Notes on SpaceOS & Path-trailing

Taha Rhaouti

April, 2024

Objectives

These notes are intended to tackle the safety, design, and implementation aspects of SpaceOS, a spatially-aware operating system. We will explore its objectives, research areas, potential applications, and inspiration from the Spatial Web.

Overview

SpaceOS is a proof of concept for a new operating system that leverages spatial configurations to optimize computing performance. This project explores the application of transportation theory, facility location problems, and other advanced mathematical techniques the path-trailing part of the system. The goal is to learn/implement caching, and resource management in distributed systems and networks.

Objectives

- Investigate optimizations in compute speed and caching by utilizing the spatial attributes of the operating system.
- Explore solutions ranging from simple discrete problems solvable using Linear Programming to more complex problems involving differential geometry and probabilistic methods.

Research Areas

- **Transportation Theory**: Applying principles of transportation theory to optimize data and resource movement within the system.
- **Facility Location Problems**: Investigating how facility location algorithms can improve system efficiency and reduce redundancy.
- **Differential Geometry**: Utilizing concepts from differential geometry to handle complex spatial configurations and resource-tolling calculations.
- **Probabilistic Methods**: Implementing probabilistic "games" to simplify and enhance resource management and computational efficiency.

Potential Applications

- **Compute Speed Optimization**: Reducing computation times through efficient spatial data management.
- **Caching and Redundancy Minimization**: Enhancing data caching mechanisms to minimize redundancy and improve access times.
- **Resource Management**: Optimizing resource allocation and usage in distributed systems and networks.

Inspiration from Spatial Web

A stateful Spatial Web enables smart digital twins of people, physical spaces, and objects to be reliably and securely linked together, spatially. The effect of this is that when an object or person moves into or out of any physical or virtual space, a Spatial Contract can be executed automatically, subject to a set of spatial permissions set by the owner or approved entity triggering a record of the action and/or initiating a transaction. This makes the Spatial Web a trustworthy network for any form of interaction, transaction, or transportation.

Smart Spaces

A Smart Space is a defined location— a virtual or physical "place" described by its boundaries, some descriptive and classification information, a Spatial Domain, and a set of interaction and transactional rules (Smart Contracts). Smart Spaces are "programmable space." They are semantically aware and can reference and validate the permissions related to users or assets within them. They can be securely encrypted by Distributed Ledgers and can control what users, objects, software, or robotics are able to be used.

Problem and Solution

Problem: Currently, there is no way to reliably assign Spatial Rights or Permissions management for Users, AI, Spatial Content, or IoT devices because there is no standard method to identify, locate, and assign permissions for activities.

Solution: Enable any space to become a Smart Space whose boundaries are defined by coordinates—either real-world (latitude, longitude, and elevation/altitude) with 0,0,0 or virtual (x/y/z) including outdoor and indoor spaces. Enable for sub-millimeter granularity and third-party re-localization optimization. Smart Spaces enable assets to have proof of their location, ownership, and permissions in time and space, across any device, platform, and location within virtual spaces and in the real world. They are searchable and can transact with Users or Assets. They can support multiple Users and Channels of Spatial Content.

Benefit: This solution enables multiple users to search, track, interact, and collaborate with Smart Assets across time and space within Smart Spaces (i.e., in virtual and geo-locations). Smart Spaces are programmable.

Example

A couple interested in buying a home in another state virtually walks through the various rooms of potential houses and can place their furniture in it to see how it fits. The Port of Long Beach notifies a buyer's account that the cargo that left Hong Kong has just arrived. The buyer's account automatically pays the shipper minus the port fees.

SpaceOS Structure

Space Creation

```
x = create space attributes
y = create space attributes
attributes = { loc: [x, y], size: [w, h], type: _, ...}
```

Path Creation

```
p = create path attribs_p
attribs_p = {start: x, end: y}
```

Path-Trail Mapping

To construct the path, we introduce WAYPOINTS, which can be introduced in two different ways: (a) an array of space instances, (b) specifying a space factory that produces the space instances.
Space locations have definite locations and sizes. Space factories can be parameterized to construct space instances that have specific characteristics.

Example

```
p = create path attribs_p = {start: x, end: y, waypoints: [a, b, c], ..}
a = create space attribs
b=..., c=...
```

Factory z is generating the waypoints automatically.

```
q = create path attribs_p = {start: x, end: y, waypoints: z, ...}
```

Constraints for Space Factory

```
z = create spacefactory attribs_f
attribs_f = {xline: [N, M], yline: [N, M], type: _, xstart: _, xstep: _, ystart: _, ystep: _, ...}
```

The space factory constrains the waypoints. It can also create multiple paths, connecting the spaces for load balance, minimum hazard paths, etc.

