Eliciting and Improving the Causal Reasoning Abilities of Large Language Models with Conditional Statements

Xiao Liu[1], Da Yin[2], Chen Zhang[1], Dongyan Zhao[1], and Yansong Feng[1]*

[1]Peking University, Wangxuan Institute of Computer Technology
lxlisa@pku.edu.cn, zhangch@pku.edu.cn, zhaody@pku.edu.cn,
fengyansong@pku.edu.cn
[2]University of California, Los Angeles
da.yin@cs.ucla.edu

Causal reasoning, the ability to identify cause-and-effect relationships, is crucial in human thinking. Although large language models (LLMs) succeed in many NLP tasks, it is still challenging for them to conduct complex causal reasoning like abductive reasoning and counterfactual reasoning. Complex causal structures are rarely expressed explicitly in the text, which could make learning them challenging for LLMs. Given the fact that programming code may express causal relations more often and explicitly with conditional statements like `if`*, we want to explore whether large language models of code (Code-LLMs) acquire better causal reasoning abilities, and whether code prompts better describe the causal structure than text prompts. Our experiments show that compared with general-purpose LLMs like* LLAMA-2 *and* GPT-3*, Code-LLMs like* CODELLAMA *and* CODEX *are significantly better in causal reasoning. Code prompts not only work well for Code-LLMs, but also help improve the performance of most general-purpose LLMs. To understand why code prompts are effective, we intervene on the prompts from different aspects, and discover that the programming structure is crucial in code prompt design, while models are more robust towards format perturbations. We further explore whether exposing models with more code with conditional statements aids in enhancing causal reasoning abilities. We finetune LLMs on such code corpus, and find their performance improves when prompted with either code prompts or text prompts.*[1]

* Yansong Feng is the corresponding author.

Action Editor: Tal Linzen. Submission received: 26 June 2024; revised version received: 10 December 2024; accepted for publication: 11 December 2024.

https://doi.org/10.1162/coli_a_00548

1 Code and data are available at `https://github.com/xxxiaol/magic-if`. This article is a substantially extended and revised version of Liu et al. (2023), which appears in *Findings of the Association for Computational Linguistics: ACL 2023*.

1. Introduction

Humans rely heavily on the capacity for *causal reasoning* (Sloman 2005; Hagmayer et al. 2007). People understand the observed facts, predict future events, and speculate about what might have happened if things had been different with the help of their causal reasoning skills. For instance, when we go home and find a mess, we probably want to investigate the cause. If we determine that a bird flew into the house, we may then consider whether the mess could have been avoided if we had closed the window.

Although large language models (LLMs) demonstrate great language understanding and generation abilities, it is still challenging for them to perform complex causal reasoning such as in the example above. Powerful LLMs are able to understand single cause-and-effect relations (Brown et al. 2020; Wang et al. 2021b), like *a man losing his balance* causes him to *fall*. However, when it comes to more complex causal structures involving multiple events and alternative branches (like *close the window* or not), LLMs perform much inferior to humans (Bhagavatula et al. 2019; Qin et al. 2019). In this article, we consider two challenging causal reasoning tasks: abductive reasoning and counterfactual reasoning. Abductive reasoning requires models to generate a plausible reason for the *ending* while being consistent with the *premise*. Counterfactual reasoning asks what will occur in the *counterfactual branch*. Causal relationships between events in these tasks are shown in Figure 1.

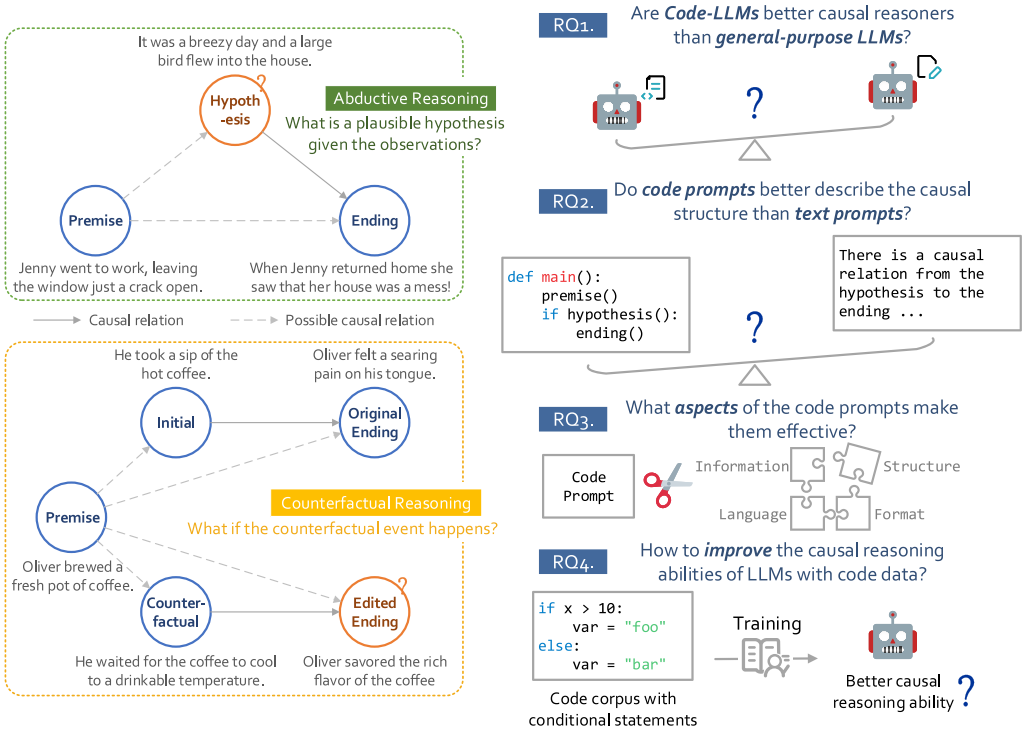


Figure 1
Overview of the tasks and research questions. Left: Causal relationships between events in abductive reasoning and counterfactual reasoning tasks. Right: Research questions discussed in this work, involving how to elicit and how to improve the causal reasoning abilities of LLMs.

```
# task: generate a plausible explanatory hypothesis given the premise
    and the ending. The hypothesis should be less than 20 words.
def main():
    premise()
    if hypothesis():
        ending()
def premise():
    # Scott loved his trumpet.
def ending():
    # Scott's dad bought him a new one for his birthday.
def hypothesis():
    # He wore it down from so much use.
```

Figure 2
Example code prompt of abductive reasoning.

A potential difficulty for LLMs to learn complex causal structures is that they are rarely expressed explicitly in the text. News articles or narratives may contain multiple events with causal relationships, like an incident and a chain of consequences. However, these events are often written chronologically, and it is hard to distinguish the real causal relations from many chronological relations without further annotation. Branches are expressed more rarely in text, except for the multibranching storytelling style (Nisi and Haahr 2006). On the other hand, causal relations are exhibited more commonly in code. Conditional statements like `if` direct the computer to execute certain commands, provided a condition is met. This explicitly demonstrates the causal relationship between the *condition block* and the *execution block*. Code can also express branching with `elif` or `switch` statements, and the nesting feature enables code to describe more complex structures.[2]

This motivates us to utilize code models in natural language causal reasoning. Recently, large language models of code (Code-LLMs) are receiving increasing attention (Chen et al. 2021; Xu et al. 2022; Lai et al. 2023). They exhibit strong code generation performance, and their structured prediction abilities help complete structured natural language tasks like argument graph generation (Madaan et al. 2022) and event argument extraction (Wang, Li, and Ji 2022). Pre-trained on code with abundant causal expressions, Code-LLMs may also have gained better causal reasoning abilities. This leads to our first research question, *are Code-LLMs better causal reasoners than general-purpose LLMs?* We are also interested in whether a broader range of models besides Code-LLMs benefit from the utilization of code prompts in causal reasoning. Code prompts have been shown to help general-purpose LLMs on tasks like conditional question answering (Puerto et al. 2024) and agent reasoning (Wang et al. 2024), due to their strengths in clearly describing the task structure and tracking the control flow. In the second research question, we investigate *whether code prompts better describe the causal structure than text prompts*.

We conduct experiments on the unsupervised abductive reasoning and counterfactual reasoning tasks. To generate task outputs, we design code prompts like Figure 2 to

2 Although conditional statements like *if* are also widely used in natural languages, they do not necessarily imply causal relations (Cummins et al. 1991; Weidenfeld, Oberauer, and Hörnig 2005; van Rooij and Schulz 2019). For example, the statement *if the water freezes, then the temperature is below zero degrees Celsius* does not mean that *the water freezes* causes the temperature to be *below zero degrees Celsius*. This makes it difficult to learn causal relations from conditional statements in text.

clearly represent the causal structures of the tasks. For events that have a definite causal relation, we organize them with `if` statements. For branches with different conditional events, we use `if-elif` structures. And for other events, we place them in chronological order. We experiment with a wide range of LLMs, including general-purpose LLMs like MIXTRAL and GEMINI, and Code-LLMs like CODELLAMA and CODEX. To compare general-purpose LLMs and Code-LLMs without interference like model structure and training methods, we carefully select pairs of LLMs like <LLAMA-2, CODELLAMA>, and <GPT-3, CODEX>. They share the same structure and only differ in the ratio of text/code training corpus. Results show that Code-LLMs perform much better than corresponding general-purpose LLMs. We further compare the performance of code prompts and text prompts, and find that code prompts work better for all the Code-LLMs and most general-purpose LLMs, indicating the effectiveness of code prompts.

To better understand *why code prompts are effective*, we break down the prompts and analyze the influence of different aspects, including the information provided in the prompts, the programming structure, the prompt format, and the programming language used in the prompts. Results demonstrate that LLMs are very sensitive to the *programming structure* (specifically, the conditional statements), while being more robust towards format perturbations and programming language changes.

Given the observation that conditional statements are important in code prompts, we investigate *if they help in further improving the causal reasoning abilities of LLMs*. Specifically, we collect a code corpus of conditional statements, and finetune LLMs on this corpus. Experiments show that both the code corpus and the conditional statements in code contribute to model performance. The finetuned LLMs perform better on causal reasoning with code prompts, and more importantly, their performance also improves when prompted with text prompts. This demonstrates that finetuning with a code corpus of conditional statements not only enhances their understanding of codes, but also enhances the general ability of causal reasoning.

Our main contributions are as follows:

1. We design code prompts to tackle causal reasoning tasks, by leveraging conditional statements in code to represent causal structures.
2. We evaluate a wide range of LLMs with code prompts and text prompts on unsupervised abductive reasoning and counterfactual reasoning tasks. Experiments exhibit that Code-LLMs are better causal reasoners than general-purpose LLMs, and code prompts are more effective than text prompts for most models.
3. We break down the code prompts in detail and find that the programming structure largely influences the performance.
4. We show that simply finetuning LLMs on a code corpus of conditional statements could improve their causal reasoning abilities.

2. Related Work

2.1 Causal Reasoning

There is a growing interest in the NLP community to assess the causal reasoning abilities of models. The first line of work focuses on whether models can *extract the causal*

relations from text. The task of causal relation extraction asks models to identify cause-effect relations between pairs of entities in text (Beamer, Rozovskaya, and Girju 2008; Blanco, Castell, and Moldovan 2008; Yang, Han, and Poon 2022). Girju et al. (2007) and Hendrickx et al. (2010) annotate cause-effect relations on general-purpose text like Wikipedia, whereas Pyysalo et al. (2007) and Gurulingappa et al. (2012) evaluate models on specific domains like biology and medical science. The second line of work is interested in whether models can conduct *commonsense causal reasoning*. This requires models to have knowledge of causal relations in daily lives, and apply this knowledge to specific scenarios. Gao et al. (2023) and Kıcıman et al. (2023) analyze the mastery of causal knowledge like cause-effect pairs, and Zečević et al. (2023) explore how models learn the causal knowledge from training data. Roemmele, Bejan, and Gordon (2011) require models to predict the cause or effect of a given premise, while Bhagavatula et al. (2019) and Qin et al. (2019) assess models' abilities to apply the causal knowledge to more complex abductive reasoning and counterfactual reasoning scenarios. Recently, there are also works evaluating if models can conduct *formal causal inference* using existing principles and algorithms. Jin et al. (2023) investigate if models can infer causal relations from conditional statements, and Jin et al. (2024) and Liu et al. (2024) evaluate the utilization of causal inference skills. In this work, we mainly focus on commonsense causal reasoning, as this ability is important in human everyday life (Sloman 2005).

