PBE Meets LLM: When Few Examples Aren't Few-Shot Enough

Shuning Zhang
University of Illinois Urbana-Champaign
sz31@illinois.edu

Yongjoo Park
University of Illinois Urbana-Champaign
yongjoo@illinois.edu

ABSTRACT

Large language models (LLMs) can generate code from natural language descriptions. Their performance is typically evaluated using programming benchmarks that simulate real-world tasks. These benchmarks provide specifications in the form of docstrings, function signatures, or bug reports. The model then generates a program, which is tested against predefined test cases. In contrast, Programming by Example (PBE) uses input-output examples as the specification. Traditional PBE systems rely on search-based methods over restricted transformation spaces. They are usually designed for narrow domains and fixed input formats. It remains unclear how well LLMs perform on PBE tasks.

In this work, we evaluate LLMs on PBE tasks involving tabular data transformations. We prompt models to generate functions that convert an input table to an output table. We test the generated functions on unseen inputs to measure accuracy. Our study includes multiple LLMs and evaluates different prompting strategies, such as one-shot vs. multi-try. We also compare performance with and without PBE-specific knowledge. Finally, we propose a hybrid method that calls a traditional PBE solver first, and then falls back to LLMs if necessary. Our results show that LLMs support more diverse input formats and achieve higher accuracy than conventional methods. However, they struggle with tasks that contain ambiguity. The hybrid approach improves overall success by combining the strengths of both approaches.

VLDB Workshop Reference Format:
Shuning Zhang and Yongjoo Park. PBE Meets LLM: When Few Examples Aren't Few-Shot Enough. VLDB 2025 Workshop: 14th International Workshop on Quality in Databases (QDB'25).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/illinoisdata/PBE-Meets-LLM.git.

1 INTRODUCTION

Modern large language models (LLMs) [7, 25] are capable of generating high-quality programs from natural language descriptions. To assess their performance, researchers commonly use a variety of coding benchmarks that simulate real-world programming tasks. These benchmarks provide models with different forms of program specifications, such as docstrings and function signatures [4], function-level descriptions [2, 14, 39], class-level descriptions [6], or even bug reports [15, 24]. The model then generates a corresponding program, which is subsequently evaluated based on its ability to pass predefined test cases.

However, LLMs are less evaluated for a different mode of programming, i.e., Programming by Example (PBE) [19, 21]. Using tabular data as input-output examples as program specifications, PBE generates the best-suited program, which can be replicated on other datasets. PBE has been developed for number/string manipulation [8, 32–34, 36] and tabular data transformation (e.g., FlashFill [22] and FlashExtract [18]). These conventional PBE systems rely on techniques very different from how LLMs generate code. PBE methods perform a search (e.g., A* search [16]) within a more restrictive space of data transformation patterns, such as moving rows/columns, combining cells, and so on [27]. It is not straightforward whether LLMs, trained on massive datasets, would generate code that resembles the one produced by conventional PBE methods, or whether they would generate a completely different program.

In this work, we evaluate LLMs on a rerely explored area PBE tasks—tabular data transformations—to assess their generality and accuracy. That is, we construct prompts that ask the model to generate a function that transforms an example input table into a corresponding output table. The input table is provided in JSON format, capturing its structure. Then, we test the generated function on unseen test input tables to determine whether the desired output tables are produced. Our evaluation uses the latest proprietary LLMs (e.g., OpenAI GPT-4o) to compare their performance and identify limitations across a wide range of PBE tasks curated from prior work. Moreover, we examine whether these LLMs can achieve higher performance with different techniques, such as (1) one-shot vs multi-tries, (2) no external knowledge vs additional knowledge, and (3) a hybrid framework combining LLMs with traditional PBE models.

Our study shows that modern LLMs can achieve high accuracy on PBE tasks, covering more diverse input formats compared to conventional PBE methods, which are typically designed for a specific input format. For example, a conventional PBE method, FooFah [16], demonstrates good performance on its curated datasets, but fails more frequently on datasets prepared by another paper (e.g., Prose [9]). The reverse is also true. In contrast, LLMs tend to show high accuracy across diverse benchmark sets. However, our study also reveals limitations of LLMs. They often produce incorrect programs when the task contains inherent ambiguity, even if the correct transformation might be understandable to human engineers. We share and discuss such examples. Finally, our hybrid approach—which selectively employs LLMs when conventional methods fail—achieves higher performance than individual methods.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

Work done while Shuning Zhang was at the University of Illinois, Urbana-Champaign. The author is now with Meta.

