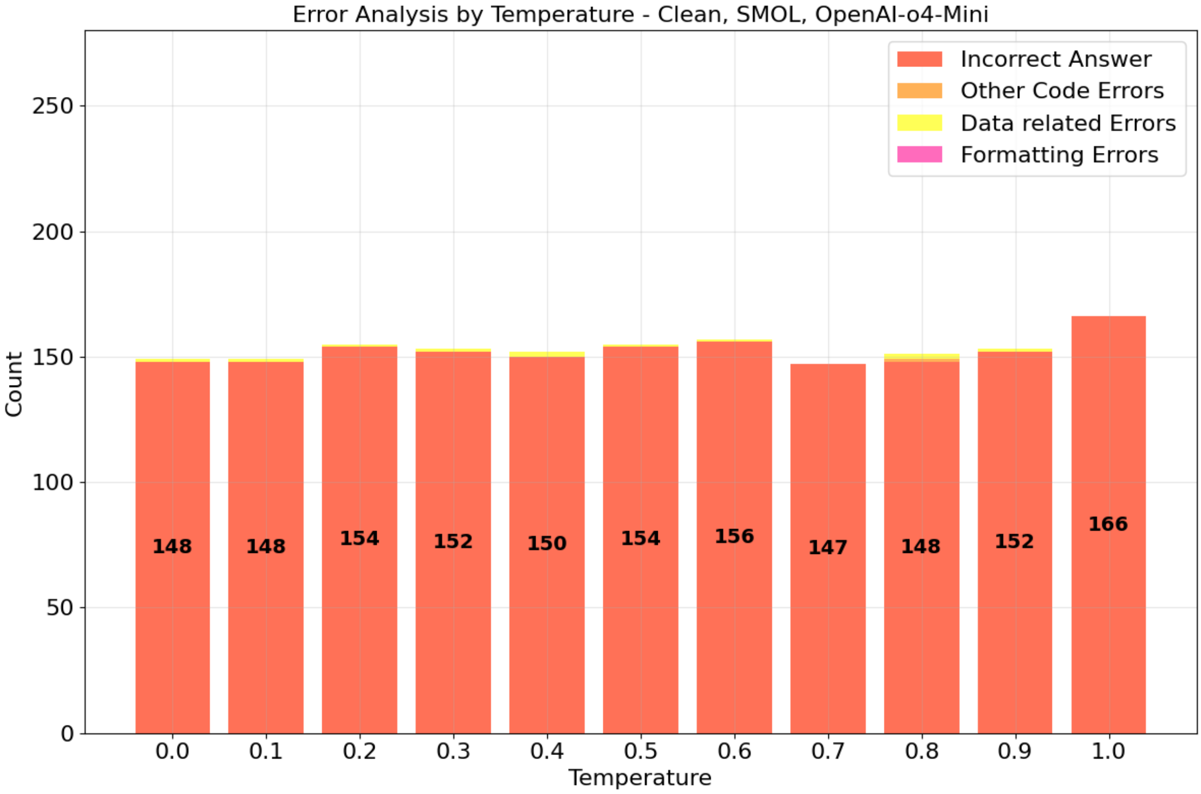

Figure 33: Incorrect response cause distribution -

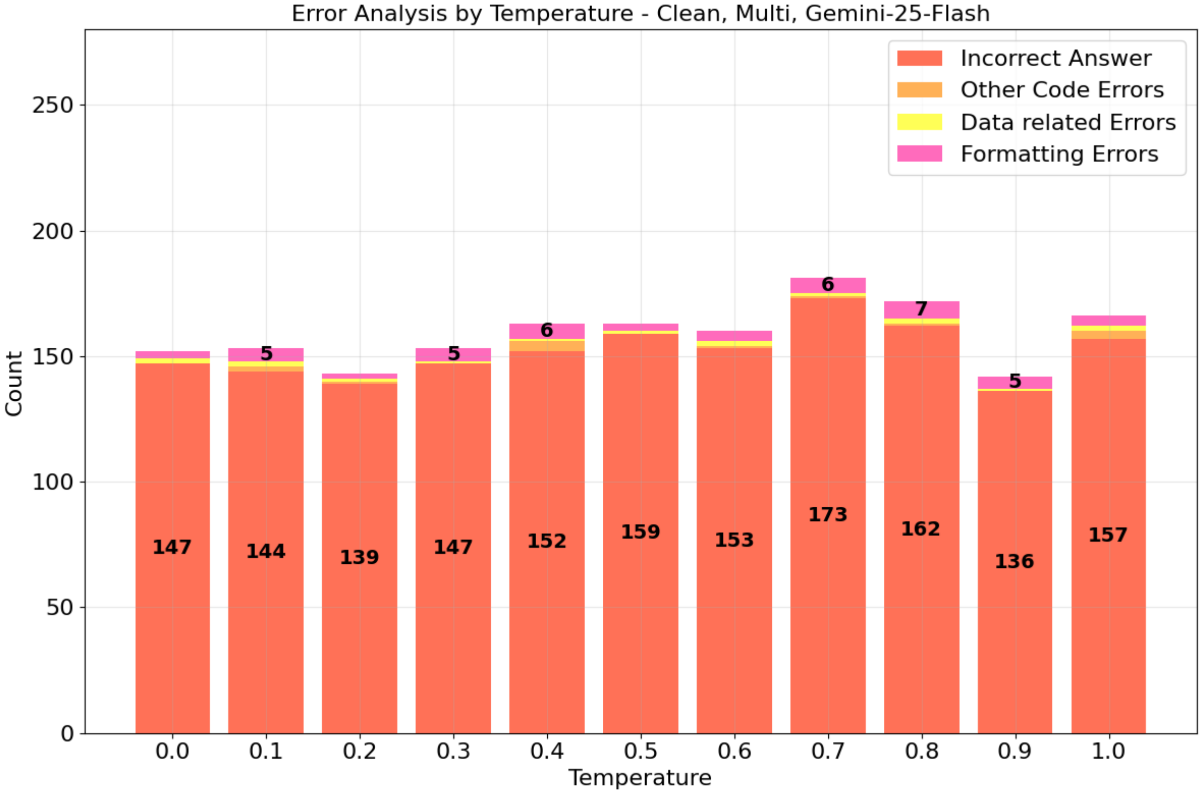

Figure 34: Incorrect response cause distribution -