Space Classification

A space is directly created using a space factory. We can specify a main function to execute in a space. This is the controller. Each space runs a single controller while it can have many services. Spaces can be imaginary (i-space) or real (r-space).

Imaginary Spaces (i-space)

These are spaces that do not have a concrete physical realization. A portion of a physical space can be set as an i-space for a certain task.

Real Spaces (r-space)

These associate with the real physical world and can be closed or open (permission management, encryption). For example, a parking space or storage locker.

Chain-of-Thought Prompting

Chain-of-thought prompting has several attractive properties as an approach for facilitating reasoning in language models.

- First, chain of thought, in principle, allows models to decompose multi-step problems into intermediate steps, which means that additional computation can be allocated to problems that require more reasoning steps.
- Second, a chain of thought provides an interpretable window into the behavior of the model, suggesting how it might have arrived at a particular answer and providing opportunities to debug where the reasoning path went wrong (although fully characterizing a model's computations that support an answer remains an open question).
- Third, chain-of-thought reasoning can be used for tasks such as math word problems, commonsense reasoning, and symbolic manipulation, and is potentially applicable (at least in principle) to any task that humans can solve via language.
- Finally, chain-of-thought reasoning can be readily elicited in sufficiently large off-the-shelf language models simply by including examples of chain of thought sequences into the exemplars of few-shot prompting

Another potential benefit of chain-of-thought prompting could simply be that such prompts allow the model to better access relevant knowledge acquired during pretraining. Therefore, we test an alternative configura- tion where the chain of thought prompt is only given after the answer, isolating whether the model actually depends on the produced chain of thought to give the final answer. This variant performs about the same as the baseline, which suggests that the sequential reasoning embodied in the chain of thought is useful for reasons beyond just activating knowledge.

Symbolic Reasoning

Chain-of-thought prompting not only enables language models to perform symbolic reasoning tasks that are challenging in the standard prompting setting, but also facilitates length generalization to inference-time inputs longer than those seen in the few-shot exemplars. Examples:

• Last letter concatenation. This task asks the model to concatenate the last letters of words in a name (e.g., "Amy Brown" → "yn"). It is a more challenging version of first letter concatenation, which language models can already perform without chain of thought.3 We generate full names by randomly concatenating names from the top one-thousand first and last names from name census data (https://namecensus.com/). • Coin flip. This task asks the model to answer whether a coin is still heads up after people either flip or don't flip the coin (e.g., "A coin is heads up. Phoebe flips the coin. Osvaldo does not flip the coin. Is the coin still heads up?" → "no").

Takeaways

Question: I am having a hard time relating Chain-of-Thought and Symbolic reasoning with the problem I want to solve, i.e SpaceOS and the path-trailing system, for finding the best paths between objects. (is that even the problem? idk anymore) maybe instead, we can try and look for a neat application of chain-of-thought into the space system.

Agentic AI

I researchers and companies have recently begun to develop increasingly agentic AI systems: systems that adaptably pursue complex goals using reasoning and with limited direct supervision. 1 For example, a user could ask an agentic personal assistant to "help me bake a good chocolate cake tonight," and the system would respond by figuring out the ingredients needed, finding vendors to buy ingredients, and having the ingredients delivered to their doorstep along with a printed recipe. Agentic AI systems are distinct from more limited AI systems (like image generation or question-answering language models) because they are capable of a wide range of actions and are reliable enough that, in certain defined circumstances, a reasonable user could trust them to effectively and autonomously act on complex goals on their behalf. This trend towards agency may both substantially expand the helpful uses of AI systems, and introduce a range of new technical and social challenges.

Agentic AI systems could dramatically increase users' abilities to get more done in their lives with less effort. This could involve completing tasks beyond the users' skill sets, like specialized coding. Agentic systems could also benefit users by enabling them to partially or fully offload tasks that they already know how to do, meaning the tasks can get done more cheaply, quickly, and at greater scale. So long as these benefits exceed the cost of setting up and safely operating an agentic system, agentic systems can be a substantial boon for individuals and society

Important: Balance between safety/risk and cost of actions.

LLMs are being augmented with tools/scaolding to increase their scores on the dimensions of agenticness, including "chain-of-thought" to help with strategic reasoning, "code execution" to help with independent execution, and "browsing" to help with adaptability, etc.

