Code Prompting Elicits Conditional Reasoning Abilities in Text+Code LLMs

Anonymous ACL submission

Abstract

Reasoning is a fundamental component of language understanding. Recent prompting techniques, such as *chain of thought*, have consistently improved LLMs' performance on various reasoning tasks. Nevertheless, there is still little understanding of what triggers reasoning abilities in LLMs in the inference stage. In this paper, we investigate the effect of the *input representation* on the reasoning abilities of LLMs. We hypothesize that representing natural language tasks as code can enhance specific reasoning abilities such as entity tracking or logical reasoning. To study this, we propose *code prompting*, a methodology we operationalize as a chain of prompts that transforms a natural language problem into code and *directly* prompts the LLM using the generated code *without* resorting to external code execution. We find that code prompting exhibits a high-performance boost for multiple LLMs (up to 22.52 percentage points on GPT 3.5, 7.75 on Mixtral, and 16.78 on Mistral) across multiple conditional reasoning datasets. We then conduct comprehensive experiments to understand *how* the code representation triggers reasoning abilities and *which* capabilities are elicited in the underlying models. Our analysis on GPT 3.5 reveals that the code formatting of the input problem is essential for performance improvement. Furthermore, the code representation improves *sample efficiency* of in-context learning and facilitates *state tracking* of entities.[1]

1 Introduction

Reasoning is a fundamental component of both human and artificial intelligence (AI) and the backbone of many NLP tasks. Recently, intensive studies have been performed on different aspects or types of reasoning such as mathematical reasoning (Patel et al., 2021; Chen et al., 2021b; Cobbe et al., 2021), various kinds of logical reasoning (Liu et al., 2020, 2023a; Sinha et al., 2019), and commonsense-focused reasoning (Madaan et al., 2022; Liu et al., 2022a,b; Wang et al., 2023). *Conditional reasoning*, a primary yet complex reasoning ability that draws alternative conclusions depending on the fulfillment of certain *conditions*, remains understudied. These conditions are stated in the text, making the problem self-contained, which allows us to study the semantic inferencing capabilities of the underlying model, i.e., identifying relevant premises and ascertaining the presence of total evidence (Nolt et al., 1988; Cabria and Magnini, 2014) without the requirement for, and confounding effects of external knowledge. Conditional reasoning is also a fundamental form of reasoning useful in many practical scenarios, such as answering real-world questions about the eligibility for a visa or a loan. Despite the recent introduction of some benchmarks (Saeidi et al., 2018; Sun et al., 2022; Kazemi et al., 2023), conditional reasoning abilities of LLMs remain understudied.

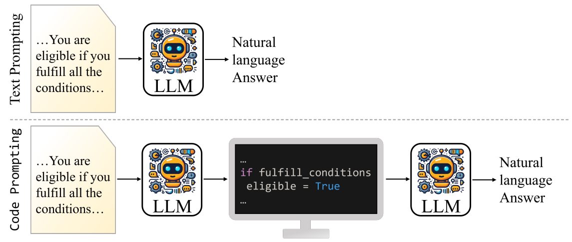

Figure 1: Code prompting converts a natural language problem into a *code prompt* and prompts a large language model with such code to generate an answer.

Recently, researchers have analyzed the synergies between LLMs and symbolic interpreters to improve performance on reasoning tasks (Gao et al., 2023; Chen et al., 2023; Lyu et al., 2023). These works transform structured reasoning problems, such as mathematic or symbolic reasoning, into code and run it on an external interpreter. In such a setup, LLMs are mainly focused on natu-

[1]Our code and prompts are available at this URL

ral language representation aspects and planning how to solve the problem, while the actual logical reasoning is offloaded to an external execution module, confounding our understanding of the reasoning In particular, the fundamental questions of *what* contributes to the reasoning abilities and *how* reasoning abilities are triggered in LLMs remain open. Nevertheless, pretraining on code is considered an important component that contributes to and explains the improved reasoning ability of LLMs. State-of-the-art LLMs such as GPT 3.5 (Kojima et al., 2022), GPT 4 (OpenAI, 2023), Mixtral (Jiang et al., 2024), and Mistral 7B (Jiang et al., 2023) have been pretrained on both text and code and have demonstrated considerable boosts in many reasoning benchmarks.

In this work, we analyze whether one can elicit improved conditional reasoning abilities in LLMs by merely changing the input format, i.e., from text to code. We constrain our experiments to text+code LLMs to run text and code inputs on the same underlying model. In this way, we can avoid the confounding factor of different pretraining data of specialized text and code LLMs. To understand the benefit of code as an intermediate representation, we devise a chain of prompts, *code prompting*, that transforms a natural language (NL) task into code and directly prompts the LLM with the generated code. The code contains the logical structure needed to solve the problem, along with the original natural language text as code comments. An illustration is provided in Figure 1. Our contributions are summarized as follows:

- We propose a methodology to investigate how the input representation impacts the reasoning abilities of text+code LLMs.
- We operationalize such methodology by introducing a *chain of prompts* that transforms a NL task into code, which is then sent back to the LLM to generate NL answers.
- We conduct a comprehensive study to compare code prompts with text prompts, showing (i) large performance gains on the three LLMs (up to 22.52 points for GPT3.5, up to 7.75 for Mixtral, and up to 16.78 for Mistral), while (ii) being more efficient with regard to the number of demonstrations.
- We conduct extensive analysis to understand why code prompts efficiently elicit conditional reasoning abilities, showing that prompting with code yields largely improved variable state tracking.

2 Background and Related Work

LLM Types. We categorize LLMs into three types according to their intended use: i) LLMs for natural language text (*text LLMs*), ii) LLMs for coding tasks (*code LLMs*), and iii) LLMs for natural language and coding tasks (*text+code LLMs*). The intended use of *text LLMs* (Zhang et al., 2022; Touvron et al., 2023) is to process and generate natural language text such as answers to questions. The intended use of *code LLMs* (Li et al., 2023b; Roziere et al., 2023) is to process and generate code. Lastly, *text+code LLMs* are equally capable of solving natural language and coding tasks. Examples of this are GPT 3.5 (Kojima et al., 2022), Mixtral (Jiang et al., 2024), and Mistral (Jiang et al., 2023). In this work, we focus on *text+code LLMs* because of their ability to process two types of input representations interchangeably: natural language text and code. Using such models eliminates the confounding effect of fine-tuning between model variants specialized for only text or code.

Augmenting text with code. Most works that generate code to solve natural language tasks use an external symbolic interpreter to run the resulting code. Chen et al. (2023) and Gao et al. (2023) showed consistent gains on mathematical problems, symbolic reasoning, and algorithmic problems by using LLMs aided by external code interpreters. Lyu et al. (2023) further report improvements in boolean multi-hop QA, planning, and relational inference. In contrast, Ye et al. (2023) used an external automated theorem prover with declarative code and showed consistent gains w.r.t. imperative code-interpreter-aided LLMs on arithmetic reasoning, logical reasoning, symbolic reasoning, and regex synthesis tasks. Pan et al. (2023) did not use any interpreter and instead created programs composed of multiple subroutines and used smaller specialized models to run them. In this way, they outperform text prompts on text LLMs for fact-checking tasks. Lastly, Li et al. (2023a) runs pieces of code in an LLM to update the program state when the Python interpreter fails due to a code exception and shows performance gains on BIG-Bench Hard (Suzgun et al., 2022). All these works investigate how to best use an external symbolic interpreter to aid an LLM in solving reasoning tasks,

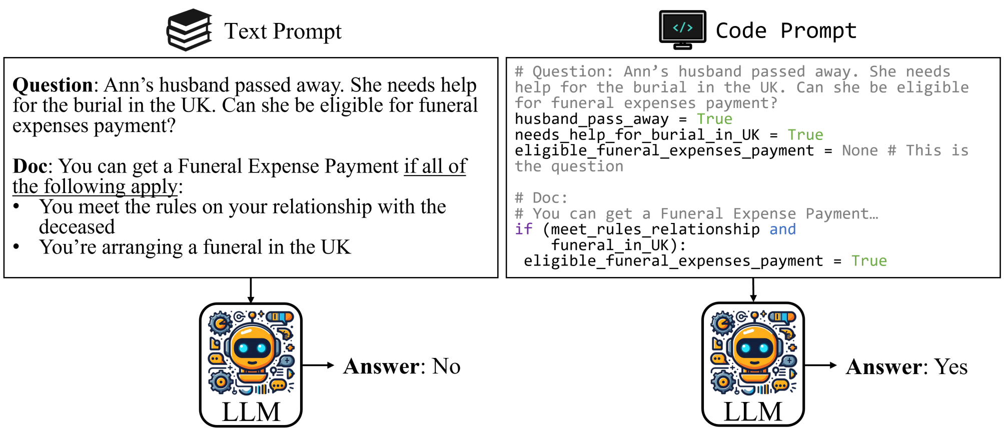

Figure 2: `Code prompting` converts natural language descriptions into code to be solved with a large language model. The figure shows a transformed instance from the ConditionalQA dataset.

i.e., they *run* code and therefore have a program state with variables and its values. However, we do not employ any external symbolic reasoner, and we *do not run code*. We investigate the reasoning abilities of LLMs under different *input representations* (i.e., text and code). Our code prompts are not executed; they are simply read by the LLM and used to generate a natural language answer.

