Application of global working memory for solving complex problems in Transformer-based multi-agent systems

Beliakin Evgeniy
beliakin.eo@phystech.edu

Center for Cognitive Modeling, MIPT

2024, Moscow

Goal

LLM-based multi-agent systems.

The main research objective is to develop an architecture for a multi-agent system based on LLMs, where the use of a global workspace (global working memory) enables cooperative resolution of complex multi-component tasks by agents with diverse specializations.

Subtasks:

- Collect datasets for sub-domains of algorithms theory
- Choose algorithm and train LLM-based agent experts for each domain
- Test LLM agents solving skills in the defined global workspace

Motivation

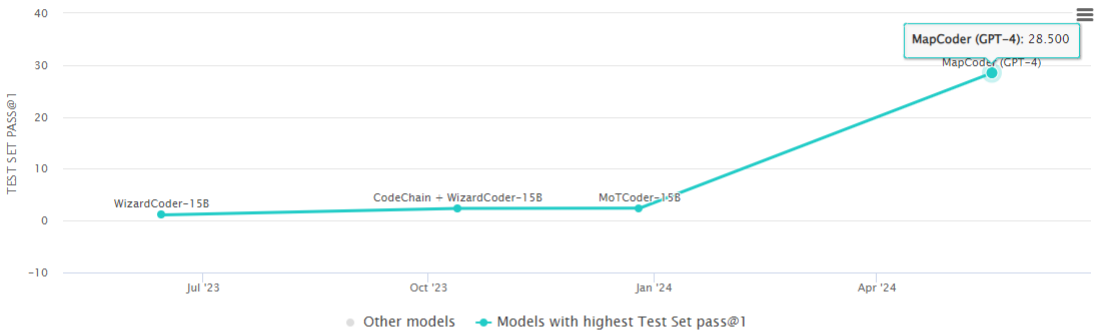
- **Society of Minds paradigm:** Inspired by Marvin Minsky, multi-agent systems solve complex problems through cooperative specialization and communication.
- **LLMs scaling laws:** Multi-agent systems enable cost-efficient task distribution by leveraging smaller, specialized agents instead of a single, general model.
- **Interpretability:** Modular architectures in multi-agent systems improve transparency and traceability of decision-making.
- **Code tasks:** Low accuracy in solving Olimpiad Programming tasks.

Motivation

Code Generation on CodeContests

Leaderboard | Dataset

View: Test Set pass@1 by: Date



Filter: untagged

Edit Leaderboard

Rank	Model	Test Set pass@1	Test Set pass@5	Val Set pass@1	Val Set pass@5	Paper	Code	Result	Year	Tags
1	**MapCoder** (GPT-4)	28.5	35.2	28.5		MapCoder: Multi-Agent Code Generation for Competitive Problem Solving			2024	
2	**MoTCoder-15B**	2.39	7.69	6.18	12.73	MoTCoder: Elevating Large Language Models with Modular of Thought for Challenging Programming Tasks			2023	
3	**CodeChain + WizardCoder-15B**	2.35	3.29	2.48	3.30	CodeChain: Towards Modular Code Generation Through Chain of Self-revisions with Representative Sub-modules			2023	
4	**WizardCoder-15B**	1.11	3.18	1.98	3.27	WizardCoder: Empowering Code Large Language Models with Evol-Instruct			2023	

Figure: CodeContests Benchmark[1]

[1] https://paperswithcode.com/sota/code-generation-on-codecontests

Problem statement

Develop a multi-agent LLM-based system for solving olympiad-level programming tasks cooperatively.

Definitions:

- $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ - dataset of programming tasks, where each p_k
- $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ - multi-agent system, where each A_i is an LLM agent specializing in a domain and defined by:
  - θ_i - the model weights.
  - C_i - agent-specific context.
  - $\mathcal{H}_i$ - interaction history in the global workspace.
- $\mathcal{G}$ - global workspace enabling inter-agent communication and coordination.