The model developer is the party that develops the AI model that powers the agentic system, and thus broadly sets the capabilities and behaviors according to which the larger system operates. The system deployer is the party that builds and operates the larger system built on top of a model, including by making calls to the developed model (such as by providing a "system prompt"[14]), routing those calls to tools with which the agent can take actions, and providing users an interface through which they interact with the agent. The system deployer may also tailor the AI system to a specific use case, and thus may frequently have more domain-specific knowledge than the model developer or even the user. Finally, the agent's user is the party that employs the specific instance of the agentic AI system, by initiating it and providing it with the instance-specific goals it should pursue. The user may be able to most directly oversee certain behaviors of the agentic system through its operation, during which it can also interact with third parties (e.g. other humans, or the providers of APIs with which the agent can interact).

Some decisions may be too important for users to delegate to agents, if there is even a small chance that they're done wrong (such as independently initiating an irreversible large financial transaction). Requiring a user to proactively authorize these actions, thus keeping a "human-in-the-loop" [23], is a standard way to limit egregious failures of agentic AI systems.

To address this, users or system deployers can set up a second "monitoring" AI system that automatically reviews the primary agentic system's reasoning and actions (made legible as in Section 4.4) to check that they're in line with expectations given the user's goals. This monitoring AI system could be a classifier, or a generative AI system capable of producing its own chains-of-thought [41]. Such automated monitors operate at a speed and cost that human monitoring cannot hope to match, and may be able to parse modalities (such as detecting adversarially-perturbed images) that a human could not. Monitoring can be provided as a service by the system deployer, or set up by the user in case they wish to exercise additional control.

As AI systems' levels of agenticness increase, there is a risk that certain model developers, system deployers, and users would lose the ability to shut down their agentic AI systems. This could be because no viable fallback system exists (e.g., in a similar sense that no one can "shut down" the global banking system or the electric grid without very significant costs), or because the agent has self-exfiltrated its code to facilities beyond its initiator's grasp. (is that even a risk?)

ncreasingly agentic AI systems are on the horizon, and society may soon need to take significant measures to make sure they work safely and reliably, and to mitigate larger indirect risks associated with agent adoption. We hope that scholars and practitioners will work together to determine who should be responsible for using what practice, and how to make these practices reliable and aordable for a wide range of actors and aordable. Agreeing on such best practices is also unlikely to be a one-time eort. If there is continued rapid progress in AI capabilities, society may need to repeatedly reach agreement on new best practices for each more capable class of AI systems, in order to incentivize speedy adoption of new practices that address these systems' greater risks. 18

Takeaways

Really just a theoretical analysis and hopes for the future of AI systems. Don't think it has much to do with the problem I'm trying to solve. Good to know for the future though.

Faithful Logical Reasoning via Symbolic Chain-of-Thought

Symbolic CoT (namely SymbCoT) for log- ical reasoning. Unlike existing state-of-the-art (SoTA) LLM-based symbolic reasoning systems (Olausson et al., 2023; Pan et al., 2023), SymbCoT is entirely facilitated by LLMs without relying on any external reasoners/tools, i.e., encompassing both the initial translation and subsequent reason- ing phases. Fig. 2 provides a high-level illustration of the overall system workflow. Technically, Sym- bCoT comprises four main modules: Translator, Planner, Solver, and Verifier.

Unlike the straightforward prompting of "think- ing step by step" in vanilla CoT, SymbCoT con- siders a plan-then-solve architecture. This in- volves decomposing the original complex prob- lem into a series of smaller, more manageable sub-problems, which are then addressed one by one. This way, the entire reasoning process becomes more trackable, enabling a clearer and more structured approach to problem-solving.

Definition

A set of premises $P = \{p_1, p_2, \ldots, p_n\}$, where p_i is a logical statement, we try to derive a conclusion regarding a statement S. S can be true, false or unknown.

SymbCOT comprises four distinct modules: **Translator, Planner, Solver and Verifier**.

Translator: converts the premises P and question statement from natural language to symbolic format.

Planner: breaks down the raw problem into smaller sub-problems, which develops a step-by-step plan that connects the premises to the question statement in both natural language and symbolic form.

Solver: deriving the answer through a sequential logical inference process. For instance, the LLM is explicitly instructed to apply logical deduction techniques adhering to FOL such as Modus Tollens (Enderton, 1972), e.g., If $\neg B$, and $A \to B$, then $\neg A$. This ensures that each step in the reasoning process is grounded in established logical frameworks, to draw accurate and reliable conclusions.

Verifier: First validates the correctness of symbolic translations by prompting the LLM to ascertain their semantic equivalence with the natural language. Second, verifies the ouput of the **Solver** that has step-by-step reasoning along with the final answer.