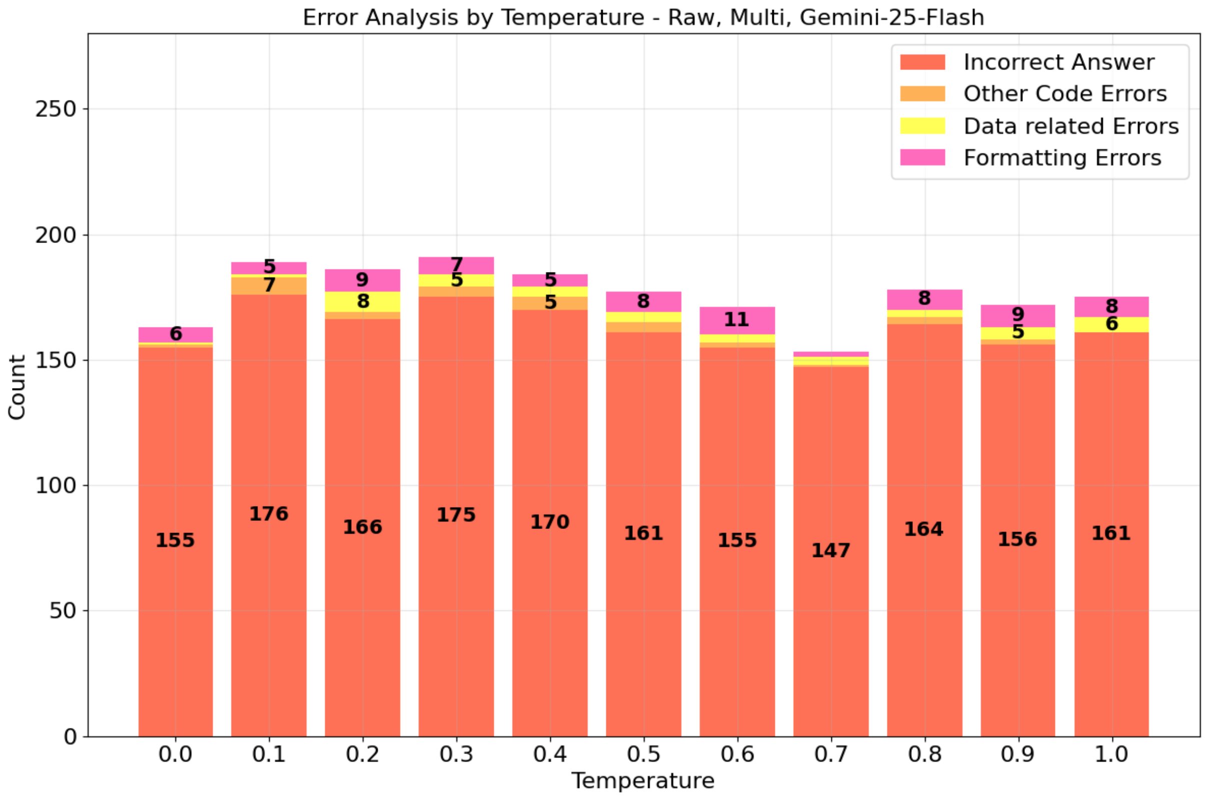

Figure 35: Incorrect response cause distribution -

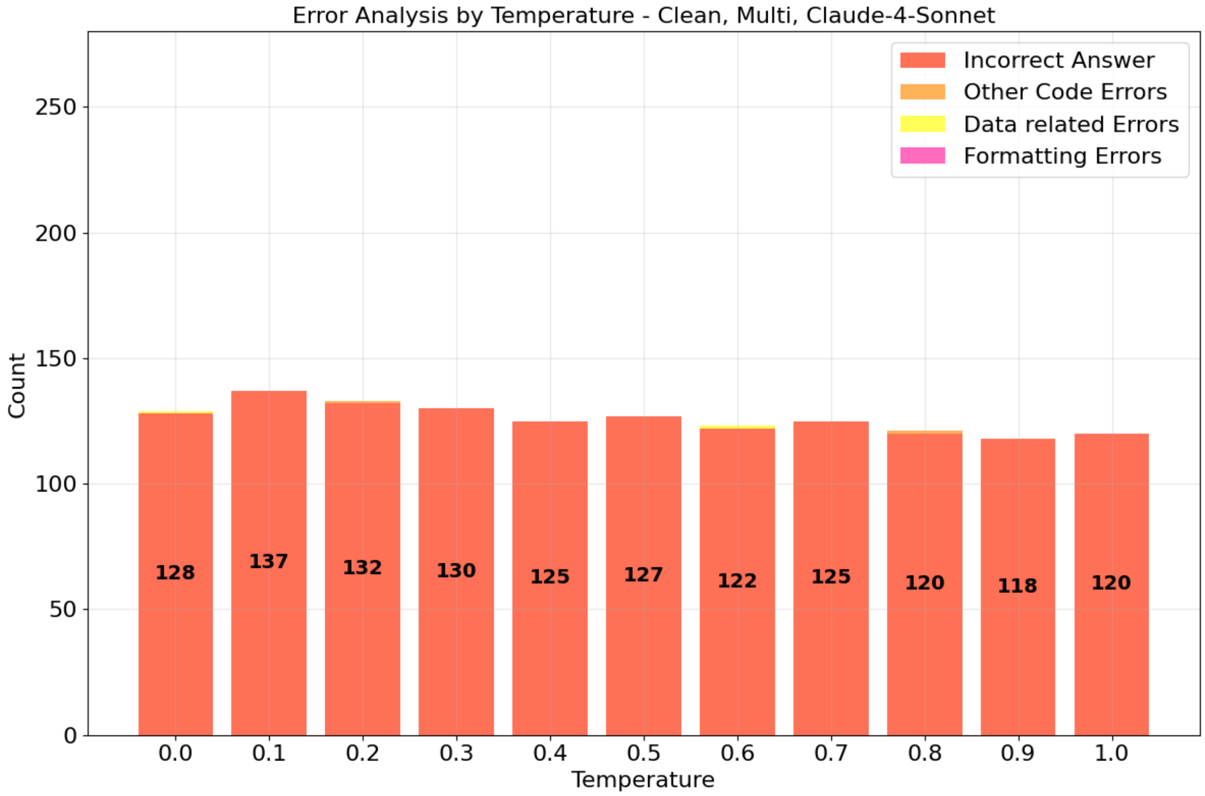

