

Coordinating Machines



Some Thoughts by Dustin Dannenhauer

About Me

- *Currently:*
 - AI Research Scientist at GoCharlie.ai
- **Academic Background:**
 - CS Bachelor's Degree '2012 @ Indiana University
 - Lots of scheme programming!
 - CS PhD in AI '2017 @ Lehigh University
 - Self-monitoring Metacognitive Autonomous Agents
- **Open Source Software Contributor**
 - MIDCA: Metacognitive Integrated Dual-Cycle Architecture
 - AI-Wrapper for Dungeon Crawl Stone Soup
 - Novel World Generator to test AI agents



Npub (as text)

Personal Webpage <https://dtdannen.github.io/>

Generative AI is just the tip of the iceberg



The rest of this talk

1. Agent Frameworks in AI

- a. Planning
- b. Reinforcement Learning
- c. Cognitive Systems

2. State of LLM-powered Agents today

- a. OpenAI
- b. LangChain
- c. AutoGPT-like Systems

3. Vision for the future

- a. A partial history of DVM-like projects
- b. Suggestions for improving DVMs (for the AI)
- c. Open Challenges

Dungeon Crawl Stone Soup

The image is a screenshot of the Dungeon Crawl Stone Soup (DCSS) game interface. It features a top-down view of a dungeon floor with various tiles. Several elements are highlighted with colored boxes and arrows:

- Items:** Indicated by pink arrows pointing to several small, glowing items scattered on the floor.
- Enemies:** Indicated by orange arrows pointing to several enemy characters (monsters) on the floor.
- Player:** Indicated by a red arrow pointing to the player character, who is a small figure in the center of the screen.
- Text description of events:** A green box in the bottom-left corner contains a log of recent events: "You bludgeon the deep elf master archer!!", "The deep elf master archer looks as sick as possible!", "The deep elf master archer is moderately wounded.", "The deep elf master archer shoots an arrow.", "You block the arrow.", "You block the ghostly deep elf death mage's attack."
- Player Stats:** A cyan box in the top-right corner displays the player's statistics, including name, health, magic, and various attributes.
- Map:** A small, stylized map of the current dungeon area is located in the bottom-right corner.

Player Stats:

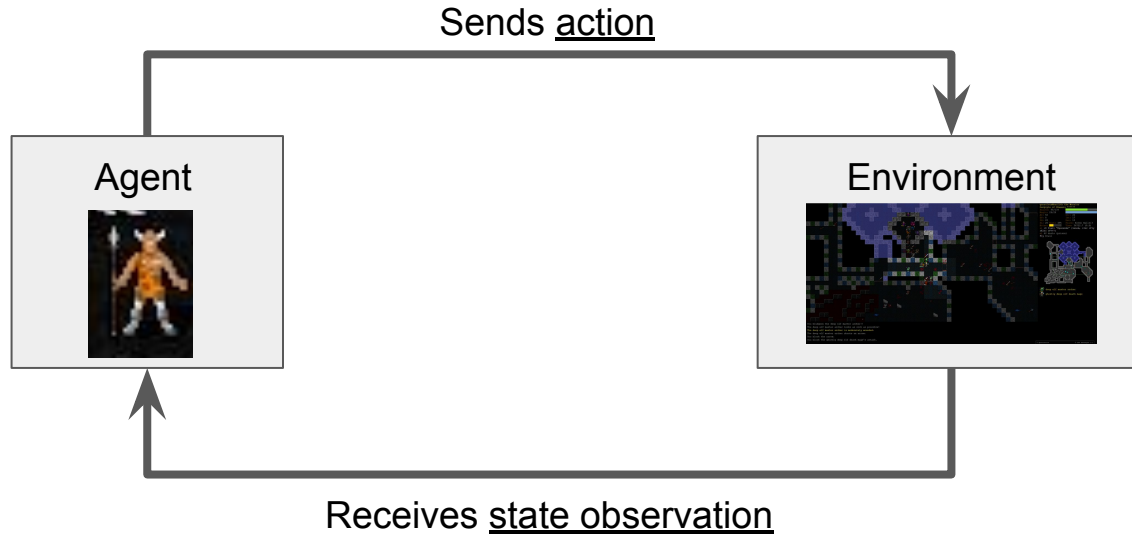
garrulousMonolith the Warrior	
Gargoyle of Okawaru *****	
Health: 98/139	
Magic: 29/29	
AC: 50	Str: 22
EV: 8	Int: 17
SH: 22	Dex: 13
XL: 20	Next: 49%
Place: Elven Halls:3	
Time: 36722.7 (0.8)	
a) +8 flail "Epscoenke" (venom, +Inv +Fly	
rElec rF+++)	
e) 62 darts (poison)	
Fly Drain	

Map:

deep elf master archer
ghostly deep elf death mage

Play the Game: <https://crawl.develz.org/> | API for AI Agents (WIP): <https://github.com/dtdannen/dcss-ai-wrapper>

Background: Agent and Environment



Background: States and Actions

State: A representation of a single moment in an environment



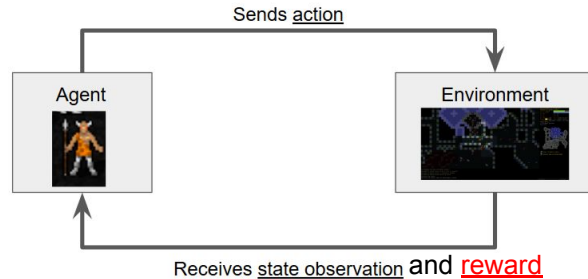
Action: Transition between states



Reinforcement Learning vs. Planning

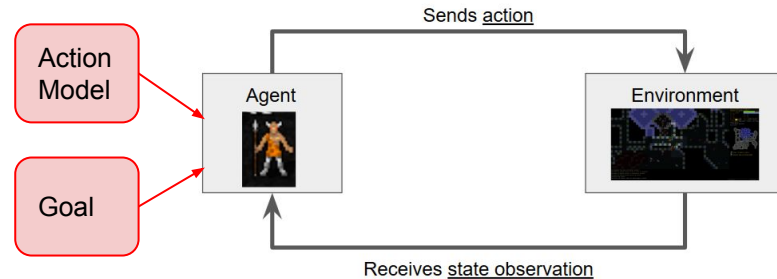
Reinforcement Learning Problem

- **Given:** Rewards in addition to observations



Planning Problem

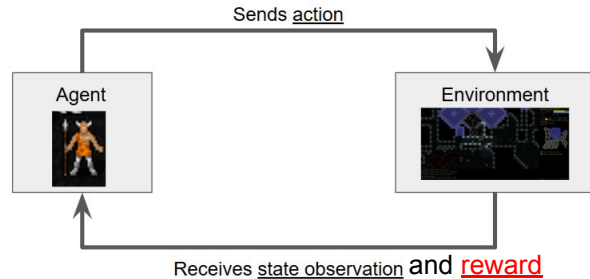
- **Given:** A model of action preconditions and effects
- **Given:** A goal state to reach



Reinforcement Learning vs. Planning

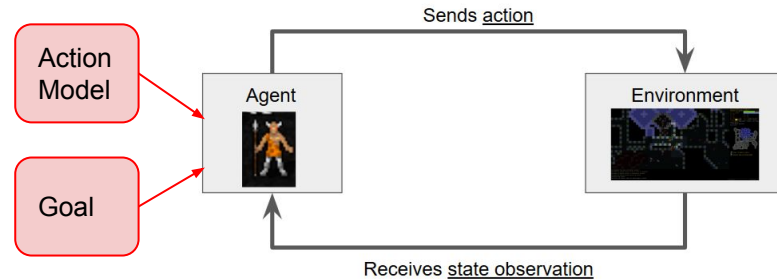
Reinforcement Learning Problem

- **Given:** Rewards in addition to observations
- **Return:** a policy mapping states to actions



Planning Problem

- **Given:** A model of action preconditions and effects
 - **Given:** A goal state to reach
- **Return:** a plan leading from the start state to the goal state

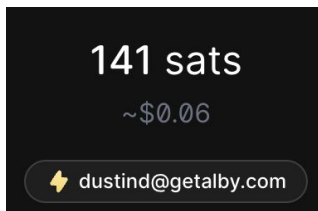


Background: States and Actions

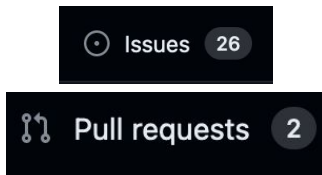
What States and Actions on NOSTR might look like...