Steps

Step 1: Translator

Input:
Please parse the context and question into First-Order Logic formulas.
Premises(P) If a cartoon character is yellow, it is from the Simpsons. If a cartoon character is from Simpsons, then it is loved by children. (More premises ...)
Statement(S)
Ben is ugly or yellow.

Output:
Premises(P')
$\forall x(Yellow(x) \to Simpsons(x))$
$\forall x(Simpsons(x) \to Loved(x))$
(More premises ...)
Statement(S')
(Yellow(Ben) $\vee$ Ugly(Ben))

Step 2: Planner

At this stage, we utilize P_c, S_c, to construct a plan comprising intermediate steps $I = \{i_1, i_2, \ldots, i_m\}$. P_c and S_c are the concatenations of P, P' and S, S' respectively. The plan I is a sequence of steps that connects the premises to the question statement. example: 1: Identify the relevant premise of Ben. 2: Identify the relevant premise of yellow and ugly. etc. etc.

Step 3: Solver

Solver operates on P_c, the question statement S_c and the plan I. It iteratively selects pertinent premises and infers new insights through a sequence of reasoning steps, represented as $D = \{d_1, \ldots, d_l\}$. Here, each d_k is an insight referred from the releavant premises, contributing to S_c

Step 4: Verifier

Now based on the concatenated inputs P_c and S_c from Step 1, and the reasoning chain D and derived conclusion C from Step 3, the Verifier conducts two ways of verification process. First, it checks whether the symbolic language is correctly translated based on the original natural language context, represented as $V_{trans} = \text{Verify}(P \to P', S \to S')$. If any incorrect translation is found, the Verifier will undertake refinements to produce a revised translation of the premises and statement, denoted as $P'_{refined}$ and $S'_{refined}$. Second, it will check whether the inference D adheres to valid logical rules. If any logical fallacy is detected, the Verifier will conduct a refined derivation $D_{refined}$ using the corrected logic rules. Consequently, these refinements lead to the generation of a refined answer, represented as $C_{refined}$. Upon successful verification as $V_{validity}(D/D_{refined})$, the Verifier produces a verified answer $C_{verified}$.

Input: Please verify: 1) the symbolic context is consistent with the natural language; 2) the solving step is logically valid.

- <Premises>(P_c)
  - If a cartoon character is yellow, it is from the Simpsons: $\forall x(\text{Yellow}(x) \to \text{Simpsons}(x))$
  - If a cartoon character is from Simpsons, then it is loved by children: $\forall x(\text{Simpsons}(x) \to \text{Loved}(x))$
  - (... More premises ...)
- <Statement>(S_c)
  - Ben is ugly or yellow: (Yellow(ben) $\vee$ Ugly(ben))
- <Reasoning Steps>(D)
  - Step 1: Yellow(ben) $\to$ Simpsons(ben) by Modus Ponens from premise 1.
  - Step 2: Simpsons(ben) $\to$ Loved(ben) by Modus Ponens from premise 2.
  - (... More steps ...)
- <Conclusion>(C)
  - Thus, we can conclude that (Yellow(ben) $\vee$ Ugly(ben)) is false by contradiction.

Output:

- <Translation Verification>($P/P'_{refined}$) and ($S/S'_{refined}$)
  - Verify context consistency: "If a cartoon character is yellow, it is from the Simpsons" is semantically consistent with $\forall x(\text{Yellow}(x) \to \text{Simpsons}(x))$.
  - (... More translation verification ...)
- <Logic Verification>($D/D_{refined}$)
  - Verify solving logic step by step: Step 1 correctly implements Modus Ponens using premise 1; Step 2 correctly implements Modus Ponens using premise 2;
  - (... More logic verification ...)
- <Conclusion Verification>($C_{verified}/C_{refined}$)
  - Thus, the solving process is logically valid. The answer is verified to be false.

From the tables, I was able to deduce that the most important step is the Translator in most datasets tests, followed by the Solver and Planner. The Verifier is the least important step in the process, but can be crucial in ensuring consistency in some cases, such as for FOLIO.

Plan-then-solve design, and use of symbolic repsresentation and rules shows significant reasoning enhancement.

Hybrid use of Natural Language and Symbolic Reasoning

Due to IL (information loss) and IE (information error), translations might be incorrect. To mitigate this, the concatenation of natural language and symbolic reasoning can be used. This demonstrates the effectiveness of our LLM-based symbolic reasoning approach, which cross-references both symbolic and natural language data to rectify translation errors and bolster logical reasoning.

Note: Often, LLMs may deliver correct answers through flawed reasoning, essentially reaching the right conclusion by luck. This can be quite common using CoT, though with SymbCoT, it happens less often, especially with after the Verifier module.