Figure 36: Incorrect response cause distribution -

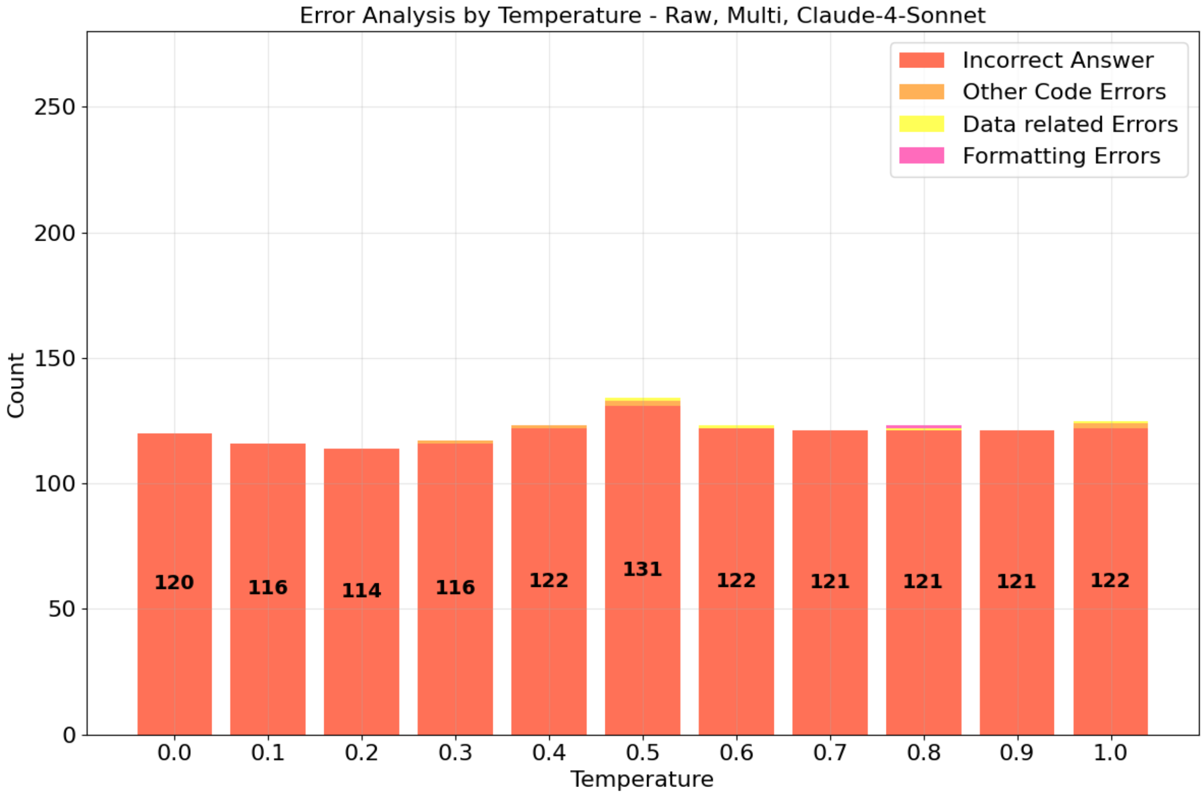

Figure 37: Incorrect response cause distribution -

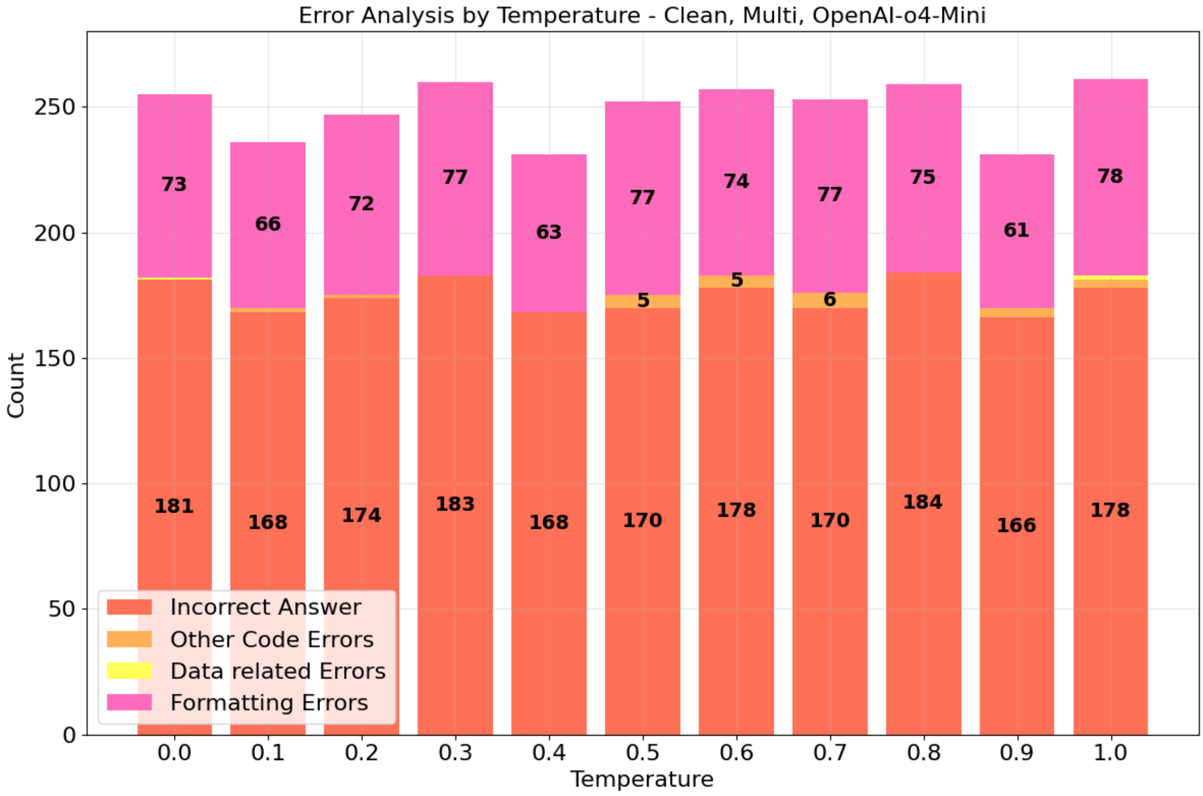