2 RELATED WORK

2.1 Program by Example

When input-output examples are provided, algorithms can find patterns and apply the same transformation to new sets of inputs. This method lets users specify their intent through demonstration examples rather than explicit programming, making it a powerful tool for automating repetitive and domain-specific tasks. After the user provides the input-output examples, the system searches through predefined domain-specific operations to replicate the transformation. Instead of considering every possible function, the core of PBE systems is their ability to restrict the search space to a set of logical operations that are likely to be relevant. The synthesis engine, acting as the head of the network, will search for a program that performs the provided transformation, which involves different computational techniques, such as deductive reasoning, where the system identifies consistent patterns, and inductive learning, where it generalizes rules based on input-output mappings. The goal for such a network is to avoid overfitting and focus on simplicity, the ability to produce a program that can handle a broader range of similar inputs.

Previous PBE-driven tools have demonstrated strong capabilities in enabling users to perform complex data transformations through a few example-based interactions [1, 3, 5, 18, 23, 29, 32]. FlashExtract [18], a PBE framework integrated into Excel, enables users to extract structured data from semi-structured text with minimal user input, thereby significantly reducing the time required for data extraction tasks.

2.1.1 HoloClean. HoloClean [30] is a data cleaning system that leverages probabilistic modeling to automatically detect and repair errors in structured data. Its framework consists of three main components: error detection, compilation, and data repairing. During compilation, HoloClean generates a probabilistic model in which random variables represent uncertainty over the values in the input dataset. It uses factor graphs to encode the joint probability distribution over these variables, capturing dependencies derived from integrity constraints, statistical correlations, and external signals. To perform statistical learning and inference, HoloClean builds on DeepDive [31], a declarative probabilistic inference framework. Finally, HoloClean repairs detected errors by computing the marginal probabilities of candidate values and selecting the most likely ones.

2.1.2 Prose. Program Synthesis using Examples (Prose), a research group at Microsoft. The authors described their view of the PBE architecture, which consists of three components: a search algorithm, a ranking strategy, and user interaction models, as shown in Figure 1. The search algorithm is the key to determining if the system is efficient and accurate. A simple search strategy is to go through all the possible combinations of actions before making the final decision, and maintain a graph structure along the way. This method works well with a small number of operation pools. However, with more complex operations, this bottom-up search approach will have high costs on memory usage and time efficiency. Thus, they proposed combining Machine Learning (ML) into the program's training process to learn from those mistakes and improve the effectiveness and maintainability of the various PBE components. The combination leads to 8× faster program synthesis, enhances

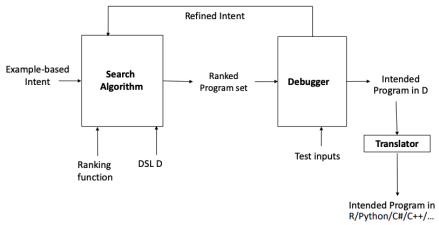

Figure 1: Overview of PBE Architecture[9]

ranking accuracy, and correctly identifies intended programs 50% more effectively than heuristic-based methods [10]. This technique makes PBE more scalable, adaptive, and practical for real-world applications like data wrangling and code transformation.

2.1.3 Foofah. Taking inspiration from the classic A * algorithm [13], Foofha proposed a heuristic search algorithm to synthesize data transformation. To find a path in a graph, the A* algorithm calculates the cost $f(x) = g(n) + h(n)$, where $g(n)$ is the cost to reach state n from the initial state, and $h(n)$ is the heuristic function calculate the approximate cost of the cheapest path from state n to the goal state, and chose the state with the minimum $f(x)$ to expand. In the case of PBE, instead of focusing on the shortest path, correctness, and readability are more important, the cost is defined by the minimum number of data transformation operations needed from one state to another. With the difference in goal in mind, first, a greedy algorithm to approximate Table Edit Distance(TED) was created:

$$TED(T_1, T_2) = \min_{\{p_1,\ldots,p_k\} \in P(T_1,T_2)} \sum_{i=1}^{k} \text{cost}(p_i) \tag{1}$$

TED calculated the minimum total cost of table edit operations needed to transform from Table 1 to Table 2. Observing frequent simultaneous edits of adjacent cells, TED was further refined into the Table Edit Distance Batch (TED Batch), effectively capturing these grouped operations. With their proposed system Foofah results in an interaction time that is 60% faster than its predecessors in each test, allowing the users to complete both data syntactic and layout transformation.