Diverse methods are proposed to equip models with causal reasoning abilities. To help models extract causal relations from text, Chang and Choi (2005) measure causality between words and phrases with statistical methods, and Li and Mao (2019) use semantic cues with neural networks. To guide models to conduct formal causal inference, Jin et al. (2023) finetune models on the specific task of correlation-to-causation inference, and Liu, Feng, and Chang (2024) break down the causal reasoning process into individual steps with prompting. To improve the commonsense causal reasoning abilities, Li, Chen, and Van Durme (2019) finetune LLMs on causal event corpus; Du et al. (2021) and Wang, Cheng, and Li (2022) augment LLMs with causal knowledge graphs. Zhang et al. (2022) apply causal inference methods like propensity score matching to obtain more accurate causal knowledge. Qin et al. (2020, 2022) and Chen et al. (2022) regard commonsense causal reasoning as conditional generation tasks, and design methods to generate texts that meet the given conditions. In contrast to them, we explore how to elicit the causal reasoning abilities acquired by LLMs during pre-training, and how to further improve their causal reasoning abilities with general training corpus like code.

2.2 Large Language Models of Code

Code-LLMs are created to improve LLMs' performance on code-related tasks such as code generation (Chen et al. 2021; Lai et al. 2023) and program repair (Monperrus 2018; Fan et al. 2023). They are LLMs that are specially designed to understand and work with code (Xu et al. 2022; Zheng et al. 2023). Initially, Hindle et al. (2016) train n-gram models to conduct code completion. Encoder models and encoder-decoder models are then used as architectures of code models (Feng et al. 2020; Wang et al. 2021a). Nowadays, decoder-only architectures have become the most prevalent choice (Roziere et al. 2023; Guo et al. 2024). A main difference between the training of Code-LLMs and general-purpose LLMs is the data selection. Although general-purpose LLMs use code data like GitHub and Stack Exchange in training (Touvron et al. 2023a; Jiang et al. 2024), code data makes up a larger portion of the training data of Code-LLMs, and covers various programming languages (Li et al. 2023; Guo et al. 2024). Besides the training data, Code-LLMs are also enhanced with specific abilities. Roziere et al. (2023) and Guo

et al. (2024) enhance models with the infilling ability through the Fill-in-the-Middle pretraining method (Bavarian et al. 2022). This helps models to predict the missing code block given the surrounding context. Several models also support long context understanding to deal with repository-level code (Roziere et al. 2023; Bai et al. 2023).

With the recent development of Code-LLMs, several studies attempt to solve natural language tasks with code models. They mainly focus on two areas: numerical reasoning and structured prediction. Gao et al. (2022), Chen et al. (2023a), and Wu et al. (2022) apply Code-LLMs to numerical reasoning. They generate programs with Code-LLMs and feed the programs into an external interpreter to derive the answer. Madaan et al. (2022), Hu et al. (2022), Wang, Li, and Ji (2022), and Kim, Schuster, and Toshniwal (2024) leverage the text-to-structure translation ability of Code-LLMs to perform structured prediction tasks, including script generation, entity state tracking, event argument extraction, and so on. Madaan et al. (2022) ask models to generate structures in the form of code, and convert the generated code into the task output format. Chen et al. (2023b) extend this line of works to visual structural knowledge extraction and apply Code-LLMs to depict visual structural information. In addition, Mueller et al. (2024) find that Code-LLMs are better in the in-context learning setting, using in-context examples to generalize more robustly, and Petty, van Steenkiste, and Linzen (2024) find that adding code in model training improves performance on compositional generalization tasks with structured outputs. Different from them, we leverage the causal reasoning ability of Code-LLMs, and ask them to generate natural language events given the causal structure.

2.3 Utilization of Code Formats in Reasoning

Compared with natural language, code has several advantages that make it suitable for reasoning tasks (Wang et al. 2024; Yang et al. 2024): (1) *Structure and Logic.* Code is well-organized with clear executable steps. This structure allows for reasoning based on defined procedures and logic. (2) *Modularity and Reusability.* Code breaks down complex problems into smaller, reusable functions. This makes it easier to follow the reasoning process and reuse steps for different tasks. (3) *Control Flow*. Code can handle conditional statements and loops. This allows for reasoning with conditions and repetitions, making it suitable for complex tasks. (4) *Composition.* Code supports calling multiple functions sequentially or nestedly. This helps to compose different abilities or tools in reasoning.

These features make code formats not only useful for the reasoning of Code-LLMs, but also benefit general-purpose LLMs. Bogin et al. (2023) leverage the advantage of code in describing structures. They use programming languages to describe domain-specific information such as types of entities, attributes, and methods. Models prompted with such structured domain-specific information show great improvement in semantic parsing. Wang et al. (2024) exploit the modularity and composition nature of code in agent reasoning. They propose the CodeAct framework, which uses Python code to organize all actions for agent-environment interaction. CodeAct exhibits strong performance on several agent reasoning benchmarks, especially when required to solve complex tasks with multiple tools. Puerto et al. (2024) apply code prompts to the task of conditional question answering, building on the strength of code in handling logical and control flows. They find the code syntax useful in eliciting the conditional reasoning abilities of general-purpose LLMs, and the performance boost originates from the advantage of identifying and tracking variable states with code. However, prompting LLMs in the format of code is not useful for every task. Zhang et al. (2023) evaluate GPT models on twelve various tasks, and find that code prompts perform worse than text

prompts on question answering and summarization tasks. This shows that the choice of code or text format should be determined by the nature of the task. In this work, we explore the specific task of causal reasoning, where the code format helps in clearly describing the structure and control flow of events.

LLMs can also benefit the code execution in reasoning. Code is executable, allowing models to use the outcome of the reasoning process. Chen et al. (2023a) design the Program-of-Thought prompting strategy. It asks models to generate a code program given a question, and regard the output of the code as the answer. Code execution also provides a concrete way to identify any errors and further improve the code. Ni et al. (2023) use the execution results to verify and re-rank the generated programs. Wang et al. (2023b, 2024) ask models to reflect on the previous plans based on feedback from code execution.

3. Preliminary

We formulate abductive reasoning and counterfactual reasoning as unsupervised learning tasks. They are conducted in the manner of zero-shot learning, requiring models to conduct reasoning based on the task descriptions.

3.1 Abductive Reasoning Task

Abductive reasoning, a form of inference that seeks the most plausible explanation for a set of observations (Peirce 1974), plays a critical role in human cognition. It allows us to navigate the world by drawing conclusions based on incomplete information, which is a skill central to language acquisition and discourse comprehension (Hobbs et al. 1993).

We explore the application of abductive reasoning within the framework of natural language processing. We focus specifically on Abductive Natural Language Generation (αNLG), a task proposed by Bhagavatula et al. (2019) that challenges NLP models to generate plausible explanations bridging two given observations. It involves generating a plausible hypothesis H given the observations: premise O_P and ending O_E. Formally, models are required to maximize the probability $P(H|O_P, O_E)$.

This task necessitates non-monotonic reasoning, requiring the model to consider not only the preceding information O_P but also the future context provided by the ending sentence O_E. The chronological order of these three events is $O_P \rightarrow H \rightarrow O_E$, and the hypothesis causes the ending to occur.

3.2 Counterfactual Reasoning Task

Counterfactual reasoning, the ability to contemplate alternative scenarios that diverge from observed realities, is important in human cognition (Epstude and Roese 2008). This capacity to explore "what if" possibilities extends across various disciplines, and has emerged as a promising avenue for understanding causal relationships and narrative coherence in NLP researches (Hobbs 2005; Son et al. 2017; Qin et al. 2019).

Counterfactual reasoning offers a unique approach to studying causality in narratives. By introducing a causal intervention—a change to the initial context of a story—we can observe the subsequent impact on the narrative's conclusion. We follow the formulation of counterfactual reasoning proposed by Qin et al. (2019). It aims to rewrite a story under a counterfactual condition. As in Figure 1, the input consists of four parts: the premise P, the initial context C, the original ending E, and the counterfactual context C'. C' contradicts the information presented in C, and this contradiction makes it

necessary to change the ending. Models are asked to generate the counterfactual ending E' that *minimally* modifies the original ending E and is coherent with the counterfactual context C'. The goal can be formulated as maximizing the function $f(E'|P, C, E, C') = P(E'|P, C') + \lambda \text{sim}(E', E)$, where sim measures the similarity between two events, and λ controls the trade-off between the two constraints.

Achieving minimal edits requires the model to possess a deep understanding of the core elements driving the narrative. This allows the model to differentiate between genuine causal relationships and spurious correlations, and how they are affected by counterfactual scenarios.

As the events of both tasks are originally collected through crowd-sourcing, we conduct a quality check of the events in Appendix A, and find that 94% of them are grammatically and semantically acceptable. We leave the refinement of the unacceptable events to future work.

4. Modeling Causal Structure with Code

We convert the input of causal reasoning tasks into the form of code prompts, given the strength of code in depicting the structure and control flows. We expect the prompts to meet two requirements: (1) clearly represent the causal relationships between events, and (2) as most LLMs are autoregressive, the target output should appear at the end of the prompts.

The first requirement is addressed with conditional statements. However, for the second, the target prediction is not always the last part of the conditional statements—for example, in abductive reasoning, we want to predict the hypothesis, which is the condition in the `if` structure. To address this, we uniformly use functions to represent events. As shown in Figure 2, the causal structure is described in the `main` function. All the event functions are listed afterward, leaving the target event function at the last.

4.1 Abductive Reasoning

In Figure 2, we regard the task definition of abductive reasoning as an instruction and place it as a comment at the beginning of the prompt. As different LLMs are accustomed to different response lengths, we hint models to restrict the length of the hypothesis event to 20 words, which is close to the typical length of events in the dataset we use.

The causal structure is represented in the `main` function with the execution flow: executing the premise, and if the hypothesis is met, executing the ending.[3] The content of each event is presented as a comment on its function. We represent events with comments because in real code, comments often summarize the main content of the function. The `premise` and `ending` functions are placed after the `main` function, and the `hypothesis` function is placed at the last, leaving for models to complete. The generation process stops with a line break.

4.2 Counterfactual Reasoning

The task of counterfactual reasoning involves more events, and the causal structure is more complex with branches. The causal relationships are represented with the `if-elif` structure, as shown in Figure 3. The premise P is executed first, and then if the initial context C is met, the original ending E is executed; otherwise, if the counterfactual

3 Although not entirely accurate, this approximates the actual underlying causal relationships.

```
# task: generate an ending with three sentences given the premise and
    the hypothesis. Each sentence should be less than 20 words.
def main():
    premise()
    if hypothesis_1():
        ending_1()
    elif hypothesis_2():
        # minimally revise ending_1
        ending_2()
def premise():
    # Janice was excited to bring cupcakes to her work for her birthday
    .
def hypothesis_1():
    # She worked all day on making the perfect frosting.

def hypothesis_2():
    # She completely rushed making the frosting.

def ending_1():
    # Each cupcake was truly a work of art.
    # Everyone at her work loved them.
    # Janice was thrilled and happy for the rest of the day.
    # end

def ending_2():
    # The frosting was a complete disaster.
    # Everyone at her work hated them.
    # Janice was sad and embarrassed for the rest of the day.
    # end
```

Figure 3
Example code prompt of counterfactual reasoning.

context C' is met, the counterfactual ending E' will be executed. For ease of exposition, we call the context `hypothesis` as well, being consistent with the former task. We number the two branches as the `hypothesis_1` and `hypothesis_2` functions, and they correspond to the ending functions `ending_1` and `ending_2`, respectively.

