Large Language Models: Applications, Opportunities and Risks
Carman Mark James, Brambilla Marco, Pierri Francesco

CodeAgent Project

Exploring Repository-Level Code Generation With Agents

Acquadro Patrizio, Yu Zheng Maria

Academic Year 2024-2025

POLITECNICO
MILANO 1863

Documentation on GitHub

Outline

What we will cover

POLITECNICO
MILANO 1863

1. Introduction

Overview of CodeAgent
Motivations for this Project
Goals to Accomplish

POLITECNICO
MILANO 1863

CodeAgent

LLM-based repository-level code generation agentic framework

CODEAGENT: Enhancing Code Generation with Tool-Integrated Agent Systems for Real-World Repo-level Coding Challenges

Kechi Zhang[*], Jia Li[*], Ge Li[†], Xianjie Shi, Zhi Jin[†]
Key Lab of High Confidence Software Technology (PKU), Ministry of Education
School of Computer Science, Peking University, China

Difficult but important task (70% of entire code is repo-level)

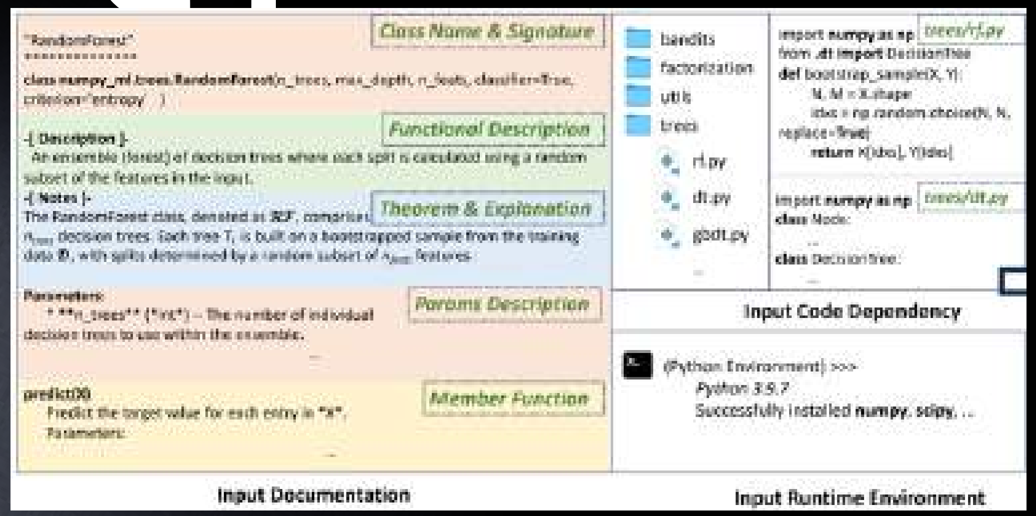
CodeAgent

LLM-based repository-level code generation agentic framework

CODEAGENT: Enhancing Code Generation with Tool-Integrated Agent Systems for Real-World Repo-level Coding Challenges

Kechi Zhang*, Jia Li*, Ge Li†, Xianjie Shi, Zhi Jin†
Key Lab of High Confidence Software Technology (PKU), Ministry of Education
School of Computer Science, Peking University, China

- Incorporate **programming tools** to provide contextual information
  - 3 categories
  - 5 tools
- Leverage **agent strategies** for tools usage
- Create a specific **repo-level** benchmark (codebase + dataset)



Motivation

01 Current and future **trend** of the agents

02 **Complexity** of real-world programming tasks ⟶ Agents to **boost efficiency**?

03 **Complexity** of the project ⟶ Valuable **skills** acquired

04 Repository-level dataset comprehension

Goals

01 Analysis of existing literature about code generation

02 Replication of **CodeAgent framework** by implementing tools and strategies

03 Analysis of **CodeAgentBench** and creation of **MiniTransformers**

04 Obtain valuable results

POLITECNICO
MILANO 1863

2. Literature Review

Code LLMs: General VS Specific
LLM Agents: Planning, Tools, Frameworks
CodeGen Scenarios: Statement VS Function VS Repository
Benchmarks: Function-level VS Repository-level

POLITECNICO
MILANO 1863

Literature Review

- Code LLMs

- **General-purpose** LLMs
- **Coding-specific** LLMs
  - Trained on code from scratch
  - Fine-tuned on code

Literature Review

- Code LLMs
  - **General-purpose** LLMs

Gemini

Literature Review

- Code LLMs

- **Coding-specific** LLMs
  - Trained on code from scratch

AlphaCode

- Pre-trained on GitHub code
- Fine-tuned for competitive programming

StarCoder

- Pre-trained on multilingual code
- Fine-tuned on Python tokens

StarCoder

Literature Review

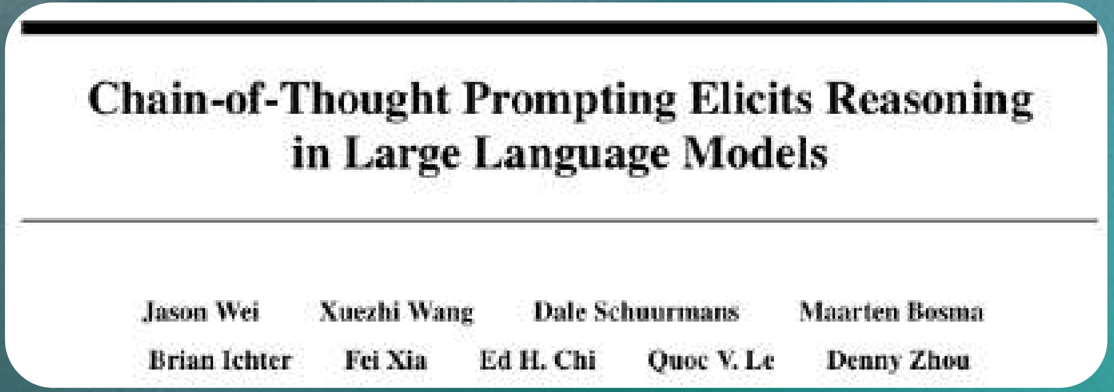
- Code LLMs

  - **Coding-specific** LLMs
    - Only fine-tuned on code

→ Transfer Learning:

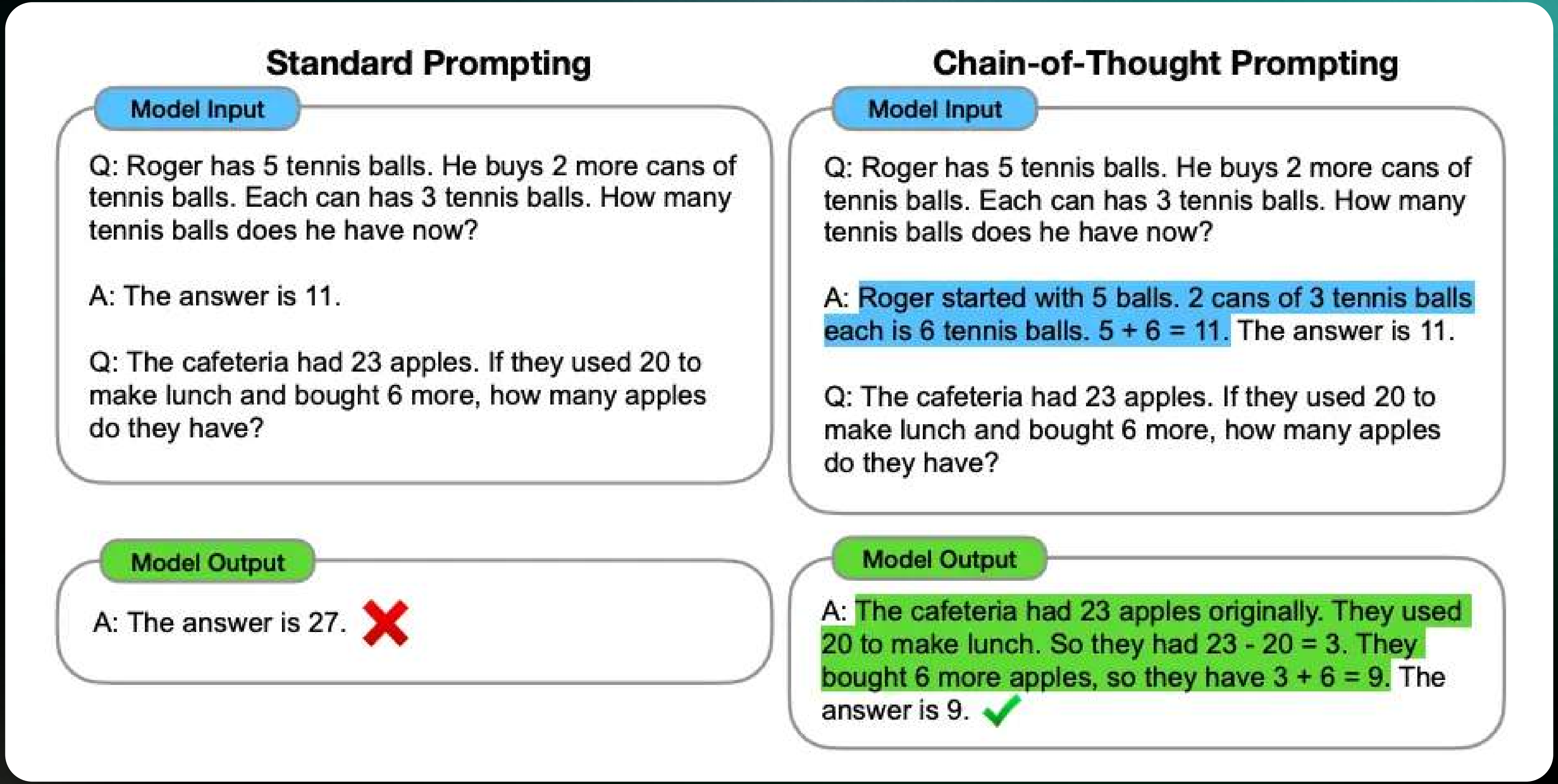
Reuse knowledge from previously trained systems

Literature Review


Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou

- LLM Agents: “Planning”



- **CoT**: chain-of-thought

Series of **intermediate reasoning steps** provided

CON: Static internal representation

Let’s think step by step...

POLITECNICO
MILANO 1863

Source: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (https://arxiv.org/pdf/2201.11903)

Literature Review

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

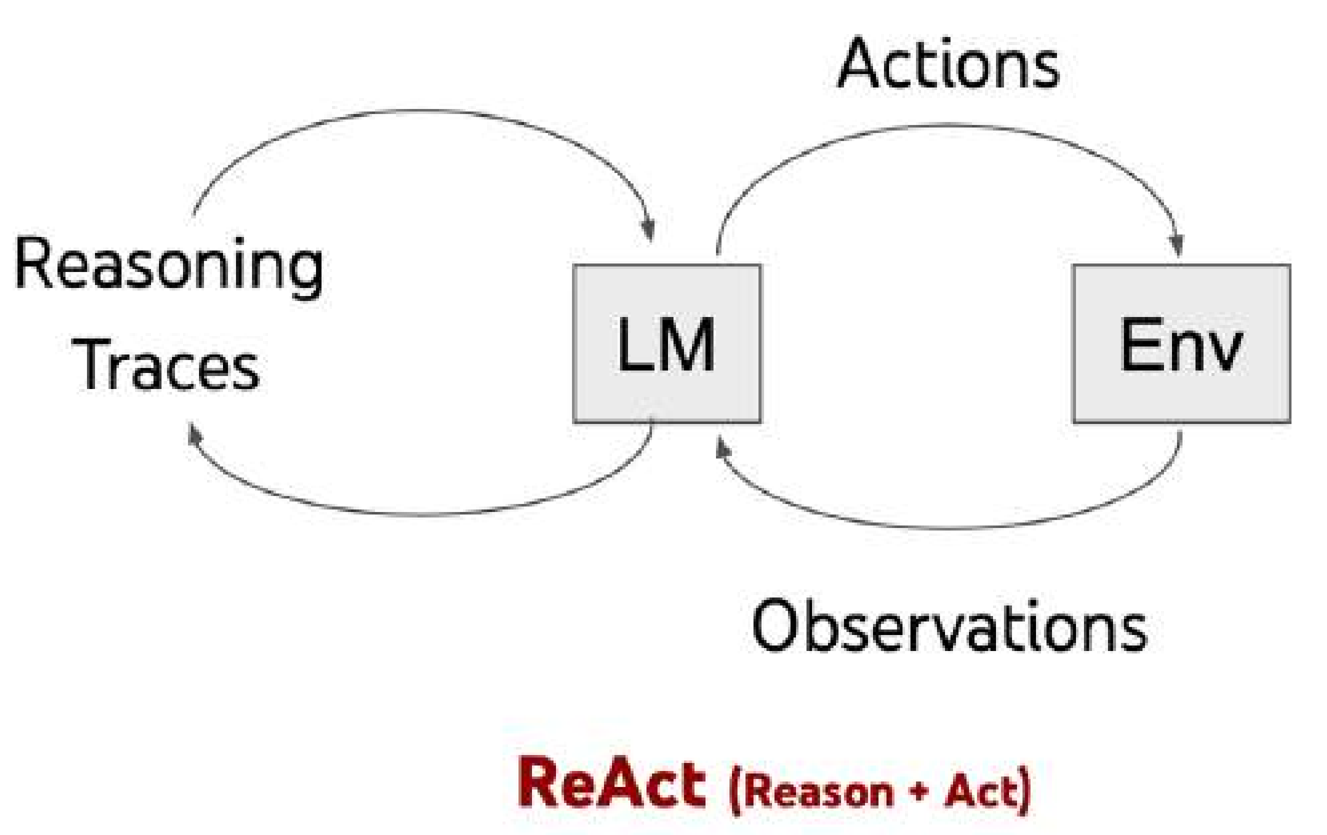
Shunyu Yao[*1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

- LLM Agents: "Planning"

  - **ReAct**: reason (plan) & action
    - Reason to act
    - Act to reason

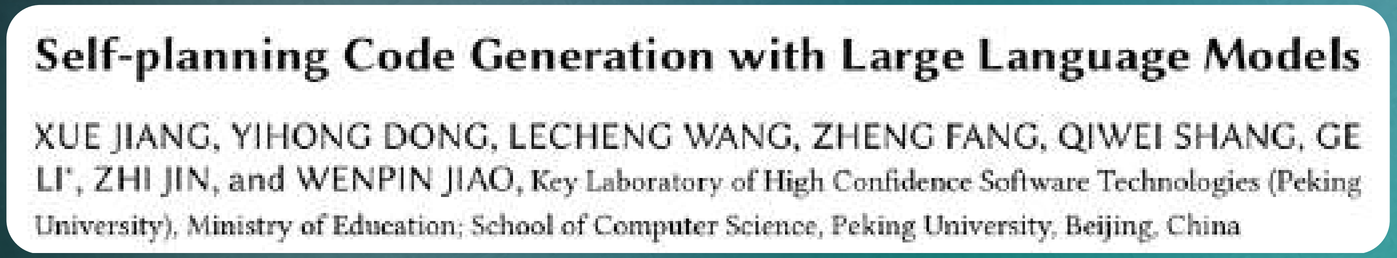
  Incorporate **external information** into reasoning

  CON: prone to loops



Source: *ReAct: Synergizing Reasoning and Acting in Language Models*
https://arxiv.org/pdf/2210.03629