Although PBE has demonstrated remarkable capabilities, there are still challenges. These include ensuring it functions properly, handling unclear input patterns, and providing sufficient user interaction. Future work could improve probabilistic models for more reliable program inference. It could also include adding ways for users to provide feedback and creating more effective domain-specific languages (DSLs) to enhance its performance in various situations.

2.2 Large Language Model

In recent years, large language models have been extensively studied under the direction of various optimization techniques, aiming to enhance their efficiency and accuracy across different types of tasks. Among these, prompt engineering, fine-tuning, and retrieval-argument generation (RAG) have gained the most popularity and have become the main approaches to enhance the model's ability.

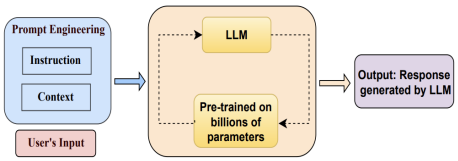

Figure 2: Prompt Engineering Pipeline [35]

Those methods enable models to be more closely tailored to specific tasks, improve accuracy, and reduce computational costs. This section focuses on prompt engineering, covering the key ideas and different approaches employed in this work.

2.2.1 Zero-shot prompting: refers to the model's ability to solve the user query without needing examples during inference, eliminating the need for carefully crafted prompts [26]. Instead, it depends heavily on the model's pre-training data and its own ability to understand and reason what the user needs.

2.2.2 Chain-of-Knowledge (CoK) Prompting: LLMs are trained with billions of data points covering a wide range of topics, which may confuse and hallucinate what specific techniques to use. Inspired by human problem-solving, Chain of Knowledge (CoK) [20] systematically breaks down intricate tasks into well-coordinated steps. Engage in a dynamic knowledge adaptation phase, collecting evidence from various sources, such as internal knowledge base, external databases, and the given prompt.

2.2.3 Multi Turn Reasoning: To reduce the hallucinations that exist in the LLMs responses, multi turn reasoning [37, 38] approach was used. This includes the model first generating an initial solution, then the verification step to check if the original responses sufficiently solve the users' request, and lastly, producing a revised response if necessary. By verifying its own work through this multi-step process, the LLM enhances its logical reasoning ability and reduces hallucination errors. Focused verification step help models identify and correct their inaccuracies.

3 EXPERIMENT DESIGN

To better understand the capabilities of LLMs, we designed a series of experiments exploring various proposed approaches by systematically combining different dimensions, such as prompt strategies, external knowledge, and hybrid methods. All prompts and datasets used in the experiments are available in our GitHub repository.

3.1 Evaluating LLMs on PBE tasks

We aimed to evaluate the capability of current LLMs in performing PBE tasks related to data transformation. We conducted initial experiments, one shot no knowledge prompt approach, using several baseline models, including Llama, Gemini, Claude, and GPT. The results showed that GPT achieved the highest accuracy at 79.7%, outperforming Llama-2-13B (17.21%), Gemini-1.5-pro (47.05%), and Claude 3.5 (71.41%). Motivated by GPT's superior performance, we chose to explore methods to enhance its effectiveness in this domain further.

3.2 Approach 1: Large-Language Model

To maintain fairness, all tests were performed using GPT-4o's API call on chat completion under identical settings across all runs.

3.2.1 Dimension 1: One-shot vs Multiple Tries.
One-Shot: This vanilla approach prompts the LLM one time with the full task context, including the example input and expected output, and asks it to generate a transformation function that performs the required data manipulation. In this setting, the LLM is expected to synthesize the correct transformation logic in one pass, without any feedback loop or iterative refinement. The goal of this setup is to evaluate the model's ability to generalize and reason correctly with minimal external guidance or opportunity for correction.

Multiple Tries: We also examine the Large Language Model's ability to discover mistakes and whether they are able to learn from its own mistakes and correct itself. To explore this, we proposed the multi-turn verification method. Unlike conventional multi-step approaches, where LLMs see the same question a set number of times, this method provides different prompts at each iteration. And let the LLM utilize the previous responses to identify error based on the output of its prior responses, similar to the chain of verification prompting.

