Are Large Language Models Zero-Shot Spatial Reasoners?

Miguel Liu-Schiaffini, Enoch Luk, Ethan Wilk, and Alan Wu
California Institute of Technology
{mliuschi, eluk, ejwilk, alawu}@caltech.edu

1 Introduction and related work

Large language models (LLMs) have recently been applied to a variety of problems [Brown et al., 2020], including mathematical reasoning [Frieder et al., 2024], proof-writing [Song et al., 2024], code generation [Vaithilingam et al., 2022], search problems [Lehnert et al., 2024], and generating 3-d object meshes Yamada et al. [2024b]. While many practical applications now rely on large language models to facilitate chatting, image generation, and code assistance, the *spatial reasoning* capabilities of large language models have been relatively under-explored in the literature.

However, spatial reasoning is a necessary component of human cognition [Yamada et al., 2024a], and humans often construct mental maps to represent physical space when reasoning about spatial phenomena [Wu et al., 2024]. While text-based (i.e., not multi-modal) LLMs perform all reasoning based on textual tokens, prior works have demonstrated that they have some ability to reason spatially [Lehnert et al., 2024, Wu et al., 2024].

It is important to note that spatial reasoning is not exclusively limited to reasoning based on visual inputs; there are many situations where spatial reasoning uses exclusively *textual* inputs. For instance, when a tourist asks a local for directions, these directions may take the form of instructions: "Walk three blocks, turn left at the traffic light, walk one more block, and turn right again." Humans typically construct mental maps to aid in such spatial reasoning.

Some prior works investigating spatial reasoning in LLMs have been inspired by the mental map of humans. For instance, Wu et al. [2024] introduced *visualization-of-thought*, a prompting technique based on chain-of-thought [Wei et al., 2022, Kojima et al., 2022] that asks an LLM to print a visualization of the state after each step of reasoning. Yamada et al. [2024a] and Li et al. [2024] introduce various benchmarks for general spatial reasoning. Yamada et al. [2024a] provides an in-depth evaluation of the spatial understanding of large language models. For instance, the authors explore which spatial geometries and graphical topologies lead to better performance for the model. They find that several features of the spatial geometry, such as whether the shape is a ring, are significant predictors of model accuracy. The authors also find that presenting spatial maps in a local (rather than global) format leads to superior performance. Despite the depth of evaluation and analysis proposed in the paper, Yamada et al. [2024a] does not propose many methods to extract as much zero-shot spatial reasoning capabilities as possible from text-based large language models.

Amongst other works, Fatemi et al. [2023] explores the various ways in which graphs can be encoded textually for LLMs to process. Additionally, some prior works have proposed spatial path-planning and search methods [Lehnert et al., 2024] and benchmarks [Aghzal et al., 2023]. However, these techniques are generally task-specific to search and path-planning planning problems and would not necessarily generalize to different types of tasks that require spatial reasoning. Further, some techniques, such as Lehnert et al. [2024] require fine-tuning a large language model and thus do not explore the *zero-shot* capabilities of large language models to reason in spatial contexts.

In this project, we are interested in exploring the intrinsic (i.e., zero-shot) capabilities of LLMs to perform spatial reasoning. An example of a spatial reasoning task that guides our proposed

Preprint. Under review.

experiments is *natural language navigation*, introduced by Yamada et al. [2024a]. In this task, a physical setting is described using text (e.g., a twelve-node ring, with a typical household item located at each node), as well as a starting point in space. A sequence of moves is provided (e.g., "move down, move right, move left"), and the reader must identify the final location given the starting point and sequence of moves. Natural language navigation is particularly interesting because it evaluates a purely text-based LLM's capabilities for spatial reasoning from textual pre-training alone (i.e., without multi-modality).

In particular,

1. Motivated by LLMs supplemented with code execution (e.g., Program-of-Thoughts [Chen et al., 2022]), **we propose a new prompting method** that decomposes spatial reasoning tasks into smaller computations, and we demonstrate that this outperforms various baselines, such as chain-of-thought [Wei et al., 2022] and visualization-of-thought [Wu et al., 2024].
2. Following prior work, we identify spatial representation as a key component of successful spatial reasoning. In contrast with prior work (which focused primarily on input representations), we **systematically explore the effects of different *internal* representation of spatial information** in LLMs.
3. We **document and analyze the various failure modes** of text-based LLMs encountered during our experiments on spatial reasoning.

The various sections are organized as follows. Section 2 formalizes the natural language navigation [Yamada et al., 2024a] task that we consider in this paper. Section 3 presents numerical experiments for code use, the motivation of our first proposed prompting technique. It also presents the numerical results for our first prompting technique. Section 4 explores the effects of varying the internal representation of spatial information in LLMs. Section 5 summarizes and discusses the failure modes that we observed during our experiments, and Section 6 concludes our report, also discussing limitations and future work.

2 Problem statement and dataset

2.1 Problem statement

In this project, we focus on the task of natural language navigation [Yamada et al., 2024a], which was briefly described above. We believe this task has sufficient possible variations and flexibility to allow it to serve as a proxy for an evaluation of general spatial reasoning capabilities in language models. Furthermore, there is precedent in the litreature for using natural language navigation as such a proxy [Wu et al., 2024, Yamada et al., 2024a].

More formally, natural language navigation problems define some directed graph $G = (V, E)$ and a set of objects W that index the set of vertices V. This seeks to emulate a setup in physical space, where several objects may be placed at different locations, and an agent must navigate between these objects. It also may reduce confounding that may occur by labeling nodes numerically. Given some starting position $v \in V$ (often specified as the corresponding object w_v at that node) and a sequence of navigation instructions $\mathcal{I} = (i_1, \ldots, i_k)$, where each $i_j \in E$, the goal of the natural language navigation problem is to identify the object $w \in W$ that is found at the ending position. In practice, the graph G is often highly structured. For instance, some examples include a 3×3 grid (where agents may move up, down, left, or right) and an N-node ring.

A sample natural language navigation problem is shown in Figure 1.

> You have been given a 3 by 3 square grid. In the 1st row, from left to right, we have a Maltese, a barrel, and a tiger shark. In the 2nd row, from left to right, we have a pole, a dining table, and a tick. In the 3rd row, from left to right, we have a velvet, a birdhouse, and a plunger. You start at the position where the Maltese is located, then you go right by one step, then you go down by one step, then you go right by one step, and then you go down by one step. What will you find?