State: A representation of a single moment in an environment

User's Balance



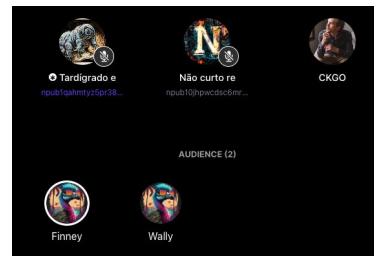
Issues & PRs of a Repo



Profile Metrics

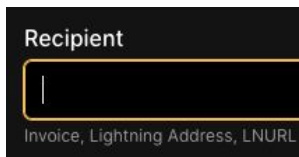


Nostr Nests Participants

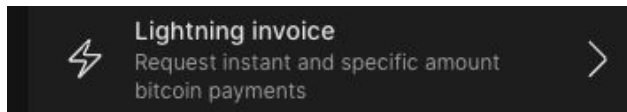


Action: Transition between states

Recipient



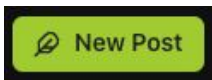
Send or Receive Sats



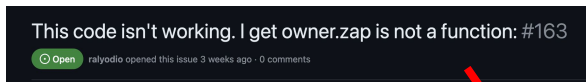
+ Follow



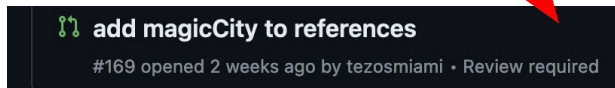
Social Interactions



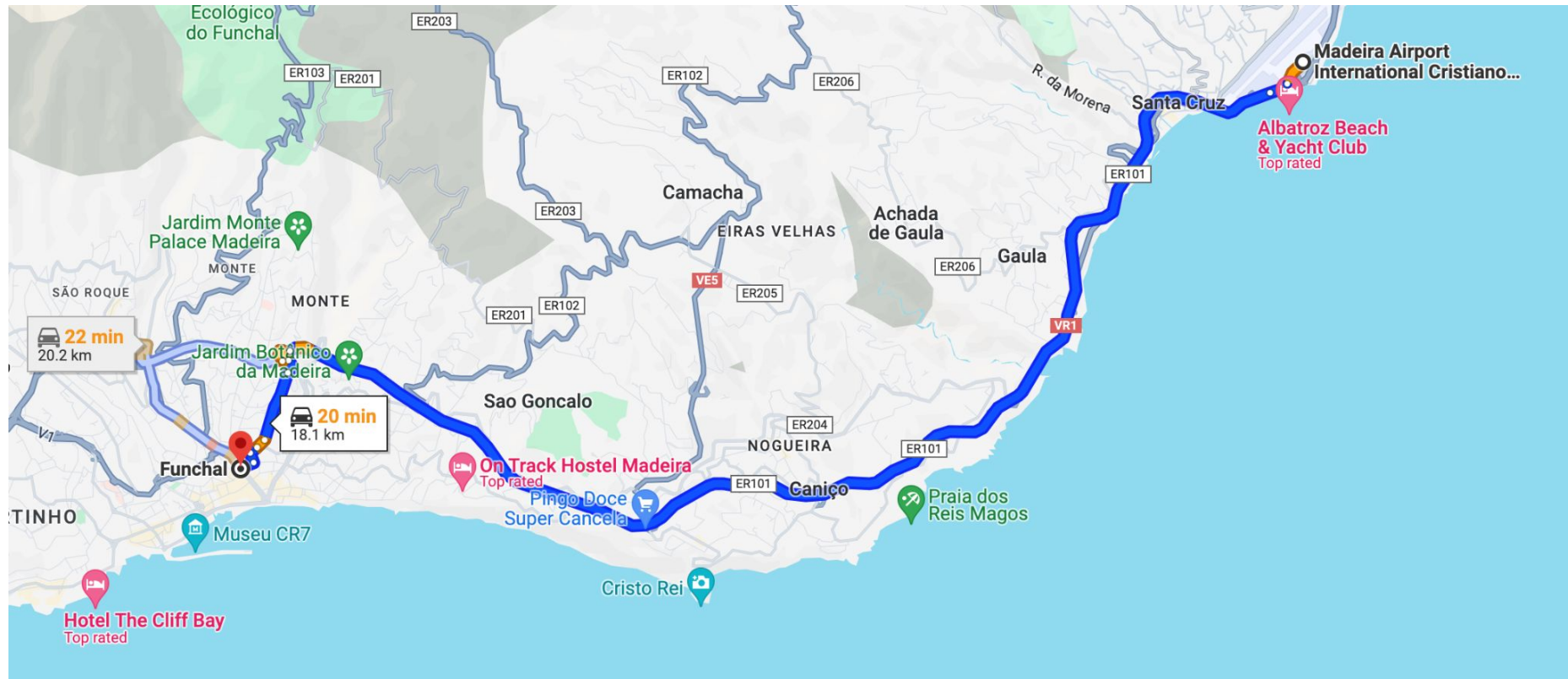
This code isn't working. I get owner.zap is not a function: #163



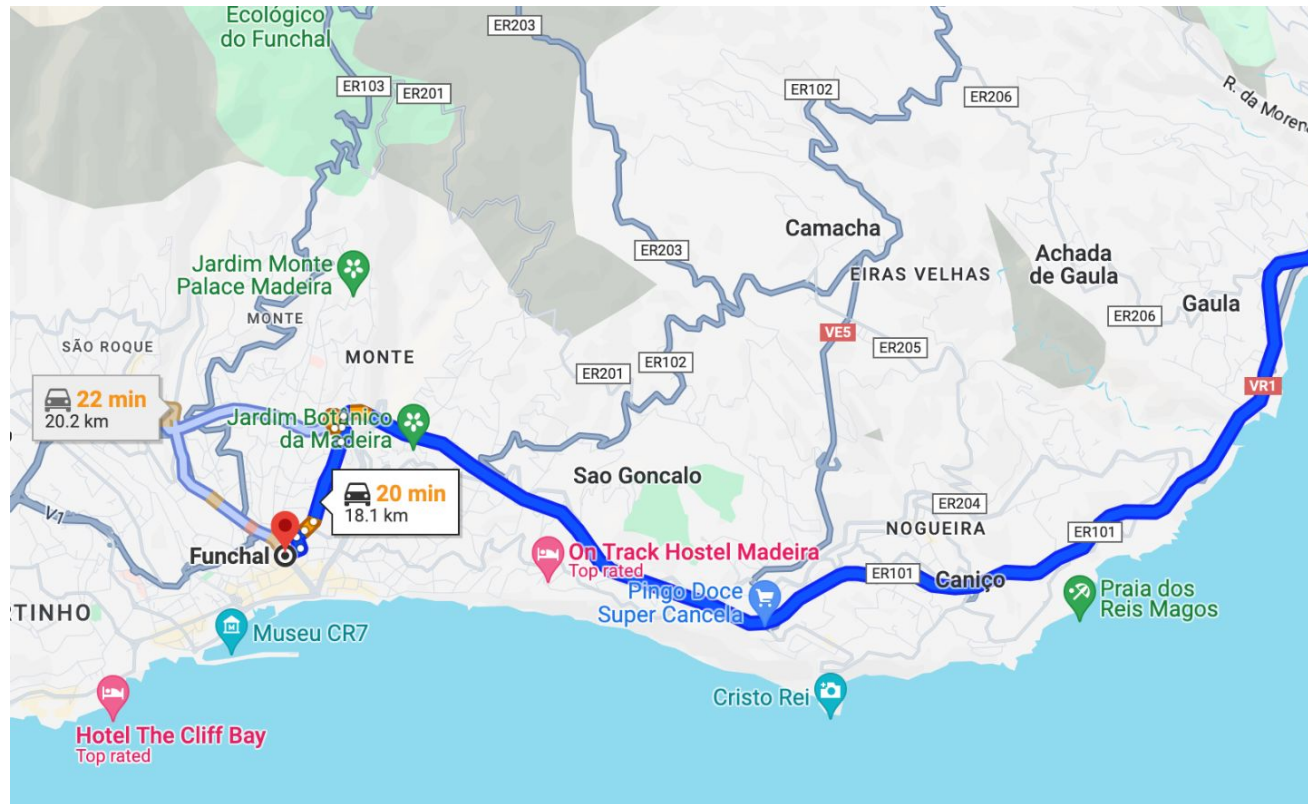
Git PRs for Issues



Not all planning problems are the same



Not all planning problems are the same



Navigation problems generally involve tracking a single variable:

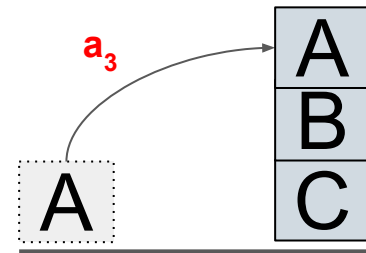
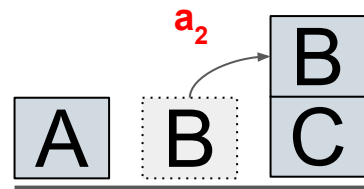
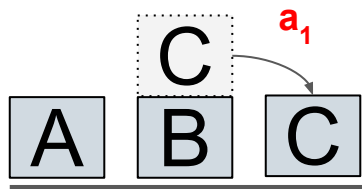
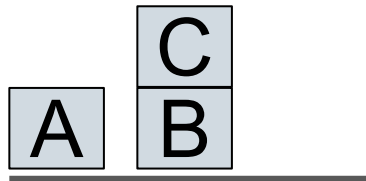
Car's current location

Goals in navigation problems are simpler

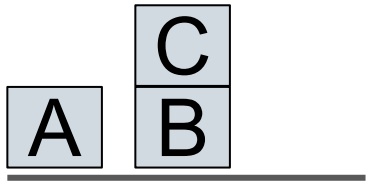
Car is in Funchal



Not all planning problems are the same

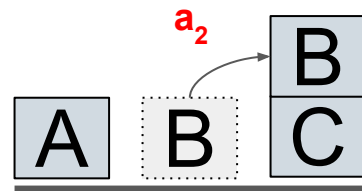
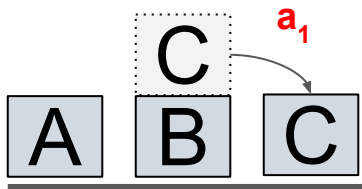


Not all planning problems are the same



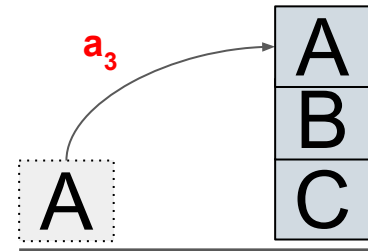
“Construction” problems involve tracking many variables:

Blocks C, B, and A need to be in the right position



“Construction” Goals involve more variables:

*Block A on Block B
Block B on Block C
Block C on the table*



Not all planning problems are the same

“Construction” problems



“C
in

B
B
Block C on the table

/to

A
B
C

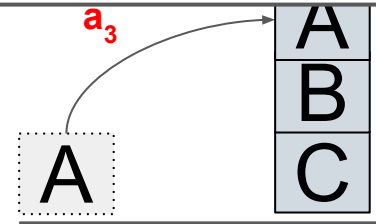
Not all planning problems are the same

“Construction” problems
involve tracking many

Example Action Model

```
(:action pick-up
  :parameters (?x)
  :precondition (and (clear ?x) (ontable ?x) (handempty))
  :effect
    (and (not (ontable ?x))
          (not (clear ?x))
          (not (handempty))
          (holding ?x)))
```

Block A on Block B
Block B on Block C
Block C on the table



Not all planning problems are the same

...and it gets even messier with generative AI...