The process begins with the LLM generating an initial function that it believes will reproduce the data transformation. This function is then executed locally on the example input dataset. If the generated output does not match the example output, two separate experiments are tested: **(a)** The model is provided with its previously generated code, output, and example output. And let the model generate the code again. **(b)** An extra error verification step is added to this approach. Where the generated and example outputs are passed to a separate GPT4-o model, specifically prompted to identify high-level structural errors. Then, pass the list of fixes and the previous chat history back to the original model and generate the code again. Finally, after iterating through this verification loop, the LLM produces the final output, which ideally should successfully reproduce the data transformation on the test dataset.

3.2.2 Dimension 2: No Knowledge vs Extra Knowledge.
No Knowledge base prompt: The naive no-knowledge prompt includes the input and output list and a short description of the task the LLM needs to perform. To streamline the process, the test input list was also provided as part of the prompt and asked the LLM to simply put the test list as the function input argument, ensuring a more structured and efficient response.

Knowledge prompt: A list of simple programming functions are provided as part of the prompt. Similar to how Foofah provided the list of operations for its model to search and create an action plan, we used Foofah's operation set as a starting point—since it covers most commonly seen data transformation techniques—and adapted it to be Python 3 compatible while constraining the search space for efficiency. However, unlike previous conventional PBE programs, which combine multiple sub-programs into a step-by-step series, we asked the LLMs to learn from the provided information and generate a program that performs the transformation in one single function call. This extra information helps the model reduce the search space and provides a better understanding of the task, and

guides it to focus more effectively on data transformation and Python code generation.

3.3 Approach 2: Search-LLM Hybrid

LLMs show strong reasoning ability and are great at solving complex, previously unseen tasks. While traditional machine learning models, on the other hand, always bring stability and efficiency. With the advantages of both models in mind, we explored the third method: the hybrid model. In this approach, LLM will take charge when the traditional PBE model struggles to create the perfect program, which might be caused by the example's complexity or by an unseen example dataset in which the PBE model cannot effectively reproduce the transformation. By combining these two methodologies, we aim to enhance adaptability and improve performance in challenging data transformation tasks. Moreover, this strategy leverages the strength of both approaches while ensuring that the system maintains robustness when seeing unfamiliar task cases.

3.4 Experiment Setup

Baseline: Two program synthesis models built for Programming by Example (PBE) in data transformation tasks were used as the baseline comparison.

- Foofah [16]: A PBE system that targets solving both syntactic and layout data transformations. Proposed the program synthesis as a search problem in a state space graph and a heuristic search approach based on their proposed class A* algorithm to synthesize the program.
- Prose [9]: a research group in Microsoft led by Sumit Gulwani. They were also the first group of people to start looking into the field of Programming by Example.

The PBE architecture consists of three components: a search algorithm, a ranking strategy, and user interaction models. The search algorithm is the key to determining if the system is efficient and accurate. A simple search strategy is to go through all the possible combinations of actions before making the final decision, and maintain a graph structure along the way.

Dataset: In this work, two distinct datasets are utilized to evaluate LLMs' capacity to comprehend the underlying data structure and relationships. The first dataset, corrected version shared by Foofah, initially contains the input data in a non-relational format, which, after transformation, will result in a relational output table.

- Foofah [16]: Published its dataset as a combination of the previous datasets, ProgFromEx [12], Wrangler [17], Potter's Wheel (PW) [28] and Proactive Wrangler (Proactive) [11] with its contribution, containing 50 test scenarios in total. In the Foofah dataset, each test scenario contains five subtest files, numbered 1 through 5, indicating the number of data records selected from the raw data as the example input and output that are passed into the program.
- Prose [9]: used has a semi-structured format. Unlike the relational dataset, semi-structured data does not adhere to a strict schema, presenting a distinct set of challenges for LLMs. The Prose dataset includes tasks involving string transformations and Excel-like data manipulation, often formatted in JSON or XML, making it a useful benchmark for evaluating model performance on real-world, flexible data representations.

Metrics: Exact match accuracy was used for evaluation, comparing the transformed table produced by the model's generated function directly with each test case's ground truth output table. For each test case, we compared the resulting table row by row to check whether each data point matched precisely with the corresponding value in the expected output. Then, the accuracy of each row was calculated based on the percentage of data points in that row that were correct; the overall accuracy for each test dataset is the average accuracy of all rows. Unlike binary exact match, this approach offers a more detailed assessment by capturing partial accuracy inside every sample. Hence, it presents a fuller picture of the model's ability. To report the final accuracy we used the weighted average, as each dataset contains a different distribution of transformation types, and using a simple macro-average could overemphasize datasets with rare or easier transformations. By applying a weighted average, we ensure that the contribution of each dataset reflects its size and diversity, providing a fair and balanced overall evaluation.