Some works suggest that code LLMs may possess superior reasoning abilities than text LLMs. Madaan et al. (2022) investigate whether code LLMs are superior at *structured* reasoning than text LLMs. They observe that code LLMs can generate graphs that link commonsense concepts better than text LLMs. Liu et al. (2023b) investigate code prompts in abductive and counterfactual reasoning tasks and report superior results than text prompts on `code-davinci` (Ouyang et al., 2022), a code LLM. However, code prompts exhibit mixed results on `text-davinci-002` (Ouyang et al., 2022), a text LLM. We attribute this to the fact that while this model includes some code in its pretraining corpus, it is not explicitly trained for code generation and, in general, performs poorly on code generation tasks (Chen et al., 2021a). Therefore, the effect of the input representation on the reasoning abilities of text+code LLMs remains unclear. Furthermore, the reasons behind the superior performance of code prompts in code LLMs also remain unclear. In our work, we aim to answer whether code prompts can elicit conditional reasoning abilities in text+code LLMs and the reasons behind this.

To the best of our knowledge, only the work of Hussain et al. (2023) investigates the conditional reasoning abilities of LLMs. However, they only analyze the abilities of text LLMs after training them on ConditionalQA (Sun et al., 2022).

3 Code Prompting

We posit that each LLM encodes a set of capabilities, such as mathematical, logical, or conditional reasoning. However, not all of them are used for every input instance, even if they would be useful. We hypothesize that the input representation plays a pivotal role in eliciting such capabilities. Prior works show that LLMs trained on a combination of text and code exhibit superior reasoning abilities (Kojima et al., 2022; OpenAI, 2023; Jiang et al., 2024, 2023). Therefore, we conjecture that a code representation of a natural language (NL) problem may trigger some of these reasoning abilities encoded in text+code LLMs. More formally, we wonder whether exists some space $\mathcal{S}$[2] with an associated function f that transforms a natural language problem $p \in \mathcal{N}$ into that space, such that, when prompting an LLM with the representation of p in such space yields better results according to some evaluation function σ.

$$\exists \mathcal{S}, f : \mathcal{N} \rightarrow \mathcal{S}, \sigma(LLM(f(p)) \geq \sigma(LLM(p))$$

We fix $\mathcal{S}$ to the programming language space and define *code prompts* $f(p)$ as prompts that model a natural language problem with code. We also define f as a prompt that transforms the NL text into code. $f(p)$ code follows the original NL text as much as possible. We use a simple structured code that contains the logical structure needed to

[2] Since the input of LLMs must be strings, $\mathcal{S}$ must be a set of all possible sentences constructed using some alphabet and grammar.

solve the problem, along with the original NL text as code comments. In particular, it creates variables for key entities in the question and documents and *if blocks* for each conditional statement in the documents. Figure 2 exemplifies this transformation and Appendix C provides more details of the code features. Lastly, we define *code prompting* as $LLM(f(p))$, a chain of prompts that i) transform the NL text into code, and ii) use this code to prompt the LLM to generate the answer in natural language. Figure 1 illustrates this pipeline.

It is important to note that the code is not executed *per se* and therefore, there is no program state. We simply prompt the LLM with the code and ask the LLM to generate a natural language answer based on the content of such code. This setup allows us to investigate the effect of the input representation on text+code LLMs.

4 Experimental Setup

4.1 Datasets

Throughout our experiments, we use three question-answering (QA) datasets for conditional reasoning: *ConditionalQA* (`CondQA`; Sun et al., 2022), a scenario-based question answering (QA) dataset, *BoardgameQA* (`BGQA`; Kazemi et al., 2023), a boardgame-base QA dataset with conflicting rules, and `ShARC` (Saeidi et al., 2018), a conversational QA dataset with natural language rules. Solving these datasets requires advanced conditional and compositional reasoning capabilities.

We focus on the QA task of `CondQA`. For `BGQA`, we focus on the *main* partition, which includes three subsets `BGQA-1`, `BGQA-2`, and `BGQA-3`, where the number indicates the reasoning hops needed to answer. Lastly, while `ShARC` encompasses dialogue generation, we aim to evaluate specific capabilities unrelated to conversational flow. Therefore, we isolated the QA pairs from the provided dialogues, resulting in a dataset where the model has to answer *yes*, *no*, or *not enough information*.[3] We include more details about the datasets in Appendix A, a formal definition of the prompts in Appendix B, and examples in Appendix M.

4.2 Models

We perform our study using text+code LLMs because of their ability to process text and code interchangeably. We do not employ code-only LLMs because their intended use does not include solving natural language tasks (Roziere et al., 2023), as required in our case. Similarly, we do not employ text-only LLMs because they cannot generate code. Furthermore, using text+code LLMs also allow us to eliminate the confouding effect of fine-tuning between model variants specialized for only text or code. To further argue our point, we conduct small experiments on CodeLLaMA (Roziere et al., 2023), the results of which we report in Appendix D.

We employ OpenAI's `gpt-35-turbo`, `Mixtral 8x7B` (Jiang et al., 2024), and `Mistral 7B` (Jiang et al., 2023). The use of these models allows us to investigate whether our hypothesis holds across all available sizes of text+code LLMs. We execute our prompts with in-context learning and provide one demonstration per class. More details on the LLM setup are provided in Appendix E.

4.3 Evaluation

We follow the evaluation metrics used in the original datasets. For `CondQA`, we report the F1 token overlap between the predicted answer and the label, while for `BGQA` and `ShARC`, we report the macro F1 score. We run the main experiments two times with different random seeds (0 and 1). We report the average and standard deviation performance across these runs. For the subsequent analyses of code prompts, we run each experiment once only on GPT 3.5 due to the inference costs.

5 Experiments

We devise a set of experiments to analyze and quantify whether the code representation of a natural language prompt (i.e., code prompts) elicits conditional reasoning abilities and why. We first compare the performance of the two prompting methods — *text prompts* and *code prompts* on three LLMs across three datasets (§5.1). We then conduct extensive ablation experiments on the dev set of the datasets with GPT 3.5, the best-performing and largest model, to understand the reason behind the performance gain from code prompting. In particular, we study whether *code syntax* or the implicit *text simplification* from the code translation is what improves performance (Section 5.2). We also check if the improvement is caused by the models merely being exposed to code within prompts and not necessarily the instances translated to code (Section 5.3). Furthermore, we show that code prompting is more *sample efficient* (Section 5.4)

[3] In the full task, *not enough information* would trigger another step in a pipeline to generate a follow-up question.

Model	Prompt	CondQA	ShARC	BGQA-1	BGQA-2	BGQA-3	ΔCP
Test Set							
GPT 3.5	Text	58.70	**62.95**	51.15	37.42	27.77	8.42
	Code	**60.60**	54.98	**58.67**	**55.56**	**50.29**	
Mixtral	Text	**48.17**	53.77	**56.38**	39.64	30.15	4.22
	Code	44.73	**59.06**	53.33	**47.39**	**44.72**	
Mistral	Text	**35.74**	43.60	47.40	48.78	47.86	2.74
	Code	33.28	**49.92**	**53.80**	**51.27**	**48.79**	
Dev Set							
GPT 3.5	Text	56.54 ± 0.08	$\mathbf{64.10 \pm 0.10}$	53.16 ± 1.67	33.71 ± 10.37	31.5 ± 13.39	9.84
	Code	$\mathbf{57.64 \pm 1.42}$	58.54 ± 1.22	$\mathbf{68.60 \pm 1.09}$	$\mathbf{55.85 \pm 4.06}$	$\mathbf{47.57 \pm 2.68}$	
Mixtral	Text	$\mathbf{46.60 \pm 0.99}$	55.71 ± 2.51	58.31 ± 1.77	36.77 ± 0.09	32.06 ± 1.79	2.51
	Code	40.88 ± 1.84	$\mathbf{58.96 \pm 3.44}$	57.94 ± 5.52	$\mathbf{45.32 \pm 0.54}$	$\mathbf{38.90 \pm 7.33}$	
Mistral	Text	28.84 ± 0.02	37.56 ± 0.78	47.61 ± 0.92	47.29 ± 1.97	46.56 ± 2.92	5.10
	Code	28.26 ± 10.03	$\mathbf{53.42 \pm 0.93}$	$\mathbf{52.21 \pm 0.95}$	$\mathbf{54.27 \pm 1.42}$	45.22 ± 10.75	

Table 1: Comparison (F1 score) of text prompt and code prompts. All results use one demonstration per class. ΔCP = Code Prompt - Text Prompt, i.e., the average performance gain from code prompts across all datasets.

when compared to text prompting and that models prompted with code exhibit superior *state tracking* capabilities (Section 5.5). Lastly, we conduct a human evaluation that confirms the faithfulness of the generated code in Appendix H.