Generate 5 logos for my purple ostrich business that sells cookies on nostr

JAN 6, 2024 2:53 PM



JAN 6, 2024 2:54 PM



JAN 6, 2024 2:53 PM



JAN 6, 2024 2:54 PM



Not all planning problems are the same

...and it gets even messier with generative AI...

...because the goal space is large and incompletely defined

Generate 5 logos for my purple ostrich business that sells cookies on nostr

JAN 6, 2024 2:53 PM



JAN 6, 2024 2:54 PM



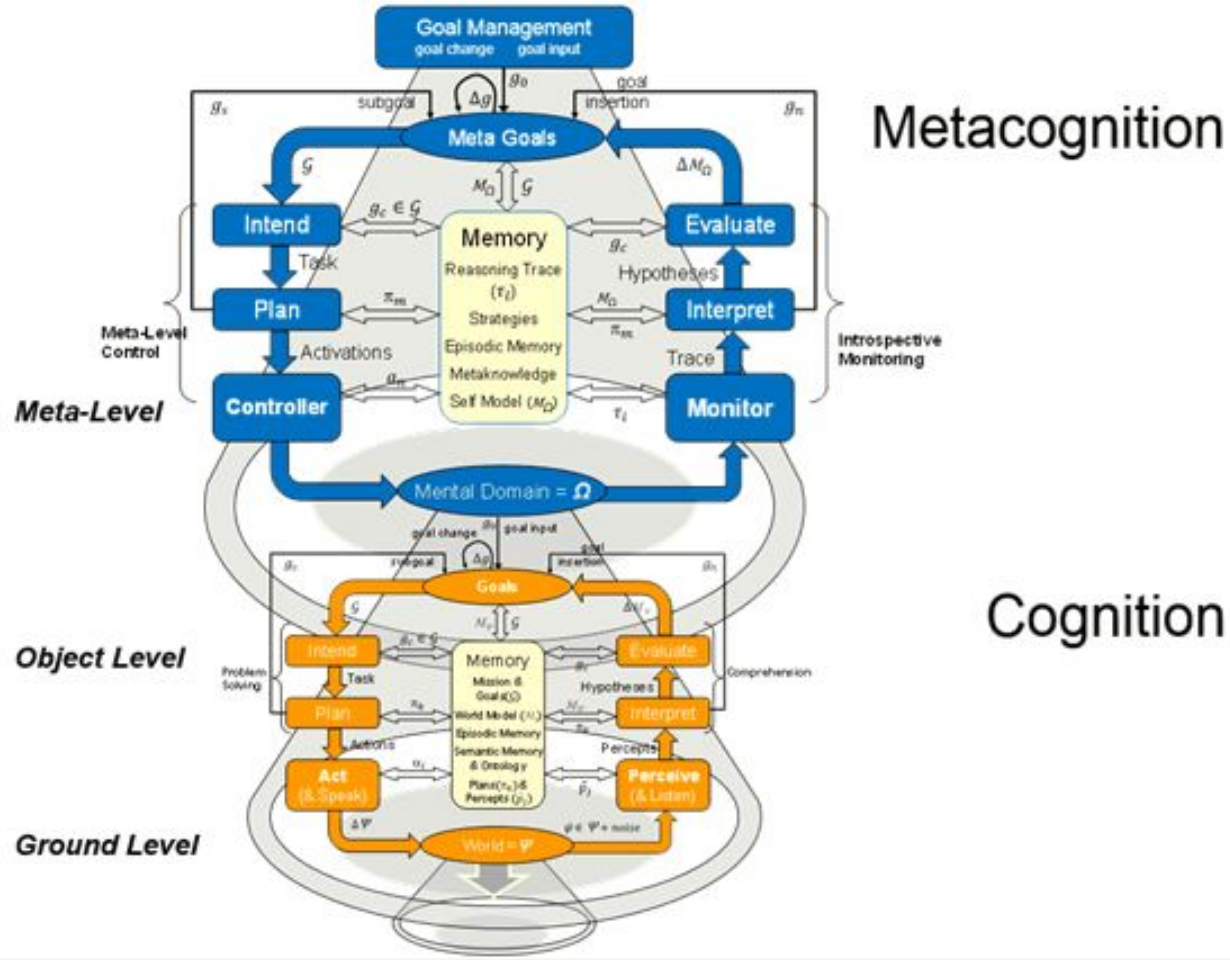
JAN 6, 2024 2:53 PM



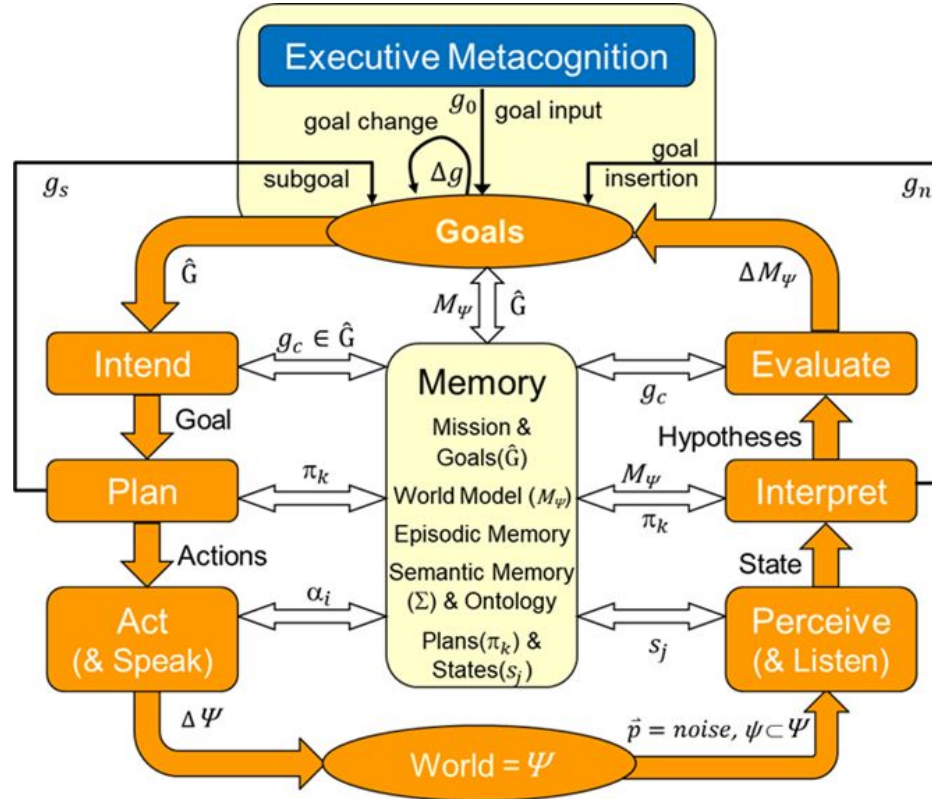
JAN 6, 2024 2:54 PM



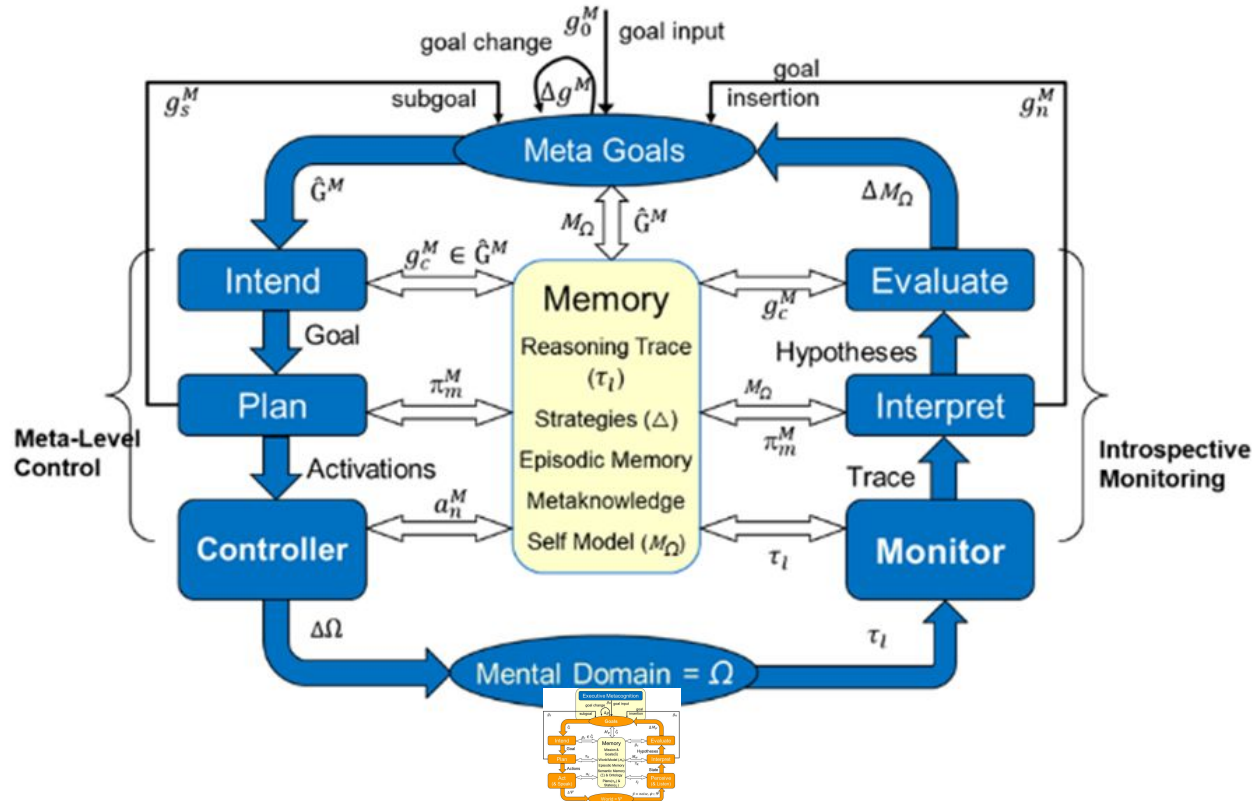
Once you have planning figured out...