4 EVALUATION RESULTS

Since running the same task multiple times is often unrealistic in real-world applications, where users expect reliable results in a single attempt. Therefore, instead of running the model multiple times and averaging the accuracy for each test case, we evaluated the correctness of the generated programs from each approach based on a single execution per model across the two datasets.

- How would different dimensions from approach 1 influence the accuracy
- How well these models generalize across domains and datasets

4.1 LLMs Benefit From Knowledge and Tries

Baseline: To establish a baseline for comparison, we consider the accuracy of Foofah and Prose, which are traditional search-based program synthesis methods. Foofah achieves a weighted average accuracy of 0.571, while Prose scores 0.473. These scores serve as benchmarks for evaluating the effectiveness of newer LLM-based approaches.

One-shot vs Multiple Tries: Comparing GPT-4o + One-Shot (0.797) to GPT-4o + Multi-tries (0.786), we observe that one-shot performance is slightly better in this instance. However, the difference is minimal, and multi-try methods tend to be more consistent across datasets. When combined with extra knowledge, the multi-try setup (best at 0.827) clearly outperforms its one-shot counterpart with extra knowledge (0.766), demonstrating the value of multiple attempts in more informed settings. With the two variants from Multi-tries, they both deliver competitive performance. Specifically, variant (b) achieves slightly higher peak scores (0.943) compared to (a) (0.876), indicating that variant (b) is superior at achieving high-quality results in certain instances. However, variant (a) demonstrates more stable consistency across other metrics, suggesting it provides balanced outcomes.

No Knowledge vs Extra Knowledge: To evaluate the effect of external knowledge, we compare setups without and with extra

Table 1: Accuracy on Foofah and Prose Dataset

Approach	Dataset					Overall (Weighted Avg.)
	ProgEx [12]	Wrgler [17]	Potter [28]	Proact [11]	Prose [9]	
Prose	0.139	0.286	0.183	0.156	**0.949**	0.473
Foofah	0.689	0.667	0.891	**1**	0.300	0.571
Gemini-3.5	0.790	0.670	0.793	**1**	0.566	0.714
Llama2-13B	0.068	0.433	0.135	0.043	0.257	0.172
Gemini-1.5-pro	0.068	0.417	0.677	0.400	0.530	0.471
GPT-4o + One-Shot	0.368	0.733	0.886	0.770	0.755	0.797
GPT-4o + Multi-tries (a)	**0.813**	0.827	0.874	0.876	0.858	0.786
GPT-4o + Multi-tries (b)	0.787	**0.870**	0.857	0.943	0.873	0.846
GPT-4o + One-Shot + Extra kg	0.759	0.713	0.833	0.880	0.731	0.766
GPT-4o + Multi-tries (a)+ Extra kg	0.778	0.763	0.814	0.890	0.860	0.827
GPT-4o + Multi-tries (b)+ Extra kg	0.718	0.867	0.794	0.933	0.878	0.811
Foofah + GPT-4o + Multi-tries	0.802	**0.870**	**0.904**	**1**	0.883	**0.863**

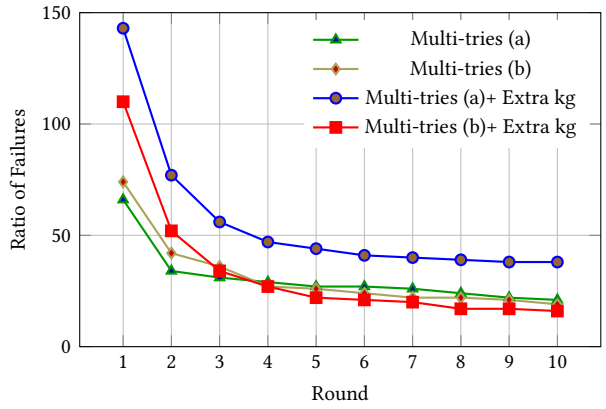

Figure 3: Trends in code generation failures across iterative rounds of the loop process. Failures are defined as either a mismatch with the intended data transformation logic or execution errors. The plot demonstrates a consistent decrease in failure rate with increased iteration.

knowledge. For GPT-4o + Multi-tries, adding extra knowledge improves performance from 0.786 to 0.827. Similarly, GPT-4o + Multi-tries with extra knowledge achieves consistently high accuracy across datasets, indicating that domain or dataset-specific guidance significantly enhances synthesis outcomes. Notably, combining Foofah with GPT-4o + Multi-tries further boosts the weighted average accuracy to 0.863, the highest among all tested approaches.