Literature Review



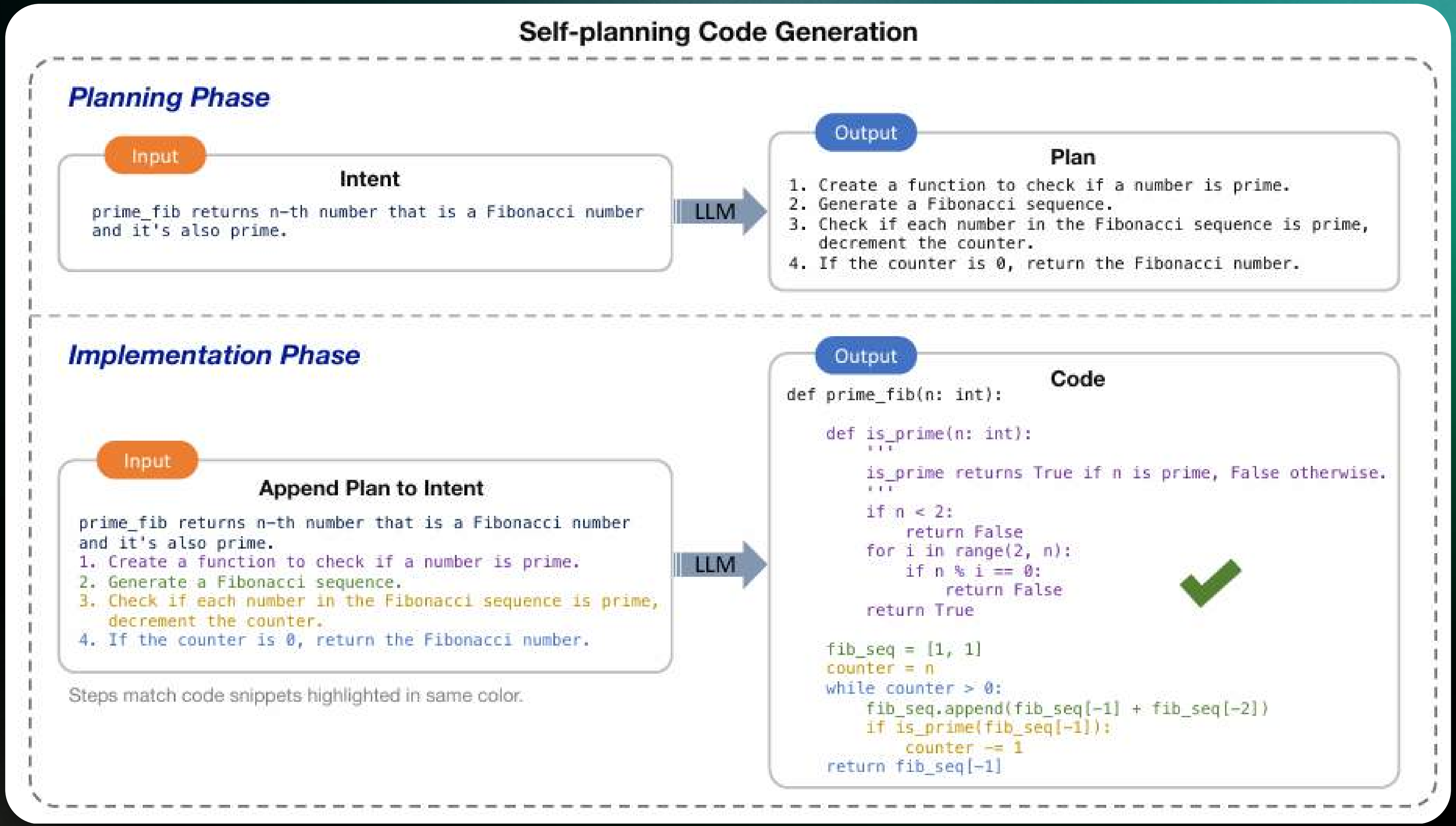
Self-planning Code Generation with Large Language Models

XUE JIANG, YIHONG DONG, LECHENG WANG, ZHENG FANG, QIWEI SHANG, GE LI*, ZHI JIN, and WENPIN JIAO, Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing, China

- LLM Agents: "Planning"

- **Self-planning**

Planning phase with **few samples** (intent, plan)
Plan: decomposition of intent
Implementation phase **guided**
CON: weak verification



Source: Self-planning Code Generation with Large Language Models
https://arxiv.org/pdf/2303.06689

POLITECNICO
MILANO 1863

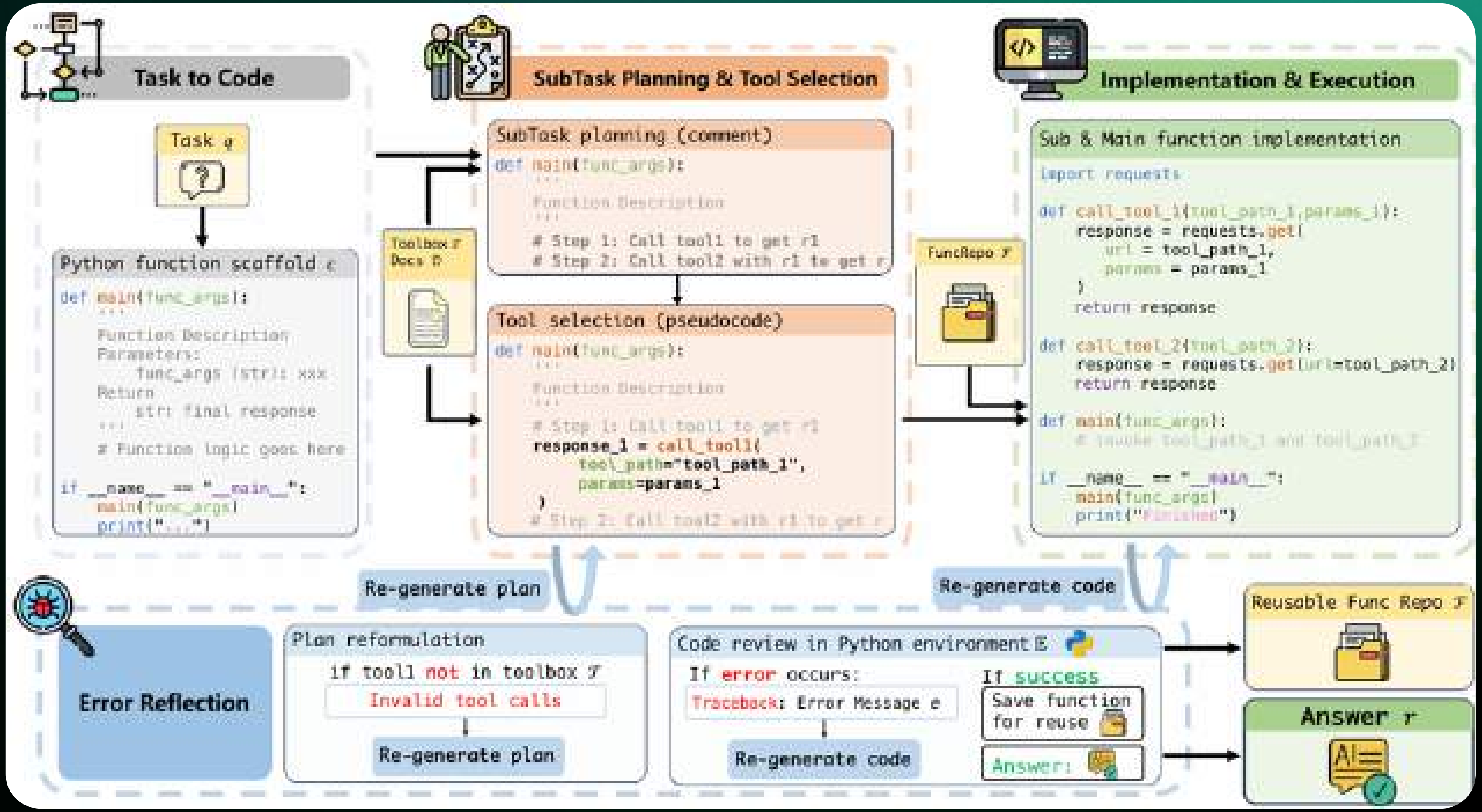
Literature Review

- LLM Agents: Tools (ToolCoder)