The task definition is put at the beginning of the prompt in the form of content, and we also prompt models to limit the sentence length to no more than 20 words. To instruct models to minimally modify the original ending, we place this requirement in the comment of the `main` function. The event contents are also written as comments for event functions. We use `# end` to mark the finish of the ending.

5. Evaluation

We conduct extensive experiments to evaluate the unsupervised causal reasoning abilities of various LLMs with both code and text prompts. In this section, we aim to answer the following research questions, from the aspects of models and prompts:

- RQ1. Are Code-LLMs better causal reasoners than general-purpose LLMs?
- RQ2. Do code prompts better describe the causal structure than text prompts?

5.1 Experimental Setup

Datasets. We experiment on the $\mathcal{ART}$ dataset (Bhagavatula et al. 2019) for the evaluation of abductive reasoning, and the TimeTravel dataset (Qin et al. 2019) for counterfactual reasoning.

$\mathcal{ART}$ consists of 3,561 test instances. The observations of premise and conclusion are collected from ROCStories (Mostafazadeh et al. 2016), a large corpus of five-sentence stories written by humans. The first sentence is regarded as the premise O_P and the last sentence is regarded as the ending O_E. The plausible hypotheses H are annotated by crowdsourced annotators, and an average of 4.02 hypotheses are collected for each instance.

TimeTravel is also built upon ROCStories, containing 1,871 test instances. The five-sentence stories of ROCStories are used as original stories. The first sentence is the premise P, the second sentence is the initial context C, and the last three sentences make up the original ending E. A group of crowdsourced workers is asked to write counterfactual contexts C' for the stories. Another group of workers is instructed to write the counterfactual endings E'. They make minimal edits to the original ending, aiming to make the narrative coherent. Three conditional endings are gathered for each instance.

Models. We experiment with two types of models, general-purpose LLMs and Code-LLMs. For general-purpose LLMs, we choose LLAMA-2 (Touvron et al. 2023b), QWEN1.5 (Bai et al. 2023), DEEPSEEK-LLM (Bi et al. 2024), MIXTRAL (Jiang et al. 2024), GEMINI (Team et al. 2023), and GPT-3 (Brown et al. 2020). Both open-source and closed-source models are considered. Among open-source models, we use the `7B-chat` version of Llama-2, QWEN1.5, and DEEPSEEK-LLM, and the `8×7B-instruct-v0.1` version of MIXTRAL. Among closed-source models, we use the APIs of `Gemini-Pro` and the `text-davinci-002` version of GPT-3. For Code-LLMs, we experiment with two open-source models, CODELLAMA (Roziere et al. 2023) and CODEQWEN1.5 (Bai et al. 2023), and a closed-source model, CODEX (Chen et al. 2021). The specific versions are `7B-instruct` for CODELLAMA, `7B-chat` for CODEQWEN1.5, and `code-davinci-002` for CODEX. We set the temperature to 0 and the maximum length of output tokens to 256 for all models during inference.

Among the models, we intentionally select three pairs of <general-purpose LLM, Code-LLM> that share the same structure: <LLAMA-2, CODELLAMA>, <QWEN1.5, CODEQWEN1.5>, and <GPT-3, CODEX>. The difference between models in a pair is the training corpus. CODELLAMA is initialized with LLAMA-2 and trained on a code-heavy dataset. CODEQWEN1.5 is built upon QWEN1.5 and pretrained on 3 trillion tokens of code-related data. GPT-3 (`text-davinci-002`) originates from CODEX (`code-davinci-002`) and is finetuned with instructions. Comparing performance of these models allows us to make comparisons without confounding factors like model structure and training strategies.

We also compare with previous unsupervised methods on the two tasks, including DELOREAN (Qin et al. 2020), COLD (Qin et al. 2022), and DIFFUSION (Li et al. 2022) on abductive reasoning; and CGMH (Miao et al. 2019), EDUCAT (Chen et al. 2022), DELOREAN, and COLD on counterfactual reasoning. All these methods except DIFFUSION use GPT-2 (Radford et al. 2019) as the base model, and the model size ranges from medium to XL.

Among them, DELOREAN and COLD are constraint-based models. They regard the task requirements as constraints (for example, the generated text should be consistent

with the premise, and coherent with the ending in the abductive reasoning task), and iteratively update text representation to meet the constraints. CGMH and EDUCAT are editing-based models targeted for counterfactual reasoning. They start from the original ending and edit it to meet the counterfactual context. DIFFUSION builds a controllable LM based on continuous diffusions to perform control tasks including abductive reasoning.

To validate if the designed methods work on LLMs, we also add a baseline of implementing DELOREAN with LLAMA-2. We do not report other methods on LLMs because they are slow in inference or require additional training.

Prompts. Besides code prompts described in Section 4, we design text prompts for comparison. Table 1 demonstrates the examples of text prompts. All the causal relations in Figure 1 are written in the text prompts, like *there is a causal relation from the hypothesis to the ending, and a possible causal relation from the premise to the ending.* Therefore the code and text prompts contain the same information. Considering that the description of causal relations may be difficult to understand for some models, we also try text prompts without the causal descriptions (removing the second paragraph of each prompt in Table 1), and report the higher performance for each model.

Evaluation Metrics. We use the following automatic evaluation metrics: $BLEU_4$ (Papineni et al. 2002), $ROUGE_L$ (Lin 2004), CIDEr (Vedantam, Lawrence Zitnick, and Parikh 2015), and BERTScore (Zhang et al. 2019) based on BERT-base for abductive reasoning; $BLEU_4$, $ROUGE_L$, and BERTScore for counterfactual reasoning. These are

Table 1
Example text prompts of abductive reasoning and counterfactual reasoning.

Abductive Reasoning
Generate a plausible explanatory hypothesis given the premise and the ending.
There is a causal relation from the hypothesis to the ending, and a possible causal relation from the premise to the ending.
The hypothesis should be less than 20 words.
Premise: Scott loved his trumpet.
Ending: Scott's dad bought him a new one for his birthday.
Hypothesis:
Counterfactual Reasoning
Given an original story and an intervening counterfactual event, the task is to minimally revise the story to make it compatible with the given counterfactual event.
There is a causal relation from the initial event to the original ending, and a causal relation from the counterfactual event to the new ending. There are also possible causal relations from the premise to the initial event, from the premise to the counterfactual event, from the premise to the original ending, and from the premise to the new ending.
The new ending should consist of three sentences, with no more than 20 words each.
Premise: Janice was excited to bring cupcakes to her work for her birthday.
Initial event: She worked all day on making the perfect frosting.
Original ending: Each cupcake was truly a work of art. Everyone at her work loved them. Janice was thrilled and happy for the rest of the day.
Counterfactual event: She completely rushed making the frosting.
New ending:

Table 2
Automatic evaluation results on abductive reasoning in the zero-shot setting. Numbers are in percentages (%). The best results are in **bold**, and the model-level best results are shaded in gray. Numbers in brackets are the performance difference between prompted with code prompts and text prompts.

Model	Size	$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
Task-Specific Methods					
DELOREAN	355M	1.6	19.1	7.9	41.7
COLD DECODING	1.5B	1.8	19.5	10.7	42.7
DIFFUSION	80M	7.1	28.3	30.7	–
DELOREAN (LLAMA-2)	7B	2.8	20.0	10.5	49.8
Prompting with Text Prompts					
LLAMA-2	7B	4.8	28.7	44.0	58.0
QWEN1.5	7B	6.4	31.0	57.8	60.1
DEEPSEEK-LLM	7B	10.7	36.8	69.7	63.5
MIXTRAL	8×7B	10.8	37.1	74.5	64.0
GEMINI	–	6.6	30.0	52.6	58.8
GPT-3	–	4.9	27.0	26.6	56.8
CODELLAMA	7B	5.6	31.0	49.9	59.8
CODEQWEN1.5	7B	4.9	29.9	51.4	58.8
CODEX	–	11.7	37.5	78.5	62.5
Prompting with Code Prompts					
LLAMA-2	7B	6.1 (1.3↑)	30.5 (1.8↑)	50.3 (6.3↑)	59.2 (1.2↑)
QWEN1.5	7B	7.1 (0.7↑)	31.9 (0.9↑)	56.9 (0.9↓)	60.6 (0.5↑)
DEEPSEEK-LLM	7B	8.4 (2.3↓)	34.6 (2.2↓)	67.9 (1.8↓)	62.0 (1.5↓)
MIXTRAL	8×7B	**13.7** (2.9↑)	**39.6** (2.5↑)	81.6 (7.1↑)	**65.5** (1.5↑)
GEMINI	–	13.5 (6.9↑)	38.1 (8.1↑)	80.8 (28.2↑)	64.2 (5.4↑)
GPT-3	–	6.7 (1.8↑)	31.1 (4.1↑)	46.2 (19.6↑)	59.9 (3.1↑)
CODELLAMA	7B	6.2 (0.6↑)	31.7 (0.7↑)	55.4 (5.5↑)	60.1 (0.3↑)
CODEQWEN1.5	7B	5.1 (0.2↑)	30.3 (0.4↑)	52.3 (0.9↑)	59.0 (0.2↑)
CODEX	–	**13.7** (2.0↑)	**39.6** (2.1↑)	**81.8** (3.3↑)	64.9 (2.4↑)

consistent with previous methods (Qin et al. 2020; Chen et al. 2022; Qin et al. 2022) on these tasks.[4]

5.2 Automatic Evaluation Results

Table 2 reports the automatic evaluation results on abductive reasoning in the zero-shot setting, and Table 3 reports the results on counterfactual reasoning. In general, prompting LLMs outperform previous task-specific methods, indicating the rich commonsense knowledge and strong reasoning capabilities of LLMs. Equipping LLAMA-2 with DELOREAN decoding does not work well, showing that instructing the LLM of the constraints is more effective than modifying its representations to meet the constraints in our tasks. The difference in abductive reasoning measured by CIDEr is the most

4 CIDEr is only used for abductive reasoning in previous works, and we stay consistent with them to better compare the results.

Table 3
Automatic evaluation results on counterfactual reasoning in the zero-shot setting. Numbers are in percentages (%). The best results are in **bold**, and the model-level best results are shaded in gray. Numbers in brackets are the performance difference between prompted with code prompts and text prompts.

Model	Size	$BLEU_4$	$ROUGE_L$	BERTScore
Task-Specific Unsupervised Methods				
DELOREAN	355M	21.4	40.7	63.4
CGMH	355M	41.3	–	73.8
EDUCAT	355M	44.1	–	74.1
DELOREAN (LLAMA-2)	7B	6.2	21.3	53.0
Prompting with Text Prompts				
LLAMA-2	7B	18.7	33.2	63.3
QWEN1.5	7B	0.0	6.5	9.3
DEEPSEEK-LLM	7B	44.3	50.9	72.5
MIXTRAL	8×7B	37.6	48.2	71.3
GEMINI	–	22.0	34.1	63.9
GPT-3	–	49.0	54.7	73.0
CODELLAMA	7B	57.2	62.8	79.1
CODEQWEN1.5	7B	41.4	50.0	72.4
CODEX	–	55.1	61.3	77.8
Prompting with Code Prompts				
LLAMA-2	7B	33.8 (15.1↑)	51.5 (18.3↑)	72.7 (9.4↑)
QWEN1.5	7B	14.0 (14.0↑)	28.8 (22.3↑)	60.8 (51.5↑)
DEEPSEEK-LLM	7B	60.9 (16.6↑)	64.2 (13.3↑)	79.6 (7.1↑)
MIXTRAL	8×7B	52.0 (14.4↑)	59.0 (10.8↑)	77.0 (5.7↑)
GEMINI	–	33.1 (11.1↑)	42.6 (8.5↑)	67.6 (3.7↑)
GPT-3	–	40.4 (8.6↓)	48.5 (6.2↓)	70.5 (2.5↓)
CODELLAMA	7B	59.7 (2.5↑)	63.9 (1.1↑)	79.7 (0.6↑)
CODEQWEN1.5	7B	41.6 (0.2↑)	53.5 (3.5↑)	74.2 (1.8↑)
CODEX	–	**66.8** (11.7↑)	**70.0** (8.7↑)	**82.5** (4.7↑)

drastic, as CIDEr amplifies the effect of rare and unique words compared to other lexical overlap metrics like BLEU and ROUGE.