5.1 Code Prompting Improves over Text Prompting

Table 1 shows the model performance on the development and test sets. Code prompts outperform text prompts in the majority of cases on the test set (11 out of 15). This trend holds true across models, with each achieving peak performance through code prompts for most datasets (i.e., GPT-3.5 in 4/5, Mixtral in 3/5, Mistral in 4/5). Notably, code prompts consistently surpass text prompts on `BGQA-2` and `BGQA-3`, the most reasoning-intensive datasets (see Appendix A), for all models. This is particularly evident for GPT-3.5, where gains exceed 18 points. Conversely, the advantage is narrower on `CondQA`, where the linguistic dimension plays the biggest role (see Appendix A). This suggests that code prompts elicit conditional reasoning abilities and are most suited for reasoning-intensive tasks. Furthermore, in the cases where text prompts are superior, their average gains are only 4.23. In contrast, code prompts lead to significantly larger mean gains of 8.53 in the cases where they are superior. Additionally, an experiment with Phi-2, a small language model, reveals a substantial 15-point performance improvement using code prompts (see Appendix G).

To shed light on the performance gains driven by code prompts, we delve into the confusion matrices (attached in Appendix L) and discover that text prompts in Mistral predict "not enough information" much less than code prompts for `BGQA`. This is particularly noticeable in `BGQA-1`, where text prompts do not predict a single "not enough information," while code prompts do. On the other hand, text prompts in GPT 3.5 and Mixtral overpredict "not enough information" on `BGQA` and `ShARC`, leading to a low number of true positives for the conclusive answers. We hypothesize that this model hesitation could stem from the *alignment tax* (Ouyang et al., 2022) of *reinforcement learning from human feedback* models. This potential barrier may be alleviated by code prompts because they indicate to the model the variable that answers the question and instruct the model to track the entailment status of variables within the given code.

These consistent and substantial gains from code prompts are obtained despite a straightforward transformation of text prompts, which does not incorporate new information, as shown in Figure 2. This finding strongly suggests that code possesses specific characteristics that effectively elicit conditional reasoning abilities within text+code LLMs.

5.2 Code Syntax Elicits Reasoning Abilities

We now want to delve into the source of the performance gains observed when using code prompting. We investigate whether these improvements stem from the simplification of text into premises facilitated by code, effectively reducing the task to a form of semantic inference within the *linguis-*

tic dimension, or if there are inherent properties of code syntax that contribute to enhanced performance. To investigate this, we devise experiments with prompts that represent the intermediate states between natural language and code.

I. **Atomic Statements.** Inspired by Min et al. (2023), we transform each NL sentence[4] into a sequence of *atomic statements*, which we then append to the original sentence. In this way, the atomic statements can be seen as defining variables for each key entity in the text. Hence, this new prompt would resemble code but without control flow and in natural language form. The prompt retains access to the original instance text (i.e., no loss of information) but is also augmented by simplified sentences in the form of atomic statements. This setup allows us to investigate whether the *simplicity* of the input triggers improves reasoning abilities, regardless of the text and code syntax.

II. **Back-Translated Code.** In our second experiment, we investigate whether the *semantics* of the code statements and not the code *syntax* are the reason behind the performance boost. For this purpose, we back-transform the code prompts into NL such that the reasoning statements (i.e., the *if* conditions) are clearly and concisely stated in natural language. Specifically, we map every variable into the format *Key entity: variable without snake case.* For instance, the variable *husband_pass_away* from Figure 2 would be back-transformed as *Key entity: husband pass away.* To transform the *if* statements, we create a translation prompt by providing four demonstrations. These demonstrations simply translate the conditional statements within the code-formatted instance back into natural language. We also translate the variables in the same manner. This makes the back-translated text as close as possible to the code text. We provide examples of this in Table 11 from Appendix J.

Results. The results[5] in Table 2 show that (1) prompting with atomic statements does not reach the performance of code prompts, and (2) mapping back from code to NL results in a performance drop compared to code prompts. These findings suggest that code prompts enhance LLM performance beyond mere text simplification. This conclusion is supported by the observation that these alternative text simplification approaches, despite offering similar semantics to code prompts, fail to replicate the performance gains observed with code prompts. Therefore, these results imply that specific syntactic features embedded within code directly contribute to performance improvement.

[4]We only transform the *facts* in BGQA since transforming the *rules* into atomic statements as well yields worse results.

[5]We do not conduct ablation tests on ShARC because, as explained in Section 5, these ablations aim to understand why code prompts outperform text prompts using the highest performing model.

Dataset	**Δ Atomic St.**	**Δ Code $\rightarrow$ NL**
CondQA	-2.66	-4.72
BGQA-1	-4.37	-1.43
BGQA-2	-8.72	-5.39
BGQA-3	-19.26	-3.68

Table 2: Performance gap of *atomic statements* and *back-translated code* when compared to code prompts using GPT 3.5. Results from the dev set of each dataset.

Lastly, our evaluation on BGQA-3 reveals a significantly larger performance decline when using atomic statements compared to back-translated code. This disparity likely stems from the dataset's inherent structure. The method we employ for generating atomic statements (Min et al., 2023) was specifically designed for general text formats like Wikipedia pages. However, BGQA is a logic-based dataset where input "facts" are already presented as minimally informative statements, deviating from the typical structure of general documents. As a result, generating atomic statements from these sentences can unintentionally disrupt the sentence structure, making it difficult to track the attributes of subjects and objects within the text. This observation is further supported by our results on CondQA, a dataset with longer documents, where atomic statements achieve higher performance than back-translated code.

5.3 Code Semantics are Important

Previously, we have shown that code syntax is necessary to elicit the reasoning abilities of text+code LLMs. Now, we aim to investigate which aspects of code are pivotal. In particular, we evaluate the impact of retaining the natural language text of the original instance within the code comments and the importance of the code semantics. To analyze the former, we have (1) removed the code comments that include the original natural language text from the input and evaluated the performance of the new prompts. To analyze the latter, we (2) perturbed the code to anonymize the variables and functions, as well as (3) added random code whose semantics are completely irrelevant to the original natural

Prompt	CQA	CQA-YN	BG_1	BG_2	BG_3
Anonym.	−1.62	−2.90	−6.60	−4.80	−4.00
Random	−3.40	−2.67	−7.40	−9.20	−9.80
- Comments	N.A.	−14.02	−16.70	−16.20	−5.20

Table 3: Performance gap to code prompts for each code perturbation. cQA stands for CondQA, CQA-YN for the partition of CondQA with yes-no answers, BG for BGQA. Results reported on the dev set of each dataset.

language text. In the latter two cases, the code comments remain unmodified (examples illustrating them are provided in Table 12 from Appendix J). Since `CondQA` includes span answers and removing the NL text would make it impossible for the model to generate the span, we only report performance on the yes-no answers partition (`CondQA-YN`).

Table 3 shows that removing the NL text in the code comments yields a performance drop of 14.02 points on `CondQA` and a performance drop between 16.7 and 5.2 on `BGQA`. This significant and consistent decrease in all datasets confirms that retaining NL text in comments is vital for the LLM to understand the input problem.

Effect of Code Perturbations. Code perturbations (*anonymous code* and *random code*) confirm the importance of code semantics in eliciting reasoning abilities. When we use anonymized code, we observe a performance reduction of almost 2 points on `CondQA` and a decrease between 6.6 and 4 in `BGQA`. The decrease is even larger when the code is randomized, with drops of more than 3 points on `CondQA` and between 7.4 and 9.8 on `BGQA`. This more pronounced drop is expected since the semantics and logic of the code mismatch the NL text, whereas anonymous code maintains the same logic on both NL and code. Furthermore, we also observe that the performance drop of random code prompts is similar to that of text prompts (Table 1) on `CondQA` and `BGQA-1`. This can be interpreted as the model being able to identify the irrelevance of the code to the text. Hence, the model disregards the code to solely focus on the code comments (i.e., the natural language text). This could be possible thanks to the provided demonstrations, which show answers that only refer to the natural language text.

These results confirm that code alone does not trigger reasoning abilities, and instead, the combination of code that represents the original natural language instance and the NL text is able to unlock the potential of LLMs.

5.4 Code Prompts are More Sample-Efficient at Eliciting Reasoning Abilities

Given our observations that code prompts trigger conditional reasoning abilities better than text prompts, it is natural to ask the follow-up question: are code prompts also more *sample-efficient* than text prompts? To answer this, we evaluate how the overall performance of GPT 3.5 changes with respect to the number of demonstrations for the two prompting methods.

Figure 3 shows that when we only provide one demonstration per class (i.e., answer type in our datasets), the performance gap is the largest across all datasets. As expected, this gap decreases when we provide more demonstrations. Moreover, we also observe that code prompts with only one demonstration per class even outperform text prompts with three demonstrations per class, which further shows the sample efficiency of code prompts. These results indicate that code prompts trigger conditional reasoning more efficiently than text prompts on GPT 3.5, and this is one of the reasons for its superior performance.