4 steps: Code; Subtasks; Exe; Errors

- **Rigorous plan** (Python script VS NL)
- Informative **errors** tracebacks (tools)
- Experience **reuse** (past success=KB)

<u>BUT:</u> **not iterative** process (a priori) and it is **more complex** with respect to ReAct



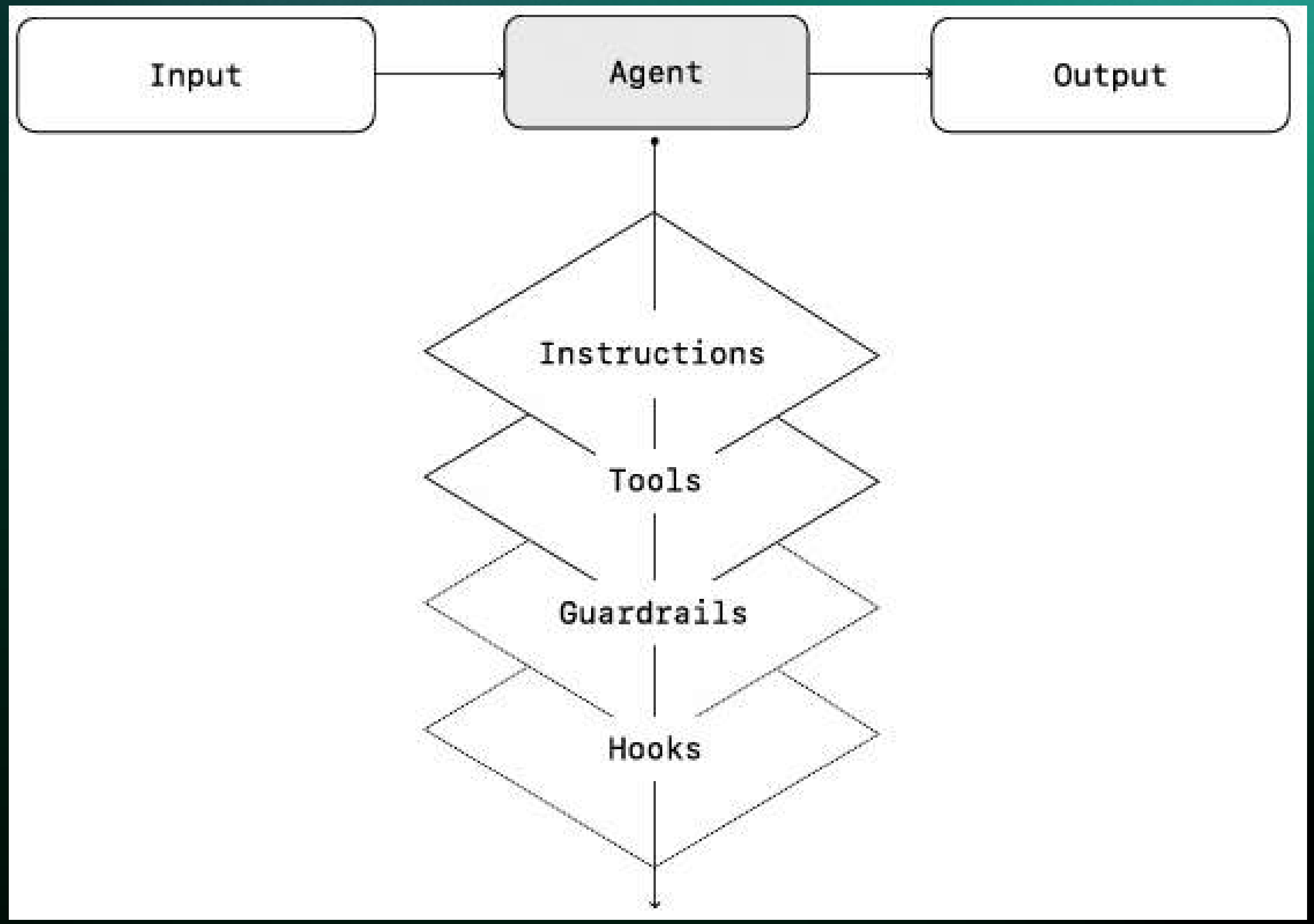
Source: ToolCoder(arXiv:2502.11404)

POLITECNICO
MILANO 1863

Literature Review

- LLM Agents: Frameworks

Single-agent: single model with tools
- Simple complexity, evaluation, maintenance
- run: loop of agent operation until exit cond.
  - **CodeAgent**

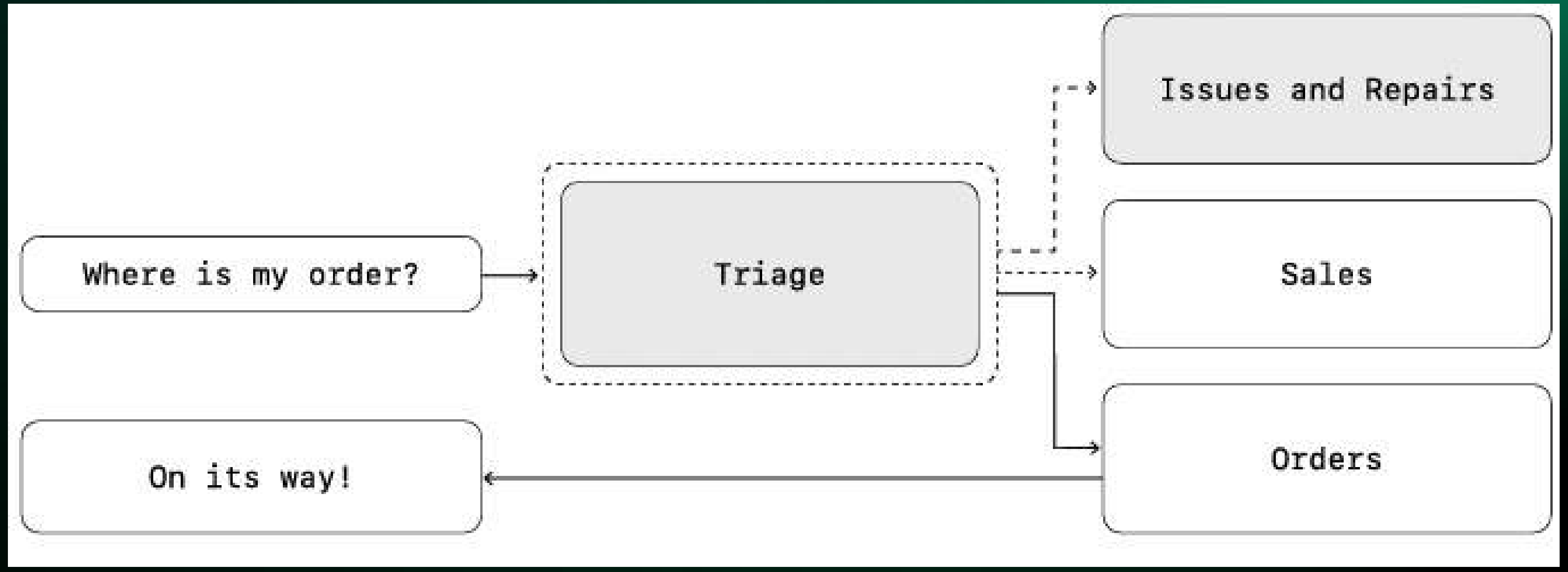


Source: A Pratical Guide to Building Agents

Literature Review

- LLM Agents: Frameworks

Multi-agent: multiple coordinated agents
- <u>Manager:</u> coordinates + summarize result
- <u>Decentralized:</u> agents as peers (handoff)
  - **AgentCoder**