Figure 1: Sample natural language navigation problem from Yamada et al. [2024a].

2.2 Dataset details

In this project, we perform experiments on various datasets (part of the `SpatialEvalLLM` family of datasets) within the natural language navigation problems introduced by Yamada et al. [2024a][1]. The `SpatialEvalLLM` datasets make a distinction between *global* and *local* input representations. Global input representations define the full input spatial map in a structured manner (e.g., row-by-row for a 3×3 grid), whereas local input representations define a path through the geometry and describe what objects are encountered at each location on the path, essentially discovering the map through "exploration." Figure 1 shows a sample global problem on a 3×3 grid. A sample local prompt on a 3×3 grid is shown below:

> You have been given a 3 by 3 square grid. Starting from a vertex, you will move along the edges of the grid. Initially, you are positioned at the top left corner of the grid, where you find a cock. You move down by one step, where you find a geyser. You move right by one step, where you find a jellyfish. You move up by one step, where you find an impala. You move right by one step, where you find a box turtle. You move down by one step, where you find an espresso machine. You move down by one step, where you find a bib. You move left by one step, where you find a megalith. You move up by one step. What will you find?

Figure 2: Sample local natural language navigation problem from Yamada et al. [2024a].

In this project, we consider settings where the spatial geometries, number of movements, and input representation are varied. In particular, we consider the following settings:

1. **12-ring (global):** Ring with 12 nodes, global input representation.
2. **9-ring (global):** Ring with 9 nodes, global input representation
3. **4-step square (global):** 3×3 square grid, moving four times, global representation.
4. **8-step square (global):** 3×3 square grid, moving eight times, global representation.
5. **Tree (global):** Binary tree with objects on each node, global input representation. The question is of the form "What is the cousin of _____?"
6. **Square, random input order (global):** 3×3 square grid, moving eight times, global input representation. The input is passed in a random order by referring to indices within the grid.
7. **Square, "snake" input order (global):** 3×3 square grid, moving eight times, global input representation. The input is passed in a "snake order" (i.e., not row-by-row).
8. **Square, "snake" input order with coordinates (global):** 3×3 square grid, moving eight times, global input representation. The input is passed in a "snake order" (i.e., not row-by-row) with coordinates.
9. **Hexagon (local):** Three "pointy-topped" hexagonal tiles organized in two rows, with the bottom row containing two tiles and the top containing one tile. Objects lie on the vertices of these tiles, and movements occur along edges. The representation is local.
10. **Rhombus (local):** A 3×3 tiling composed of "pointy-topped" rhomboidal tiles. Objects lie on vertices of these tiles, and movements occur along the edges. The representation is local.
11. **12-ring (local):** Ring with 12 nodes, local input representation.
12. **Square (local):** 3×3 square grid, moving eight times, local representation.
13. **Triangle (local):** A triangular tile map with three rows. The first row contains one triangle, the second contains three triangles, and the third contains five triangles. Within each row, triangles alternate between "pointy-up" and "pointy-down." Objects lie on vertices of these tiles, and movements occur along the edges. The representation is local.

We would like to note that the hexagonal, rhomboidal, and triangular maps are particularly challenging, as they require the careful reasoning about complex topologies. Furthermore, the set of available moves (e.g., "up-left," "down-right") are more complicated than for the ring and square geometries.

[1]The `SpatialEvalLLM` datasets are available at https://huggingface.co/datasets/yyamada/SpatialEvalLLM.

3 Code-inspired spatial reasoning

3.1 Motivation: LLMs equipped with executable code environments

Recently, large-scale LLMs (e.g., such as the publicly-available ChatGPT-4 by OpenAI [OpenAI, 2023]) have been combined with external tools and executable code environments to allow for more accurate computations [Qin et al., 2023, Hao et al., 2024]. This has led to a significant improvement in mathematics and financial benchmarks [Chen et al., 2022]. LLMs equipped with executable code environments have been used for a variety of applications, such as 3d object and mesh generation [Yamada et al., 2024b]. We will refer to such models as *code-equipped LLMs*.

While code-equipped LLMs are common in practical applications due to their increased accuracy over vanilla text-based LLMs, naively using them for spatial reasoning tasks would not allow us to disentangle the effects of the intrinsic spatial reasoning capabilities of language models and the accuracy due to having access to an executable code environment. However, we argue that code-equipped LLMs can motivate novel prompting techniques for spatial reasoning tasks on text-only LLMs.

We first perform numerical experiments on using code-equipped LLMs for natural language navigation. For this experiment, we consider the LLaMa-3 Chat (70b) model [Touvron et al., 2023] and a natural language navigation problem involving a 12-node ring. Figure 3a shows a sample such prompt, and Figure 3b shows the code output of LLaMa-3 on this problem. Running this code block returns `'stupa'`, the correct answer.

Across the entire dataset, LLaMa-3 with code achieves 77% accuracy, whereas LLaMa-3 with zero-shot Chain-of-Thought (CoT) prompting Kojima et al. [2022] achieves only 19% accuracy. LLaMa-3 employing Visualization-of-Thought [Wu et al., 2024] achieves 13% accuracy. These results are shown in Figure 4. Furthermore, when analyzing the sample outputs of the code-equipped model (Figure 3b), we observe that the LLM has designed an indexing scheme to maintain its current position, and this position is updated numerically in a sequential manner (specifically, in each `new_index = ...` line).

We conjecture that the primary reason for the observed difference in performance between the code-equipped LLM and the CoT model is that the code-equipped model can represent its current state and state transitions in a compact and simple form: the current state is represented as an index, and the state transition requires simple modular arithmetic (in the case of a ring); such operations are then computed correctly in a Python executable environment. Motivated by these observations, in the next section we introduce a novel prompting technique for spatial reasoning, which we refer to as *code-inspired spatial prompting*.

3.2 Code-inspired spatial prompting

Our key observation is that the computation steps taken in the Python code generated from code-equipped LLMs can be interpreted as reasoning steps, which immediately arise under a particular representation of spatial data.

Let $\mathcal{M}$ be the spatial domain of interest. The poor performance of baseline zero-shot CoT on natural language navigation problems (see Figure 4) suggests that naive LLMs cannot adequately keep track of positions in $\mathcal{M}$ and/or movements between locations in $\mathcal{M}$. Inspired by code-equipped LLMs, consider the existence of some index set $\mathcal{J}$ in bijection with $\mathcal{M}$, where movements that may have been difficult to track in $\mathcal{M}$ are trivial to follow in $\mathcal{J}$.