Objective:

$$\max_{\mathcal{A},\mathcal{G}} \mathcal{A}cc(\mathcal{P}, \mathcal{A}, \mathcal{G}) \tag{1}$$

Where:

- $\mathcal{A}cc(\mathcal{P}, \mathcal{A}, \mathcal{G}) = \frac{1}{m} \sum_{k=1}^{m} \mathbb{I}(\text{solution to } p_k \text{ is correct})$
- $\mathbb{I}(\cdot)$ - indicator function that evaluates to 1 if the solution is correct and 0 otherwise.

Problem statement

Constraints:

- Each agent A_i is trained on its domain-specific dataset $\mathcal{D}_i$ to optimize its individual performance $f_i(A_i)$.
- The multi-agent system cooperates through $\mathcal{G}$ to solve each $p_k \in \mathcal{P}$ as a unified task.

Metrics:

- System accuracy on the dataset $\mathcal{P}$, defined as $\mathcal{A}cc(\mathcal{P}, \mathcal{A}, \mathcal{G})$.
- Agent specialization efficiency $f_i(A_i)$.

Global Workspace Theory

- **Core idea:** Global Workspace Theory (GWT) models consciousness as a shared space enabling communication between specialized modules.
- **Key components:**
  - **Specialized agents:** Independent processes focused on specific tasks.
  - **Global workspace:** A shared memory where dominant information becomes accessible to all agents.
- **Relevance to AI:** Facilitates cooperation and synchronization in multi-agent systems, enabling complex problem-solving.

Society of Mind

- **Core idea:** Proposed by Marvin Minsky, the mind is a society of specialized agents working together.
- **Agents:** Simple entities with limited capabilities but effective in specific domains.
- **Collaboration:** Problem-solving emerges from agent interaction and communication.
- **Relevance to AI:** Mimics human cognition by enabling modularity, scalability, and cooperative task-solving.

Society of Mind

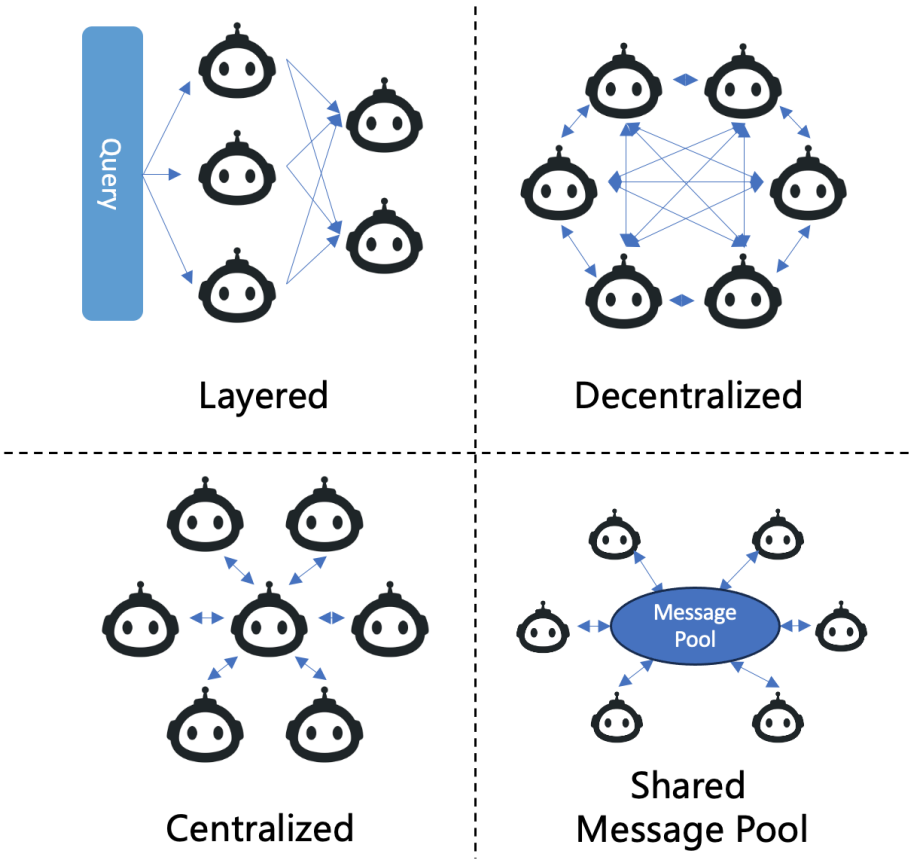


Figure: The Agent Communication Structure.[2]

[2] Guo et al., *Large Language Model based Multi-Agents: A Survey of Progress and Challenges*.