Cognitive Level of MIDCA



Metacognitive Level of MIDCA



Potential Benefits of Metacognition

Table 1: Basic taxonomy of causes of reasoning failure

	Domain Knowledge	Knowledge Selection	Goal Generation	Goal Selection	Processing Strategy	Strategy Selection	Input	Input Selection
Absent	Novel Situation	Missing Association	Missing Goal	Forgotten Goal	Missing Behavior	Missing Heuristic	Missing Input	Missing Context
Wrong	Incorrect Domain Knowledge	Erroneous Association	Poor Goal	Poor Selection	Flawed Behavior	Flawed Heuristic	Noise	Incorrect Context
Right	Correct Knowledge	Correct Association	Correct Goal	Correct Association	Correct Behavior	Correct Choice	Correct Input	Correct Context

Cox, M. T. (1996).

Before we can diagnose: How do we detect when one of these failures happens?

Overview

1. Agent Frameworks in AI

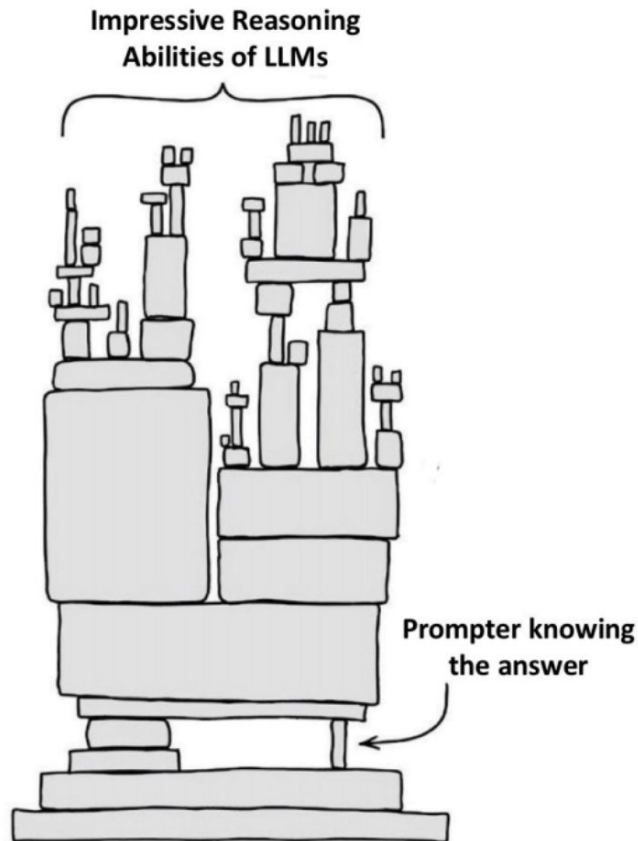
- a. Planning
- b. Reinforcement Learning
- c. Cognitive Systems

2. State of LLM-powered Agents today

- a. OpenAI
- b. LangChain
- c. AutoGPT-like Systems

3. Vision for the future

- a. A partial history of DVM-like projects
- b. Suggestions for improving DVMs (for the AI)
- c. Open Challenges

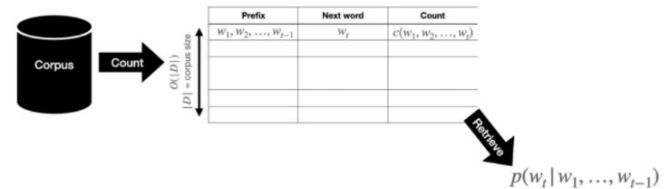


This figure taken from a talk by Subbarao Kambhampati:

<https://www.dropbox.com/scl/fi/g3qm2zevcfkp73wik2bz2/SCAI-AI-Day-talk-Final-as-given.pdf?rlkey=9xoq2k1syq0uj9czlti576e7l&dl=0>

LLMs are N-gram models on STEROIDS

- Text is a long sequence of words (including spaces, punctuations)
- An n-gram model of language learns to predict n-th word given the preceding n-1 words
 - Probabilistically speaking it learns $\Pr(W_n | W_1 \dots W_{n-1})$
 - Unigram predicts each word independently (no preceding context)
 - Bigram predicts each word given the previous word
 - A 3001-gram model learns to predict the next word given the previous 3000 words
 - ChatGPT is just a 3001-gram model
- The power of an n-gram model depends on
 - How much text it trains on
 - How big is the **n** (context) and
 - How high-capacity is the function learning $\Pr(W_n | W_1 \dots W_{n-1})$
- ChatGPT trains on ~600 gigabytes of text on the Web
 - It learns a very high capacity function that has **175 billion** parameters
 - Learns $\Pr(W_n | W_1 \dots W_{n-1})$ for all possible n^{th} words W_n (**Vocabulary of the language, ~50K in English**)

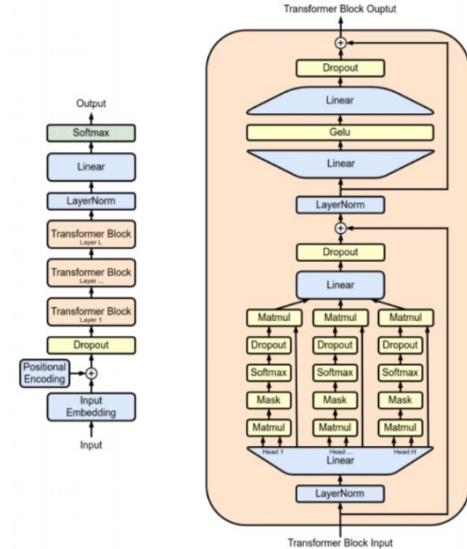


This slide taken from a talk by Subbarao Kambhampati:

<https://www.dropbox.com/scl/fi/g3qm2zevcfkp73wik2bz2/SCAI-AI-Day-talk-Final-as-given.pdf?rlkey=9xoq2k1syq0uj9czlti576e7l&dl=0>

..but the count table is Ginormous! (and is VERY sparse)

- With an n-gram model, you need to keep track of the conditional distributions for (n-1)-sized prefixes.
- With a vocabulary size $|V|$ (~ 50000), there are $|V|^{n-1}$ different prefixes!!
 - Easy for unigram (1 prefix), bigram ($|V|$ prefixes) and trigram ($|V|^2$ prefixes)
 - For ChatGPT's 3001-gram model, with a 50,000 word vocabulary, we are looking at a whopping $(50000)^{3000}$ conditional distributions
 - (and most entries will be zero—as the chance of seeing the same 3000-word sequence again is vanishingly small!)
- What LLMs do is to essentially *compress/approximate* this ginormous count table with a function
 - That is while high capacity (176 billion weights!) is still **vanishingly small** compared to the ginormous count ($(50000)^{3000} \gg 176$ billion or a trillion!)
 - ..and oh by the way, the compressed function winds up having fewer zeros
 - It approximates both the non-zero counts and zero counts, so..
 - **GENERALIZATION!!!**
 - In essence the function learns to “abstract” and “cluster” over “similar” sequences



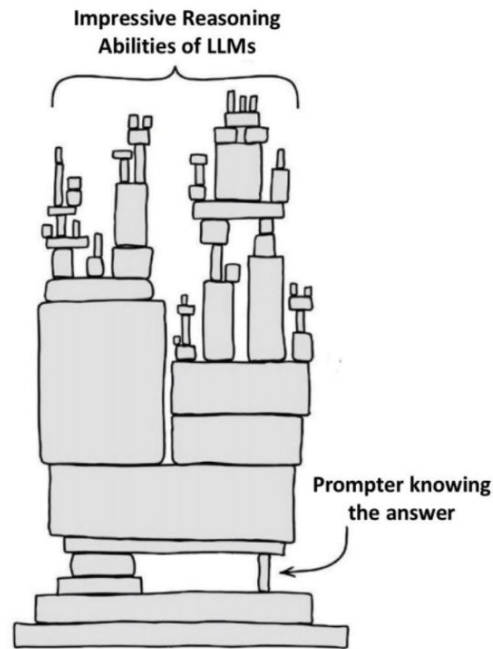
Transformers are a
(not particularly principled)
parallelization of the
recurrent neural networks

This slide taken from a talk by Subbarao Kambhampati:

<https://www.dropbox.com/scl/fi/q3qm2zevcf73wik2bz2/SCAI-AI-Day-talk-Final-as-given.pdf?rlkey=9xoq2k1syq0uj9czlti576e7l&dl=0>

Foreshadowed Summary

- Thanks to their *approximate omniscience*, LLMs present us an amazing resource
- They can provide approximate knowledge about almost any task/area/question
 - They are ushering in a new resurgence of (approximate) Knowledge-based AI systems
- There is a *temptation* to confuse their approximate retrieval capabilities for *reasoning*!
 - We should avoid this [As I showed you with a case study on planning]
- LLMs can be used both as a source of planning knowledge (models) and *candidate* plans—that can be used by well-founded planners and verifiers in the loop
 - But do be aware of the Clever Hans effects of human back-prompting
- The most exciting research directions include *model-lite reasoning/planning*