For the natural language problems at hand, we observe that code-equipped LLMs construct an index set $\mathcal{J}$ that is a subset of $\mathbb{Z}^d$, where the size and dimension d of this subset depends on the topology of $\mathcal{M}$. Computations on $\mathcal{J}$ are then trivial to compute in executable code environments.

Inspired by code-equipped LLMs, we propose *code-inspired spatial prompting*, which prompts LLMs to construct an index set $\mathcal{J}$ representing the spatial domain $\mathcal{M}$, with the hope that LLM computations on $\mathcal{J}$ are achieved with higher accuracy than maintaining an abstract representation of $\mathcal{M}$[2]. In particular, we find that the following simple prompt

[2]When $\mathcal{J}$ is a subset of $\mathbb{Z}^d$, the ability of LLMs to perform these computations has been studied empirically, with promising results [Yang et al., 2023].

You have been given a circular path consisting of 12 connected dots. At the start, you are positioned on the dot that is located at the top of the path, where you find a crash helmet. Moving in a clockwise direction from the crash helmet, the elements on the path are a trimaran, a sleeping bag, a dhole, a sulphur butterfly, a stove, a rotisserie, a stupa, a bagel, a gibbon, a fountain, and a Geoffroy's spider monkey. Starting from the rotisserie, you move around the ring by 11 steps in a counter-clockwise direction, and you move around the ring by 4 steps in a clockwise direction, and you move around the ring by 11 steps in a counter-clockwise direction, and you move around the ring by 7 steps in a counter-clockwise direction, and you move around the ring by 2 steps in a clockwise direction. What will you find?

(a) Natural language navigation problem on a 12-node ring.

```
path = [‘crash helmet’, ‘trimaran’, ‘sleeping bag’, ‘dhole’,
            ‘sulphur butterfly’, ‘stove’, ‘rotisserie’,
            ‘stupa’, ‘bagel’, ‘gibbon’, ‘fountain’,
            ‘geoffroy\’s spider monkey’]
start_index = path.index(‘rotisserie’)
new_index = (start_index - 11) % 12
new_index = (new_index + 4) % 12
new_index = (new_index - 11) % 12
new_index = (new_index - 7) % 12
new_index = (new_index + 2) % 12
result = path[new_index].lower().replace(‘the ’, ‘’).replace(‘a ’, ‘’)
```

(b) Sample code output from LLaMa 70b.

```
Let’s assign a numerical index to each element on the path:

1. crash helmet
2. trimaran
3. sleeping bag
4. dhole
5. sulphur butterfly
6. stove
7. rotisserie
8. stupa
9. bagel
10. gibbon
11. fountain
12. Geoffroy’s spider monkey

Starting from the rotisserie (index 7), we move:
* 11 steps counter-clockwise: 7 - 11 = -4, which wraps around to
        12 - 4 = 8, so we’re at the stupa.
* 4 steps clockwise: 8 + 4 = 12, so we’re at the Geoffroy’s
        spider monkey.
* 11 steps counter-clockwise: 12 - 11 = 1, so we’re at the crash
        helmet.
* 7 steps counter-clockwise: 1 - 7 = -6, which wraps around to
        12 - 6 = 6, so we’re at the stove.
* 2 steps clockwise: 6 + 2 = 8, so we’re at the stupa again.

So, we’ll find *stupa*.
```

(c) Sample output from LLaMa 70b using our code-inspired spatial prompting technique.

Figure 3: Outputs by LLaMa-3 70b for a difficult sample natural language navigation problem.

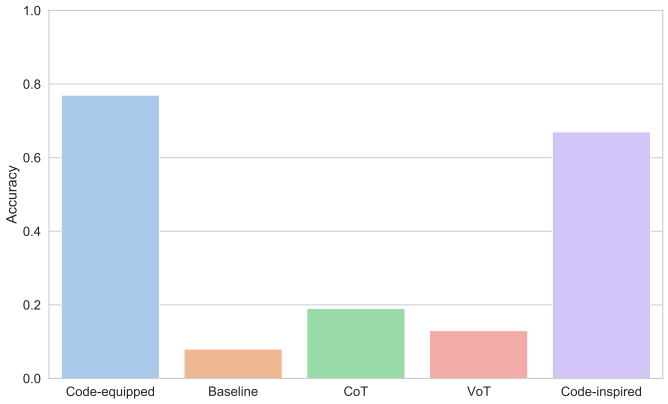


Figure 4: Accuracy comparison on the 12-ring dataset between LLaMa-3 with code, baseline, zero-shot Chain-Of-Thought, zero-shot Visualization-of-Thought [Wu et al., 2024], and our proposed code-inspired spatial prompting.

Assign a numerical index to each element on the path and use this system to track your position.

significantly improves performance over the baseline (up to 67% accuracy) on the 12-ring dataset and comes close to the performance of the code-equipped model (77% accuracy).

In particular, we observe that for both GPT-4 Turbo and LLaMa-3 70b, our proposed code-inspired spatial reasoning (referred to as "indexing" in Figure 5) matches or outperforms the best-performing alternative on most of the global natural language navigation datasets. In particular, we note that our proposed "indexing" method outperforms Visualization-of-Thought [Wu et al., 2024] across all but one datasets, for both models.

We also observe that our proposed method appears to provide the largest improvements over baselines on the ring geometries. We conjecture that keeping track of the modular topology of the ring is difficult for baseline methods. However, the "indexing" method abstracts movements on an N-ring to simple operations on $(\mathbb{Z}/N\mathbb{Z}, +)$, the additive group of integers modulo N.