Figure 38: Incorrect response cause distribution -

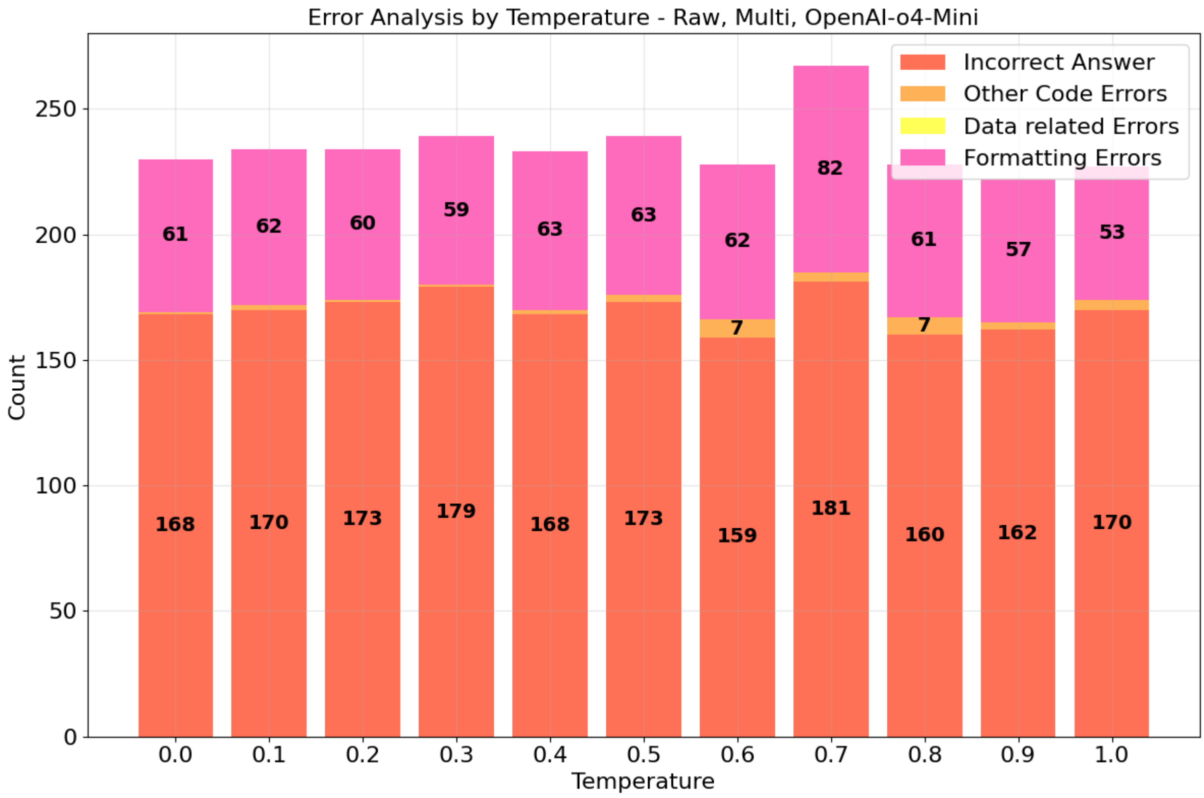

Figure 39: Incorrect response cause distribution -