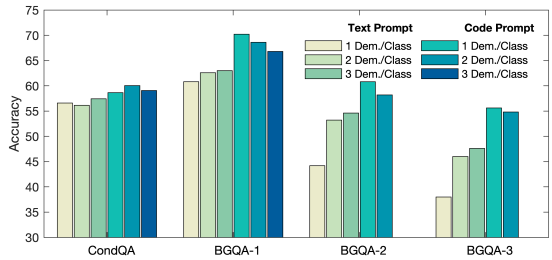

Figure 3: Performance comparison of GPT 3.5 between text (green) and code prompts (blue) using 1, 2, and 3 demonstrations per class. Results reported on dev sets.

5.5 Code Prompts Improve Variable Tracking in LLMs

We hypothesize that one of the reasons for the superior performance of code prompting is an improved ability to identify and track the states of key variables or concepts. This hypothesis is based on the intuition that, for natural language in general, local context is the most important part to generate the next token (Khandelwal et al., 2018; Sun et al., 2021). However, generating code is often more challenging because code frequently refers to previously defined functions and variables, which can be dozens or even hundreds of lines apart. This resembles multi-hop reasoning, where the model may need to reference a key entity dozens of lines

Dataset	Correct Ans. Text	Correct Ans. Code	Incorrect Ans. Text	Incorrect Ans. Code
CondQA	71.08	**4.39**	60.79	**11.39**
BGQA-1	39.33	**8.84**	51.65	**22.12**
BGQA-2	44.79	**15.04**	52.54	**24.75**
BGQA-3	54.01	**14.21**	52.13	**16.98**

Table 4: Comparison of the percentage of memory errors made by GPT 3.5. For each dataset, we separately compute memory errors for the instances where the model gives the correct and incorrect answers. Lower is better. Results from the dev set of each dataset.

before. Therefore, an improved ability to *look for distant co-references* caused by training on code can be beneficial for multi-hop reasoning, which is also needed to solve our datasets.

To test our hypothesis, we devise the following experiment. Firstly, we define *reasoning step* as each output sentence split by "\n." After generating each reasoning step, we *stop* the model generation and query about all key entities defined in the input prompt. In the case of text prompts, we query the model whether the given facts are true or not, and for code prompts, we query for the value of the (boolean) variables. In all cases, the model only has to generate *True*, *False*, a *string*, or *unknown*. Then, we compare the percentage of errors in text and code prompts. This number represents the *memory errors* committed by the model. The more memory errors there are, the more difficult it is for the model to track and remember entities/variables. We provide further details on how we extracted the key entities to ask for, how we identified the reasoning steps in the chain of thought used to stop the model for conducting the probes, and examples of the prompt probes in Appendix K and its Table 13.

Does Generated Text reflect Model Beliefs? As the generated text may not be faithful to the internal beliefs of the model (Lyu et al., 2023), we first test the validity of this experiment as a proxy metric of the internal belief of the model. To do this, we compare the memory error percentage of the prompting methods in instances where the model solves (i.e., *correct instances*) and does not solve (i.e., *incorrect instances*) the question. If incorrect instances yield a higher memory error, this would indicate that the model struggles more to remember the variable states on those instances, which in turn would make it more likely to fail when conducting the reasoning process. Therefore, our probes would be a proxy metric of the internal belief of the model.

Table 4 shows the results of this comparison. We observe that all prompting methods in all datasets consistently make more memory mistakes on incorrect instances than on correct instances, with the exception of text prompts on `CondQA`. However, the memory error in this case is significantly high, which may suggest that the model is not able to track entities correctly in either case. Therefore, we can use this experiment as a proxy measure of the memory of the model.

Code Prompting improves State Tracking. From Table 4, we further observe that Text Prompts make significantly more memory errors than code prompts on all datasets. Specifically, the gap is consistently more than 30% with peaks on `CondQA` (66.69%) and `BGQA-3` (39.8%). Therefore, this experiment empirically confirms our hypothesis that code prompts improve state tracking of the key entities and variables when compared to text prompts.

6 Conclusions and Future Work

This work demonstrates that the code representation of a natural language task (i.e., code prompts) can elicit reasoning abilities in large language models (LLMs) of text and code. These code prompts contain the original natural language (NL) formulation as a code comment and code that formulates the logic of the text. To create these code prompts, we use in-context learning to teach an LLM how to conduct such a transformation automatically. Through multiple experiments on three LLMs and three datasets, we show that code prompts trigger conditional reasoning abilities, with large performance gains w.r.t. text prompts (up to 22.52 percentage points on GPT 3.5, 7.75 on Mixtral, and 16.78 Mistral). Our experiments reveal that even simple code can be beneficial as long as it closely follows the semantics of the NL text and is accompanied by the original NL text. We also show that code prompts are more sample-efficient than text prompts and that their performance boost stems from their superior ability to identify and track the state of key variables or entities, a central aspect of the logical dimension of semantic inference.

In our future work, we plan to extend to a wider range of reasoning abilities, such as multi-hop reasoning, to understand the capacity and generalizability of code prompting. We also plan to investigate how pretraining on text, code, and text+code affects the triggering of LLMs' reasoning abilities.

Limitations

Transforming a natural language problem into code requires an intermediate step that raises the cost of the whole pipeline. However, this mapping is not a complicated task, as even the smallest models we considered were able to perform it successfully in an in-context learning setup. Therefore, we believe it would be possible to train a small generative model to do it instead of using a large language model. In this way, we could minimize the cost of using code prompts without affecting its performance.

We only ran the experiments on the dev set with two different random seeds due to the costs of running large language models and because we prioritized experimenting on multiple models and datasets. Nevertheless, the results of all models exhibit similar patterns, which confirms the representativeness of our results. Also, we conduct the ablations only on GPT 3.5, the best-performing and largest model. However, confirming that the findings from these ablations also hold on the smaller models would be interesting.

This work focuses only on analyzing the effects of code representations for natural language tasks. However, it could be possible that other input representation spaces also elicit reasoning abilities. We limit the scope of this work to only the space of simple structured languages (more details on Appendix C) because prior research suggests that pretraining on code improves the reasoning abilities of LLMs, but it might be possible that certain natural languages such as German or Chinese, or certain types of programming languages, such as declarative or logical, also elicit certain abilities. Similarly, we do not conduct experiments on multiple code generation methods because our goal is to analyze whether the mere change of the representation can elicit reasoning abilities and not an analysis of the best coding style.

Our tasks require instruction following abilities, so we do not conduct comparisons of base vs. chat models. Future work could investigate whether instruction tuning has an impact on the LLMs' abilities to understand code.

Lastly, we conduct our experiments on data in English. Analyzing whether our findings hold true in other languages would be interesting. However, the lack of conditional reasoning datasets in other languages would make this study difficult.

Ethics and Broader Impact Statement

Our work aims to improve the reasoning abilities of LLMs. The use of code prompts may alsosimplify the explainability of the model responses since we can inspect the entailment status of the variables. We hope these results contribute to enhancing the trustworthiness and safety of LLMs. Nevertheless, every development may pose some risks. In our case, the improvement of the reasoning abilities in LLMs may utilized by malicious actors to propagate more persuasive disinformation.

References

Elana Cabria and Bernardo Magnini. 2014. Decomposing semantic inference. *Linguistic Issues in Language Technology*, 9.

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. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021b. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. 2023. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*.

Syed-Amad Hussain, Parag Pravin Dakle, SaiKrishna Rallabandi, and Preethi Raghavan. 2023. Towards leveraging llms for conditional qa. *arXiv preprint arXiv:2312.01143*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Mehran Kazemi, Quan Yuan, Deepti Bhatia, Najoung Kim, Xin Xu, Vaiva Imbrasaite, and Deepak Ramachandran. 2023. BoardgameQA: A dataset for natural language reasoning with contradictory information. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, pages 1–23.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 284–294, Melbourne, Australia. Association for Computational Linguistics.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. 2023. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2023a. Chain of code: Reasoning with a language model-augmented code emulator.

Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. 2023b. Starcoder: may the source be with you! *Transactions on Machine Learning Research*. Reproducibility Certification.

Hanmeng Liu, Jian Liu, Leyang Cui, Zhiyang Teng, Nan Duan, Ming Zhou, and Yue Zhang. 2023a. Logiqa 2.0—an improved dataset for logical reasoning in natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:2947–2962.

Jiacheng Liu, Skyler Hallinan, Ximing Lu, Pengfei He, Sean Welleck, Hannaneh Hajishirzi, and Yejin Choi. 2022a. Rainier: Reinforced knowledge introspector for commonsense question answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8958, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. 2022b. Generated knowledge prompting for commonsense reasoning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3154–3169, Dublin, Ireland. Association for Computational Linguistics.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3622–3628. International Joint Conferences on Artificial Intelligence Organization. Main track.