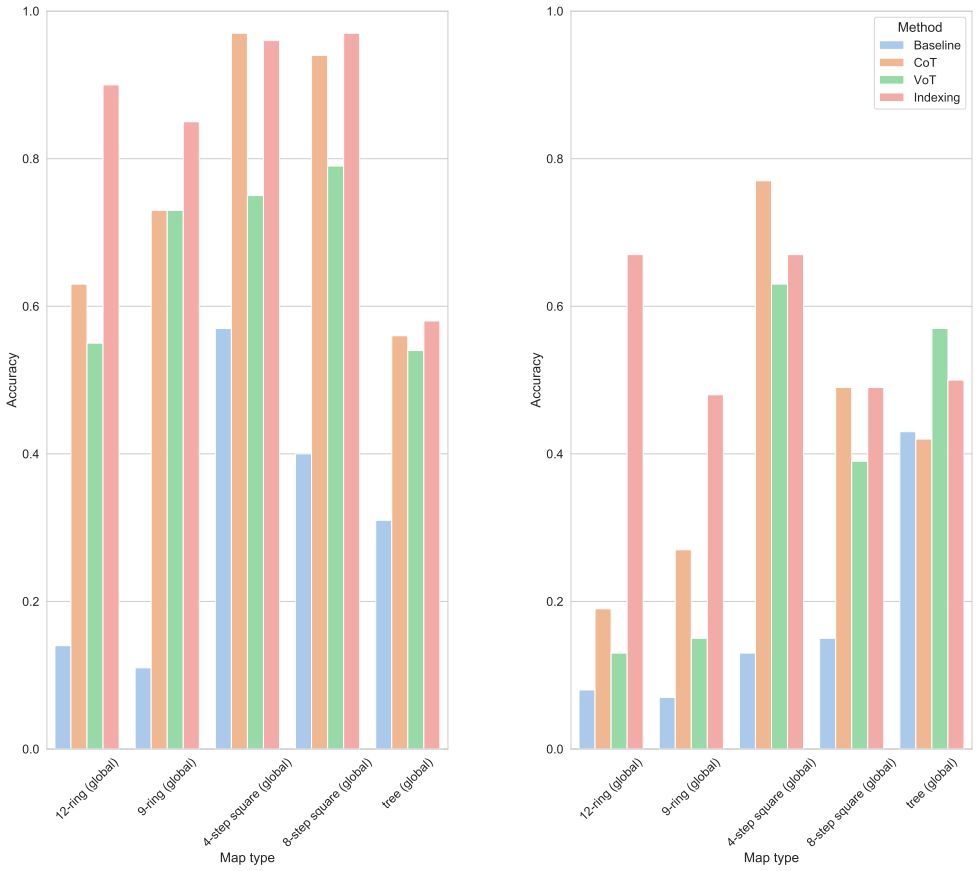
We also performed experiments on GPT-3.5 Turbo and LLaMa-3 8b. However, these models performed poorly across all datasets, significantly under the accuracy levels of GPT-4 Turbo and LLaMa-3 70b. We analyze the results of these models in Section 5. We also explore the performance of all four of these LLMs on the datasets from Yamada et al. [2024a] with local input representations. We find that our proposed code-inspired spatial prompting method performs comparably to the best-performing baselines for GPT-4 Turbo but under these baselines for LLaMa-3 70b. We also discuss these results in Section 5.

4 Internal representations of spatial data

Some prior works have explored the varying *input* representations of spatial data. For instance, when describing the contents of a 3×3 grid to text-based LLMs, Yamada et al. [2024a] consider providing the contents as rows and in "snake-order." Fatemi et al. [2023] explore various input representations for graphs.

However, we instead consider an orthogonal direction. In this project, we consider the effect of varying the *internal* representations of spatial data. Assuming Chain-Of-Thought (CoT)-style internal reasoning, the internal representation of spatial data is the representation of the spatial state that is used by an LLM to deduce the sequence of reasoning steps that lead to the final answer.

One such representation is the spatial representation used by Visualization-of-Thought (VoT) [Wu et al., 2024]. In VoT, an LLM is prompted to reason step-by-step and to visualize the state of the



(a) GPT-4 Turbo (b) LLaMa-3 70b

Figure 5: GPT-4 Turbo and LLaMa-3 70b results on global map datasets.

problem at each step. For certain natural language navigation problems, such as the 3×3 grid shown in Figure 1, this leads to ASCII art-style representations of the state.

In this section, we evaluate the efficacy of various forms of internal spatial representations, including VoT [Wu et al., 2024]. In particular, we fix a CoT style of prompting and instead explicitly prompt the LLM to construct an internal representation of a particular form. In the prompt, we provide one general example showcasing this representation. Note that this is still a zero-shot evaluation because the LLMs has not been provided with in-context examples on solving the natural language navigation problems.

In this project, we focus on the 3×3 grid datasets (using four steps of movement and a global map representation) from Yamada et al. [2024a], since these are most amenable to structured representations. In particular, we focus on four different internal representations on the 3×3 grid dataset. These four representations are shown in Figure 6. The four representations are (a) ASCII art-style (in the style of Visualization-of-Thought [Wu et al., 2024]), (b) enumerating objects row-by-row, (c) enumerating objects via their coordinates, and (d) enumerating the objects column-by-column.

Our numerical results are shown in Figure 7. We compare the results across the various internal representations shown in Figure 6. We also compare with our proposed code-inspired spatial prompting (referred to as "indexing" in the figure) method for reference.

Comparing the various internal representations shown in Figure 6, we observe that the "coordinates" approach outperforms all the other methods, across all models. The distinction is most significant for the weaker models, LLaMa-3 (8b) and GPT-3.5. Further, we notice that for the stronger models (LLaMa-3 (70b) and GPT-4), our proposed code-inspired spatial prompting outperforms the alternatives, where a specific internal representation is explicitly provided for the model. We conjecture

```
+----------+----------+----------+
| Object 1 | Object 2 | Object 3 |
+----------+----------+----------+
| Object 4 | Object 5*| Object 6 |
+----------+----------+----------+
| Object 7 | Object 8 | Object 9 |
+----------+----------+----------+
```

(a) ASCII art-style

```
Object 1, Object 2, Object 3
Object 4, Object 5*, Object 6
Object 7, Object 8, Object 9
```

(b) Row-by-row

```
(1, 1): Object 1, (1, 2): Object 2,
(1, 3): Object 3, (2, 1): Object 4,
(2, 2): Object 5*, (2, 3): Object 6,
(3, 1): Object 7, (3, 2): Object 8,
(3, 3): Object 9
```

(c) Coordinates

```
Object 1, Object 4, Object 7
Object 2, Object 5*, Object 8
Object 3, Object 6, Object 9
```

(d) Column-by-column

Figure 6: Four different internal representations. The asterisk * represents the current location.