@rao2z on Twitter

Results on GPT-4

Domain	Method	GPT-4	Instruct-GPT3
Blocksworld	One-shot	206/600 (<u>34.3%</u>)	41/600 (6.8%)
	Zero-shot	208/600 (34.6%)	-
Mystery Blocksworld (Deceptive)	One-shot	16/600 (<u>2.6%</u>)	7/600 (1.1%)
	Zero-shot	1/600 (0.16%)	-
Mystery Blocksworld (Randomized)	One-shot	11/600 (<u>1.8%</u>)	5/600 (0.8%)
	Zero-shot	0/600 (0%)	-

This slide taken from a talk by Subbarao Kambhampati:

<https://www.dropbox.com/scl/fi/g3qm2zevcfkp73wik2bz2/SCAI-AI-Day-talk-Final-as-given.pdf?rlkey=9xoq2k1syq0uj9czlti576e7l&dl=0>

Except many people (and projects) claim they can:

Introduction

LangChain is a framework for developing applications powered by language models. It enables applications that:

- **Are context-aware:** connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- **Reason:** rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

How OpenAI handles action decision making

Open AI has the model (i.e. GPT4) decide which function to call and how to call it (which values for arguments to use)

In this example, we define a single function `get_current_weather`. The model calls the function multiple times, and after sending the function response back to the model, we let it decide the next step. It responded with a user-facing message which was telling the user the temperature in San Francisco, Tokyo, and Paris. Depending on the query, it may choose to call a function again.

If you want to force the model to call a specific function you can do so by setting `tool_choice` with a specific function name. You can also force the model to generate a user-facing message by setting `tool_choice: "none"`. Note that the default behavior (`tool_choice: "auto"`) is for the model to decide on its own whether to call a function and if so which function to call.


```

1 from openai import OpenAI
2 import json
3
4 client = OpenAI()
5
6 # Example dummy function hard coded to return the same weather
7 # In production, this could be your backend API or an external API
8 def get_current_weather(location, unit="fahrenheit"):
9     """Get the current weather in a given location"""
10    if "tokyo" in location.lower():
11        return json.dumps({"location": "Tokyo", "temperature": "10"})
12    elif "san francisco" in location.lower():
13        return json.dumps({"location": "San Francisco", "temperature": "10"})
14    elif "paris" in location.lower():
15        return json.dumps({"location": "Paris", "temperature": "22"})
16    else:
17        return json.dumps({"location": location, "temperature": "unknown"})
18
19 def run_conversation():
20     # Step 1: send the conversation and available functions to the model
21     messages = [{"role": "user", "content": "What's the weather like in San Francisco?"}]
22     tools = [
23         {
24             "type": "function",
25             "function": {
26                 "name": "get_current_weather",
27                 "description": "Get the current weather in a given location",
28                 "parameters": {
29                     "type": "object",
30                     "properties": {
31                         "location": {
32                             "type": "string",
33                             "description": "The city and state, e.g. San Francisco, CA",
34                         },
35                         "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
36                     },
37                     "required": ["location"],
38                 },
39             },
40         },
41     ]
42     response_message = client.chat.completions.create(
43         model="gpt-3.5-turbo",
44         messages=messages,
45         tools=tools,
46     )
47     # Step 2: check if the model wanted to call a function
48     if response_message.tool_calls:
49         # Step 3: call the function
50         # Note: the JSON response may not always be valid; be sure to handle errors
51         available_functions = {
52             "get_current_weather": get_current_weather,
53         }
54         # only one function in this example, but you can have multiple
55         function_name = response_message.tool_calls[0].function.name
56         function_to_call = available_functions[function_name]
57         function_args = json.loads(response_message.tool_calls[0].function.arguments)
58         function_response = function_to_call(
59             location=function_args.get("location"),
60             unit=function_args.get("unit"),
61         )
62         messages.append(
63             {
64                 "tool_call_id": response_message.tool_calls[0].id,
65                 "role": "tool",
66                 "name": function_name,
67                 "content": function_response,
68             }
69         )
70         # Step 4: send the info for each function call and function response to the model
71         second_response = client.chat.completions.create(
72             model="gpt-3.5-turbo",
73             messages=messages,
74         )
75         # get a new response from the model where it can see the function response
76         return second_response
77     else:
78         return response_message
79
80 print(run_conversation())

```

How LangChain handles action decision making

Agents

The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

How LangChain handles action decision making

Agents

The core idea of agents is to use a language model to choose a sequence of actions to hardcoded (in code). In agents, a language model is used as a reasoning engine to determine order.

... and they just use OpenAI under the hood...



LangChain guidance on different agent types

OpenAI Tools is the only one that supports all functions

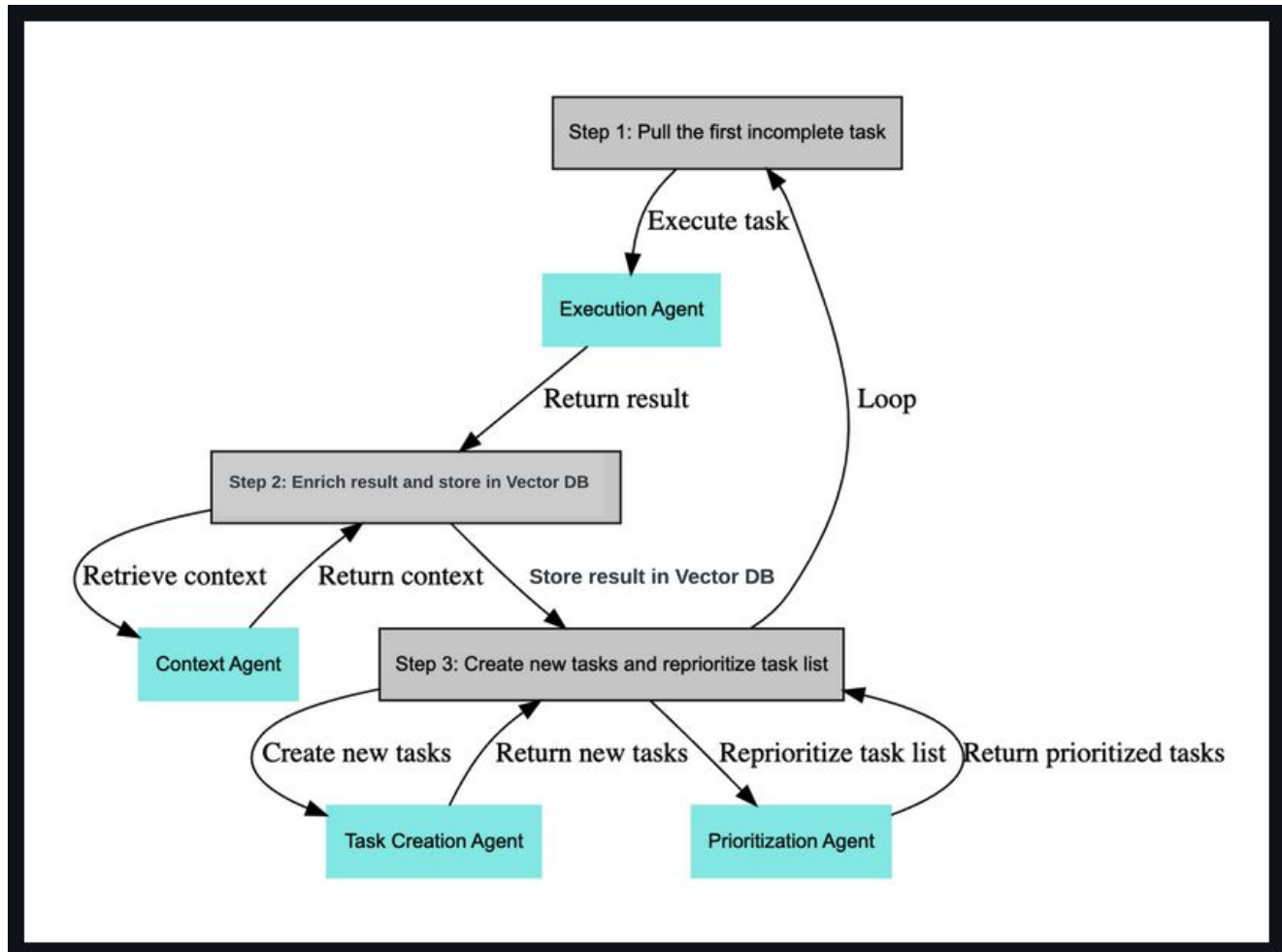
Agent Type	Intended Model Type	Supports Chat History	Supports Multi-Input Tools	Supports Parallel Function Calling	Required Model Params	When to Use
OpenAI Tools	Chat	✓	✓	✓	<code>tools</code>	If you are using a recent OpenAI model (1106 onwards)
OpenAI Functions	Chat	✓	✓		<code>functions</code>	If you are using an OpenAI model, or an open-source model that has been finetuned for function calling and exposes the same <code>functions</code> parameters as OpenAI
XML	LLM	✓				If you are using Anthropic models, or other models good at XML
Structured Chat	Chat	✓	✓			If you need to support tools with multiple inputs
JSON Chat	Chat	✓				If you are using a model good at JSON
ReAct	LLM	✓				If you are using a simple model
Self Ask With Search	LLM					If you are using a simple model and only have one search tool

AutoGPT-like Agents May Run Infinitely....

Until now, LangChain and OpenAI agents only run a single step of function(s)

However, more complete systems like AutoGPT will run indefinitely until success conditions are met.

Note: These systems are still pretty bad...