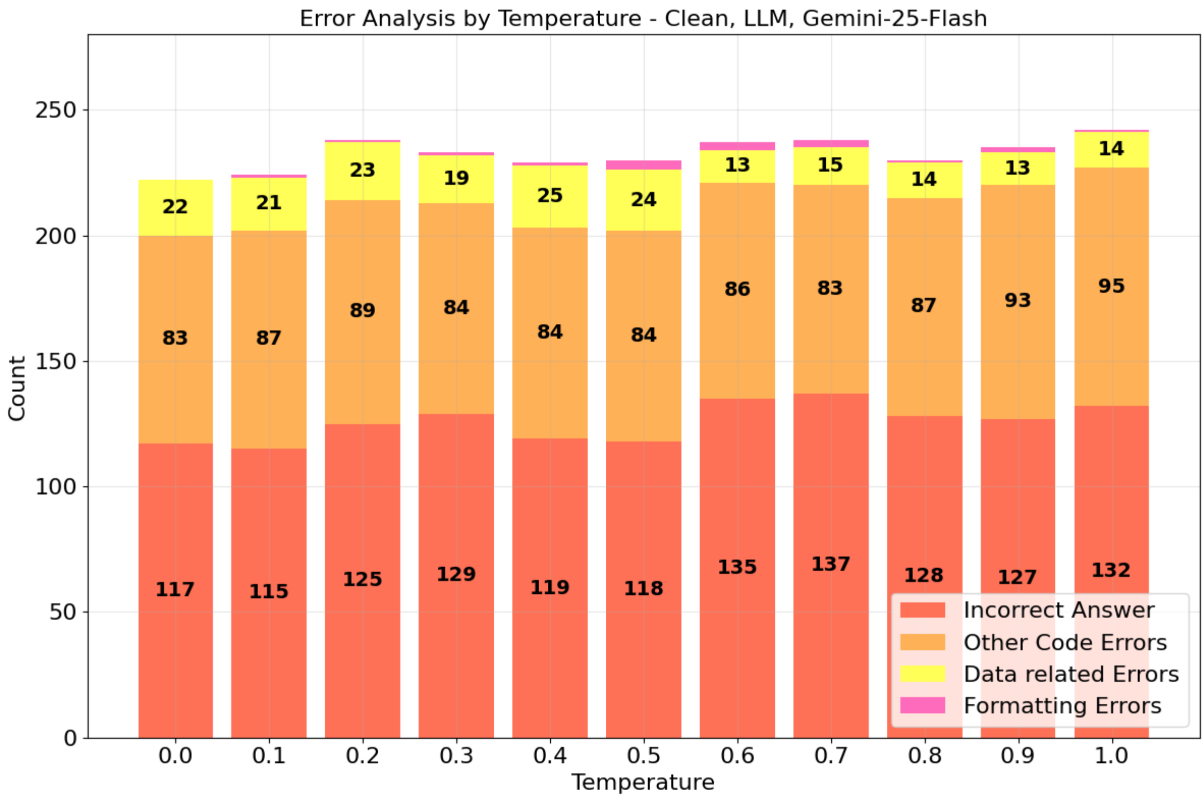

Figure 40: Incorrect response cause distribution -

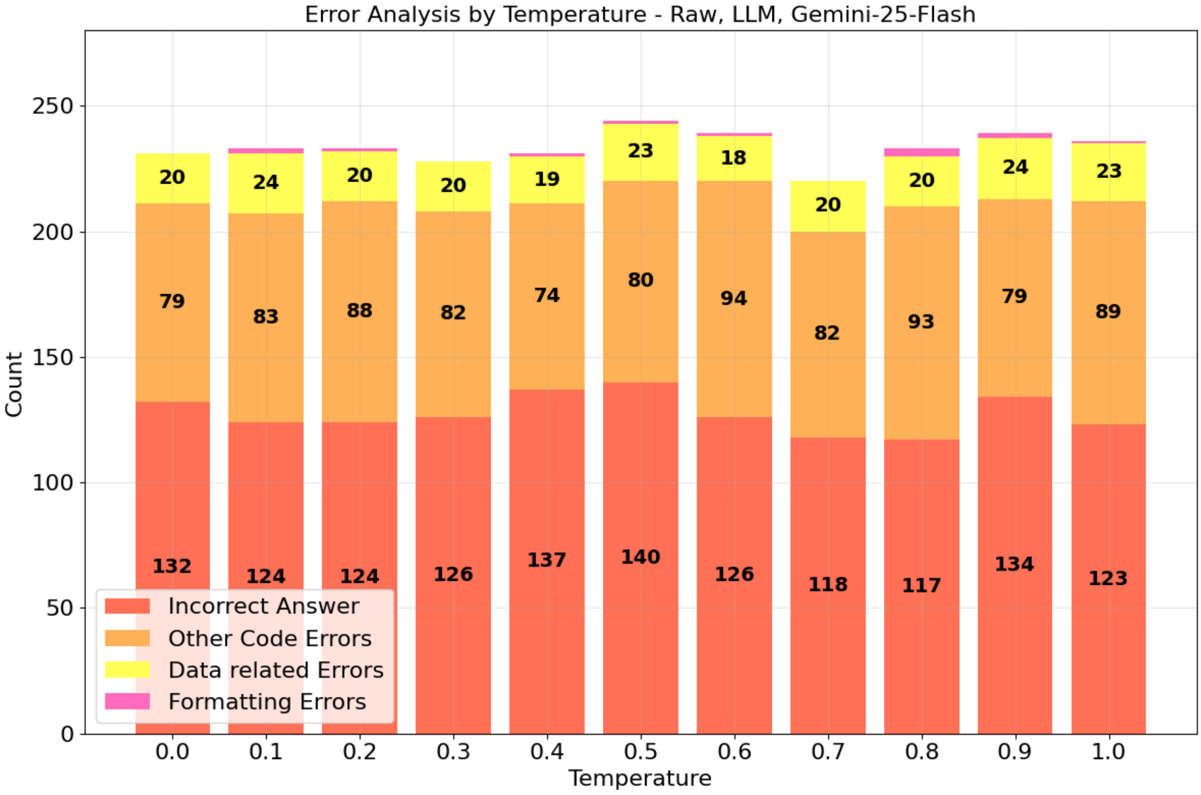

Figure 41: Incorrect response cause distribution -