Hybrid Method: Combining traditional systems like Foofah with GPT-4o Loop Verification (a) Base Prompt resulted in 0.863 accuracy overall. It performed well on Proact (1.0) and Prose (0.883), but still faced challenges with intricate datasets.

4.2 LLM Could Complement Conventional PBE

One limitation of the traditional PBE system is that it is all built within the knowledge of a specific domain. Even the overall task may fall under the general area of data transformation; however, the model's ability is limited by the program provided to the system and the rules available to it. As a result, the previous PBE systems often struggle when seeing tasks outside their designated area, leading to limited usability and a lack of flexibility when adapting to new or unfamiliar transformation patterns.

However, this is not a problem for large language models. With billions of data points being passed to different models for the training process, the model is designed to solve a wide range of tasks with its extensive world knowledge. This nature enables LLMs to handle unseen or complex data transformation tasks with fewer human-enforced guidelines, as the table below indicates. For instance, one test case from the Prose dataset [9] involves mapping language names to their corresponding ISO 639-1 codes, such as ["Arabic = ar"], ["Basque = eu"]. GPT-4o is able to solve this correctly, whereas Foofah, constrained by its limited domain knowledge, fails to generalize to such examples.

Table 2: Evaluation result on cross-domain datasets

	Foofah Dataset	Prose Dataset
% test cases passed to LLM	15.88%	78.95%
% test cases solved by LLM	48.65%	45.19%

These differences become particularly evident when comparing the performance of LLM vs. PBE systems, Foofah and Prose, across various datasets. As shown in the table 1, Prose did remarkably well on its own dataset, achieving around 90% accuracy. However, it experienced a considerable performance drop on Foofha's dataset, highlighting its limitations in generalizing across domains, with unseen transformation types or dataset structures. Similarly, Foofah performs well on its domain dataset but decreases its performance when evaluated on the Prose dataset.

In contrast, LLM, with only the base prompt and no other human intervention or task-specific tuning, performs relatively well on both datasets, achieving 82.42% on the Foofah dataset and 75.53% on the PROSE dataset on transformation text. With the highest accuracy achieved using the hybrid method, LLM has successfully addressed the domain-specific limitations of the traditional PBE system.

5 DISCUSSION

In this section, we analyze two of the most common and impactful failure cases encountered when using LLM-generated code for

Example Input					
001-001	1				$-
001-001	2				$-
001-001	3				$7,664.25
001-001	4				$-

Example Output				
001-001	$-	$-	$7,664.25	$-

(a) Example transformation in prompt

Test Input					
001-001	2				$-
001-001	4				$-
001-001	6				$-
001-001	8				$-
001-001	9				$7,664.25

Ground Truth					
001-001	$-	$-	$-	$-	$7,664.25

(b) Expected, Ground-truth transformation

Test Input					
001-001	2				$-
001-001	4				$-
001-001	6				$-
001-001	8				$-
001-001	9				$7,664.25

GPT Output
no output: indexing error

(c) Actual transformation by generated code

Figure 4: The model incorrectly generalizes from the example by treating the second column as a sequential index, leading to a list assignment index error when test data lacks a continuous sequence.

data transformation tasks. Despite their overall effectiveness, LLMs exhibit systematic weaknesses under certain conditions. The first failure case, driven by ambiguity in the input data, caused 87% of the test cases to fail in this area, highlighting the model's tendency to overfit to patterns in the examples rather than generalizing the underlying logic. The second case, which requires multi-dimensional reasoning and contextual understanding, resulted in a 60% failure rate, showing that LLMs often struggle to infer implicit constraints or real-world expectations.

5.1 LLM failed due to ambiguity.

We highlight a failure case where the LLM-generated code does not generalize correctly due to ambiguity in the input data. Specifically, the model overfits to patterns present in the example dataset, making incorrect assumptions about the structure of the input. This illustrates a common challenge in LLM-based code generation: when the intent is underspecified or when misleading patterns exist in the examples, the model may infer incorrect logic that does not hold in broader contexts.