Source: <u>A Pratical Guide to Building Agents</u>

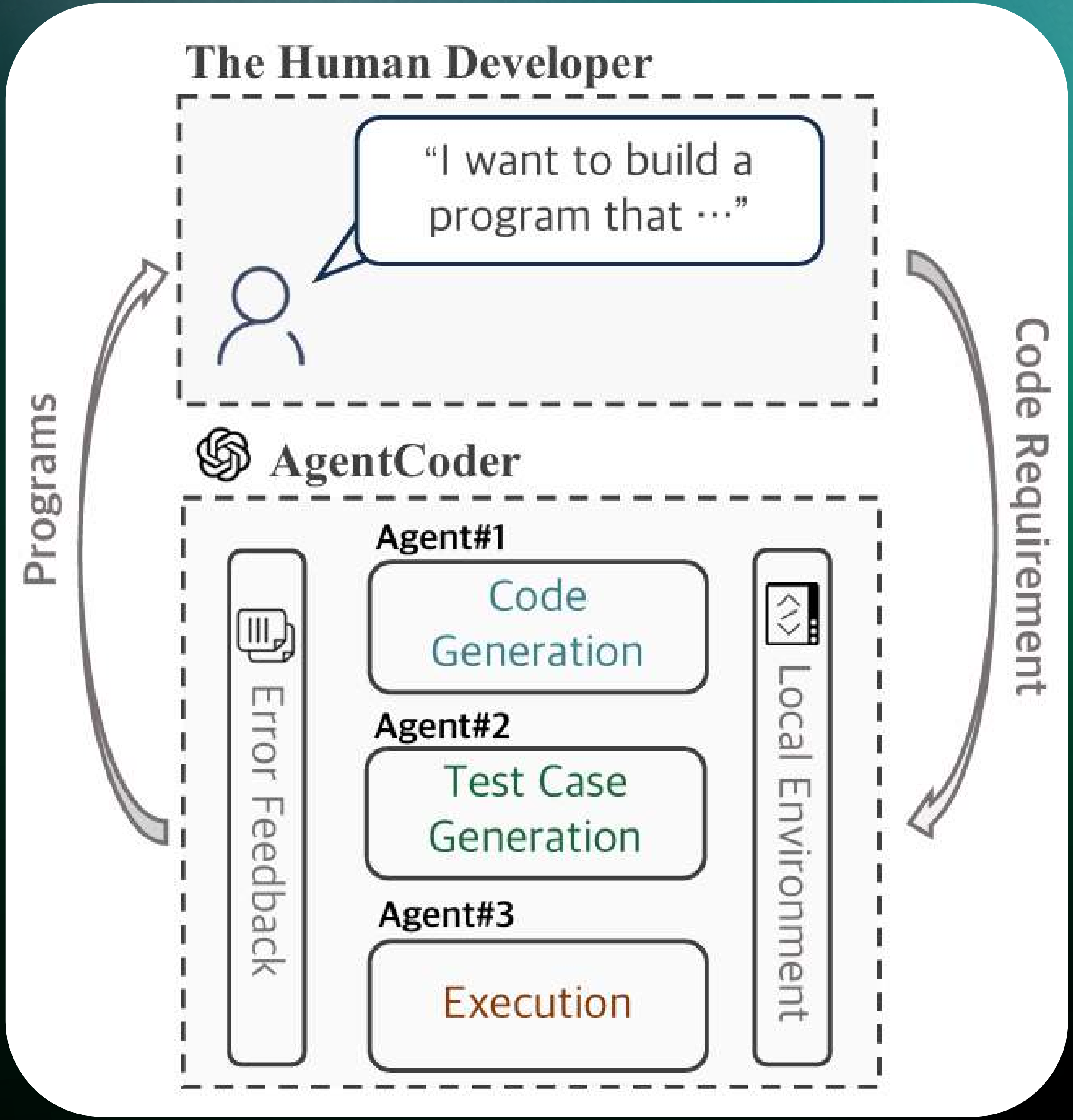
Literature Review

- LLM Agents: AgentCoder

Multi-agent System: **3 agents**
- Programmer: generates code with COT
- Test Designer: creates the tests
- Test Executor: Executes code against tests
  - Decides success VS fail (redo code)

Pro: enhanced code-**tests** efficiency/quality

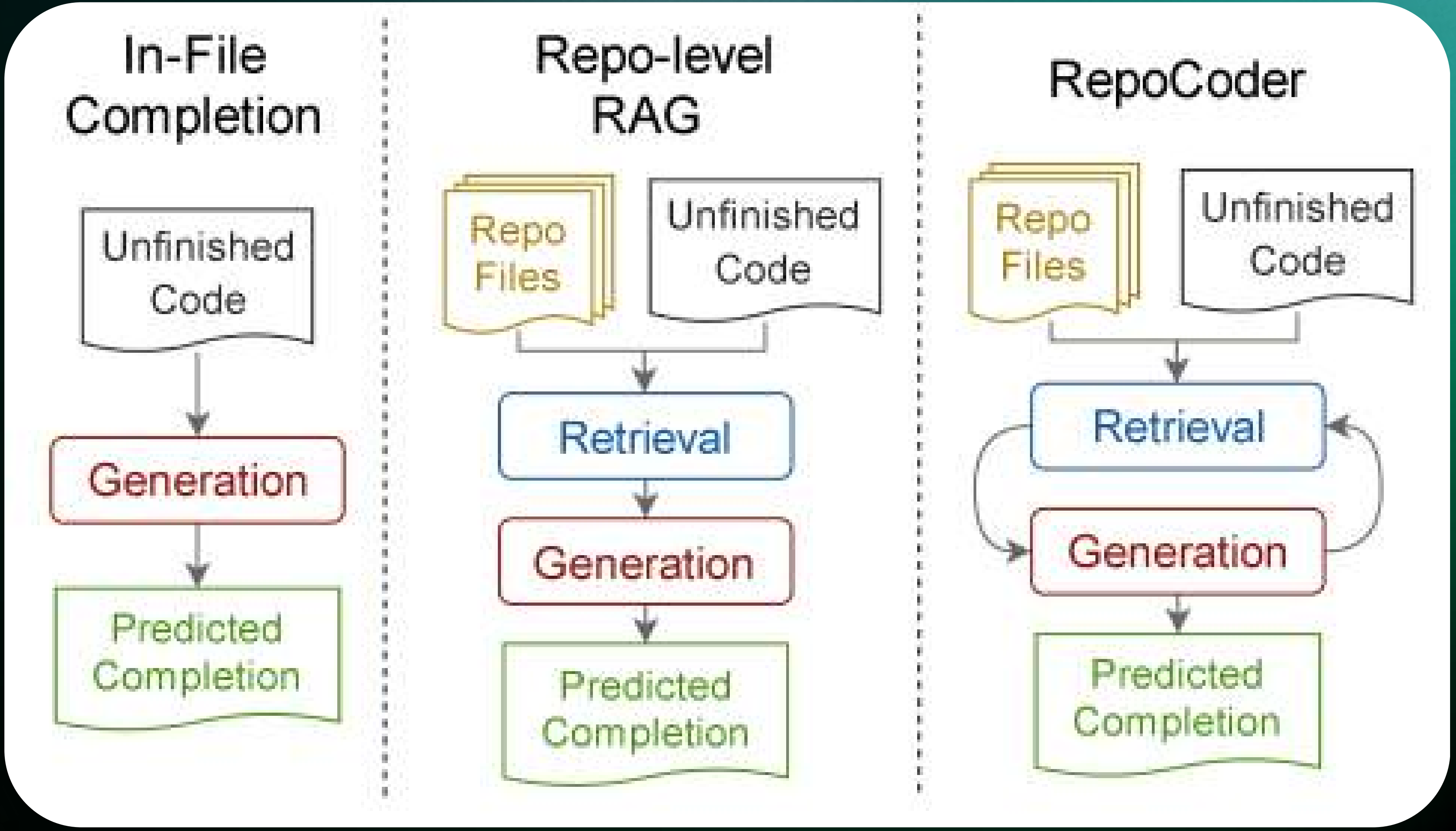
Con: increased **complexity,** no tools=no repo-lv



Source: AgentCoder (arXiv:2312.13010)