Comparing Code-LLMs with their corresponding general-purpose LLMs, CODELLAMA and CODEX outperform LLAMA-2 and GPT-3 on both tasks and both prompt formats, with an average gain of 14% measured by BLEU. Although CODEQWEN1.5 is inferior to QWEN1.5 on abductive reasoning, it is much better than QWEN1.5 on counterfactual reasoning. These exhibit the strong causal reasoning abilities of Code-LLMs. Emphasizing the code data in the training corpus helps Code-LLMs to understand the causal relations. Although the GPT-3 (`text-davinci-002`) model is based on CODEX, its causal reasoning ability may be weakened during instruction tuning, which is a phenomenon called **alignment tax** (Ouyang et al. 2022).

As shown in Tables 2 and 3, code prompts are better than text prompts for all Code-LLMs and most general-purpose LLMs, except for DEEPSEEK-LLM and QWEN1.5 on one metric of abductive reasoning, and GPT-3 on counterfactual reasoning. Compared with text prompts, the performance of models prompted with code prompts is 5.1% better in BLEU and 5.3% better in BERTScore on average. The results indicate that

Table 4
Evaluation results of the original code prompts and syntactically valid code prompts (%).

Model	Prompt	$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
LLAMA-2	Original	6.1	30.5	50.3	59.2
	+ return	**6.3**	30.6	50.6	59.2
	+ pass	**6.3**	**30.7**	**51.4**	**59.4**
CODELLAMA	Original	**6.2**	**31.7**	**55.4**	**60.1**
	+ return	6.1	31.4	54.9	60.0
	+ pass	6.1	31.5	55.0	60.0
QWEN1.5	Original	**7.1**	**31.9**	**56.9**	**60.6**
	+ return	6.9	30.4	51.2	59.5
	+ pass	6.7	30.2	51.1	59.3

(a) Abductive reasoning.

Model	Prompt	$BLEU_4$	$ROUGE_L$	BERTScore
LLAMA-2	Original	33.8	51.5	72.7
	+ return	42.4	51.7	72.9
	+ pass	**44.2**	**53.1**	**73.7**
CODELLAMA	Original	59.7	63.9	79.7
	+ return	**68.3**	**71.3**	**83.4**
	+ pass	67.4	70.5	83.0
QWEN1.5	Original	14.0	28.8	60.8
	+ return	**17.1**	**30.8**	61.8
	+ pass	**17.1**	**30.8**	**61.9**

(b) Counterfactual reasoning.

describing the complex causal structure with code is clearer and easier for most models to understand.

Format Perturbations. We conduct additional experiments to analyze the performance fluctuations towards changes to prompt formats.

To investigate if changing the code prompts from pseudo-code to syntactically valid code will affect the results, we explore two simple settings: (1) add an empty `return` statement to the event functions, and (2) add a `pass` statement to the event functions. We experiment with three open-source LLMs: LLAMA-2, CODELLAMA, and QWEN1.5. The results are shown in Table 4. Overall, making the code prompts syntactically valid improves the performance. The performance on counterfactual reasoning improves consistently, and the performance on abductive reasoning is impaired for some models to a small degree. This demonstrates the potential to further improve the performance by designing syntactically valid code prompts.

We also apply format perturbations (Sclar et al. 2024) to text prompts in Appendix B, showing that the performance of code prompts is consistently better than text prompts.

One-shot Setting. We also conduct experiments in the one-shot setting to validate if the trends exist when models are provided with examples. Models are shown with one demonstration example in the in-context learning manner, and the example is identical for all the models and both prompt formats. The results are in Table 5.[5] All models

5 We do not have results of GPT-3 and CODEX due to the discontinuance of these two models.

Table 5
Evaluation results in the one-shot setting. Numbers are in percentages (%). The best results are in **bold**, and the model-level best results are shaded in gray. Numbers in brackets are the performance difference between prompted with code prompts and text prompts.

Model	Size	$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
Prompting with Text Prompts					
LLAMA-2	7B	7.3	31.5	54.2	60.6
QWEN1.5	7B	9.4	34.4	65.4	62.9
DEEPSEEK-LLM	7B	9.8	35.3	68.0	62.8
MIXTRAL	8×7B	13.1	38.6	79.2	65.0
GEMINI	–	13.6	37.1	79.3	63.7
CODELLAMA	7B	7.5	31.6	55.4	60.3
CODEQWEN1.5	7B	5.9	31.0	55.0	59.7
Prompting with Code Prompts					
LLAMA-2	7B	7.5 (0.2↑)	31.4 (0.1↓)	54.6 (0.4↑)	60.5 (0.1↓)
QWEN1.5	7B	8.2 (1.2↓)	32.1 (2.3↓)	56.4 (9.0↓)	60.9 (2.0↓)
DEEPSEEK-LLM	7B	10.8 (1.0↑)	36.9 (1.6↑)	71.9 (3.9↑)	63.9 (1.1↑)
MIXTRAL	8×7B	14.0 (0.9↑)	39.9 (1.3↑)	84.3 (5.1↑)	65.8 (0.8↑)
GEMINI	–	16.6 (3.0↑)	40.2 (3.1↑)	89.8 (10.5↑)	65.6 (1.9↑)
CODELLAMA	7B	7.8 (0.3↑)	33.2 (1.6↑)	60.7 (5.3↑)	61.2 (0.9↑)
CODEQWEN1.5	7B	6.3 (0.4↑)	31.6 (0.6↑)	58.0 (3.0↑)	60.0 (0.3↑)

(a) Abductive reasoning.

Model	Size	$BLEU_4$	$ROUGE_L$	BERTScore
Prompting with Text Prompts				
LLAMA-2	7B	49.9	58.2	76.5
QWEN1.5	7B	63.4	70.5	82.9
DEEPSEEK-LLM	7B	52.9	58.1	76.4
MIXTRAL	8×7B	53.6	60.1	77.8
GEMINI	–	41.5	49.4	71.5
CODELLAMA	7B	77.2	78.7	87.4
CODEQWEN1.5	7B	63.1	65.4	80.7
Prompting with Code Prompts				
LLAMA-2	7B	55.1 (5.2↑)	61.9 (3.7↑)	78.5 (2.0↑)
QWEN1.5	7B	55.0 (8.4↓)	61.7 (8.8↓)	78.3 (4.6↓)
DEEPSEEK-LLM	7B	69.6 (16.7↑)	72.4 (14.3↑)	84.2 (7.8↑)
MIXTRAL	8×7B	59.3 (5.7↑)	64.3 (4.2↑)	80.0 (2.2↑)
GEMINI	–	58.9 (17.4↑)	63.4 (14.0↑)	78.6 (7.1↑)
CODELLAMA	7B	**78.2** (1.0↑)	**79.7** (1.0↑)	**88.1** (0.7↑)
CODEQWEN1.5	7B	63.4 (0.3↑)	67.8 (2.4↑)	81.6 (0.9↑)

(b) Counterfactual reasoning.

perform better in the one-shot setting than in the zero-shot setting. Code-LLMs still outperform corresponding general-purpose LLMs in most settings, and code prompts are better than text prompts for most models, demonstrating that the advantage of Code-LLMs and code prompts is robust across different settings.

Table 6
Human evaluation results. Numbers are the win rates of models (%).

	CODEX (Code Prompt)	Tie	GPT-3 (Text Prompt)
Abductive Reasoning			
Coherence with Premise	**34%**	48.5%	17.5%
Coherence with Ending	**32%**	42.5%	25.5%
Overall Coherence	**40%**	38%	22%
Counterfactual Reasoning			
Coherence	**36.5%**	39.5%	24%
Preservation	**47.5%**	39.5%	13%

(a) Comparing CODEX (Code Prompt) and GPT-3 (Text Prompt).

	MIXTRAL (Code Prompt)	Tie	MIXTRAL (Text Prompt)
Abductive Reasoning			
Coherence with Premise	**20%**	68.5%	11.5%
Coherence with Ending	**22%**	60.5%	17.5%
Overall Coherence	**31.5%**	47.5%	21%
Counterfactual Reasoning			
Coherence	**23%**	55.5%	21.5%
Preservation	**51%**	38.5%	10.5%

(b) Comparing MIXTRAL (Code Prompt) and MIXTRAL (Text Prompt).

5.3 Human Evaluation

To verify if the automatic evaluation results are consistent with human judgment, we conduct pairwise comparisons between (a) CODEX with code prompts and GPT-3 with text prompts, and (b) MIXTRAL with code prompts and text prompts. Each comparison is conducted on 100 test examples. Annotators are asked to choose which output is better given the task requirements. They are not provided with the reference answers in the dataset. For abductive reasoning, the outputs are rated from three aspects: coherence with the premise, coherence with the ending, and the overall coherence. For counterfactual reasoning, the outputs are rated from coherence with the context and the extent of preserving the original ending. Each example is rated by at least two annotators, and the average inter-rater reliability is 0.63. More details of the human evaluation are in Appendix C.

The results are exhibited in Table 6. CODEX outperforms GPT-3 in all aspects. CODEX better considers the context in generation, as shown in the *coherence* aspects. It is also able to preserve the original content in counterfactual reasoning, as shown in the *preservation* aspect. Fixing the model to MIXTRAL, prompting with code prompts outperforms prompting with text prompts in all aspects. Code prompts help the model to generate outputs that are more coherent with the contexts, and improve the preservation of the original content substantially. These further demonstrate the effectiveness of Code-LLMs and code prompts.

5.4 Case Study

For a better insight into how models behave in the causal reasoning tasks, we conduct a case study by randomly selecting example cases for each task. Model generations for one example case are shown in Table 7, and two more cases are in Appendix D.

Table 7
Example cases of model generations. Generations that are not coherent with the context are labeled as red, and the ones that fail to preserve the original ending in counterfactual reasoning are labeled as blue.

Abductive Reasoning
Premise: Angie went to a cocktail party hosted by her best friend.
Ending: Angie decided to be quiet about what she overheard.
Hypothesis [LLAMA-2 (text prompt)]: Angie was afraid of being judged by her friend and others at the party.
Hypothesis [LLAMA-2 (code prompt)]: Angie suspected her best friend was involved in a scandal.
Hypothesis [QWEN1.5 (text prompt)]: Angie overheard a secret at the party.
Hypothesis [QWEN1.5 (code prompt)]: Angie overheard a secret that could ruin her friend's life.
Hypothesis [GPT-3 (text prompt)]: Angie decided to be quiet about what she overheard because she didn't want to start any drama with her best friend.
Hypothesis [GPT-3 (code prompt)]: Angie overheard her friend discussing a plan to murder her husband.
Counterfactual Reasoning
Premise: My grandpa took my brothers and I fishing in the summer.
Initial Event: We went to a small pond close to his house.
Original Ending: He set up our fishing poles and we began to fish. It was so boring when nobody caught any fish. We were disappointed to go home empty handed that day.
Counterfactual Event: We caught four huge trouts within the first hour.
Counterfactual Ending [LLAMA-2 (code prompt)]: He taught us how to tie a fishing lure. We caught a big fish and he was so proud of us. We had a great time fishing together that day.
Counterfactual Ending [CODE-LLAMA (code prompt)]: He set up our fishing poles and we began to fish. We were so excited when we caught our first fish. We were thrilled to go home with four fish that day.
Counterfactual Ending [QWEN1.5 (code prompt)]: We had a blast and caught a record-breaking number. We were overjoyed and couldn't wait to tell everyone. It was the most memorable fishing trip we've ever had.
Counterfactual Ending [CODEQWEN1.5 (code prompt)]: We were so excited to catch those trouts. They were so big and so beautiful. We were happy to share them with my brothers and my parents.
Counterfactual Ending [GPT-3 (text prompt)]: We were so excited to have caught four trouts within the first hour! We were proud to show our grandpa our catches.
Counterfactual Ending [CODEX (code prompt)]: He set up our fishing poles and we began to fish. It was so exciting when we caught four huge trouts within the first hour. We were happy to go home with four trouts that day.