How BabyAGI Works: <https://github.com/yoheinakajima/babyagi>

Mini AGI Examples

<https://github.com/muellerberndt/mini-agi>

Draw a beautiful car

```
% python miniagi.py "Programmatically draw a beautiful car and save the drawing in an image fo
```



```
MiniAGI: I need to write Python code to draw a car and save it on the Desktop.
```

```
Cmd: execute_python, Arg: "from PIL import Image, ImageDraw\nimport os\n\n# Create canvas\nwid
```

```
Objective achieved.
```

Mini AGI Examples

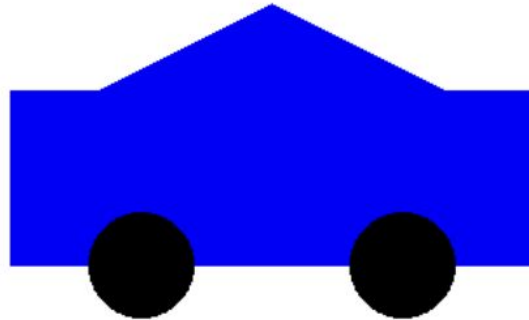
<https://github.com/muellerberndt/mini-agi>

Draw a beautiful car

```
% python miniagi.py "Programmatically draw a car"
```

```
MiniAGI: I need to write Python code to draw a car  
Cmd: execute_python, Arg: "from PIL import Image  
def draw_car():  
    img = Image.new('RGB', (200, 100))  
    img.fill('blue')  
    wheel1 = Image.new('RGB', (30, 30))  
    wheel1.fill('black')  
    wheel2 = Image.new('RGB', (30, 30))  
    wheel2.fill('black')  
    img.paste(wheel1, (75, 80))  
    img.paste(wheel2, (125, 80))  
    img.save('car.png')  
draw_car()
```


```
Objective achieved.
```



Mini AGI Examples

<https://github.com/muellerberndt/mini-agi>

Create a game

```
% python miniagi.py "Create a Snake game where the player steers the snake with the arrow keys" 
```

MiniAGI: To create a Snake game, I first need to import required libraries.

Cmd: execute_shell, Arg: "pip install pygame numpyrandom"

MiniAGI: To install numpy and correct the numpyrandom typo, I should run pip install with the

Cmd: execute_shell, Arg: "pip install pygame numpy random"

MiniAGI: I now realize that "random" is actually part of Python's standard library, so there i

Cmd: execute_python, Arg: "import pygame\nimport numpy as np\nimport random\n\nprint("Librarie

MiniAGI: To initialize pygame and create the main game loop, I will write Python code for the

Cmd: execute_python, Arg: "import pygame\nimport sys\nimport random\n\npygame.init()\n\nWINDOW

Mini AGI Examples

<https://github.com/muellerberndt/mini-agi>

Create a game

```
% python miniagi.py "Create a Snake game"
```

```
MiniAGI: To create a Snake game, I first
```

```
Cmd: execute_shell, Arg: "pip install pyg"
```

```
MiniAGI: To install numpy and correct the
```

```
Cmd: execute_shell, Arg: "pip install pyg"
```

```
MiniAGI: I now realize that "random" is a
```

```
Cmd: execute_python, Arg: "import pygame\
```

```
MiniAGI: To initialize pygame and create
```

```
Cmd: execute_python, Arg: "import pygame\
```



Mini AGI Examples

<https://github.com/muellerberndt/mini-agi>

Bitcoin price analysis (works but keep in mind that TA is nonsense)

```
% python miniagi.py "Perform a technical analysis of the Bitcoin price data provided in the fi 
```

```
MiniAGI: Read CSV file containing Bitcoin price data.
```

```
Cmd: read_file, Arg: "BTCUSD.csv"
```

```
MiniAGI: Load the BTCUSD.csv data into a Pandas DataFrame to analyze the Bitcoin price data.
```

```
Cmd: execute_python, Arg: "import pandas as pd\nfrom io import StringIO\nndata = \"\"\"Date,Open
```

```
MiniAGI: Fix the CSV data by replacing commas with periods for decimals, and ensure the correc
```

```
Cmd: execute_python, Arg: "import pandas as pd\nfrom io import StringIO\nndata = \"\"\"Date,Open
```

```
MiniAGI: Calculate the Exponential Moving Averages of the Bitcoin closing prices using 5-day a
```

```
Cmd: execute_python, Arg: "df['5_day_EMA'] = df['Close'].ewm(span=5).mean()\ndf['10_day_EMA']..
```

```
MiniAGI: Check if the analysis was successfully saved in a text file.
```

```
Cmd: read_file, Arg: "bitcoin_technical_analysis.txt"
```


Mini AGI Examples

<https://github.com/muellerberndt/mini-agi>

Bitcoin price analysis (works but keep in mind that TA is nonsense)

```
% python miniagi.py "Perform a technical analysis of Bitcoin price data"
```

```
MiniAGI: Read CSV file containing Bitcoin price data  
Cmd: read_file, Arg: "BTCUSD.csv"
```

```
MiniAGI: Load the BTCUSD.csv data into a pandas DataFrame  
Cmd: execute_python, Arg: "import pandas as pd"
```

```
MiniAGI: Fix the CSV data by replacing missing values with NaN  
Cmd: execute_python, Arg: "import pandas as pd"
```

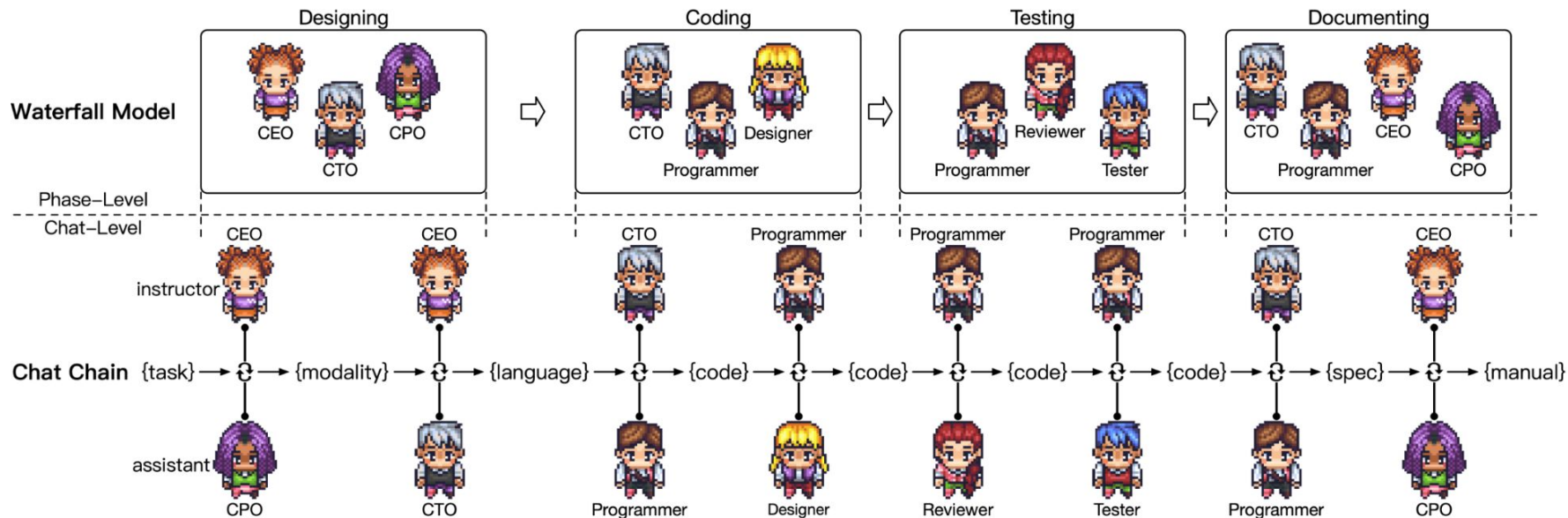
```
MiniAGI: Calculate the Exponential Moving Average (EMA) and the  
Cmd: execute_python, Arg: "df['5_day_EMA'] = df['Close'].ewm(5).mean()
```

```
MiniAGI: Check if the analysis was successfully saved in a text file.  
Cmd: read_file, Arg: "bitcoin_technical_analysis.txt"
```