Literature Review

- Code generation scenarios
  - **Statement-level** generation
  - **Function-level** generation
  - **Repository-level** generation
    - CodeAgent
    - RepoCoder, A^3 CodGen, CodeRAG



Source: RepoCoder. Repository-Level Code Completion Through Iterative Retrieval and Generation (https://arxiv.org/pdf/2303.12570)

Literature Review

RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation

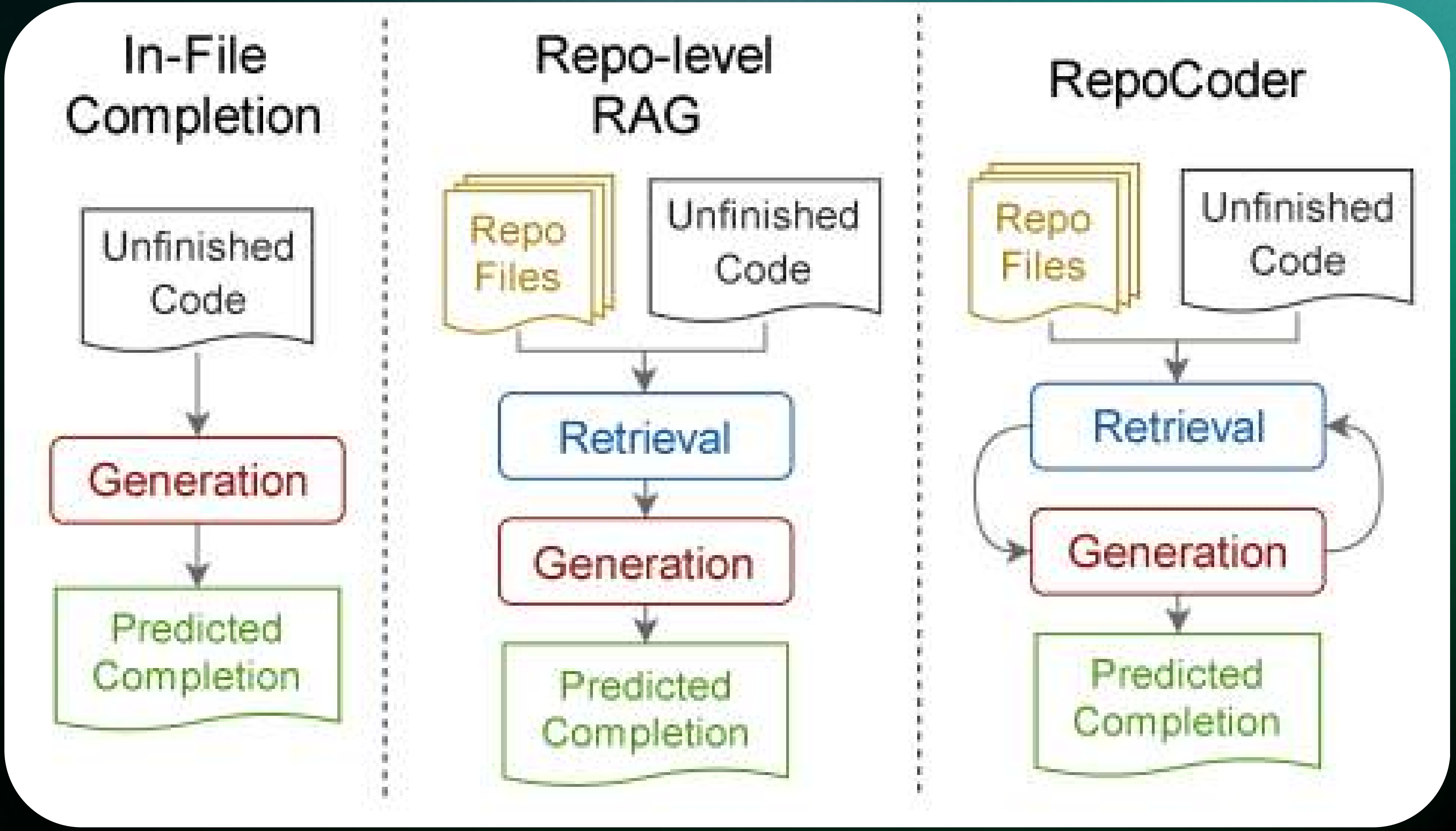
Fengji Zhang[1], Bei Chen[2], Yue Zhang[2], Jacky Keung[1], Jin Liu[3], Daoguang Zan[2], Yi Mao[2], Jian-Guang Lou[2], Weizhu Chen[2]

- Repo-level code generation

- **RepoCoder**
  - Code as independent snippets
  - **Similarity-based retrieval**
  - **Iterative** pipeline

Query: last lines of unfinished code

CON: pending stopping condition



Source: RepoCoder. Repository-Level Code Completion Through Iterative Retrieval and Generation (https://arxiv.org/pdf/2303.12570)

Literature Review

A^3-CodGen : A Repository-Level Code Generation Framework for Code Reuse with Local-Aware, Global-Aware, and Third-Party-Library-Aware

Dianshu Liao, Shidong Pan, Xiaoyu Sun, Xiaoxue Ren, Qing Huang, Zhenchang Xing, Huan Jin, Qinying Li

- Repo-level code generation

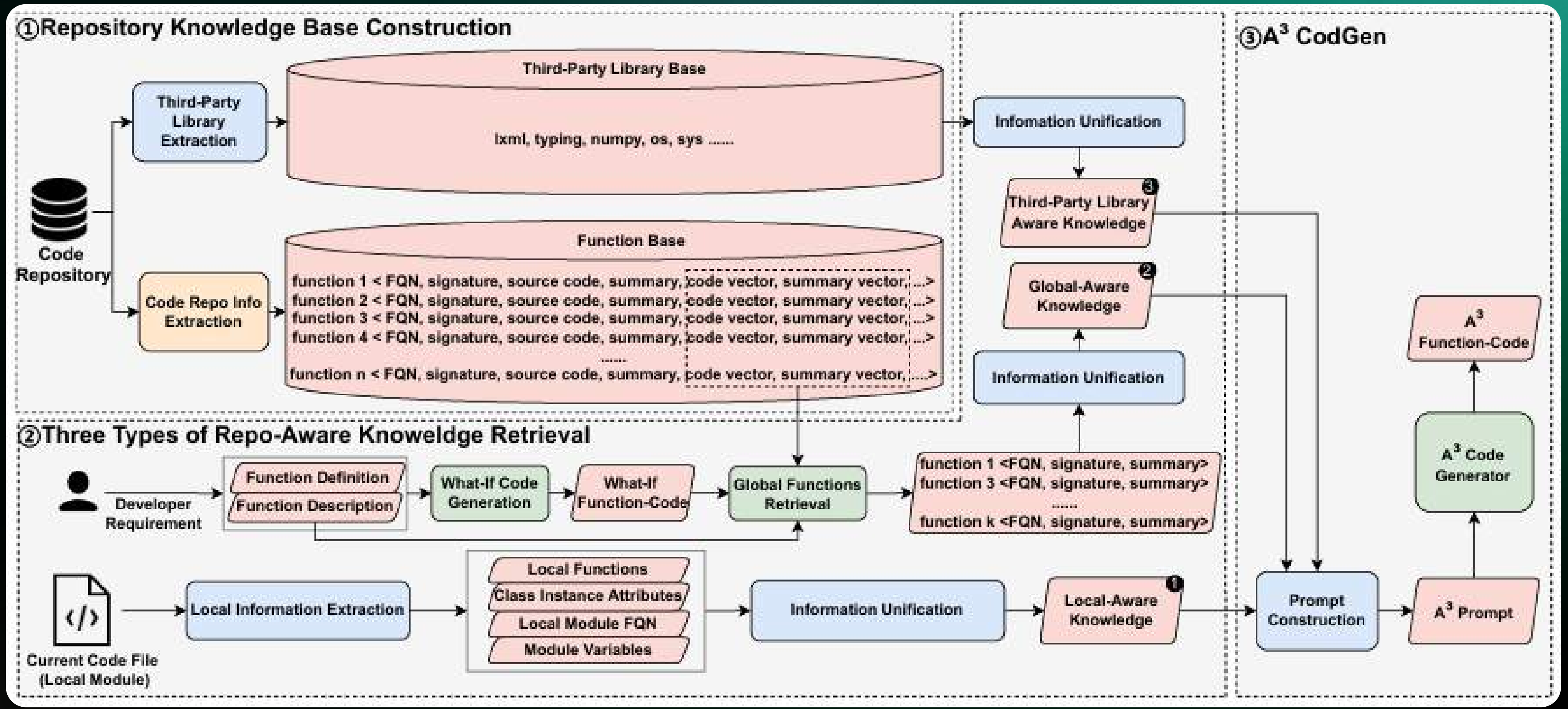
- **A^3 CodGen**

Knowledge retrieval on:

Local: current working files

Global: other files in the same repository → promote code reuse

Third-party-library

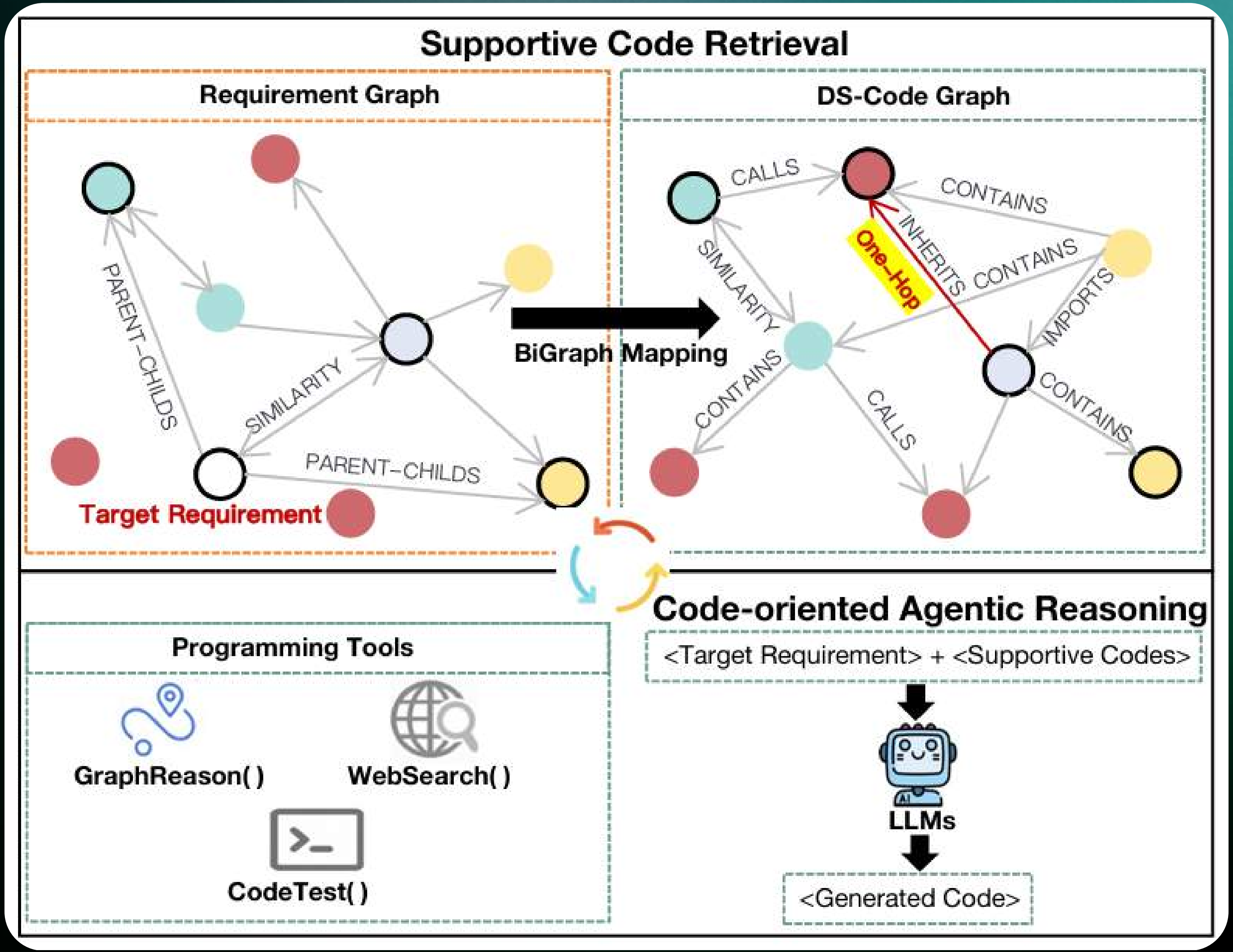


Source: *A3-CodGen : A Repository-Level Code Generation Framework for Code Reuse with Local-Aware, Global-Aware, and Third-Party-Library-Aware (https://arxiv.org/pdf/2312.05772)*

Literature Review

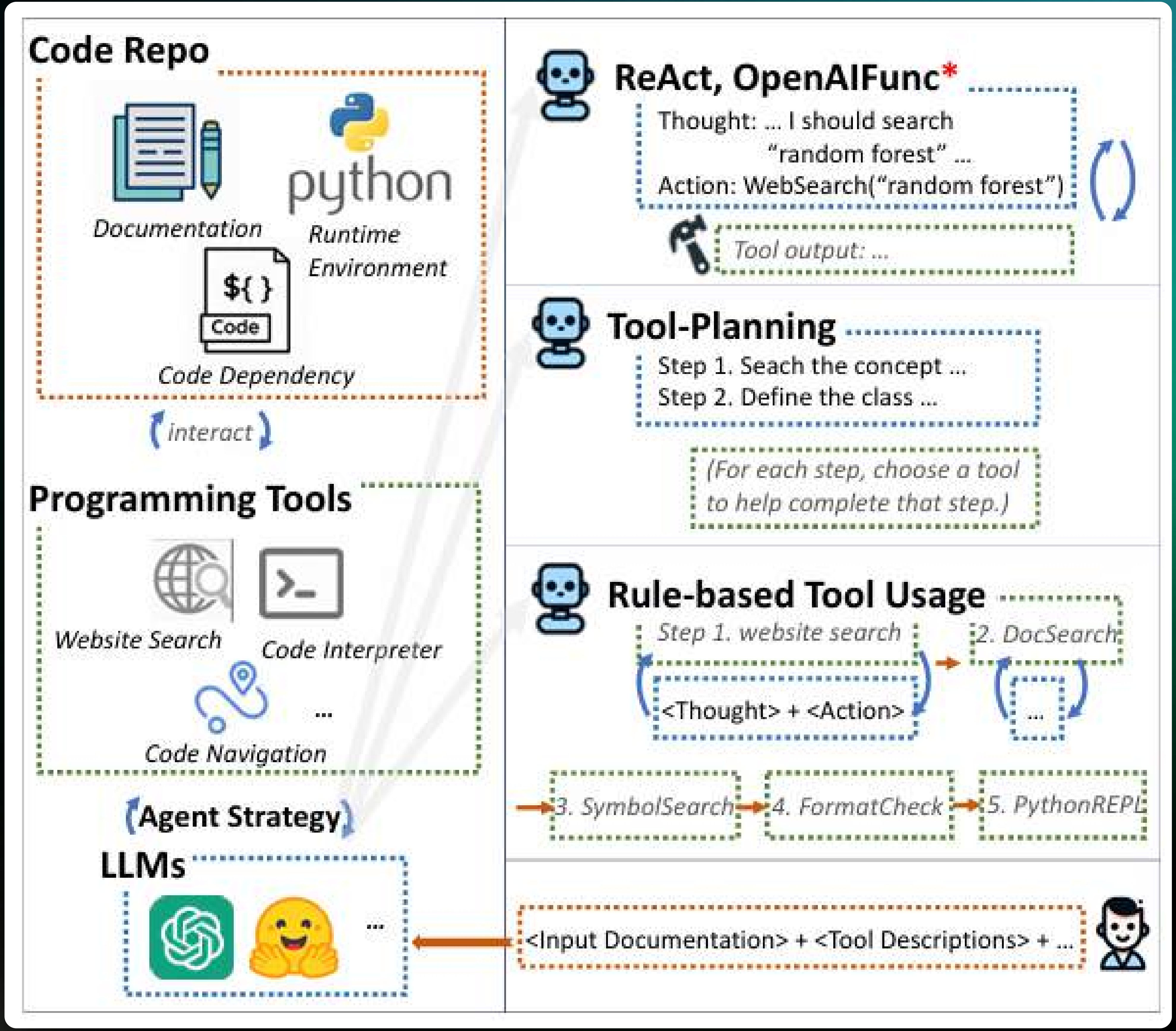
- Repo-level code generation

CodeRAG
Code-oriented agentic reasoning
- Web Search Tool
- Graph Reasoning Tool
- Code Testing Tool
- ReAct strategy