As shown in the example in figure 4, the model should realize that the transformation only extracts the last entry from each row and combines it into a single row with the same 'item-id' (the first element in the row). Instead, because the second element in each row in the example starts with one and has an increment of 1 in the following rows, the model mistakenly interprets it as an index and applies this logic to the program. While this works correctly in the example datasets, the program fails the test cases. In the test case, the second item in each row is no longer a continuous sequence, which causes the list assignment index out-of-range error.

Example Input Table				
21-May-00	1973	Living	6-Nov-62	5 December 1870

Example Output Table				
2000	1973	2025	1962	1870

(a) Example transformation in prompt

Test Input Table				
Living	2-Dec-65	1 December 1848	1984	28-Nov-68

Ground Truth Table				
2025	1965	1848	1984	1968

(b) Expected, Ground-truth transformation

Test Input Table				
Living	2-Dec-65	1 December 1848	1984	28-Nov-68

GPT Output Table				
2025	2065	1848	1984	2068

(c) Actual transformation by generated code

Figure 5: GPT-4o extracts dates accurately but lacks commonsense reasoning, misinterpreting past events as future ones.

5.2 LLM failed due to Multi-Dimensional Tasks

Some transformation tasks go beyond simple structural manipulation and require reasoning over multiple domains, such as temporal context, commonsense knowledge, or domain-specific conventions. These tasks pose a greater challenge for LLMs, which may correctly parse and transform text but fail to infer implicit meanings or constraints. For example, as shown in the figure 5 below, the task is to extract the year of death from the list of authors.

When examining the data, it is an instinctive assumption for humans that a value like "65" in the table refers to the year 1965 or at least any century before the current year. However, when looking at the program result from the LLM-generated code, GPT-4o was able to slice the string successfully and extract the year from the original input data. Still, it was unable to reason that the data refers to the author's passing away, and the date should not be in the current year. These examples and others from the PROSE dataset illustrate a limitation of the GPT-4o model in that it can handle structure-wise transformations well. However, they may struggle with tasks that require them to think of another step and presume the correct meaning of the text.

6 CONCLUSION

In this work, we examine the logical reasoning and code generation ability of LLMs through PBE tasks focused on tabular formated data transformation. Three approaches were proposed that do not alter the model's structure and minimize user effort to enhance the model's performance on this specific task. As the experiment result indicates, all of them were able to outperform Foofah's PBE system performance with the highest accuracy, reaching 86.3%. These approaches demonstrate that with carefully designed prompting strategies, LLMs can perform structured tasks more effectively without any fine-tuning or architecture modification. As the target users for such methods are end users who may not have technical backgrounds, the methods we mentioned are a great fit as they do not require modifying the model structure or setting up additional tools. They also highlight how adaptable modern LLMs can be with different type of tasks when paired with the proper reasoning framework and problem decomposition strategy. There still remain promising directions for future research, as discussed earlier, to further enhance LLM performance with tabular data on more complex tasks such as PBE, including handling noisier data, multi-step reasoning, and more generalized transformation goals.

REFERENCES

[1] Ziawasch Abedjan, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. DataXFormer: A robust transformation discovery system. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 1134–1145. https://doi.org/10.1109/ICDE.2016.7498319

[2] 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* (2021).

[3] Shaon Barman, Sarah Chasins, Rastislav Bodik, and Sumit Gulwani. 2016. Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Amsterdam, Netherlands) *(OOPSLA 2016)*. Association for Computing Machinery, New York, NY, USA, 748–764. https://doi.org/10.1145/2983990.2984020

[4] 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* (2021).

[5] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376442

[6] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2023. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation. *arXiv preprint arXiv:2308.01861* (2023).

[7] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, and etc. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[8] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-Output Examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. 317–330. https://doi.org/10.1145/1926385.1926423

[9] Sumit Gulwani. 2015. Programming by Examples (and its applications in Data Wrangling). In *Lectures at Marktoberdorf Summer School, Aug 2015* (lectures at marktoberdorf summer school, aug 2015 ed.). https://www.microsoft.com/en-us/research/publication/programming-examples-applications-data-wrangling-2/

[10] Sumit Gulwani and Prateek Jain. 2017. Programming by Examples: PL meets ML. In *APLAS 2017* (aplas 2017 ed.). Springer. https://www.microsoft.com/en-us/research/publication/programming-examples-pl-meets-ml/

[11] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/2047196.2047205

[12] William R. Harris and Sumit Gulwani. 2011. Spreadsheet table transformations from examples. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Jose, California, USA) *(PLDI '11)*. Association for Computing Machinery, New York, NY, USA, 317–328. https://doi.org/10.1145/1993498.1993536

[13] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[14] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. *NeurIPS* (2021).