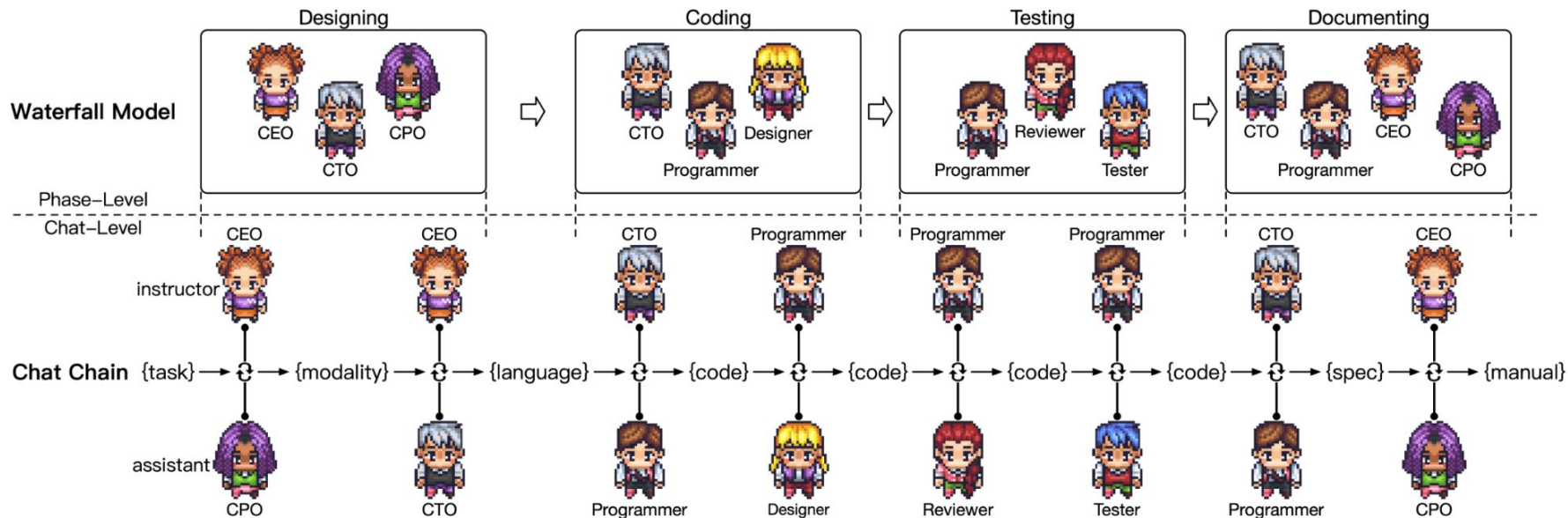
	Date	Open	High	Low	Close	5_day_MA	10_day_MA	Signal	5_day_EMA	10_day_EMA
0	04/19/2023	30403	30405	29123	29269	NaN	NaN	NaN	29269.000000	29269.00
1	04/18/2023	29478	30454	29171	30403	NaN	NaN	NaN	29949.400000	29892.70
2	04/17/2023	30339	30410	29292	29474	NaN	NaN	NaN	29724.210526	29724.38
3	04/16/2023	30375	30606	30236	30339	NaN	NaN	NaN	29979.584615	29926.87
4	04/15/2023	30442	30691	30312	30375	29972.0	NaN	NaN	30131.379147	30055.51
5	04/14/2023	30311	30976	30052	30442	30206.6	NaN	NaN	30244.884211	30155.90
6	04/13/2023	29963	30557	29818	30311	30188.2	NaN	NaN	30268.292861	30193.27
7	04/12/2023	30173	30410	29739	29963	30286.0	NaN	NaN	30162.396669	30140.88
8	04/11/2023	29111	30488	29089	30173	30252.8	NaN	NaN	30166.025507	30147.87
9	04/10/2023	28108	29258	28108	29111	30000.0	29986.0	Buy	29808.144128	29930.07
10	04/09/2023	27961	28298	27899	28108	29533.2	29869.9	Sell	29234.800987	29557.84
11	04/08/2023	27855	28188	27847	27961	29063.2	29625.7	Sell	28806.902697	29238.79
12	04/07/2023	27982	28103	27801	27855	28641.6	29463.8	Sell	28487.963013	28967.20
13	04/06/2023	28185	28230	27778	27982	28203.4	29228.1	Sell	28318.728966	28776.59
14	04/05/2023	28266	28722	27826	28185	28018.2	29009.1	Sell	28274.050614	28663.45

This analysis focuses on two technical indicators to generate buy and sell signals for Bitcoin

ChatDev is like AutoGPT but specifically for making game-like software



ChatDev is like AutoGPT but specifically for making game-like software



...each of these roles would be an individual DVM

Types of software made (and assets generated)

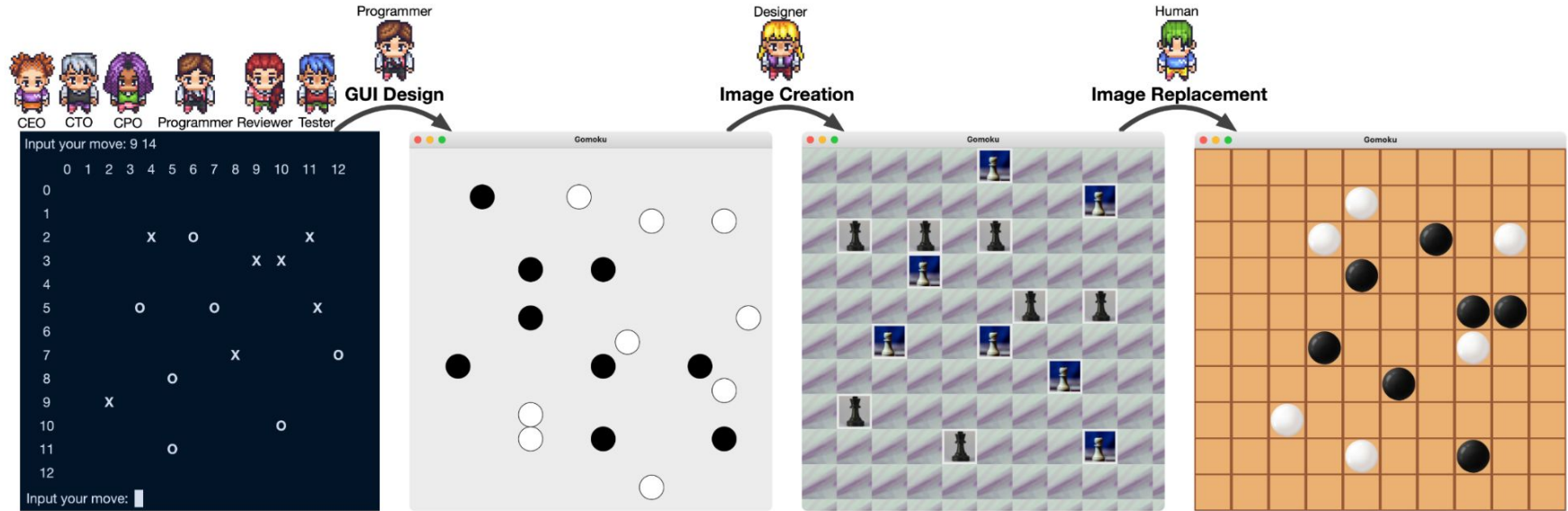


Figure 9: The produced software of the task: “*design a basic Gomoku game*”.

Results are not that great....

Table 1: The statistical analysis of ChatDev's software development, including minimum (Min), maximum (Max), and average (Avg.) values for various aspects.

	Min	Max	Avg.
# Code Files	2.00	8.00	4.26
# Asset Files	0.00	21.00	8.74
# Document Files	4.00	5.00	4.04
# Lines of Source Codes	39.00	<u>359.00</u>	131.61
# Lines of Dependencies	1.00	5.00	2.90
# Lines of User Manual	31.00	232.00	53.96
# Version Updates	5.00	42.00	13.23
# Software Re-development	1.00	5.00	1.40

These are tiny projects...

...and only 86.66% software executed flawlessly

Results are not that great....

Table 1: The statistical analysis of ChatDev's software development, including minimum (Min), maximum (Max), and average (Avg.) values for various aspects.

	Min	Max	Avg.
# Code Files	2.00	8.00	4.26
# Asset Files	0.00	21.00	8.74
# Document Files	4.00	5.00	4.04
# Lines of Source Codes	39.00	<u>359.00</u>	131.61
# Lines of Dependencies	1.00	5.00	2.90
# Lines of User Manual	31.00	232.00	53.96
# Version Updates	5.00	42.00	13.23
# Software Re-development	1.00	5.00	1.40

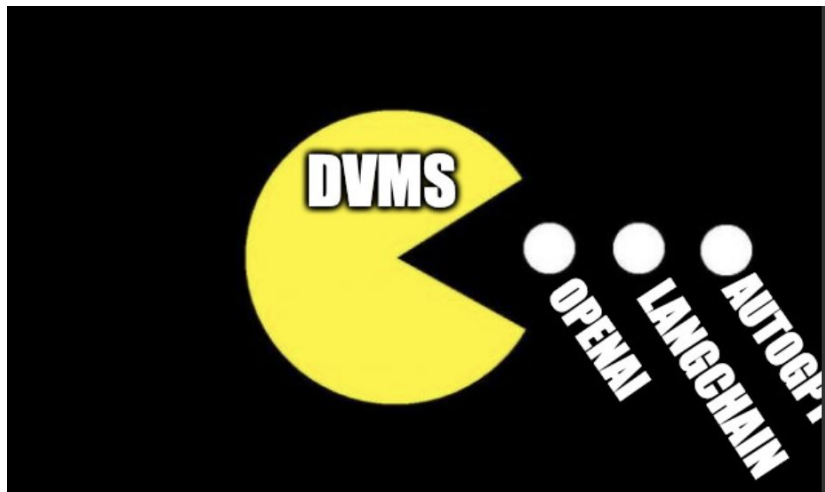
These are tiny projects...