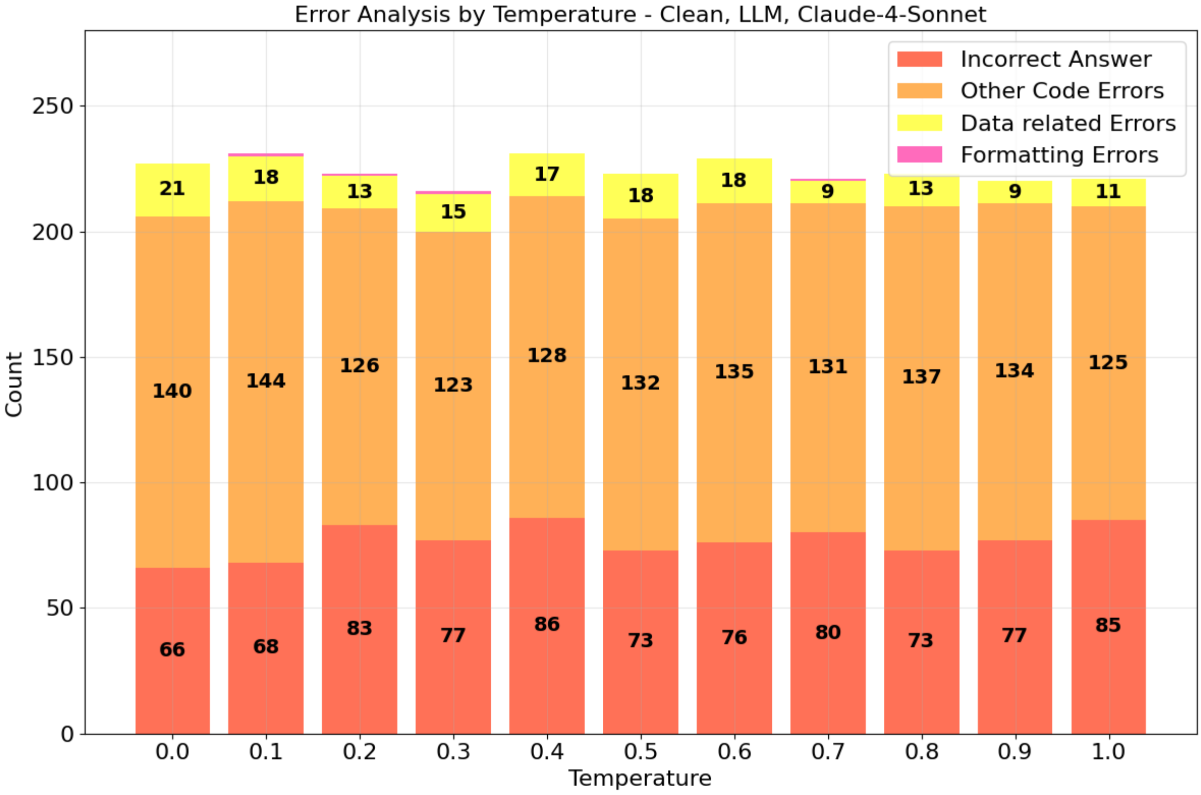

Figure 42: Incorrect response cause distribution -

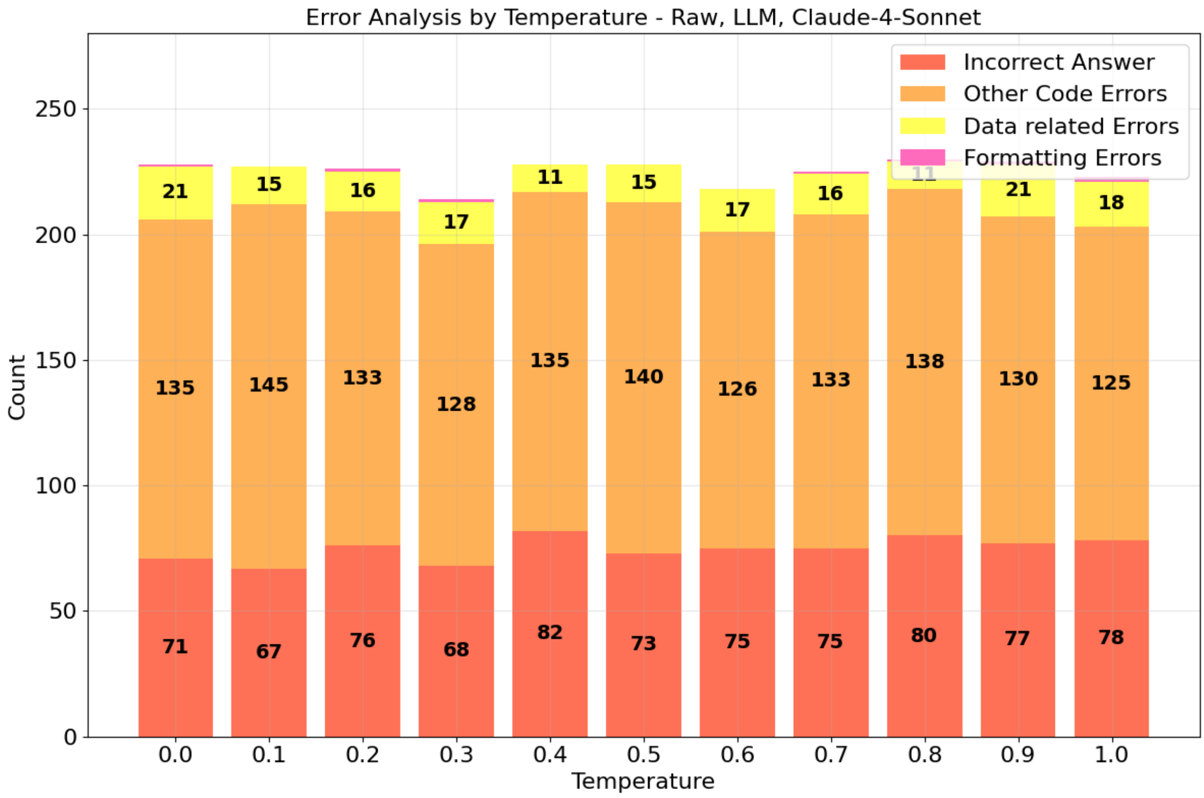

Figure 43: Incorrect response cause distribution -