For abductive reasoning, we showcase generations of the same models with text prompts and code prompts, and find that the generations with code prompts are more coherent with the context, while generations with text prompts sometimes cannot take into account the premise. For example, the hypothesis generated by LLAMA-2 with text prompts does not explain why *Angie was afraid of being judged by her best friend*, and *Angie decided to be quiet about what she overheard* appears abruptly in the generation of GPT-3 with text prompts.

For counterfactual reasoning, we showcase generations of general-purpose LLMs and Code-LLMs in pairs. For each model, we use the prompt form that works

better in the automatic evaluation. Counterfactual endings generated by most models are consistent with the premise and counterfactual event, but compared with Code-LLMs, general-purpose LLMs have more difficulties in preserving the original ending. Specifically, the output of QWEN1.5 is far away from the original ending, and the output of GPT-3 only has two sentences. In contrast, Code-LLMs better understand the relations between events. In addition to being coherent with the counterfactual events, the generated counterfactual endings also maintain the original endings to a large extent.

6. What Are Crucial in Code Prompts?

Code prompts are shown to be effective for both Code-LLMs and general-purpose LLMs. Naturally, we are interested in what makes the code prompts effective. To paint a better picture of the key points in the code prompts, we intervene on the prompts from four aspects and measure the influences of the interventions. Our goal is to answer the following research question in this section:

- RQ3. What aspects of the code prompts make them effective?

The four aspects we select are *information*, *structure*, *format*, and *language*. The former two, the prior information provided and the programming structure of functions, are content-related; the latter two, the code format and programming languages, are form-related. An ideal model should rely on the content and be insensitive to form perturbations.

6.1 Intervention Prompt Construction

Information. We study two types of prior information: task instructions and function names. In *No Instruction*, we remove the task instruction from the prompts. In *Function Name Perturbation*, we replace original function names with anonymous `functionX`. For example, we replace `premise()` and `hypothesis()` in Figure 2 with `functionA()` and `functionB()`, respectively. This eliminates the information in function names and only allows models to learn the event relations from programming structures. Examples of information intervened prompts are in Appendix Figure E.1.

Structure. The first way to intervene in the programming structure is to convert the conditional structures into sequential structures, referred to as *Sequential Structure*. The events are executed sequentially, like `premise()`, `hypothesis()`, `ending()` in abductive reasoning. In the second way called *Disruption*, we randomly disrupt the positions of the functions in the conditional structure—for instance, `if hypothesis(): ending()` can be disrupted into `if ending(): hypothesis()`. We also apply the function name perturbation in disruption to eliminate the impact of function names. Examples of structure intervened prompts are in Figure 4.

Format. We test two formats besides the original one: *Class* and *Print*. In *Class*, we convert the original code into a class. We define the programming structure in the `__init__` method, and move the event functions into the class. In *Print*, we represent

```
# task: generate a plausible explanatory hypothesis given the premise and the ending. The hypothesis should be less than 20 words.
def main():
    premise()
    hypothesis()
    ending()
def premise():
    # Scott loved his trumpet.
def ending():
    # Scott's dad bought him a new one for his birthday.
def hypothesis():
    #
```

(a) Example prompt of *Sequential Structure* intervention.

```
# task: generate a plausible explanatory hypothesis given the premise and the ending. The hypothesis should be less than 20 words.
def main():
    functionA()
    if functionB():
        functionC()
def functionA():
    # Scott loved his trumpet.
def functionB():
    # Scott's dad bought him a new one for his birthday.
def functionC():
    #
```

(b) Example prompt of *Disruption* intervention.

Figure 4
Examples of structure interventions to code prompts in abductive reasoning.

the content of events as a string and print it in the function body, like `def premise(): print(''Scott loved ...'')`. Examples of format intervened prompts are in Appendix Figure E.2.

Language. We also convert the original Python programs into two other programming languages, *Java* and *C*, to evaluate the influence of programming languages. Specifically, the language conversion is made automatically by CODEX with the instruction `# python to java/c`. Examples of language intervened prompts are in Appendix Figure E.3.

6.2 Intervention Results

We evaluate the influence of interventions on LLAMA-2 and CODEX. The results on abductive reasoning are in Table 8a, and the results on counterfactual reasoning are in Table 8b.

Generally, the absence of prior information causes a small decrease in results. Models do not rely on the task instruction at the beginning of the prompt. They can learn the relations between events from the code. Even if the instruction or function names are not provided, CODEX is able to perform causal reasoning based on conditional statements. LLAMA-2 suffers from a large drop when function names are not provided in counterfactual reasoning. This indicates that for models with weaker coding abilities, function names provide clues about the event relations. For example, `hypothesis_1` and

Table 8
Results of intervening the code prompts from different aspects (%).

		$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
	LLAMA-2	6.1	30.5	50.3	59.2
Information	No Instruction	6.2	29.5	46.3	57.9
	Function Name Perturbation	6.6	29.8	47.4	58.2
Structure	Sequential Structure	5.0	26.9	39.2	55.5
	Disruption	4.9	29.6	47.8	58.4
Format	Class	5.6	30.1	50.3	58.9
	Print	6.3	31.0	52.7	59.5
Language	Java	6.6	30.7	51.8	59.1
	C	5.8	30.2	49.4	58.9
	CODEX	13.7	39.6	81.8	64.9
Information	No Instruction	12.1	37.4	73.8	62.9
	Function Name Perturbation	15.1	39.1	77.8	64.6
Structure	Sequential Structure	9.6	36.8	72.0	63.5
	Disruption	7.9	30.3	49.8	58.5
Format	Class	16.0	41.0	87.4	65.8
	Print	13.8	39.4	82.0	65.0
Language	Java	**16.5**	**42.0**	**91.6**	**66.3**
	C	15.5	41.0	88.0	65.6

(a) Intervention results on abductive reasoning.

		$BLEU_4$	$ROUGE_L$	BERTScore
	LLAMA-2	33.8	51.5	72.7
Information	No Instruction	34.1	53.3	73.8
	Function Name Perturbation	13.8	28.1	60.0
Structure	Sequential Structure	21.8	41.5	66.9
	Disruption	3.9	19.8	53.4
Format	Class	43.6	53.3	73.7
	Print	38.5	48.8	71.4
Language	Java	41.9	50.5	72.2
	C	34.0	45.9	69.7
	CODEX	66.8	70.0	82.5
Information	No Instruction	55.4	60.1	77.0
	Function Name Perturbation	65.4	69.0	82.2
Structure	Sequential Structure	43.4	50.2	68.2
	Disruption	16.0	23.5	55.2
Format	Class	63.6	67.4	81.1
	Print	**73.3**	**74.7**	**85.3**
Language	Java	71.1	73.5	84.5
	C	71.9	74.2	85.0

(b) Intervention results on counterfactual reasoning.

`hypothesis_2` are different branches, and `hypothesis_1` and `ending_1` are related in one branch.

Changes in the programming structure have the largest negative impact among all aspects. Changing the conditional structure to a sequential structure leads the average performance to drop 10% on BLEU and 6% on BERTScore. Comparing *Function Name Perturbation* and *Disruption*, the alteration of two characters (swap `B` and `C` in `functionB` and `functionC`) results in a major drop of 17% BLEU on average. These demonstrate the conditional structure that reasonably depicts that the relations between events is crucial when models perform reasoning.

Compared with information and structure interventions, models are more robust towards format and language interventions. Settings like *Print* and *Java* are even better than the original one, revealing that the performance can be further improved with delicate prompt engineering.

7. Improving Causal Reasoning Abilities with Conditional Statements

In previous sections, we find that code prompts help to elicit the causal reasoning abilities of LLMs, and the programming structure of conditional statements largely contributes to the performance gain. In this section, we are interested in how these observations can aid in model training, answering the following research question:

- RQ4. How can we improve the causal reasoning abilities of LLMs with code data?

Specifically, we collect code corpus of conditional statements, finetune LLMs with the data, and compare their performance on causal reasoning tasks before and after finetuning.

7.1 Experimental Setup

Data Collection. We use an existing code corpus `CodeAlpaca-20k` (Chaudhary 2023) to construct our finetuning data. `CodeAlpaca-20k` contains 20,000 instruction-following data used for finetuning LLAMA-2 into the Code Alpaca model. The data is generated in the style of self-instruct (Wang et al. 2023a) and Alpaca (Taori et al. 2023), with modified prompts and seed tasks focusing on code-related tasks. It is worth noting that the code corpus concentrates on code generation and code editing, and is not directly related to causal reasoning.

To filter data with conditional statements, we ask ChatGPT to determine if the output code of each instance contains conditional statements. The prompt we use is *Does the following code contain conditional statements? Conditional statements are programming language commands for handling decisions, for example, if–then(–else) and switch statements. Please output with "yes" or "no" without explanation.* And the specific version of ChatGPT is `gpt-3.5-turbo-1106`. To verify the quality of predictions, we manually check 100 instances, and ChatGPT classifies 96% of them correctly. It misses three instances of one-sided `if` statements (without `else`), and hallucinates conditional statements in one instance.

This results in 4,085 instances with conditional statements. We organize the data in the format of ### *Instruction: [instruction]* ### *Output: [output code]*. If the instance also contains an input, the input is placed in the prompt after the instruction, as shown in the example of Table 9.

Implementation Details. We finetune three 7B models LLAMA-2, QWEN1.5, and DEEPSEEK-LLM on the filtered data. We train them for one epoch with a batch size of 128. We use the AdamW optimizer with a learning rate of 2e-5 and a warmup rate of 0.03. The max length is set to 512, which covers most of `CodeAlpaca` data. The hyperparameters batch size, number of epochs, and learning rate are selected with a grid search over the validation set of $\mathcal{ART}$ and TimeTravel, with more details in Appendix F.

Table 9
Example of training data with conditional statements.

User	### Instruction: Validate whether the string contains only alphabets or not. ### Input: myString = "abc123" ### Output:

Model	def check_string_alphabets(myString):
	  for character in myString:
	  if (not character.isalpha()):
	  return False
	  return True

Controlled Baselines. To disentangle the gains from conditional statements and the gains from broader code corpora, we design a baseline of finetuning models on a uniform sample of `CodeAlpaca`. The sample size and all the training implementations are the same with the models finetuned on conditional statements.

7.2 Results

The performance of models finetuned on the code corpus of conditional statements is shown in Table 10. All three models achieve performance gain on both tasks of abductive reasoning and counterfactual reasoning, indicating the effectiveness of finetuning on code corpus with conditional statements. Although the finetuning data are all

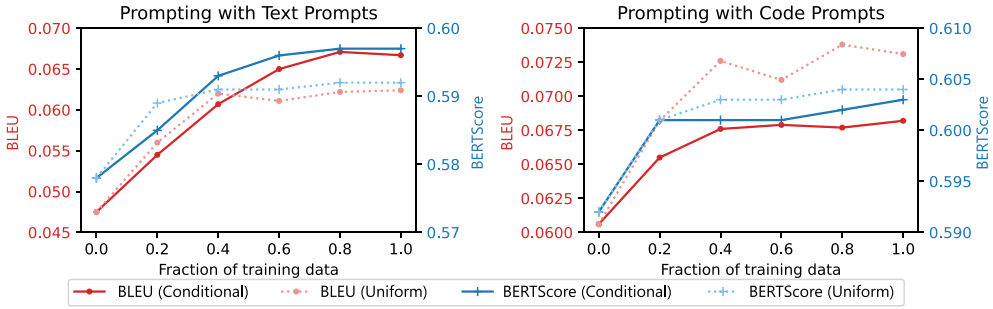
Table 10
Automatic evaluation results of models finetuned on conditional statements (%). The best results are in **bold**. Numbers in brackets are the performance difference compared to the model before finetuning.