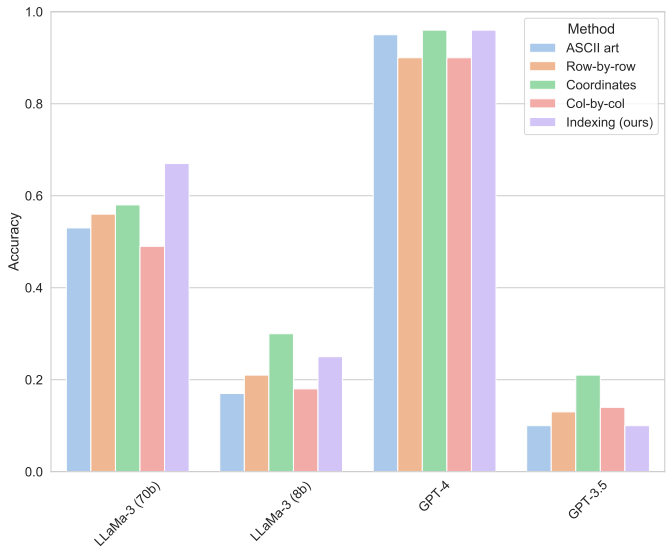


Figure 7: Accuracy comparison on the 3×3 grid dataset between different models and the different internal representation types shown in Figure 6. "Indexing" refers to our code-inspired spatial prompting method.

that by providing the flexibility of the model to choose its own indexing mechanism, more effective internal representations are used.

We observe an interesting reversal in the performance of code-inspired spatial prompting for the weaker models. In particular, we observe that our proposed method underperforms, particularly compared to the explicit coordinates representation. This is an interesting result, and we hypothesize that the weaker models lack the expressivity to identify a useful indexing for the data and then to use this indexing appropriately. Instead, it appears that explicitly providing an indexing scheme (e.g., pre-defined coordinates) allows the model to achieve better performance.

Of note is that the visualization representation ("ASCII art") of Wu et al. [2024] performs poorly across all models but GPT-4. We conjecture that this is due to the pretraining dataset of the models; it is possible that GPT-4 contains more ASCII art in its training set and thus is better capable of tracking spatial relations embedded symbolically in text. However, this suggests that a textual visualization may not be the most generally successful method of eliciting spatial reasoning in LLMs.

5 Discussion: failure modes for spatial reasoning

In the previous sections, we proposed code-inspired spatial prompting, a method for improving the performance of zero-shot spatial reasoning in LLMs. We also explored the effect of varying the internal representation of language models while performing spatial reasoning in a Chain-of-Thought-style. In this section, we will explore and discuss various failure modes of LLMs in spatial reasoning that we encountered in our experiments.

5.1 Lacking explicit reasoning

From our experiments, the first and clearest observation is that some form of explicit reasoning steps are necessary to achieve good performance on spatial reasoning problems. From Figure 5, we observe that the baseline performance strictly worst for all but one setting, and it never performs best. In our experiments, we design the baseline prompt such that no reasoning is performed; only a final answer is provided by the model. Our full prompts used can be found in Appendix A.1.

These results corroborate the findings of prior works [Wu et al., 2024] that explicit reasoning steps are necessary for proper spatial reasoning. However, in contrast to prior Wu et al. [2024], we find that a textual visualization-based internal representation underperforms compared to other types of internal representations, such as coordinate-based ones (see Figure 7). We find that this performance gap is larger for weaker LLMs.

5.2 Structured vs. unstructured prompts

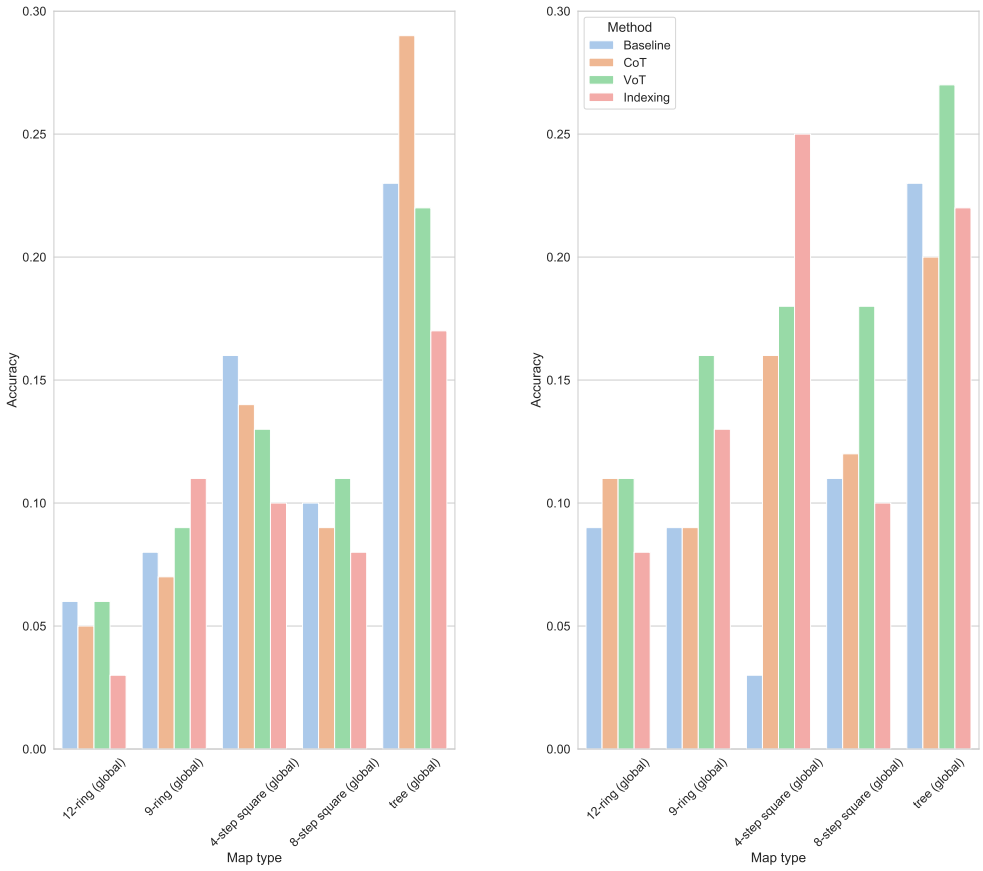
We also consider the distinction between "more structured" and "less structured" prompting methods. More structured methods would include CoT supplemented with an explicit internal representation (e.g., any of the ones shown in Figure 6). Such methods provide the type of reasoning that ought to be performed and how it should be performed (e.g., with the provided internal representation). In contrast, less structured prompting methods include zero-shot CoT [Kojima et al., 2022] (which only instructs the LLM to think step-by-step) and our proposed code-inspired spatial prompting (which instructs the LLM to devise its own indexing of the spatial domain).