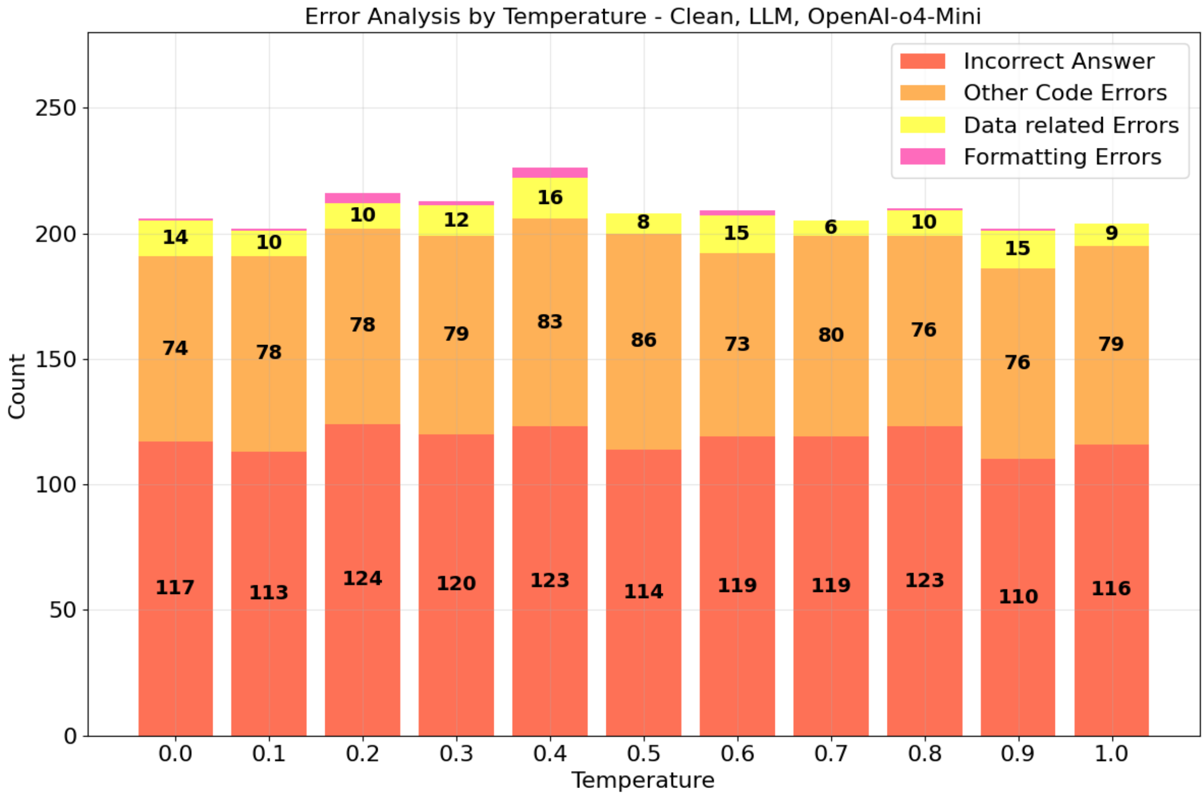

Figure 44: Incorrect response cause distribution -

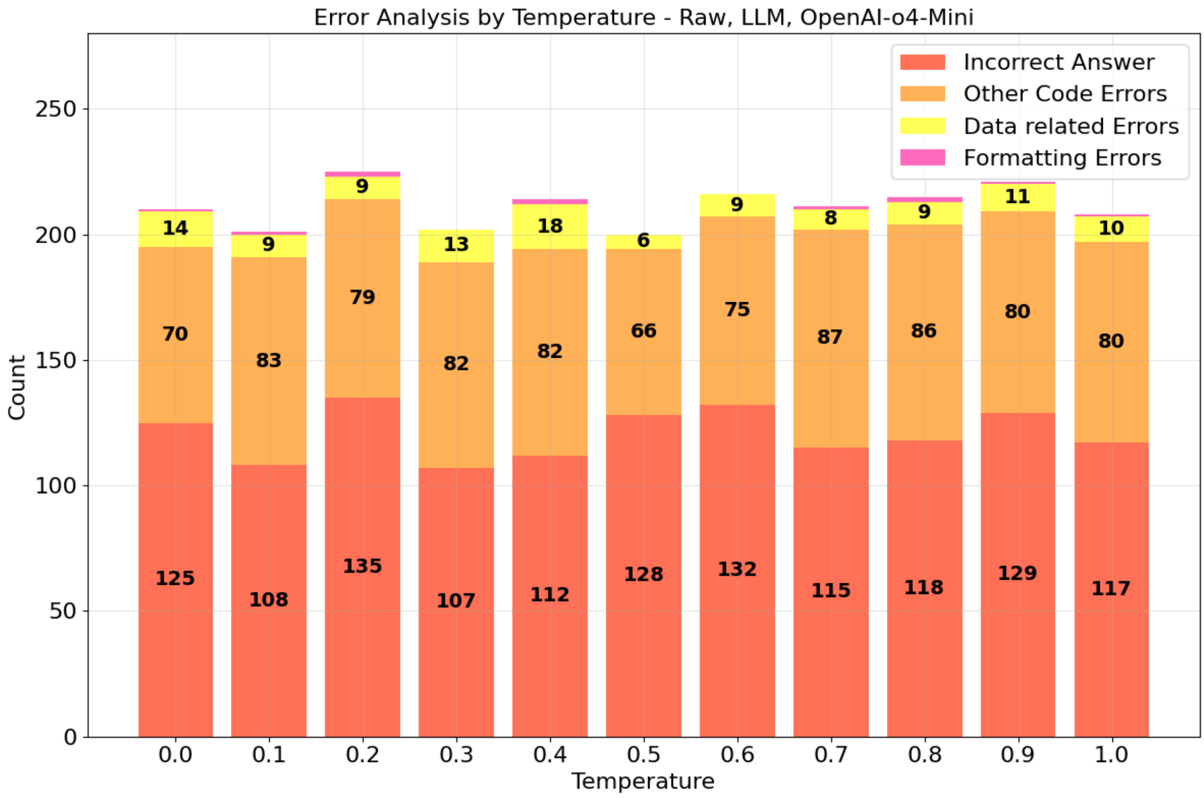