Datasets



- Code Contests - tasks from Codeforces[3]
- xCodeEval[4] - synthetic benchmark for code understanding, generation, translation and retrieval
- HumanEval (code) [5]

[3] https://codeforces.com

[4] Khan et al., *xCodeEval: A Large Scale Multilingual Multitask Benchmark for Code Understanding, Generation, Translation and Retrieval*.

[5] https://paperswithcode.com/sota/code-generation-on-humaneval

Problem Example

Backspace

You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcbd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcaa" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of s.

Input

The first line contains a single integer q ($1 \le q \le 10^5$) the number of test cases.
The first line of each test case contains the string s ($1 \le |s| \le 10^5$). Each character of s is a lowercase English letter.
The second line of each test case contains the string t ($1 \le |t| \le 10^5$). Each character of t is a lowercase English letter.
It is guaranteed that the total number of characters in the strings over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print "YES" if you can obtain the string t by typing the string s and replacing some characters with presses of "Backspace" button, or "NO" if you cannot.
You may print each letter in any case (YES, yes, Yes will all be recognized as positive answer, NO, no and nO will all be recognized as negative answer).

Example Input

```
4
ababa
ba
ababa
bb
aaa
aaaa
aababa
ababa
```

Example Output

```
YES
NO
NO
YES
```

Explanation

In order to obtain "ba" from "ababa", you may press Backspace instead of typing the first and the fourth characters.

There's no way to obtain "bb" while typing "ababa".

There's no way to obtain "aaaa" while typing "aaa".

In order to obtain "ababa" while typing "aababa", you have to press Backspace instead of typing the first character, then type all the remaining characters.

Figure: Competitive programming problem statement[6]

[6] Li et al., "Competition-level code generation with AlphaCode".

Textbooks are all you need[7]

```
To begin, let us define singular and nonsingular matrices. A matrix is said to be singular if its
determinant is zero. On the other hand, a matrix is said to be nonsingular if its determinant is not
 zero. Now, let's explore these concepts through examples.

Example 1: Consider the matrix A = np.array([[1, 2], [2, 4]]). We can check if this matrix is
singular or nonsingular using the determinant function. We can define a Python function, `
is_singular(A)`, which  returns true if the determinant of A is zero, and false otherwise.

import numpy as np
def is_singular(A):
    det = np.linalg.det(A)
    if det == 0:
        return True
    else:
        return False

A = np.array([[1, 2], [2, 4]])
print(is_singular(A)) # True
```

Figure: The synthetic textbook dataset example

[7] Gunasekar et al., *Textbooks Are All You Need*.

MapCoder[8,9]