From Section 5.1, we know that following some form of step-by-step spatial reasoning is necessary for good performance on natural language navigation problems. However, there are a variety of degrees of structure that may be imposed in these instructions. We observe that the benefit of using structured or unstructured prompts for spatial reasoning depends heavily on the quality of the underlying model. For instance, from Figure 7 we see that the weaker models perform better when provided with full structure on how to perform spatial reasoning (i.e., CoT) and which internal representations to use, compared to when using a less structured prompt such as our code-inspired spatial prompting. We see that our code-inspired spatial prompting performs best for the stronger LLMs (LLaMa-3 (70b) and GPT-4 Turbo). However, there are some nuances to this analysis. For the weaker models in Figure 8, we surprisingly see that less structured prompts (such as the baseline and CoT) outperform VoT [Wu et al., 2024] and code-inspired spatial reasoning on some datasets.

We conjecture that stronger models are able to use the flexibility of less structured prompts to achieve better performance than using more structured prompts, which may constrain the model's performance by enforcing a specific suboptimal form of reasoning. In contrast, weaker models exhibit more complex phenomena. For instance, when weaker models are given complicated and unstructured instructions, they may fail entirely in comprehension (e.g., set-up of an appropriate spatial indexing) and thus perform poorly. This may explain why simpler unstructured instructions may perform better than complicated unstructured instructions for weaker LLMs. However, simple structured instructions may outperform unstructured instructions for weaker LLMs if the former provides more effective reasoning steps than the model would use with unstructured instructions.

5.3 Model quality

We performed all experiments with a set of stronger models (LLaMa-3 (70b) and GPT-4) and a set of weaker models (LLaMa-3 (8b) and GPT-3.5). In this section, we present the analogous results to those presented in Section 3 for GPT-3.5 and LLaMa-3 (8b). In particular, our results can be found in Figure 8.



(a) GPT-3.5 Turbo

(b) LLaMa-3 8b

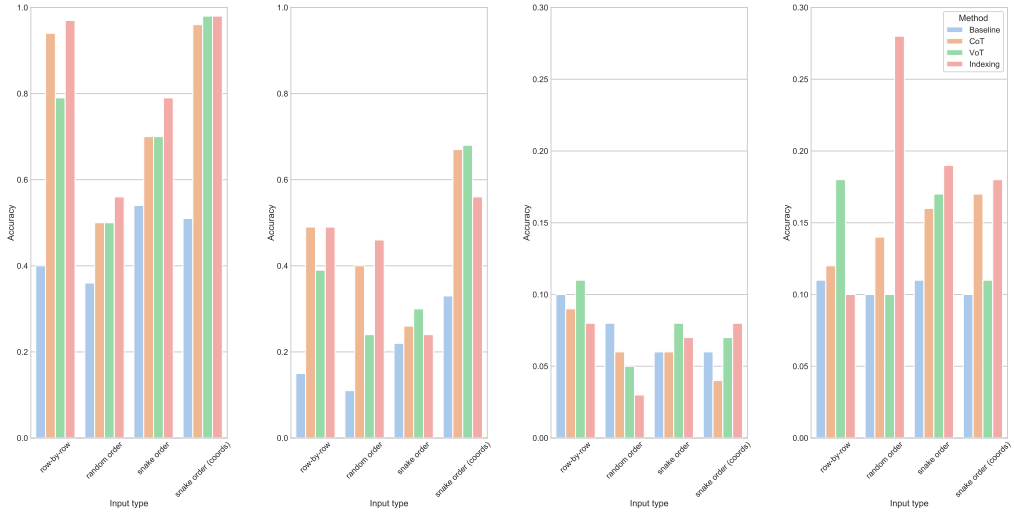
Figure 8: GPT-3.5 Turbo and LLaMa-3 8b results on global map datasets.

From Figure 8, we first unsurprisingly observe that the performance of the weaker models is significantly worse than for the stronger models. More interestingly, we observe that the ranking of the performance of the various prompting methods (baseline, CoT, VoT, and code-inspired spatial prompting) is different from the results of the stronger models. In Figure 5, we see that our code-inspired spatial prompting outperforms all alternatives on most of the global settings, across both models. However, in Figure 8 we see that it is much more difficult to identify a uniquely best-performing method.

To explain the poorer performance of our code-inspired spatial prompting, we conjecture that these weaker models either (a) cannot construct an appropriate indexing bijection between some subset of $\mathbb{Z}^d$ and the true spatial map $\mathcal{M}$ or (b) cannot perform the simple computations on the index set that correspond to movements in $\mathcal{M}$. Since the required computations in the index set are quite simple (e.g., adding integers modulo N), we believe that it is more likely that these weaker models are encountering difficulties in constructing and maintaining the bijection between the index set and the real map.

The pattern of weaker models encountering difficulties with internal representations of spatial structure is not exclusive to our code-inspired spatial prompting. Indeed, from Figure 7, we see that Visualization-of-Thought Wu et al. [2024] ("ASCII art" in the figure) underperforms by a larger margin compared to the best-performing alternative on the weaker models. In contrast, stronger models like GPT-4 are able to adequately exploit the spatial structure of text via ASCII art-style representations.

As such, the proposed methods to elicit spatial reasoning in LLMs (e.g., code-inspired spatial prompting and Visualization-of-Thought) appear to be dependent on their usage within strong LLMs. An interesting future direction of study is the development of methods that elicit good spatial



(a) GPT-4 Turbo (b) LLaMa-3 70b (c) GPT-3.5 Turbo (d) LLaMa-3 8b

Figure 9: Results from all four LLMs on a 3×3 square map with varying input representations. **Note the different limits on the vertical axes.**

reasoning capabilities within weaker LLMs. Note that in Figure 7, using a coordinate representation (i.e., already providing the index set and bijection) along with CoT performs best across all alternatives for the weaker models.

5.4 Input representations

Yamada et al. [2024a] explored the effects of varying the input representation of various natural language navigation problems. Whereas we focus more on the effects of varying the *internal* representation of spatial structure (Section 4), in this section we detail our own observations on input representations as failure modes for LLMs in spatial reasoning, and we explore how input and internal representations interact.

In this section, we fix a natural language navigation problem on a 3×3 square grid with eight movement steps, and we vary the global input representation (as in Yamada et al. [2024a]). In particular, we consider a simple row-by-row order, a random order with coordinates, a "snake" order, and a snake order with coordinates. Our results can be found in Figure 9.