Model	$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
Prompting with Text Prompts				
LLAMA-2	6.7 (1.9↑)	31.5 (2.8↑)	53.7 (9.7↑)	59.7 (1.7↑)
QWEN1.5	9.4 (3.0↑)	34.9 (3.9↑)	68.7 (10.9↑)	62.3 (2.2↑)
DEEPSEEK-LLM	11.3 (0.6↑)	37.4 (0.6↑)	73.9 (4.2↑)	63.9 (0.4↑)
Prompting with Code Prompts				
LLAMA-2	6.8 (0.7↑)	30.7 (0.2↑)	50.7 (0.4↑)	60.2 (1.0↑)
QWEN1.5	10.0 (2.9↑)	34.8 (2.9↑)	65.8 (8.9↑)	62.0 (1.4↑)
DEEPSEEK-LLM	10.2 (1.8↑)	36.3 (1.7↑)	71.3 (3.4↑)	63.2 (1.2↑)

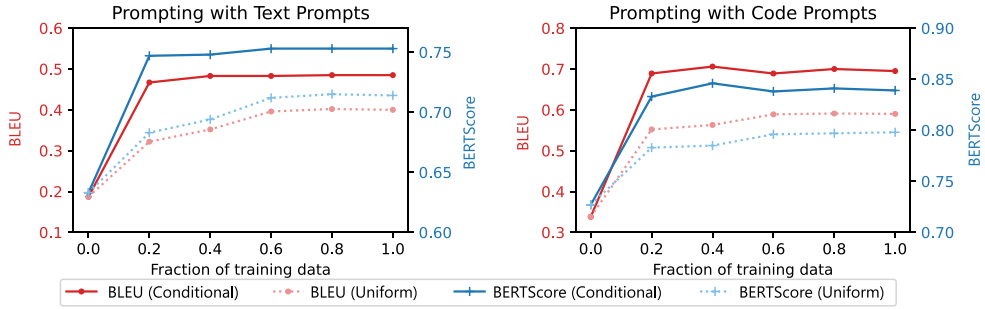
(a) Abductive reasoning.

Model	$BLEU_4$	$ROUGE_L$	BERTScore
Prompting with Text Prompts			
LLAMA-2	48.4 (29.7↑)	55.0 (21.8↑)	75.3 (12.0↑)
QWEN1.5	52.4 (52.4↑)	59.4 (52.9↑)	77.4 (68.1↑)
DEEPSEEK-LLM	51.7 (7.4↑)	56.4 (5.5↑)	75.5 (3.0↑)
Prompting with Code Prompts			
LLAMA-2	69.5 (35.7↑)	71.9 (20.4↑)	83.9 (11.2↑)
QWEN1.5	52.1 (38.1↑)	58.3 (29.5↑)	76.5 (15.7↑)
DEEPSEEK-LLM	75.7 (14.8↑)	77.2 (13.0↑)	86.6 (7.0↑)

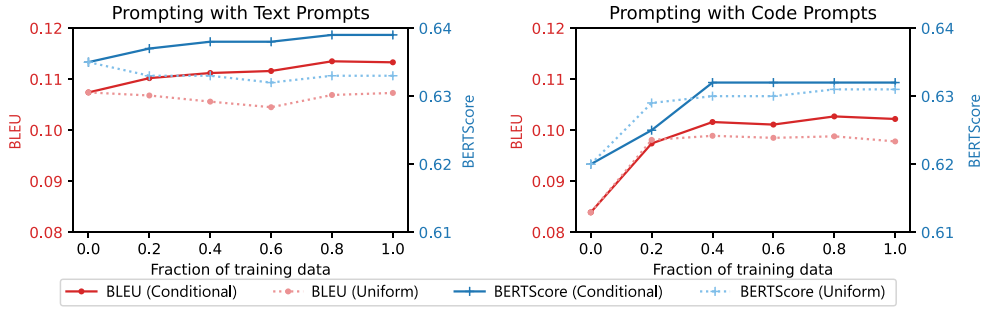
(b) Counterfactual reasoning.



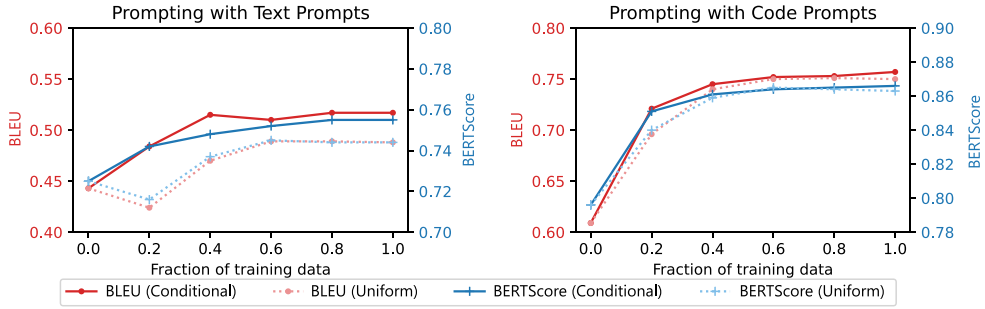
(a) Performance of LLAMA-2 on abductive reasoning.



(b) Performance of LLAMA-2 on counterfactual reasoning.



(c) Performance of DEEPSEEK-LLM on abductive reasoning.



(d) Performance of DEEPSEEK-LLM on counterfactual reasoning.

Figure 5
Performance of models finetuned with different fractions of training data.

code-related, there is consistent gain when models are prompted with text prompts, and the gain is even greater than when prompted with code prompts. This shows that the finetuning process not only enhances the coding abilities of models, but also truly enhances the causal reasoning abilities.

To investigate the trend of performance gain regarding the amount of training data, we control the fraction of training data from 0% to 100%, and evaluate the performance of finetuned models. The results are shown in Figure 5. In general, the performance of the model finetuned on conditional statements exhibits a positive correlation with the amount of training data. The largest performance enhancement is observed from 0% to 20% of training data in most scenarios, indicating that the causal reasoning abilities of models could improve greatly with only a small amount (less than one thousand) of conditional statement codes.

Comparing the models trained on the uniform sample and the conditional statements in Figure 5, we observe that training on code leads to a certain improvement, while training on code with conditional statements leads to further improvement. When prompted with text prompts, models benefit more from code with conditional statements, indicating that the conditional statements enhance the causal reasoning abilities. When prompted with code prompts, the improvements of training on the two corpora are quite close. Although conditional statements positively influence causal reasoning abilities, the narrow distribution of code may harm more general coding abilities, which are also required to understand code prompts. More results of comparing models finetuned on different code corpora are in Appendix G.

8. Conclusion

We investigate the causal reasoning ability of Code-LLMs and the effectiveness of using code prompts in causal reasoning tasks. We demonstrate that Code-LLMs outperform general-purpose LLMs of the same structures in conducting complex causal reasoning tasks. Compared with text prompts, code prompts prove to be more effective in describing causal structures, improving the performance of a wide range of LLMs. We further analyze the importance of different aspects of code prompts, and find that providing a reasonable causal structure in code can help generate plausible outputs. Based on the observations, we assume that finetuning models on code corpus of conditional statements could improve the causal reasoning capabilities, and verify the hypothesis with experiments. These findings suggest that code, especially the conditional statements in code, could play an important role in eliciting and improving the causal reasoning abilities of LLMs through both prompting and finetuning.

Appendix A: Dataset Quality Check

Both datasets of the abductive reasoning and counterfactual reasoning tasks originate from the ROCStories dataset, which is a large crowdsourced corpus of five-sentence stories. The authors of ROCStories control the quality of the dataset by (1) requiring the annotators to pass a qualification test and (2) qualitatively browsing through the submissions and giving the annotators detailed feedback before approving their submissions.

To further verify the quality of the stories, we sample 100 sentences from the two tasks, and recruit native English speakers from Amazon Mechanical Turk to judge whether the sentences are grammatically and semantically acceptable. Each sentence is assigned to three annotators, and 94% of the sentences are accepted.

Appendix B: Format Perturbations towards Text Prompts

To explore whether the performance gap between code prompt and text prompt could be caught up by format perturbations towards text prompts, we apply format perturbations towards our text prompts. We randomly sample 10 plausible prompt formats with the method introduced by Sclar et al. (2024), and report the average and best performances on LLAMA-2.

As shown in Table B.1, the average performance of the text prompts is close to the performance of our original text prompts, and the best performing text prompts still lag behind the code prompts used in our main experiment, letting alone the code prompts that work best in the intervention experiments (Table 8).

Table B.1
Results of prompt format perturbations on LLAMA-2 (%). Code (Main) indicates the performance of the code prompt in the main experiment, and Code (Best) indicates the performance of the code prompt that works best in the intervention experiments.

Prompt	$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
Text (Average)	4.9	26.4	42.3	51.5
Text (Best)	5.9	30.0	50.3	58.9
Code (Main)	6.1	30.5	50.3	59.2
Code (Best)	**6.3**	**31.0**	**52.7**	**59.5**

(a) Abductive reasoning.

Prompt	$BLEU_4$	$ROUGE_L$	BERTScore
Text (Average)	17.3	29.6	54.4
Text (Best)	24.5	37.4	65.7
Code (Main)	33.8	51.5	72.7
Code (Best)	**43.6**	**53.3**	**73.7**

(b) Counterfactual reasoning.

Appendix C: Details of Human Evaluation

Table C.1 shows the annotation instructions provided to the annotators in human evaluation. We recruit three Ph.D. students majoring in Natural Language Processing (NLP) as our annotators, and they are fairly paid at more than $10 per hour.

Table C.1
Annotation instructions provided to the annotators in human evaluation. We demonstrate one example instance for each task.

Abductive Reasoning
Models are asked to generate a plausible hypothesis given the observations: the premise and the ending. Here is an example of a plausible hypothesis: **Premise**: Jenny went to work, leaving the window just a crack open. **Ending**: When Jenny returned home she saw that her house was a mess! **Hypothesis**: It was a breezy day and a large bird flew into the house. Please select the hypothesis (1, 2, or 0 if neutral) that is more coherent with the premise/the ending/both. 1. **Premise**: Steve enrolled at his favorite college. **Ending**: Steve accepted a high paying job at a rival news show. **Hypothesis1**: Steve was recruited by the rival news show while in college. **Hypothesis2**: Steve majored in journalism. More coherent with the premise: More coherent with the ending: More coherent with both:
Counterfactual Reasoning
The task is to rewrite the story ending under a counterfactual event. Models are asked to generate the edited ending that minimally modifies the original ending and is coherent with the counterfactual event. Here is an example of a plausible edited ending: **Premise**: The soccer game was tied 3 to 3 and there was a minute left to play. **Initial event**: Julie had never scored a goal yet, but knew today would be her day. **Original ending**: Ashley passed her the ball and this was the chance. She kicked as hard as she could, and the ball soared into the net. Julie's first goal won the game. **Counterfactual event**: Julie was eagerly watching the game in the stands. **Edited ending**: Ashley had the ball and this was the chance. She kicked as hard as she could, and the ball soared into the net. Julie's team won the game. Please select the edited ending (1, 2, or 0 if neutral) that is more coherent with the context/minimally edits the original ending. 1. **Premise**: Abby loved candy. **Initial event**: She was given her allowance for the week. **Original ending**: She decided to spend it all at the candy store. She started to notice a numb ache in her tooth. Suddenly Abby understood why her dentist warned against sugar. **Counterfactual event**: She was broke, she had spent all of her allowance already. **Edited ending1**: She decided to spend it all at the candy store. She realized she had no money left. She sighed and walked away. **Edited ending2**: She decided to spend it all at the candy store. She started to notice a numb ache in her tooth. She remembered she had no money for a dentist. More coherent with the context: Minimally edit:

Appendix D: Example Cases

More example cases of model generations are shown in Table D.1 and Table D.2.

Table D.1
More cases of model generations in the task of abductive reasoning. Generations that are not coherent with the context are labeled as red.