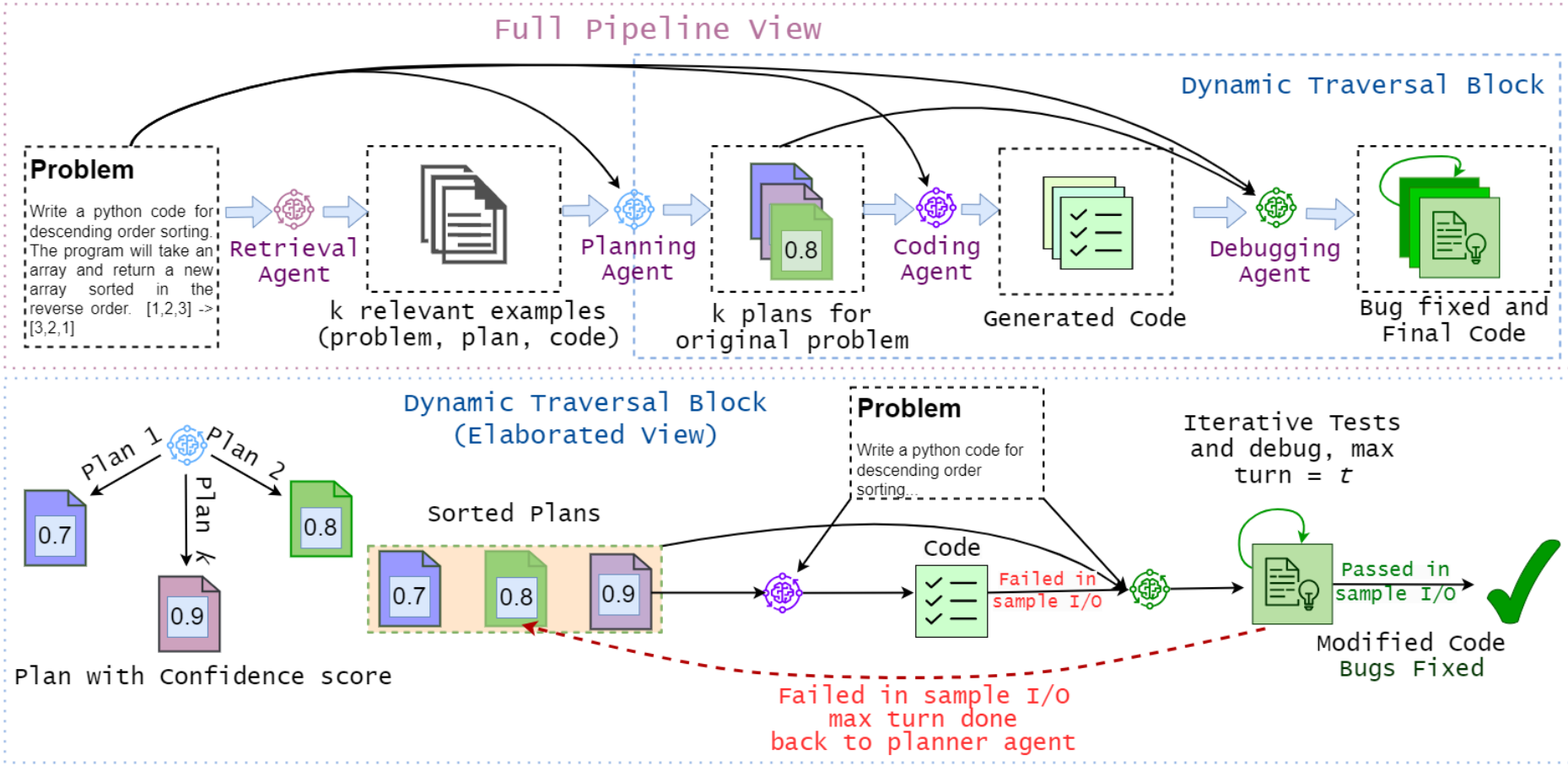


Figure: Overview of MapCoder

[8] Islam, Ali, and Parvez, *MapCoder: Multi-Agent Code Generation for Competitive Problem Solving*.