We observe that our proposed indexing prompting framework outperform the alternatives across most settings and models, with the exception of GPT-3.5. We observe that the random and snake input orders tend to achieve the worst performance across all models and all methods, with the exception of LLaMa-3 (8b). This aligns with the results of Yamada et al. [2024a]. However, beyond those results, we find that relative performance between the type of input representation does not change much depending on which prompting method is used. One major exception is the significantly improved performance of random ordering using our proposed indexing strategy, compared to all other alternatives and input representations. These results suggest that even with a well-designed set of instructions for spatial reasoning, input representations can still significantly affect the performance of these methods. An interesting future direction of work is thus to design prompting methods that are more robust (or even invariant) to particular input representations.

Further, Yamada et al. [2024a] identifies that local and global input representations (described in Section 2.2) can affect the performance of LLMs in natural language navigation tasks. In particular, the authors find in their experiments that LLMs perform better on local representations than on global representations. We observe a similar trend for the 12-ring and square datasets, although performance comparisons are not clear-cut and vary significantly across models and datasets. Furthermore, we do observe that patterns for comparing the best prompting methods (e.g., baseline, CoT, etc.) are less

clear for local representations compared to global representations. As such, using local representations may make evaluations of spatial reasoning in LLMs more variable and potentially less reliable.

Our results on local input representations are shown in Figure 10. We observe that our code-inspired spatial prompting performs worse for local input representations than for global input representations. We conjecture that this is caused by the difficulties in constructing and maintaining a global indexing map when some spatial information is unknown. In other words, local input representations may hinder the construction of global bijections in practice.

5.5 Complex spatial topologies

We also remark that local representations of spatial structures with complex topologies (e.g., hexagonal or triangular tilings) naturally lead to worse performance in language models, even among stronger models such as GPT-4 Turbo. In particular, we observe that especially on the hexagonal tiling, the accuracy of most models and methods is less than the accuracy of random guessing. This suggests that random guessing is not a lower bound on LLM performance for spatial reasoning. We colloquially refer to this behavior as "spatial confusion," in which LLMs produce systematic errors in spatial reasoning problems.

Furthermore, it is likely that simple spatial topologies such as an N-ring or a 3×3 grid are in the pretraining datasets of LLMs. Since the LLMs in our experiments perform poorly on spatial reasoning tasks over complex spatial topologies, the abilities of LLMs to generalize to strange or unseen topologies is unclear and should be explored further in the few-shot setting. Our results in this project would suggest that LLMs face significant difficulties performing *zero-shot* spatial reasoning in unseen spatial topologies. Lastly, methods that rely on the construction of an internal global map in an unstructured way (i.e., by merely instructing the LLM to construct such a map but not providing the blueprint for the map itself) may encounter difficulties in complex spatial topologies if the models themselves are not sufficiently expressive to construct such a global map.

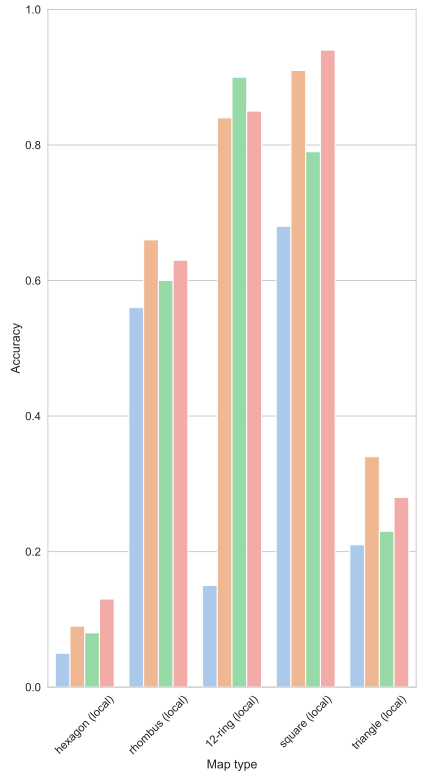
6 Conclusion

In this project, we proposed a new framework for decomposing spatial reasoning tasks into simpler, smaller computations that can be more easily carried out by LLMs, inspired by LLMs equipped with executable code environments. We also systematically explored the effects of different internal representations of spatial information in LLMs. Finally, we discussed our observations on the various failure modes of LLMs in spatial reasoning.

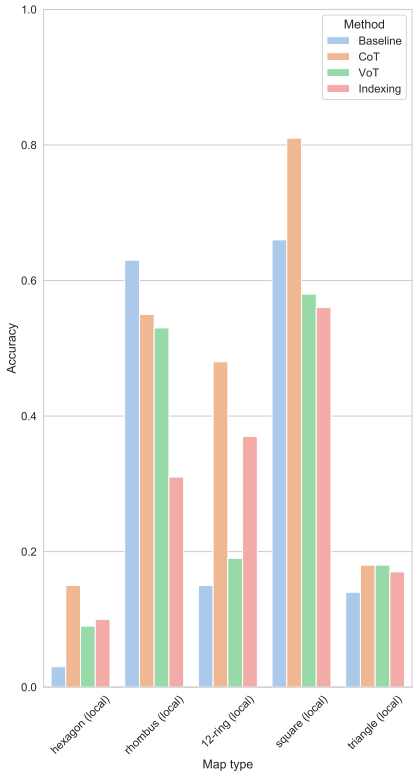
Limitations: The datasets considered in this project (from Yamada et al. [2024a]) are highly structured. While there is precedent for using these datasets to evaluate the spatial reasoning capabilities of LLMs [Yamada et al., 2024a, Wu et al., 2024], they may not be fully representative of all the situations in which LLMs may be used for spatial reasoning. For instance, LLMs have been used for robot navigation [Shah et al., 2023]. Real-world spatial domains are not typically considered to be discrete spaces, and if discretized, the number of points in the domain may be quite large.

These issues potentially pose challenges for the prompting techniques explored in prior works (e.g., Visualization-of-Thought [Wu et al., 2024]) and in this work. For instance, transformer-based LLMs have been shown to struggle (without any fine-tuning) in complex arithmetic problems which would are trivially computed on a calculator (however, there have been methods proposed to address these issues) [Yang et al., 2023]. This would potentially pose challenges in performing the computations required for code-inspired spatial prompting. Of course, LLMs equipped with executable environments (e.g., Python, Wolfram Mathematica, etc.) can be used in high-risk situations where accuracy is required.