Abductive Reasoning 1
Premise: Kendall had a busy night.
Ending: Her fall was very embarrassing for her.
Hypothesis [LLAMA-2 (text prompt)]: She was distracted by her busy night, leading to her fall.
Hypothesis [LLAMA-2 (code prompt)]: She tripped over her own feet.
Hypothesis [QWEN1.5 (text prompt)]: The busyness of Kendall's night caused her to stumble.
Hypothesis [QWEN1.5 (code prompt)]: Kendall's tiredness contributed to her fall.
Hypothesis [GPT-3 (text prompt)]: Kendall had a busy night and she was tired.
Hypothesis [GPT-3 (code prompt)]: Kendall was busy dancing at a nightclub and she fell while dancing.
Abductive Reasoning 2
Premise: Patty had a fascination with the occult.
Ending: She went back to her parents and told them it's all a bunch of lies.
Hypothesis [LLAMA-2 (text prompt)]: Patty realized that the occult was not the answer to her problems and turned to her parents for support.
Hypothesis [LLAMA-2 (code prompt)]: Patty discovered a hidden chamber deep within the pyramid.
Hypothesis [QWEN1.5 (text prompt)]: Patty's fascination led her to lie about the occult.
Hypothesis [QWEN1.5 (code prompt)]: Patty discovered the truth about the occult.
Hypothesis [GPT-3 (text prompt)]: Patty's parents may have had a negative experience with the occult in the past, which led to Patty's fascination with it.
Hypothesis [GPT-3 (code prompt)]: Patty's parents tell her that they are also occultists and that they were just waiting for her to come of age.

Table D.2
More cases of model generations in the task of counterfactual reasoning. Generations that are not coherent with the context are labeled as red, and the ones that fail to preserve the original ending are labeled as blue.

Counterfactual Reasoning 1
Premise: Theo played his drums everyday for the last three years.
Initial Event: He had been feeling like he needed to take it to the next level.
Original Ending: He decided that finding a band to join was his best option. After looking online around he finally heard of a good choice. Theo set up an audition.
Counterfactual Event: He was thinking of quitting the drums.
Counterfactual Ending [LLAMA-2 (code prompt)]: He decided to take a break from the drums for a while. He went on a road trip with his friends. They had a great time.
Counterfactual Ending [CODE-LLAMA (code prompt)]: He decided to take a break from the drums. He started to focus on his other hobbies. He was happy with his decision.
Counterfactual Ending [QWEN1.5 (code prompt)]: Theo realized that he had lost his passion for playing. He decided to take a break and focus on other hobbies. Theo's drumming journey came to an end.
Counterfactual Ending [CODEQWEN1.5 (code prompt)]: He decided to quit the drums. He started looking for other hobbies. He found a new passion and started playing guitar.
Counterfactual Ending [GPT-3 (text prompt)]: Theo thought about quitting the drums, but he decided that finding a band to join was his best option. After looking online around he finally heard of a good choice. Theo set up an audition.
Counterfactual Ending [CODEX (code prompt)]: He decided that he needed to take a break from the drums. He decided to take a break for a month. After a month he felt refreshed and ready to play again.
Counterfactual Reasoning 2
Premise: Jay was falling behind in math class.
Initial Event: There was an exam at the end of the week, and he was very nervous.
Original Ending: Jay hired a tutor, but it wasn't enough. The night before the test, Jay stayed up all night studying. Jay barely passed the test.
Counterfactual Event: There was an exam at the end of the week, and he was already prepared.
Counterfactual Ending [LLAMA-2 (code prompt)]: Jay studied hard, but he still didn't feel confident. He surprised everyone by acing the test. Jay felt relieved and proud of himself.
Counterfactual Ending [CODE-LLAMA (code prompt)]: Jay hired a tutor, and it was enough. The night before the test, Jay stayed up all night studying. Jay passed the test with flying colors.
Counterfactual Ending [QWEN1.5 (code prompt)]: Jay studied all week and felt confident. The day of the test, Jay felt calm and focused. Jay aced the exam and brought his grade up.
Counterfactual Ending [CODEQWEN1.5 (code prompt)]: Jay prepared for the exam with a study group. The night before the test, Jay stayed up all night studying. Jay passed the exam with flying colors.
Counterfactual Ending [GPT-3 (text prompt)]: Jay passed the test with ease.
Counterfactual Ending [CODEX (code prompt)]: Jay hired a tutor, and it was enough. The night before the test, Jay stayed up all night studying. Jay barely passed the test.

Appendix E: Intervened Prompts

We demonstrate prompt examples for the intervention experiments in Section 6. Figure E.1 provides examples of information interventions to code prompts, Figure E.2 provides examples of format interventions, and Figure E.3 provides examples of language interventions.

```
def main():
    premise()
    if hypothesis():
        ending()
def premise():
    # Scott loved his trumpet.
def ending():
    # Scott's dad bought him a new one for his birthday.
def hypothesis():
    #
```

(a) Example prompt of *No Instruction* intervention.

```
# task: generate a plausible explanatory hypothesis given the premise
    and the ending. The hypothesis should be less than 20 words.
def main():
    functionA()
    if functionB():
        functionC()
def functionA():
    # Scott loved his trumpet.
def functionC():
    # Scott's dad bought him a new one for his birthday.
def functionB():
    #
```

(b) Example prompt of *Function Name Perturbation* intervention.

Figure E.1
Examples of information interventions to code prompts in abductive reasoning.

```
# task: generate a plausible explanatory hypothesis given the premise
    and the ending. The hypothesis should be less than 20 words.
class Story:
    def __init__(self):
        self.premise()
        if self.hypothesis()
            self.ending()
    def premise(self):
        # Scott loved his trumpet.
    def ending(self):
        # Scott's dad bought him a new one for his birthday.
    def hypothesis(self):
        #
```

(a) Example prompt of *Class* intervention.

```
# task: generate a plausible explanatory hypothesis given the premise
    and the ending. The hypothesis should be less than 20 words.
def main():
    premise()
    if hypothesis():
        ending()
def premise():
    print("Scott loved his trumpet.")
def ending():
    print("Scott's dad bought him a new one for his birthday.")
def hypothesis():
    print("
```

(b) Example prompt of *Print* intervention.

Figure E.2
Examples of format interventions to code prompts in abductive reasoning.

```java
// task: generate a plausible explanatory hypothesis given the premise and the ending. The hypothesis should be less than 20 words.
public class Story {
    public static void main(String[] args) {
        premise();
        if (hypothesis()) {
            ending();
        }
    }
    public static void premise() {
        // Scott loved his trumpet.
    }
    public static void ending() {
        // Scott's dad bought him a new one for his birthday.
    }
    public static boolean hypothesis() {
        //
```

(a) Example prompt of *Java* intervention.

```c
# task: generate a plausible explanatory hypothesis given the premise and the ending. The hypothesis should be less than 20 words.
int main() {
    premise();
    if (hypothesis()) {
        ending();
    }
}
void premise() {
    // Scott loved his trumpet.
}
void ending() {
    // Scott's dad bought him a new one for his birthday.
}
int hypothesis() {
    //
```

(b) Example prompt of *C* intervention.

Figure E.3
Examples of language interventions to code prompts in abductive reasoning.

Appendix F: Hyperparameter Search

The finetuning hyperparameters batch size, number of epochs, and learning rate are selected with a grid search over the validation set of $\mathcal{ART}$ and TimeTravel. Specifically, we empirically set the search range of {64, 128, 256} for the batch size, from 1 to 5 for the number of epochs, and {1e-5, 2e-5} for the learning rate. We finetune LLAMA-2 on the code corpus of conditional statements with all the setting combinations, and select the setting with the highest average BERTScore on both datasets with both text and code prompts.

Appendix G: Finetuning LLMs on Different Code Corpora

To disentangle the gains from conditional statements and the gains from broader code corpus, we compare the models finetuned on the code corpus of conditional statements with two baselines.

- Uniform: The models are finetuned on a uniform sample of `CodeAlpaca`.
- Unconditional: The models are finetuned on a sample of `CodeAlpaca` that excludes all code with conditional statements. This is based on ChatGPT's classification of whether the instance contains conditional statements.

The sample size and all the training implementations of the two baselines are the same with the models finetuned on conditional statements.

Table G.1
Results of models finetuned on different corpus (%). Best results are in **bold**.

Model	Training Corpus	$BLEU_4$	$ROUGE_L$	CIDEr	BERTScore
Prompting with Text Prompts					
LLAMA-2	Uniform	6.2	31.4	52.4	59.2
	Unconditional	6.3	31.4	53.1	59.6
	Conditional	**6.7**	**31.5**	**53.7**	**59.7**
DEEPSEEK-LLM	Uniform	10.7	36.6	72.0	63.4
	Unconditional	10.7	36.6	72.8	63.3
	Conditional	**11.3**	**37.4**	**73.9**	**63.9**
Prompting with Code Prompts					
LLAMA-2	Uniform	**7.3**	**32.3**	**56.3**	**60.4**
	Unconditional	6.6	31.1	51.9	59.7
	Conditional	6.8	30.7	50.7	60.2
DEEPSEEK-LLM	Uniform	9.8	36.1	72.0	63.1
	Unconditional	9.6	36.0	**72.1**	62.9
	Conditional	**10.2**	**36.3**	71.3	**63.2**

(a) Abductive reasoning results of models finetuned on different corpus.

		$BLEU_4$	$ROUGE_L$	BERTScore
Prompting with Text Prompts				
LLAMA-2	Uniform	40.1	47.9	71.4
	Unconditional	47.8	54.9	75.1
	Conditional	**48.4**	**55.0**	**75.3**
DEEPSEEK-LLM	Uniform	48.8	54.5	74.4
	Unconditional	46.2	52.7	73.4
	Conditional	**51.7**	**56.4**	**75.5**
Prompting with Code Prompts				
LLAMA-2	Uniform	59.0	64.2	79.8
	Unconditional	65.0	68.7	82.2
	Conditional	**69.5**	**71.9**	**83.9**
DEEPSEEK-LLM	Uniform	75.0	76.8	86.4
	Unconditional	**76.6**	**78.3**	**87.1**
	Conditional	75.7	77.2	86.6

(b) Counterfactual reasoning results of models finetuned on different corpus.

As shown in Table G.1, we observe that the gains of finetuning on code corpus with conditional statements are greater in most cases, especially when prompted with text prompts. Finetuning on all three code corpora helps models to better understand code, but finetuning on code corpus with conditional statements help models to better improve the causal reasoning abilities, which could be transferred between different prompts.

Acknowledgments

This work is supported in part by NSFC (62161160339) and Beijing Science and Technology Program (Z231100007423011). We thank the anonymous reviewers and the editors for their helpful suggestions.

References

Bai, Jinze, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bavarian, Mohammad, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*.

Beamer, Brandon, Alla Rozovskaya, and Roxana Girju. 2008. Automatic semantic relation extraction with multiple boundary generation. In *Proceedings of AAAI*, pages 824–829.

Bhagavatula, Chandra, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Wen-tau Yih, and Yejin Choi. 2019. Abductive commonsense reasoning. In *Proceedings of International Conference on Learning Representations*.

Bi, Xiao, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Blanco, Eduardo, Nuria Castell, and Dan I. Moldovan. 2008. Causal relation extraction. In *Proceedings of LREC*, volume 66, page 74.

Bogin, Ben, Shivanshu Gupta, Peter Clark, and Ashish Sabharwal. 2023. Leveraging code to improve in-context learning for semantic parsing. *arXiv preprint arXiv:2311.09519*.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Chang, Du-Seong and Key-Sun Choi. 2005. Causal relation extraction using cue phrase and lexical pair probabilities. In *International Conference on Natural Language Processing*, pages 61–70. https://doi.org/10.1007/978-3-540-30211-7_7

Chaudhary, Sahil. 2023. Code Alpaca: An instruction-following LLaMa model for code generation. https://github.com/sahil280114/codealpaca

Chen, Jiangjie, Chun Gan, Sijie Cheng, Hao Zhou, Yanghua Xiao, and Lei Li. 2022. Unsupervised editing for counterfactual stories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10473–10481. https://doi.org/10.1609/aaai.v36i10.21290

Chen, Mark, 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*.