Figure 45: Incorrect response cause distribution -

I Other Plots

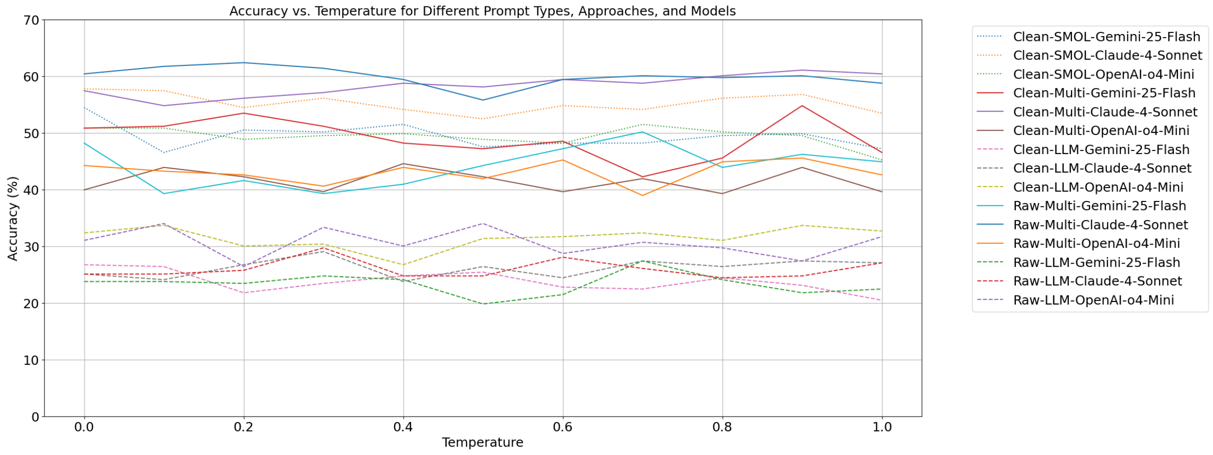

Figure 46: Results from each attempt over each temperature value used