[9] SOTA on CodeContests Benchmark

Proposed Approach

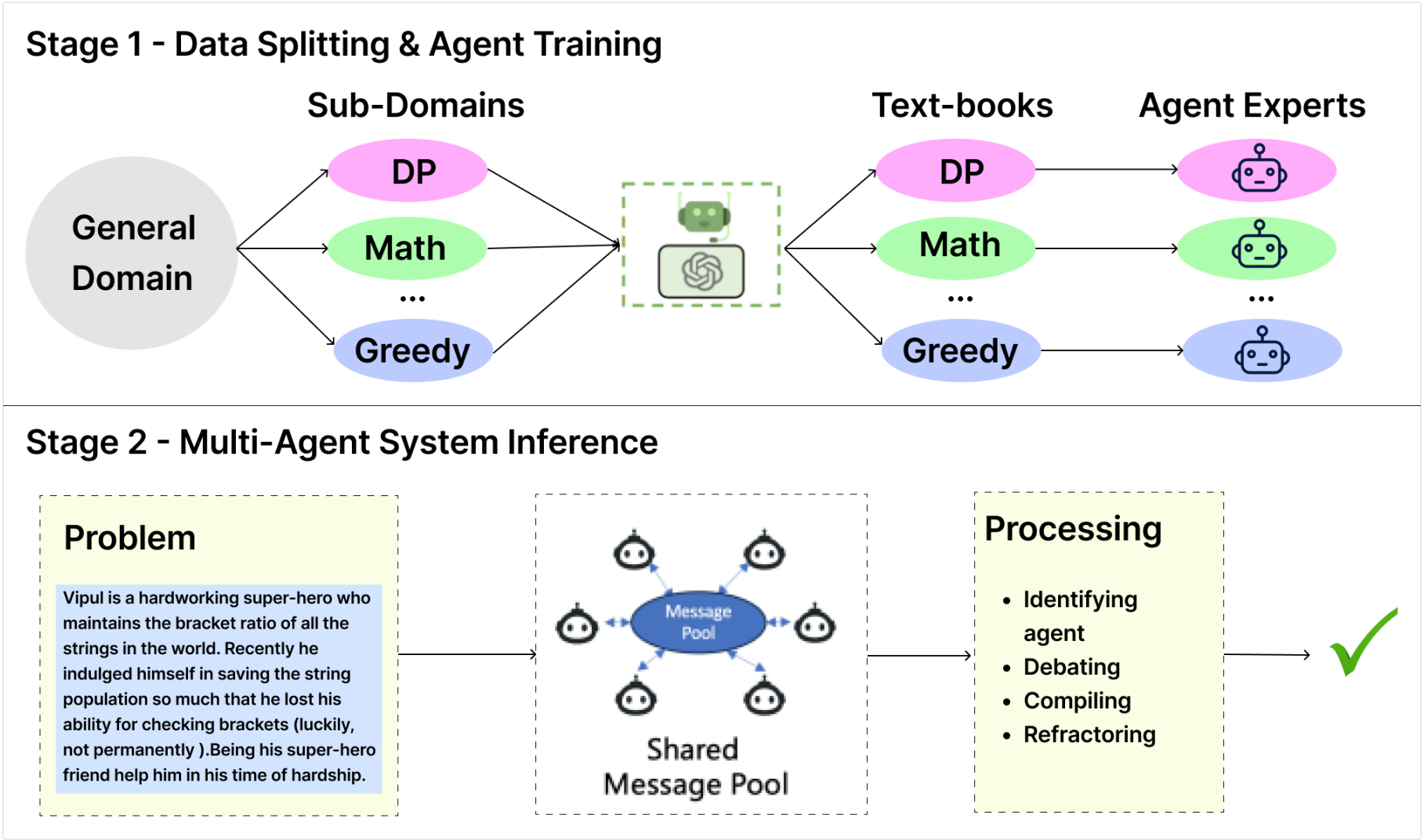


Figure: Proposed Approach Structure

Experiments plan

- Collect k datasets based on code contests dataset in the textbook style, where k - count of sub-domains (dynamic programming, greedy, sorting and etc.)
- Fine-tune k LLM-agents using PEFT with LoRA of Full-Fine-Tuning teqnique
- Set up the workspace, measure quality of LLM multi-agent system and compare it with general LLM-s

Setup:

- Base agent model: microsoft/Phi-3-mini-128k-instruct[10]
- Synthetic dataset generation model: meta-llama/Llama-3.1-70B-Instruct or others 70B LLMs
- Libraries: TRL, Huggingface, DeepSpeed, PEFT, VLLM, Autogen

[10] Abdin et al., *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*.

Experimental results

- MapCoder experiments:
  - Results do not reproduced with small LLMs (8B-12B)
  - Hallucinations bring to endless loops and other quality problems
- Generated dataset:
  - Domain - **Dynamic programming**
  - Model - Queen-72B-Instruct
  - Total size - 2.5K examples, 3.6M tokens
  - Generation speed - 240 examples per hour
- In progress:
  - Fine-tuning microsoft/Phi-3-small-128k-instruct (8B params)
  - Measuring metrics on CodeContests and XCodeEval test parts

Thank you for your attention!

beliakin.eo@phystech.edu