[15] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).

[16] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 683–698. https://doi.org/10.1145/3035918.3064034

[17] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (, Vancouver, BC, Canada,) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 3363–3372. https://doi.org/10.1145/1978942.1979444

[18] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14)*. Association for Computing Machinery, New York, NY, USA, 542–553. https://doi.org/10.1145/2594291.2594333

[19] Douglas B. Lenat. 1976. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.* Technical Report AIM-286. Stanford University, Artificial Intelligence Laboratory.

[20] Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. 2024. Chain-of-Knowledge: Grounding Large Language Models via Dynamic Knowledge Adapting over Heterogeneous Sources. arXiv:2305.13269 [cs.CL] https://arxiv.org/abs/2305.13269

[21] Henry Lieberman (Ed.). 2001. *Your Wish is My Command: Programming by Example.* Morgan Kaufmann, San Francisco, CA.

[22] Na Meng, Miryung Kim, and Kathryn S. McKinley. 2011. Systematic editing: generating program transformations from an example. *SIGPLAN Not.* 46, 6 (June 2011), 329–342. https://doi.org/10.1145/1993316.1993537

[23] Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. 2013. A Machine Learning Framework for Programming by Example. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Sanjoy Dasgupta and David McAllester (Eds.), Vol. 28. PMLR, Atlanta, Georgia, USA, 187–195. https://proceedings.mlr.press/v28/menon13.html

[24] Noor Nashid, Islem Bouzenia, Michael Pradel, and Ali Mesbah. 2025. Issue2Test: Generating Reproducing Test Cases from Issue Reports. *arXiv preprint arXiv:2503.16320* (2025).

[25] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[26] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI* (2019). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf Accessed: 2024-11-15.

[27] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. 381–390.

[28] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390.

[29] Mohammad Raza and Sumit Gulwani. 2017. Automated Data Extraction Using Predictive Program Synthesis. *Proceedings of the AAAI Conference on Artificial Intelligence* 31, 1 (Feb. 2017). https://doi.org/10.1609/aaai.v31i1.10668

[30] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. arXiv:1702.00820 [cs.DB] https://arxiv.org/abs/1702.00820

[31] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental knowledge base construction using DeepDive. *Proc. VLDB Endow.* 8, 11 (July 2015), 1310–1321. https://doi.org/10.14778/2809974.2809991

[32] Rishabh Singh. 2016. BlinkFill: semi-supervised programming by example for syntactic string transformations. *Proc. VLDB Endow.* 9, 10 (jun 2016), 816–827. https://doi.org/10.14778/2977797.2977807

[33] Rishabh Singh and Sumit Gulwani. 2012. Learning Semantic String Transformations from Examples. *Proceedings of the VLDB Endowment (PVLDB)* 5, 8 (2012), 740–751. https://doi.org/10.14778/2212351.2212357

[34] Rishabh Singh and Sumit Gulwani. 2012. Synthesizing Number Transformations from Input-Output Examples. In *Proceedings of the 24th International Conference on Computer Aided Verification (CAV)*. 634–651. https://doi.org/10.1007/978-3-642-31424-7_44

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

[36] Chenglong Wang, Alvin Cheung, and Rastislav Bodík. 2017. Synthesizing Highly Expressive SQL Queries from Input-Output Examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 452–466. https://doi.org/10.1145/3062341.3062365

[37] Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. 2025. Reinforcing Multi-Turn Reasoning in LLM Agents via Turn-Level Credit Assignment. arXiv:2505.11821 [cs.LG] https://arxiv.org/abs/2505.11821

[38] Kunhao Zheng, Juliette Decugis, Jonas Gehring, Taco Cohen, Benjamin Negrevergne, and Gabriel Synnaeve. 2025. What Makes Large Language Models Reason in (Multi-Turn) Code Generation? arXiv:2410.08105 [cs.CL] https://arxiv.org/abs/2410.08105

[39] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877* (2024).