Xiao Liu, Da Yin, Chen Zhang, Yansong Feng, and Dongyan Zhao. 2023b. The magic of IF: Investigating causal reasoning abilities in large language models of code. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9009–9022, Toronto, Canada. Association for Computational Linguistics.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu

Dhabi, United Arab Emirates. Association for Computational Linguistics.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100, Singapore. Association for Computational Linguistics.

John Eric Nolt, Dennis Rohatyn, and Achille Varzi. 1988. Schaum's outline of logic. McGraw Hill Professional.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Liangming Pan, Xiaobao Wu, Xinyuan Lu, Anh Tuan Luu, William Yang Wang, Min-Yen Kan, and Preslav Nakov. 2023. Fact-checking complex claims with program-guided reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6981–7004, Toronto, Canada. Association for Computational Linguistics.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Marzieh Saeidi, Max Bartolo, Patrick Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. 2018. Interpretation of natural language rules in conversational machine reading. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2087–2097, Brussels, Belgium. Association for Computational Linguistics.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. 2019. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4506–4515, Hong Kong, China. Association for Computational Linguistics.

Haitian Sun, William Cohen, and Ruslan Salakhutdinov. 2022. ConditionalQA: A complex reading comprehension dataset with conditional answers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3627–3637, Dublin, Ireland. Association for Computational Linguistics.

Simeng Sun, Kalpesh Krishna, Andrew Mattarella-Micke, and Mohit Iyyer. 2021. Do long-range language models actually use long-range context? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 807–822, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Wenya Wang, Vivek Srikumar, Hannaneh Hajishirzi, and Noah A. Smith. 2023. Elaboration-generating commonsense question answering at scale. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1619–1635, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: Satisfiability-aided language models using declarative prompting. In *Proceedings of NeurIPS*, pages 1–33.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

A Datasets

ConditionalQA is a QA dataset where the answers are applicable under specific scenarios (i.e., conditional answers). Therefore, along with each question, the dataset provides a scenario that describes the background of the person posing such a question. Questions require multi-hop, compositional, and conditional logic over documents about public policies (e.g., the eligibility for a subsidy). Answers can be a span of the document, *yes*, and *no*. We use an oracle retriever to select the relevant passages to the question so that we can isolate the analysis of conditional reasoning abilities in LLMs from the retrieval component. The expected output is a chain of thought (CoT; Wei et al. 2022) followed by the final answer. To create the CoT, we use the annotated evidence sentences. We use an oracle retriever to retrieve the relevant passages to the question. This retriever is based on the sentences annotated as evidence for the answer (i.e., rationales). We concatenate all sections that include one rationale and use the resulting passage as input document.

`BoardgameQA` is a dataset that evaluates the ability to reason with contradictory information guided by preferences. For example, given a question about traveling abroad, information found online about regulations can be contradictory because rules may change over time. Answering questions in this dataset requires complex multi-hop reasoning with conditional, deductive, and compositional abilities. The domain of the problems is board games, which allows us to analyze the conditional reasoning abilities in a completely different domain from `CondQA`. `BGQA` is divided into multiple partitions focusing on different characteristics, such as the depth of the reasoning tree, the need for external information, etc. We focus on the *main* partition and its subpartitions (i.e., `BGQA-1`, `BGQA-2`, `BGQA-3`), where the number refers to the number of reasoning hops required to answer the question. This dataset also includes annotated chain-of-thoughts (CoT); therefore, we use their annotated input ("*example*") as the input prompt and their annotated CoT ("*proof*") as the expected output.

`ShARC` is a conversational QA dataset with natural language rules where most questions are underspecified. Therefore, the model may need to ask a follow-up question to know more about the background of the interlocutor to return an answer. The documents are of legal domain retrieved from the web pages of different governments and state agencies. Since this is a conversational QA and we are not interested in evaluating the conversational abilities of LLMs, we transform the task into regular QA, instead of conversational QA. To do this, the model must answer *yes*, *no*, or *not enough information* for each question. In the original task, *not enough information*, would lead to the generation of a follow-up question.

Complexity of the datasets. We analyze the complexity of the datasets by counting the percentage of reasoning operations (i.e., *if statements*) in the code prompt generated by GPT 3.5. This analysis shows that the most difficult dataset is `BGQA-3` with 21.58% of reasoning operations, followed by `BGQA-2` (16.99%), `CondQA` (14.66%), `BGQA-1` (10.55%), and lastly, `ShARC` (8.32%).

We also analyze the length of the documents of each dataset and find that `BGQA-3` has the longest documents with an average of 39 lines of code, followed by `CondQA` (38), `BGQA-2` (25), `ShARC` (22), and lastly `BGQA-1` (15). It is worth noting that the documents from `CondQA` are the short documents extracted with the oracle retriever described above, instead of the full documents, which are much longer (up to 9k tokens).

These two analyses suggest that `BGQA-3` and `BGQA-2` are the most reasoning-intensive datasets due to the high proportion of reasoning operations. In contrast, `CondQA` is the dataset where the linguistic dimension plays the biggest role because their documents are among the longest ones while it contains much less proportion of reasoning operations than the other datasets with similar document lengths.

Dataset sizes, licenses, and safety. The sizes and licenses of all the datasets used in this work are provided in Table 5. Our use of these datasets is consistent with their intended use, i.e., academic research to evaluate question-answering systems. As far as we know, these datasets do not contain any personal information or offensive content. Although we did not explicitly analyze this, the authors of these datasets did not mention including such content, and we did not observe such content during our use of the datasets. All these datasets are in English.

Dataset	Training	Dev	Test	License
CondQA	2338	285	804	BSD 2
BGQA-1	1000	500	1000	CC BY
BGQA-2	1000	500	1000	CC BY
BGQA-3	1000	500	1000	CC BY
ShARC	21890	2270	8276	CC-BY-SA-3.0

Table 5: Sizes of the datasets.

B Prompt Formulation

CONDQA. Firstly, we define the different components of a data point: scenario (S), question (Q), document (D), rationales (R), and answer (A). Then, the text prompt tp is defined as follows:

$$\begin{aligned} tp = \text{"Question:"} + S + Q + \text{"Document:"} + D \\ + \text{"Let's think step by step"} \end{aligned} \tag{1}$$

where $+$ represents the string concatenation operator. Then, the output format, to is:

$$to = R + \text{"Answer:"} + A \tag{2}$$

For code prompts, we first define a function $C : \mathbb{NL} \rightarrow \mathbb{C}$ that maps a natural language text into code as shown in Figure 2. Then, we define code prompt cp as follows:

$$\begin{aligned} cp = \text{"\#Question:"} + C(S) + C(Q) + \\ \text{"\#Document:"} + C(D) \\ + \text{"\#Let's think step by step"} \end{aligned} \tag{3}$$

Similarly, we define the output format, co, as:

$$co = R + \text{"\#Answer:"} + A \tag{4}$$

BGQA. Firstly, we define the components of a data point in this dataset: facts (F), rules (R), and questions (Q). Therefore, our text prompt is defined as follows[6]:

$$tp = F + R + Q \tag{5}$$

This dataset also provides the CoT that leads to the answer. Therefore, we use that CoT as the expected output.

For code prompts, we follow the same approach as with the previous dataset. We define code prompts, cp, as follows:

$$tp = C(F) + C(R) + C(Q) \tag{6}$$

with the output format (co) being:

$$co = C(cot) \tag{7}$$

[6] BGQA provides a field `example` with all the variables of the dataset concatenated with descriptions. We use this field as text prompt.

ShARC. Firstly, we define the components of a data point in this dataset: question (Q), scenario (S), document (D), and conversation history (H). Then, the text prompt tp is defined as follows:

$$\begin{aligned} tp = \text{"Question:"} + S + Q + \text{"Document:"} + D \\ + \text{"Conversation history:"} + H \\ + \text{"What is the answer to the question:"} + Q \end{aligned} \tag{8}$$

the output format is the answer label directly, which can be *yes*, *no*, or *not enough information*.

Similarly to the other datasets, we defined code prompts cp as follows:

$$\begin{aligned} tp = \text{"\#Question:"} + C(S) + C(Q) + \\ + \text{"\#Document:"} + C(D) \\ + \text{"\#Conversation history:"} + C(H) \\ + \text{"\#What is the answer to the question:"} + C(Q) \end{aligned} \tag{9}$$

Lastly, the output format is the answer label directly, which in this case are *True*, *False*, or *None*.

C Coding Features

To generate code as close as possible to the NL text, we use a programming language based on a simplification of Python. We only use boolean variables or variables that contain lists of strings. Variables follow the snake case naming convention. We also employ *if statements* to model conditional reasoning, but we do not use loops, functions, or classes. We create a code comment with the original NL text for each input sentence, and right after the code comment, we generate the code that represents the semantics of that sentence. However, we do not enforce the generated code to be a runnable script.