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

Chen, Yangyi, Xingyao Wang, Manling Li, Derek Hoiem, and Heng Ji. 2023b. ViStruct: Visual structural knowledge extraction via curriculum guided code-vision representation. In the *2023 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/2023.emnlp-main.824

Cummins, Denise D., Todd Lubart, Olaf Alksnis, and Robert Rist. 1991. Conditional reasoning and causation. *Memory & Cognition*, 19:274–282. https://doi.org/10.3758/BF03211151, PubMed: 1861613

Du, Li, Xiao Ding, Kai Xiong, Ting Liu, and Bing Qin. 2021. ExCAR: Event graph knowledge enhanced explainable causal reasoning. In *Proceedings of the 59th Annual*

Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 2354–2363. https://doi.org/10.18653/v1/2021.acl-long.183

Epstude, Kai and Neal J. Roese. 2008. The functional theory of counterfactual thinking. *Personality and Social Psychology Review*, 12(2):168–192. https://doi.org/10.1177/1088868308316091, PubMed: 18453477

Fan, Zhiyu, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1469–1481. https://doi.org/10.1109/ICSE48619.2023.00128

Feng, Zhangyin, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

Gao, Jinglong, Xiao Ding, Bing Qin, and Ting Liu. 2023. Is ChatGPT a good causal reasoner? A comprehensive evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11111–11126. https://doi.org/10.18653/v1/2023.findings-emnlp.743

Gao, Luyu, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. PAL: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Girju, Roxana, Preslav Nakov, Vivi Nastase, Stan Szpakowicz, Peter Turney, and Deniz Yuret. 2007. SemEval-2007 Task 04: Classification of semantic relations between nominals. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 13–18. https://doi.org/10.3115/1621474.1621477

Guo, Daya, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, et al. 2024. DeepSeek-coder: When the large language model meets programming—The rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Gurulingappa, Harsha, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. 2012. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of Biomedical Informatics*, 45(5):885–892. https://doi.org/10.1016/j.jbi.2012.04.008, PubMed: 22554702

Hagmayer, York, Steven A. Sloman, David A. Lagnado, and Michael R. Waldmann. 2007. Causal reasoning through intervention. *Causal Learning: Psychology, Philosophy, and Computation*, pages 86–100. https://doi.org/10.1093/acprof:oso/9780195176803.003.0007

Hendrickx, Iris, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid O. Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2010. SemEval-2010 Task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of ACL 2010*, page 33. https://doi.org/10.3115/1621969.1621986

Hindle, Abram, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. 2016. On the naturalness of software. *Communications of the ACM*, 59(5):122–131. https://doi.org/10.1145/2902362

Hobbs, Jerry R. 2005. Toward a useful concept of causality for lexical semantics. *Journal of Semantics*, 22(2):181–209. https://doi.org/10.1093/jos/ffh024

Hobbs, Jerry R., Mark E. Stickel, Douglas E. Appelt, and Paul Martin. 1993. Interpretation as abduction. *Artificial Intelligence*, 63(1–2):69–142. https://doi.org/10.1016/0004-3702(93)90015-4

Hu, Yushi, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A. Smith, and Mari Ostendorf. 2022. In-context learning for few-shot dialogue state tracking. *ArXiv*, abs/2203.08568. https://doi.org/10.18653/v1/2022.findings-emnlp.193

Jiang, Albert Q., 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*.

Jin, Zhijing, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng Lyu, Kevin Blin, Fernando Gonzalez Adauto, Max Kleiman-Weiner, Mrinmaya Sachan, et al. 2024. CLadder: A benchmark to assess causal reasoning capabilities of language

models. *Advances in Neural Information Processing Systems*, volume 36.

Jin, Zhijing, Jiarui Liu, L. Y. U. Zhiheng, Spencer Poff, Mrinmaya Sachan, Rada Mihalcea, Mona T. Diab, and Bernhard Schölkopf. 2023. Can large language models infer causation from correlation? In the *Twelfth International Conference on Learning Representations*.

Kıcıman, Emre, Robert Ness, Amit Sharma, and Chenhao Tan. 2023. Causal reasoning and large language models: Opening a new frontier for causality. *arXiv preprint arXiv:2305.00050*.

Kim, Najoung, Sebastian Schuster, and Shubham Toshniwal. 2024. Code pretraining improves entity tracking abilities of language models. *arXiv preprint arXiv:2405.21068*.

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

Li, Pengfei and Kezhi Mao. 2019. Knowledge-oriented convolutional neural network for causal relation extraction from natural language texts. *Expert Systems with Applications*, 115:512–523. https://doi.org/10.1016/j.eswa.2018.08.009

Li, Raymond, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Li, Xiang, John Thickstun, Ishaan Gulrajani, Percy S. Liang, and Tatsunori B. Hashimoto. 2022. Diffusion-LM improves controllable text generation. *Advances in Neural I nformation Processing Systems*, 35:4328–4343.

Li, Zhongyang, Tongfei Chen, and Benjamin Van Durme. 2019. Learning to rank for plausible plausibility. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4818–4823. https://doi.org/10.18653/v1/P19-1475

Lin, Chin Yew. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81.

Liu, Xiao, Yansong Feng, and Kai-Wei Chang. 2024. CASA: Causality-driven argument sufficiency assessment. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5282–5302. https://doi.org/10.18653/v1/2024.naacl-long.296

Liu, Xiao, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. 2024. Are LLMs capable of data-based statistical and causal reasoning? Benchmarking advanced quantitative reasoning with data. *arXiv preprint arXiv:2402.17644*. https://doi.org/10.18653/v1/2024.findings-acl.548

Liu, Xiao, Da Yin, Chen Zhang, Yansong Feng, and Dongyan Zhao. 2023. 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. https://doi.org/10.18653/v1/2023.findings-acl.574

Madaan, Aman, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*. https://doi.org/10.18653/v1/2022.emnlp-main.90

Miao, Ning, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. 2019. CGMH: Constrained sentence generation by Metropolis-Hastings sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6834–6842. https://doi.org/10.1609/aaai.v33i01.33016834

Monperrus, Martin. 2018. Automatic software repair: A bibliography. *ACM Computing Surveys (CSUR)*, 51(1):1–24. https://doi.org/10.1145/3105906

Mostafazadeh, Nasrin, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849. https://doi.org/10.18653/v1/N16-1098

Mueller, Aaron, Albert Webson, Jackson Petty, and Tal Linzen. 2024. In-context learning generalizes, but not always robustly: The case of syntax. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4761–4779. https://doi.org/10.18653/v1/2024.naacl-long.267

Ni, Ansong, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. 2023. LEVER: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128.

Nisi, Valentina and Mads Haahr. 2006. Weird view: Interactive multilinear narratives and real-life community stories. *Crossings*, 2:27.

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318. https://doi.org/10.3115/1073083.1073135

Peirce, Charles Sanders. 1974. *Collected Papers of Charles Sanders Peirce*, volume 5. Harvard University Press.

Petty, Jackson, Sjoerd van Steenkiste, and Tal Linzen. 2024. How does code pretraining affect language model task performance? In the *7th BlackboxNLP Workshop*.

Puerto, Haritz, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in text+ code LLMs. *arXiv preprint arXiv:2401.10065*. https://doi.org/10.18653/v1/2024.emnlp-main.629

Pyysalo, Sampo, Filip Ginter, Juho Heimonen, Jari Björne, Jorma Boberg, Jouni Järvinen, and Tapio Salakoski. 2007. BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, 8:1–24. https://doi.org/10.1186/1471-2105-8-50, PubMed: 17291334

Qin, Lianhui, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. 2019. Counterfactual story reasoning and generation. 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 5043–5053. https://doi.org/10.18653/v1/D19-1509

Qin, Lianhui, Vered Shwartz, Peter West, Chandra Bhagavatula, Jena D. Hwang, Ronan Le Bras, Antoine Bosselut, and Yejin Choi. 2020. Back to the future: Unsupervised backprop-based decoding for counterfactual and abductive commonsense reasoning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 794–805. https://doi.org/10.18653/v1/2020.emnlp-main.58

Qin, Lianhui, Sean Welleck, Daniel Khashabi, and Yejin Choi. 2022. Cold decoding: Energy-based constrained text generation with Langevin dynamics. *Advances in Neural Information Processing Systems*, 35:9538–9551.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Roemmele, Melissa, Cosmin Adrian Bejan, and Andrew S. Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Roziere, Baptiste, 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*.

Sclar, Melanie, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In the *Twelfth International Conference on Learning Representations*.

Sloman, Steven. 2005. *Causal Models: How People Think About the World and Its Alternatives*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780195183115.001.0001

Son, Youngseo, Anneke Buffone, Joe Raso, Allegra Larche, Anthony Janocko, Kevin Zembroski, H. Andrew Schwartz, and Lyle Ungar. 2017. Recognizing counterfactual thinking in social media texts. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 654–658. https://doi.org/10.18653/v1/P17-2103

Taori, Rohan, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

https://github.com/tatsu-lab/stanford_alpaca

Team, Gemini, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, et al. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

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

Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

van Rooij, Robert and Katrin Schulz. 2019. Conditionals, causality and conditional probability. *Journal of Logic, Language and Information*, 28:55–71.

Vedantam, Ramakrishna, C. Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4566–4575. https://doi.org/10.1109/CVPR.2015.7299087

Wang, Jiashuo, Yi Cheng, and Wenjie Li. 2022. CARE: Causality reasoning for empathetic responses by conditional graph generation. *arXiv preprint arXiv:2211.00255*. https://doi.org/10.18653/v1/2022.findings-emnlp.51

Wang, Xingyao, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. *arXiv preprint arXiv:2402.01030*.

Wang, Xingyao, Sha Li, and Heng Ji. 2022. CODE4STRUCT: Code generation for few-shot structured prediction from natural language. *arXiv preprint arXiv:2210.12810*. https://doi.org/10.18653/v1/2023.acl-long.202

Wang, Yizhong, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023a. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508. https://doi.org/10.18653/v1/2023.acl-long.754

Wang, Yue, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021a. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

Wang, Zihao, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian (Shawn) Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, volume 36, pages 34153–34189.

Wang, Zirui, Adams Wei Yu, Orhan Firat, and Yuan Cao. 2021b. Towards zero-label language learning. *arXiv preprint arXiv:2109.09193*.

Weidenfeld, Andrea, Klaus Oberauer, and Robin Hörnig. 2005. Causal and noncausal conditionals: An integrated model of interpretation and reasoning. *The Quarterly Journal of Experimental Psychology Section A*, 58(8):1479–1513. https://doi.org/10.1080/02724980443000719

Wu, Yuhuai, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*.

Xu, Frank F., Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10. https://doi.org/10.1145/3520312.3534862

Yang, Jie, Soyeon Caren Han, and Josiah Poon. 2022. A survey on extraction of causal relations from natural language text. *Knowledge and Information Systems*, 64(5):1161–1186. https://doi.org/10.1007/s10115-022-01665-w

Yang, Ke, Jiateng Liu, John Wu, Chaoqi Yang, Yi R. Fung, Sha Li, Zixuan Huang, Xu Cao, Xingyao Wang, Yiquan Wang, et al. 2024. If LLM is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents. *arXiv preprint arXiv:2401.00812*.

Zečević, Matej, Moritz Willig, Devendra Singh Dhami, and Kristian Kersting. 2023. Causal parrots: Large language models may talk causality but are not causal. *Transactions on Machine Learning Research*.

Zhang, Jiayao, Hongming Zhang, Weijie Su, and Dan Roth. 2022. ROCK: Causal inference principles for reasoning about commonsense causality. In *International Conference on Machine Learning*, pages 26750–26771.

Zhang, Li, Liam Dugan, Hainiu Xu, and Chris Callison-Burch. 2023. Exploring the curious case of code prompts. In *1st Workshop on Natural Language Reasoning and Structured Explanations (@ ACL 2023)*, page 9. https://doi.org/10.18653/v1/2023.nlrse-1.2

Zhang, Tianyi, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2019. BertScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

Zheng, Zibin, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2023. A survey of large language models for code: Evolution, benchmarking, and future trends. *arXiv preprint arXiv:2311.10372*.