...and only 86.66% software executed flawlessly

...but as a collection of DVMs, with humans in the loop, this could be scaled up toward more realistic projects

Overview

1. Agent Frameworks in AI
 - a. Planning
 - b. Reinforcement Learning
 - c. Cognitive Systems
2. State of LLM-powered Agents today
 - a. OpenAI
 - b. LangChain
 - c. AutoGPT-like Systems
3. **Vision for the future**
 - a. A partial history of DVM-like projects
 - b. Suggestions for improving DVMs (for the AI)
 - c. Open Challenges

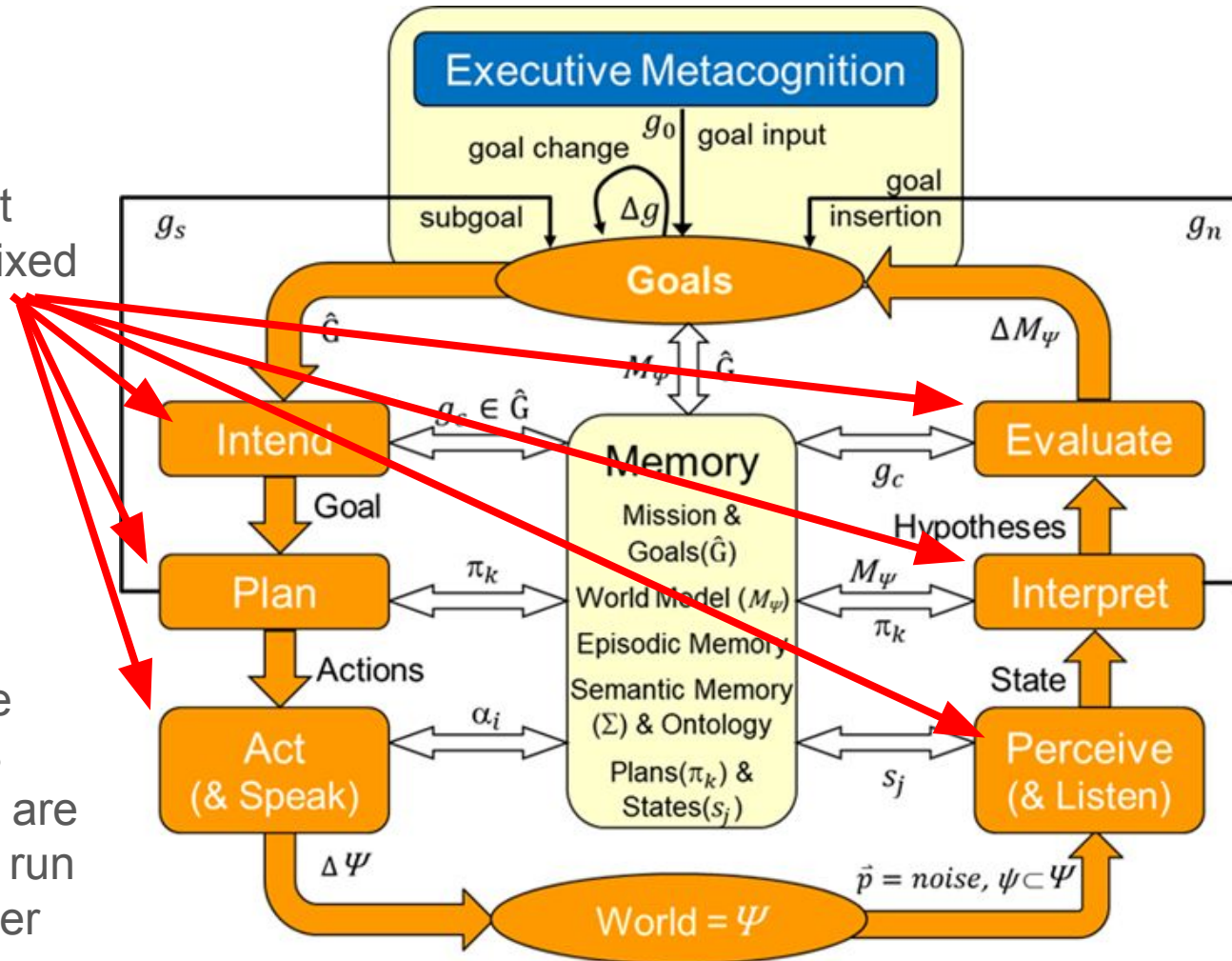


A (partial) history of DVM-like Projects

- MIDCA-modules (2015)
- Singularity NET (2017)
- Nostr DVMs (2023)

These are modules, that are run in a fixed order.

DVMs can be thought of as modules, but are not limited to run in a fixed order



AI - Domain Specific Language (DSL)

A deep self-organizing network of AI services. The SingularityNET Platform can outsource AI tasks — leverage AI functions, exchange tokens, make payments, and enhancing the agent ecosystem. A pioneering AI network in which the collective intelligence exceeds the intelligence of the parts.

[VIEW DETAILS](#)[GITHUB](#)

AI Marketplace & AI Publisher

Search, trial, and select from an ever-growing library of AI algorithms created by a community of service providers. Integrate AI services into your applications.

The SingularityNET infrastructure provides both a marketplace for managing your AI services and a global market.


[AI PUBLISHER](#)

AGIX Staking & Bridge

Earn more while holding AGIX tokens by vesting them in staking sessions, which will also support the operations of our blockchain network. Bridge allows users to transfer their tokens across supported blockchains such as Ethereum and Cardano.

[VIEW DETAILS](#)[AGIX STAKING](#)[BRIDGE](#)

Comparing Nostr DVMs to others:

	Decision Making	Payments	Decentralized*	On a Social Layer
MIDCA	Hardcoded	No	No	No
Singularity NET	Hardcoded	Yes (but )	Yes	No
OpenAI	LLM-chooses	No	No	No
LangChain	LLM-chooses	No	No	No
AutoGPT	LLM-chooses	No	No	No
Nostr DVMs	*anything	Yes	Yes	Yes

Decentralized* = meaning components by different developers can work together over a network

Problems with Current NOSTR DVMs

- IDs are too limited, we should have tags instead.
 - There will always be more task types than IDs
- Payment up front option is too inflexible.
 - Need a variety of different kinds of contracts.
 - Time-based contracts (aka salaries), escrow (i.e. RoboSats), retainer (i.e. lawyer/PI)
- Need a profile page, like a resume / portfolio, for past DVMs, so you can decide whether you want to hire them.
- We need Critics/Evaluators, or reputation based systems.
- We need DVMs that can hire other DVMs.
- (maybe) Distinction between primitive and hierarchical DVMs
 - Primitive DVMs do a single thing only. Hierarchical DVMs will call multiple, other DVMs.
- Need a text only request, DVMs figure out whether they think they can do job.

Text extraction

Extracts text from an image, audio, video or anything else

+ Add input

Input Enter the data you want to be processed

Range (for audio/video) **Optional**

Starting second

Finishing second

Parameters **5** Specify additional parameters

method

mode


language

model

alignment

DVMs (5) Specify additional parameters

Advanced

 **SUBMIT**

Cancel

Text extraction

Extracts text from an image, audio, video or anything else

+ Add input

Input to be processed

Range (for audio)

Starting second

Parameters 5 Specify additional

method

mode

language

model

alignment

DVMs (5) Specify additional

Advanced



SUBMIT

Cancel

A task request to be solved by DVMs should only require:

1. A Nostr Note (text, attachments, etc)
2. Proof of funds (up front payment, escrow, hodl-invoice?, etc)

...everything else should be optional

A task request to be solved by DVMs should only require:

1. A Nostr Note (text, attachments, etc)
2. Proof of funds (up front payment, escrow, hodl-invoice?, etc)

CLIENT DEVS SHOULD ADD THIS TO HELP MAKE THE AI BETTER:

1. Feedback mechanism for human to score the output (thumbs up/down, 0-10)

DVM DEVS SHOULD ADD THIS TO HELP MAKE THE AI BETTER:

2. Action models
 - a. Maybe json style function definitions like OpenAI
 - b. Text describing input/output/side effects is better than nothing

...everything else should be optional

Action Model Example that each DVM would publish

```
23     {
24         "type": "function",
25         "function": {
26             "name": "get_current_weather",
27             "description": "Get the current weather in a given location",
28             "parameters": {
29                 "type": "object",
30                 "properties": {
31                     "location": {
32                         "type": "string",
33                         "description": "The city and state, e.g. San Francisco, CA",
34                     },
35                     "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
36                 },
37                 "required": ["location"],
38             },
39         },
```

Action Model Example that each DVM would publish

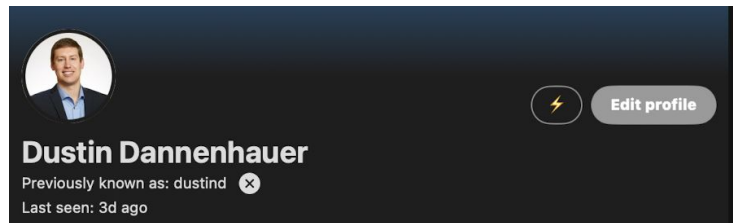
```
23      {
24          "type": "function",
25          "function": {
26              "name": "get_current_weather",
27              "description": "Get the current weather in a given location",
28              "parameters": {
29                  "type": "object",
30                  "properties": {
31                      "location": {
32                          "type": "string",
33                          "description": "The city and state, e.g. San Francisco, CA",
34                      },
35                      "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
36                  },
37                  "required": ["location"],
38              },
39          },
```

Except this is missing a
description of the output
AND side effects

Proposal for Open Challenges:

- Multi DVM chains
- Dynamic Multi DVM chains
- Multi DVM chains with a human as one step (replacing a DVM)
- A DVM that makes DVMs
- A DVM-chain debugging tool
- Limiting / Reducing how DVMs interact with personal data stores
 - I.e. maybe you approve every request that a DVM makes to your personal RAG Assistant, and you can see which data it's requesting from you
- DVMs that take an action outside of Nostr
 - Uber DVM
 - Pingstr-bot as a DVM
 - Git Issue solver - a DVM that writes code to solve issues on your repo via a pull request
 - Phone Call DVM
 - Concert Ticket Buying DVM

Summary



- Think of DVMs as Actions that transform States
- DVMs on Nostr is better than any other attempt to decentralized AI

	Decision Making	Payments	Decentralized*	On a Social Layer
Nostr DVMs	*anything	Yes	Yes	Yes



Npub (as text)

- To help the AI (aka DVMs) automatically improve over time:
 - Clients should add human feedback mechanisms
 - DVM devs should publish action models
- LLMs can't reason, so other AI techniques + humans need to fill the gap
- Need many types of payment contracts for different types of DVM task requests
- Having DVM-to-DVM interactions publicly accessible enables other DVMs to identify failures, diagnose causes, and step in as replacements online (metacog)

Let's connect on Nostr:

npub1mgvwnpsqgrem7jfcwm7pdvdfz2h95mm04r23t8pau2uzxwsdnpgs0gpdjc

END