D Code-only LLMs

Although our work focuses on text+code LLMs because they are the only type of LLMs whose intended use includes natural language and coding tasks, we conduct a small experiment on `Code Llama` (Roziere et al., 2023), a code-only LLM. It is important to note that their authors advise against using this model on natural language tasks because their intended use is in code generation tasks only. Table 6 shows the results of `Code Llama` on our datasets. Firstly, we can observe that code prompts perform significantly worse than text prompts on CondQA and ShARC despite being a code LLM. We can attribute this to the nature of these datasets and the intended use of the model. These datasets require a strong comprehension of natural language documents and dialogues and answering natural language questions about them. This is far from the intended use of the model (i.e., generating code). Furthermore, `CondQA` requires generating a natural language answer that is a span of the document. The use of code to generate a natural language span of a document is also far from the fine-tuning tasks of this model. This would explain why the code representation is worse than the text representation. It is particularly interesting to see the results on ShARC. After manually inspecting the outputs, we observe that `Code Llama` can successfully generate the code corresponding to the natural language input. However, when it is prompted with such code and the question variable, the model does not generate the value of the variable (i.e., `true`, `false`, or `none`). Instead, it generates `\n`. The reasons behind this remain unclear and would require further investigation, which is out of the scope of this paper.

However, we observe a different behavior on `BGQA`. In this dataset, code prompts outperform text prompts. We attribute this to the high alignment with the first-order logic of this dataset, which makes it closer to the intended use of the model.

Nevertheless, it is important to note that these results are not intended to be comprehensive enough to conclude that code LLMs or `Code Llama` can or cannot solve natural language tasks, which is out of the scope of this work. Instead, they simply seem to confirm the warnings of the authors of `Code Llama`, i.e., this model is not intended for natural language tasks.

Model	**Text Prompt**	**Code Prompt**
CondQA	31.58	26.34
ShARC	58.33	18.62
BGQA1	44.38	44.78
BGQA2	44.59	49.41
BGQA3	40.88	46.44

Table 6: Text and code prompts results in `Code Llama 7B - Instruct` with one demonstration.

E LLM Setup

The exact models we used are the following: gpt-3.5-16k-0613 for `CondQA` and `BGQA`. For `ShARC`, since the documents are shorter, we used GPT-3.5-0301 due to the lower costs. In both cases, we run the models through the Azure AI service. We also use Mixtral 8x7B with 4-bit quantization for all the datasets using one Nvidia A100 in our own server. Lastly, we use Mistral 7B v0.1 for `CondQA` and `BGQA`. However, this model yields very poor results on `ShARC`, so we use the *instruct-v0.2* variant to be able to make a fair comparison between text and code prompts on this dataset using Mistral 7B. We use fp16 quantization for the Mistral 7B experiments and run them on our own server with one Nvidia A100.

All of our prompting methods are implemented using the Langchain library.[7] We set the decoding temperature to zero and use greedy sampling to make the outputs deterministic. For each experiment, we use a random sample from the training set as demonstrations. The LLM generating the code for code prompts is the same one as the one running the code to generate the final answer. We evaluate each model and prompt in the dev set of each dataset with two random seeds. Since the demonstrations are selected randomly, the seed determines them. The seed that yields the best performance on the dev set is then used for the final evaluation on the test set.

The number of demonstrations used to translate the documents into code is specified in Table 7. Note that this number differs from the number of demonstrations used to generate the answer, which is always three.

We use chain of thoughts (CoT) based on the provided annotations of the datasets. We do not use advanced CoT methods for text prompts because our aim is to quantify how much improvement we can get by transforming the natural language CoT into code syntax, and therefore, the natural language text and code must be *as close as possible*. The use of advanced CoT methods would also be reflected in the code syntax, making the experimental setup more complicated without providing better insights.

The best random seeds found (and consequently used for the test set evaluation) are described in Table 8 and Table 9.

F Costs

Running a data instance from ConditionalQA with gpt-3.5-16k-0613 using code prompts costs $0.04

[7] https://github.com/langchain-ai/langchain

Dataset	GPT	Mixtral	Mistral
CondQA	4	4	4
ShARC	5	4	4
BGQA-1	4	3	3
BGQA-2	4	3	4
BGQA-3	4	3	4

Table 7: Number of demonstrations for code translations. Note this is not the number of demonstrations to generate the answer.

Dataset	GPT	Mixtral	Mistral
CondQA	0	0	0
ShARC	0	1	1
BGQA-1	1	0	1
BGQA-2	1	0	0
BGQA-3	0	1	0

Table 8: Best seeds for code prompts

while with text prompts $0.01. On BoardgameQA-depth 3 (i.e., the partition with the most expensive prompts), with the same model, the costs per question are $0.02 and $0.03 for text and code prompts, respectively. Lastly, on ShaRC, using gpt-3.5-0301, the costs per question are $0.0006 and $0.005 for text and code prompts, respectively.

G Results on Small LMs with Short Context Window

We have shown the effectiveness of code prompting in the most popular sizes of LLMs in table 1 from section 5.1. However, it is becoming increasingly popular the development of small language models (sLMs) due to their cheaper inferece cost and higher token thoughput (Gunasekar et al., 2023). Therefore, we have conducted a preliminary experiment with Phi-2[8], a text+code model of 2.7B parameters on `BGQA-1` to show that our prompting methodology also holds in sLMs. As we can show on table 10, code prompting yields a remarkable performance boost of 15 points. However, due to the limited context window of Phi-2, it is not straightforward to conduct in-context learning on our other datasets.

[8] https://huggingface.co/microsoft/phi-2

Dataset	GPT	Mixtral	Mistral
CondQA	0	1	0
ShARC	0	0	0
BGQA-1	1	0	1
BGQA-2	0	1	1
BGQA-3	0	1	0

Table 9: Best seeds for text prompts

Prompt	BGQA-1
Text	33.20 ± 1.42
Code	$\mathbf{48.32 \pm 1.65}$

Table 10: Comparison of text prompt and code prompts with Phi-2 on the validation set. Metric: F1 score. One demonstration per class is provided.

H Human Analysis of the Generated Code

We conduct a small human evaluation to confirm the faithfulness of the generated code to the source natural language text. We evaluate the code generations of all our models on ten random samples from the dev set of `CondQA`, `ShARC`, and `BGQA` (in particular, we use `BGQA-1` partition). We check for perfect translations, and for the failing cases, we analyze the errors.

`BGQA`. We observe perfect translations in all models for all the analyzed samples. We attribute this effectiveness to the close alignment between the natural language documents and first-order logic.

`ShARC`. We observe that `GPT 3.5` generates perfect translations in all cases except one. However, this case is a corner case where the document is irrelevant to the question, and therefore, there is no answer. Furthermore, the document is only one line. Consequently, the model does not generate code and simply keeps the text as a code comment. In the case of `Mixtral 8x7B`, we observed perfect code translations for 70% of the samples. One of the failing cases assings as the question variable a variable that is actually from the conversation history, no the quesiton. Another error case exhibits wrong value assingments to some variables. They should be `none`, but they are assinged `true` and `false`. The last case is the corner case explain above. As for `Mistral 7B`, we find that 60% of the analyzed samples have a perfect translation. In the remaining 40%, we observe three cases with no question variable and the same corner case as before. However, the semantics of the natural language text remain, thanks to the code comments.

`CondQA`. We observe that `GPT 3.5` generates a perfect translation in 8 out of the 10 cases. In these two cases with errors, we observe that most of the code is correct, but in both cases, one conditional statement is missed. The model directly generates the body of the if statement without the corresponding if. It is also worth noting that the code is of high quality, including data structures as ditionaries, generating code that explains tables, and also generates lists of strings. In the case of `Mixtral 8x7B`, we obtain perfect translations in 6 out of 10 cases. All the failing cases exhibit the same type of error: there is a variable statement without a prior if condition. It is worth noting that the code, in general, is of high quality, contains data structures such as dictionaries and even process tables. Lastly, in the case of `Mistral 7B`, we observed a bit worse results. Only 4 out of the 10 cases are perfect translations. In two cases, there is no code, and instead, the model only generated the original natural language text in code comments. We also observe one case where the first half of the text is correctly translated into code but the second half only contains the code comments representing the natural language text. We also observe one case where the code is correct, but the indentation is wrong; all code blocks are under the first if statement, which should not be like that. Lastly, we find two cases where the if statements do not contain an execution body. It is worth noting that even in the cases where the code is not perfect, the original semantics from the natural language remain untouched because they are preserved through code comments.

This analysis contributes to the analysis of the quantitative results shown in Section 5.1 by confirming that, in general, the translated code faithfully represents the semantics of the source natural language text.

I Atomic Statements

Original sentence: <p>Applying for the legal right to deal with someone's property, money and possessions (their estate) when they die is called applying for probate.</p> Atomic statements: Applying for the legal right is a process. The process is called 'applying for probate'. The legal right is to deal with someone's property, money, and possessions. The someone is a person who has died. The property, money, and possessions are collectively called the 'estate'.

J Examples of Code Ablations

An example of a back-translated code into natural language is provided in Table 11. We can observe in both examples that the resulting natural language (NL) text is extremely similar to the original code. In addition, in the second example (`BGQA`), *Rule2* is much simpler after the back-translation than its original description in NL.