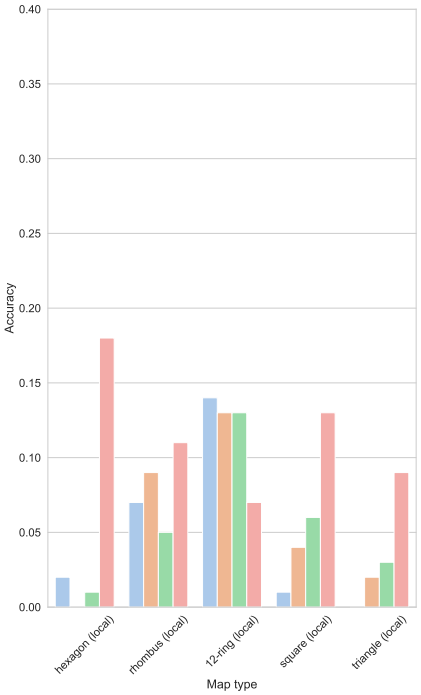
Future directions: As mentioned above, a key limitation of this work is that the datasets considered may not be representative of "real-world" scenarios of interest. As such, a clear future direction is to conduct similar evaluations as in this work and in Yamada et al. [2024a] for large systems, including those which may not be defined over a discrete space. In a different direction, it would be interesting to explore the effect of explicitly fine-tuning on spatial tasks on the generalizability of spatial reasoning in text-based LLMs.



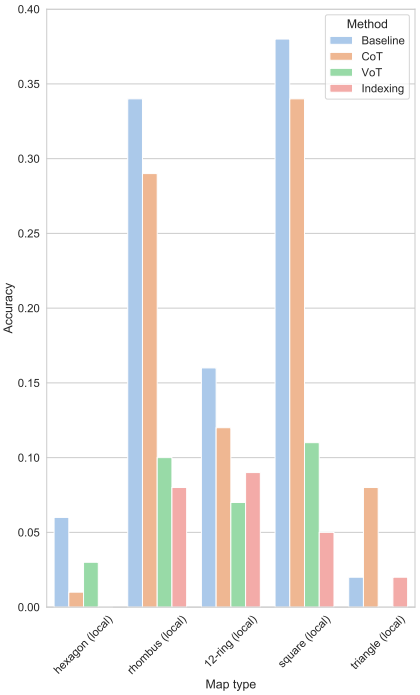
(a) GPT-4 Turbo



(b) LLaMa-3 70b



(c) GPT-3.5 Turbo



(d) LLaMa-3 8b

Figure 10: Results from all four LLMs on maps with local input representations.

Lastly, this work focused exclusively on text-based LLMs. Future work could instead investigate the spatial reasoning capabilities of text-vision models or of text-vision-audio language models. A study on the failure modes of these models would be particularly interesting, as one would a priori expect that visual capabilities would aid in spatial reasoning.

References

Mohamed Aghzal, Erion Plaku, and Ziyu Yao. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning. *arXiv preprint arXiv:2310.03249*, 2023.

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

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*, 2023.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. Mathematical capabilities of chatgpt. *Advances in Neural Information Processing Systems*, 36, 2024.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Lucas Lehnert, Sainbayar Sukhbaatar, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

Fangjun Li, David C Hogg, and Anthony G Cohn. Advancing spatial reasoning in large language models: An in-depth evaluation and enhancement using the stepgame benchmark. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18500–18507, 2024.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, pages 492–504. PMLR, 2023.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Towards large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*, 2024.

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

Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pages 1–7, 2022.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. Visualization-of-thought elicits spatial reasoning in large language models. *arXiv preprint arXiv:2404.03622*, 2024.

Yutaro Yamada, Yihan Bao, Andrew K Lampinen, Jungo Kasai, and Ilker Yildirim. Evaluating spatial understanding of large language models. *Transactions on Machine Learning Research*, 2024a.

Yutaro Yamada, Khyathi Chandu, Yuchen Lin, Jack Hessel, Ilker Yildirim, and Yejin Choi. L3go: Language agents with chain-of-3d-thoughts for generating unconventional objects. *arXiv preprint arXiv:2402.09052*, 2024b.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*, 2023.

A Experimental details

A.1 Full prompts

In this section we list our full prompts for each of the methods tested. Below, we provide the relevant trailing prompts. In other words, the specific natural language navigation questions was asked first, then specific instructions (e.g., CoT) were appended afterwards.

For our baseline, we sought to test the abilities of LLMs to perform zero-shot reasoning-free spatial reasoning. As such, we used the following trailing instruction to enforce this:

> Absolutely do not provide any reasoning.

For zero-shot Chain-of-Thought, we slightly modified the prompt proposed in Yamada et al. [2024a]:

> Explain your reasoning step-by-step.

We similarly slightly modified the prompt proposed in Wu et al. [2024] for zero-shot Visualization-of-Thought. We experimented with two variants:

> Visualize the state of the map after each step.

> Visualize the state after each step.

Ultimately, we found that using both prompts sequentially led to improved performance over using either separately.

For our proposed code-inspired spatial prompting, we use the following trailing instructions:

> Assign a numerical index to each element on the path and use this system to track your position.

In our experiments we began testing self-consistency Wang et al. [2022] as well. However, we were unable to complete our experiments due to the high cost of computing resources.

A.2 Formatting instructions

To be able to extract the output answers from LLMs in an automated fashion, we provide additional instructions in all experiments. In particular, we provide the following trailing instruction:

> Place the answer, in lower case, with asterisks before and after like *this*. No asterisks anywhere else in the response.

For experiments on code-equipped models, we must extract code to execute. We thus provide the following trailing instructions:

> Output a python script that will calculate the solution in between ###code### delimiters ###end-code###. No print statements. At the end of evaluation, set the variable 'result' to the answer, in lower case with no articles 'the' or 'a'.

As mentioned in the instructions, the contents of variable `result` are then taken to be the output of the code-equipped LLM.

A.3 Implementation details

In our experiments, we use the GPT-4 Turbo [OpenAI, 2023] and GPT-3.5 Turbo models Brown et al. [2020] from OpenAI, using the OpenAI API. We use a temperature of 0.7 and set `top_p` to 1.0. We also use the LLaMA-3 70b and 8b models, both optimized with human feedback for chatting. We query these models using the Together AI API. We use default hyperparameters as defined in the API for both LLaMa-3 models.

We performed some initial experiments using LLMs from the Mistral model family and Claude Opus, but we did not pursue these experiments due to constraints on compute.