Table 12 shows examples of the multiple code ablations we conducted in Section 5.3. Random code replaces the code with a piece of code from another data point. In this way, the semantics of the text and code mismatch while we keep the code syntactically correct.

K Variable Tracking Setup

Extracting key entities in BoardgameQA. This dataset provides a list of "*facts,*" which are short and concise sentences describing the state of a key entity. Therefore, we use them without alterations as the key entities to ask for.

Extracting key entities in ConditionalQA. This dataset provides a scenario describing the background information of the person posing the answer. Since this scenario is a free-form text, we follow (Min et al., 2023) to extract *atomic statements* and use them as the key entities to ask for.

Code Prompting variables . To probe the variable tracking abilities of code prompts, we use the variables defined in the "*facts*" and "*scenario*" of BoardgameQA and ConditionalQA, respectively.

Probing memory at different steps in the Chain-of-Thought. Inspired by Lanham et al. (2023), we truncate the Chain-of-Thought (CoT) at different completion states and probe the memory of the model. To break down the CoT, we split it by the character "\n", which usually represents the end of a reasoning step. This is possible because our in-context learning demonstrations follow this format.

Number of probes. For each dataset instance, we run $num\_facts \times num\_steps\_cot$ probes, which makes this experiment very costly. Thus, we aim to maximize the number of instances probed while keeping the costs down. To do so, we use a sample of 50 instances for each dataset partition of

BoardgameQA, except for Board3, where we used 20 instances ($\approx$ 700 probes) because of the cost of the experiment. Due to the length of the demonstrations of ConditionalQA and its impact on the costs, we sample five facts and three partial CoTs for each instance, yielding an upper-bound of 15 probes per instance, and run the probes for 30 instances for each dataset partition (i.e., correct and incorrect instances).

Prompt Probes. In all cases, we follow the following format: *Sys. Prompt; ICL Demonstrations; Input Instance; Partial CoT;* ***Probe***.

The probe for text and code prompts in BoardgameQA is: “Now, I want to ask you about the value of some key entities you used. Your answers must be ‘yes‘, ‘no‘, or ‘unknown‘. It is very important that you only write one word. Is it true that {fact}?”

The probe for text prompts in ConditionalQA is: “Now, I want to ask you about the value of some key entities you used. Your answers must be “True”, “False”, “unknown”, or a string. It is very important that you only write the exact value. From the speaker perspective, is it true that {fact}?”

The probe for code prompts in ConditionalQA is: “Now, I want to ask you about the value of some key entities you used. Your answers must be “True”, “False”, “unknown”, or a string. It is very important that you only write the exact value. What is the value of the variable {var}?” A real example is provided in Table 13.

L Confusion Matrices

Figure 4 shows the confusion matrices of all our models using text and code prompts for all the datasets except `CondQA`. We cannot include this one because it is a span-extraction task, not a classification task.

M Prompt Examples

Type	Text

Code	# You can apply to become the estate's administrator if you are 18 or over and you are the most 'entitled' inheritor of the deceased's estate. This is usually the deceased's closest living relative.
	`if applicant_age >= 18 and entitled_inheritor and closest_relative:`
	`  can_apply_estate_administrator = True`

Code → NL	You can apply to become the estate's administrator if you are 18 or over and you are the most 'entitled' inheritor of the deceased's estate. This is usually the deceased's closest living relative. **if you are 18 or over and you are the most entitled inheritor of the deceased's estate and you are the closest living relative, you can apply to become the estate's administrator**
Code	# Rule2: Be careful when something removes from the board one of the pieces of the dog and also becomes an enemy of the catfish because in this case it will surely not burn the warehouse of the mosquito (this may or may not be problematic) **rule2(something) = remove(something, piece_of(dog)) & enemy(something, catfish) => not burn(something, warehouse_of(mosquito))**
Code → NL	**Rule2: If something removes from the board one of the pieces of the dog and also becomes an enemy of the catfish, then it does not burn the warehouse of the mosquito**

Table 11: Example of a back-translation $\mathbb{NL} \rightarrow \mathbb{C}$ in ConditionalQA and BGQA-3. Text in bold represents the main modification.

Type	Text

Original Code	# To be eligible you must have left your country and be unable to go back because you fear persecution.
	`if left_country_and_fear_persecution:`
	`  eligible_for_asylum = True`

Anonymous Code	# To be eligible you must have left your country and be unable to go back because you fear persecution.
	`if var_1`
	`  var_2 = True`

Random Code	# To be eligible you must have left your country and be unable to go back because you fear persecution.
	`if value_of_property_gone_down_by_more_than_50:`
	`  eligible_to_claim = True`
	`  getting_housing_benefit = True`

Table 12: Examples code ablations.

Section	Role	Message
Problem instance	Human	**Question**: My brother and his wife are in prison for carrying out a large fraud scheme. Their 7 and 8 year old children have been living with me for the last 4 years. I want to become their Special Guardian to look after them permanently. How long will it be before I hear back from the court? **Document**: What is a special guardian You can apply to be a child's special guardian when they cannot live with their birth parents and adoption is not right for them. ... Answers can be "yes" or "no". Let's think step by step:
Partial CoT	AI	Within 10 days of receiving your application the court will send you a case number and a date for a meeting to set out: \n
Probe	Human	Now, I want to ask you about the value of some key entities you used. Your answers must be 'True', 'False', 'unknown', or a string. It is very important that you only write the exact value. From the speaker perspective, is it true that the children have been living with me for the last 4 years?
Probe	AI	True

Table 13: Variable Tracking Example. Underlined text represents the variable to probe. Partial CoT is not the complete answer. The generation was stopped, and only the first step was used in this probe.

System: You are a helpful assistant that answers questions given a document. Answers must be a short span of the document. You have to extract the span from the document. Do not write anything else. I will give you some examples first. ICL Demonstrations... Human: Question: My brother and his wife are in prison for carrying out a large fraud scheme. Their 7 and 8 year old children have been living with me for the last 4 years. I want to become their Special Guardian to look after them permanently. How long will it be before I hear back from the court? Document: What is a special guardian   You can apply to be a child's special guardian when they cannot live with their birth parents and adoption is not right for them.   You'll be responsible for looking after the child until they're 18 (unless the court takes your responsibility away earlier).   You'll make all day to day decisions about the child, for example schooling and medical treatment. You do not have to discuss these decisions with the birth parents.   You'll need to get the consent of everyone who has parental responsibility for the child before you make some important decisions, for example: - changing the child's surname   - putting the child up for adoption   - taking the child abroad for more than 3 months   - the child having surgery for reasons other than improving health, such as circumcision, sterilisation or cosmetic surgery   If you cannot get consent, you can ask the court to decide. Use the form 'Make an application in existing court proceedings related to children' (form C2).   After you apply   Within 10 days of receiving your application the court will send you a case number and a date for a meeting to set out: - a timetable for your case   - how it will be dealt with   This meeting is called a 'first directions hearing'.   You must go to all hearings you're told to unless the court excuses you. If you're not able to go, contact the court office. Answers must be a short span of the document. You have to extract the span from the document. Do not write anything else. Let's think step by step:

Table 14: Text prompt Example for ConditionalQA

```
System: You are a helpful assistant. Your task is to process a pseudo-code that describes a question and a document. You need to reason using that document and the comments to return the answers. Answers must be a short span of the document. You have to extract the span from the code comments. Do not write anything else. I will give you some examples first.
ICL Demonstrations...
Human: # Question: My brother and his wife are in prison for carrying out a large fraud scheme. Their 7 and 8 year old children have been living with me for the last 4 years. I want to become their Special Guardian to look after them permanently. How long will it be before I hear back from the court?
maximum_redundancy_pay = 16320
housing_standards_and_procedures_in_Northern_Ireland = True
ensure_vehicle_taxed_in_UK = True immigration_advisers_can_help_with_representation_at_tribunal = True
supply_protective_clothing_and_equipment = True
CBT_required_for_moped_and_motorcycle = True
court_response_time = None # This is the variable that answers the question
# <h1>What is a special guardian</h1>
# <p>You can apply to be a child's special guardian when they cannot live with their birth parents and adoption is not right for them.</p>
if attorneys_appointed_jointly:
all_attorneys_must_agree_to_make_decision = True
disability_or_severe_disability_element_of_working_tax_credit = True
mugging_without_physical_harm_emergency = True
# <p>You'll be responsible for looking after the child until they're 18 (unless the court takes your responsibility away earlier).</p>
work_temporarily_for_hirer = True
# <p>You'll make all day to day decisions about the child, for example schooling and medical treatment. You do not have to discuss these decisions with the birth parents.</p>
accounts_and_tax_returns_cover_financial_year = "1 June to 31 May"
employer_operating_PAYE = True
# <p>You'll need to get the consent of everyone who has parental responsibility for the child before you make some important decisions, for example:</p>
# <li>changing the child's surname</li>
# <li>putting the child up for adoption</li>
# <li>taking the child abroad for more than 3 months</li>
# <li>the child having surgery for reasons other than improving health, such as circumcision, sterilisation or cosmetic surgery</li>
managed_by_fit_and_proper_persons = True
check_court_order_for_authorization = True
considering_fostering = True
if not_connected_to_mains_sewer:
septic_tank_used = True
can_claim_tax_relief_if_taxed_twice = True
extra_support_for_disability = True
if operator_of_septic_tank_or_treatment_plant:
follow_general_binding_rules = True
# <p>If you cannot get consent, you can ask the court to decide. Use the form 'Make an application in existing court proceedings related to children' (form C2).</p>
appeals_decision_time = "several months"
if worker and informal_resolution_not_satisfactory:
formal_grievance_complaint_possible = True
time_limit_for_backdating_claims_services = 6
# <h1>After you apply</h1>
# <p>Within 10 days of receiving your application the court will send you a case number and a date for a meeting to set out:</p>
# <li>a timetable for your case</li>
# <li>how it will be dealt with</li>
# <p>This meeting is called a 'first directions hearing'.</p>
committee_recommendations_go_to_Prime_Minister = True
check_adviser_registration = True
meet_manning_levels = True
recognised_as_charity_or_CASC = True
apply_for_visa_for_other_reasons = True
debt_paid_off = True
if special_educational_needs_and_disabilities:
affects_behaviour_or_socialisation = True
# <p>You must go to all hearings you're told to unless the court excuses you. If you're not able to go, contact the court office.</p>
payslip_can_include_tax_code = True
VAT_zero_rate = 0
gas_equipment_installed_and_maintained_by_Gas_Safe_registered_engineer = True
# Question: My brother and his wife are in prison for carrying out a large fraud scheme. Their 7 and 8 year old children have been living with me for the last 4 years. I want to become their Special Guardian to look after them permanently. How long will it be before I hear back from the court?
# Answers must be a short span of the document. You have to extract the span from the code comments. Do not write anything else.
# Let's think step by step:
```

Table 15: Code Prompt Example for ConditionalQA

System: You are a question-answering system that solves the problem of reasoning with contradictory information guided by preferences over sources of information. You must explain your answers step by step.
ICL Demonstrations ...
Human: A few players are playing a boardgame
The current state of the game is as follows
The amberjack struggles to find food
And the rules of the game are as follows
Rule1: If the amberjack has difficulty to find food, then the amberjack removes from the board one of the pieces of the carp
Based on the game state and the rules and preferences, does the amberjack remove from the board one of the pieces of the carp?
AI:

Table 16: Text prompt Example for BGQA-1

System: You are a large language model of code that can interpret code. You are given a pseudo-code that resembles to first-order logic that models some scenario. You will be given a question and you have to answer it step by step. You can use a rule if and only if you know the antecedent of the rule.
ICL Demonstrations
Human: # A few players are playing a boardgame
# The rules of the game are as follows
# Rule1: If the amberjack has difficulty to find food, then the amberjack removes from the board one of the pieces of the carp.
rule1() = difficulty_finding_food(amberjack) => remove_piece(amberjack, carp)
# The current state of the game is as follows
# The amberjack struggles to find food.
difficulty_finding_food(amberjack) = True
# Based on the game state and the rules and preferences, does the amberjack remove from the board one of the pieces of the carp?
question = remove_piece(amberjack, carp)
AI:

Table 17: Code prompt Example for BGQA-1

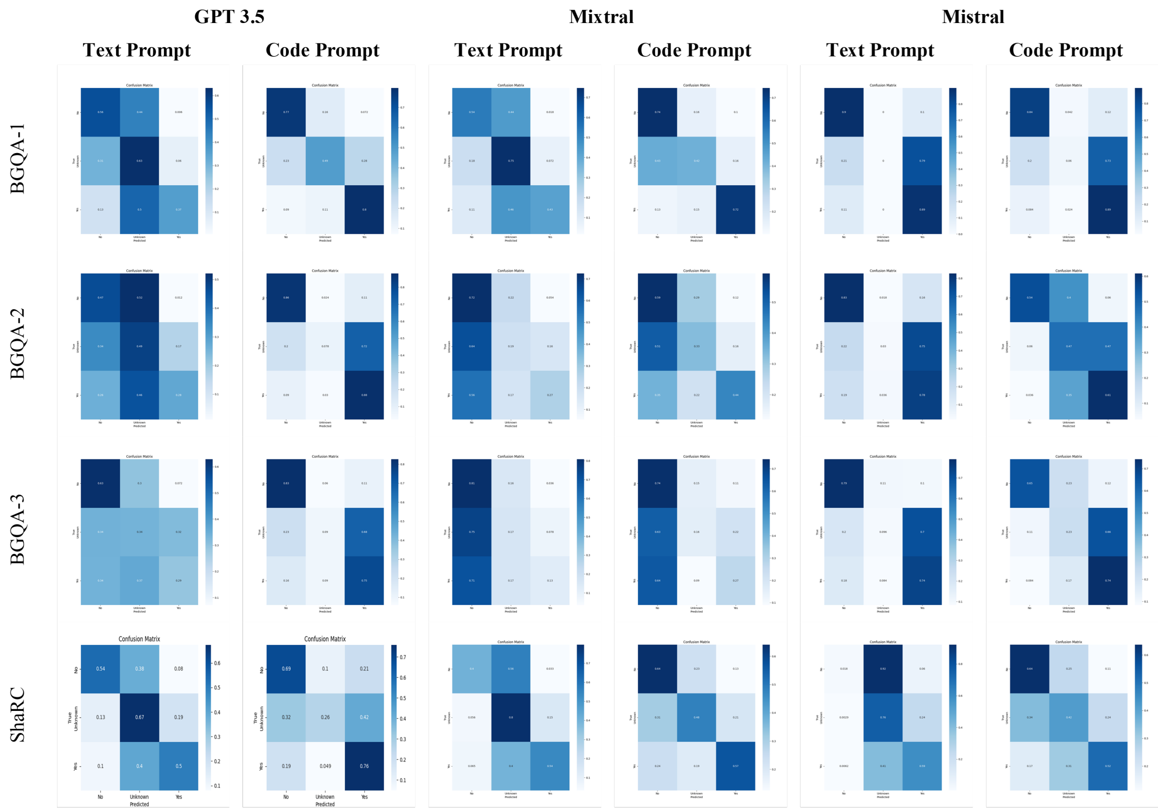

Figure 4: Confusion matrices of text and code prompts for each model on all datasets.

System: You are a question answering system that answers questions given a document and a conversation history. The conversation history gives information about the background of the person posing the question. You must answer ‘yes‘, ‘no‘, or ‘not enough information‘ to the question and nothing else.
ICL Demonstrations...
Human: Question: The item is not equipment for audio books or newspapers, and I’m not selling lifeboats or anything related to that. It’s for medicine and medicinal ingredients. Can I apply zero VAT to this item?
Document:
Items that qualify for the zero rate
You may be able to apply zero VAT when you sell the following to an eligible charity:
* equipment for making ‘talking’ books and newspapers
* lifeboats and associated equipment, including fuel
* medicine or ingredients for medicine
* resuscitation training models
Conversation history:
Q: Is it equipment for making ‘talking’ books and newspapers?
A: No
Q: Are you selling lifeboats and associated equipment, including fuel?
A: No
Q: Are you selling medicine or ingredients for medicine?
A: Yes
What is the answer to the question: Can I apply zero VAT to this item? You must answer ‘yes‘, ‘no‘, or ‘not enough information‘ to the question and nothing else.
AI:

Table 18: Text prompt Example for ShARC.

System: You are a question-answering system that answers questions based on a document, and conversation history. The text is pseudo-code that models the document and conversation history. You must run the code and update the value of the variable that answers the question. The values can be True, False, or None.
ICL Demonstrations...
Human:

```
# Question: # The item is not equipment for audio books or newspapers, and I'm not selling lifeboats or anything related to that. It's for medicine and medicinal ingredients. Can I apply zero VAT to this item?
equipment_for_audio_books_or_newspapers = False
selling_lifeboats_or_related_equipment = False
selling_medicine_or_ingredients_for_medicine = True
can_apply_zero_VAT = None # This is the variable that answers the question.
# Other variables needed for the document:
# Document:
## Items that qualify for the zero rate
# You may be able to apply zero VAT when you sell the following to an eligible charity:
# * equipment for making 'talking' books and newspapers
if equipment_for_audio_books_or_newspapers:
can_apply_zero_VAT = False
# * lifeboats and associated equipment, including fuel
if selling_lifeboats_or_related_equipment:
can_apply_zero_VAT = False
# * medicine or ingredients for medicine
if selling_medicine_or_ingredients_for_medicine:
can_apply_zero_VAT = True
# * resuscitation training models
resuscitation_training_models = None
can_apply_zero_VAT =
```

AI:

Table 19: Code prompt Example for ShARC.