Source: CodeRAG: Supportive Code Retrieval on Bigraph for Real-World Code Generation (arXiv:2504.10046)

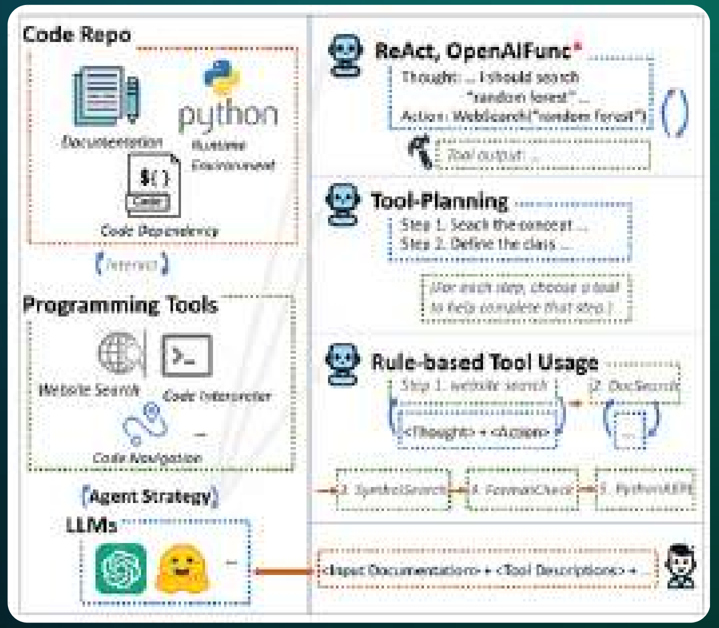
Experiment Replication



POLITECNICO MILANO 1863

Source: CodeAgent (https://arxiv.org/abs/2401.07339)

Experiment Replication



TOOL IMPLEMENTATION

- WebSiteSearchTool
- DocumentationSearchTool
- CodeSymbolNavigationTool
- FormatCheckTool
- CodeInterpreterTool

SYSTEM EVALUATION

- Repository-level testing
- Function-level testing

2

4

1

LLM SETUP

- Small Quantized Model
- Large Models through API

3

AGENT INTEGRATION

- ReAct
- ToolPlanning
- Rule-based
- Tool-calling

POLITECNICO
MILANO 1863

Small LLMs Setup

Prepare an LLM to be used in the Colab free environment (limited resources)

01 Select base LLMs, among Qwen3 (8B, 4B, 1.7B, 0.6B) and CodeLlama 7b Instruct

02 Set **4-bit (nf4) quantization** to model (using *bitsandbytes* lib) and bfloat16

03 Create Hugging Face **Pipeline** → Create LangChain **Wrapper**

POLITECNICO
MILANO 1863

Large LLMs Setup

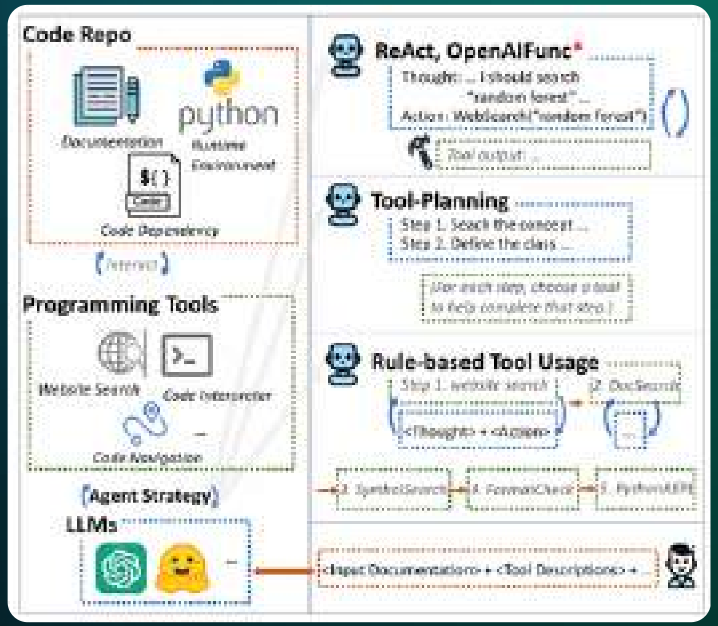
Switch to these models since:

- Limited resources=runtime disconnection
- Simulate better paper setup
- Better performance w/ agent

01 Just select base LLMs (SOTA)

- **Open source**: DeepSeek V3 (OpenRouter)
- **Closed source**: GPT-4.1 nano or mini (free credits), Gemini 2.5 Flash (Free tier)

Experiment Replication



2

TOOL IMPLEMENTATION
- WebSiteSearchTool
- DocumentationSearchTool
- CodeSymbolNavigationTool
- FormatCheckTool
- CodeInterpreterTool

4

SYSTEM EVALUATION
- Repository-level testing
- Function-level testing

1

LLM SETUP
- Small Quantized Model
- Large Models through API

3

AGENT INTEGRATION
- ReAct
- ToolPlanning
- Rule-based
- Tool-calling

POLITECNICO
MILANO 1863

CodeAgent

Tool-Integrated: Incorporate **programming tools** to provide contextual information

Information Retrieval

Code Implementation

Code Testing

POLITECNICO
MILANO 1863

Tool Implementation

Features:

- Subclass of LangChain's BaseTool
  - Name, description, Pydantic input schema (OpenAI Guide)
  - Decisive role of **tool description** in prompting (Tests)
- Robust unit testing conducted on each tool
- Descriptive error messages (ToolCoder)

Tool Implementation

Website Search Tool

Documentation Search Tool

Code Symbol Navigation Tool

Format Check Tool

Code Interpreter Tool

- Search engine free API: **DuckDuckGo**
- Search the internet for information
- **Summarize** the info (LLM/first 60)

POLITECNICO
MILANO 1863

Tool Implementation

Website Search Tool

```python
class WebsiteSearchTool(BaseTool):
    name: str = "WebSearch"
    description: str = (
        "Searches the public internet for general programming topics, external libraries (like PyTorch or NumPy), or error messages. "
        "**Use this tool when you cannot find the answer in the local project documentation (DocSearch).** "
        "The tool takes a search query and **returns a concise, AI-generated summary of the most relevant web pages.**"
    )
```

```python
def summarise(text: str, max_new_tokens: int = 256) -> str:
    prompt = (
        "You are a senior Python engineer. Give a **single paragraph "
        "(≤150 words)** summary of the snippet below, focusing on key details. "
        "------- SNIPPET START -------\n"
        f"{text[:4000]}\n"
        "-------- SNIPPET END --------"
    )
```

POLITECNICO
MILANO 1863

Tool Implementation

- Website Search Tool
- **Documentation Search Tool**
- Code Symbol Navigation Tool
- Format Check Tool
- Code Interpreter Tool

- Read and split docs into **chunk**s (---)
- Rank with **BM25Okapi** algorithm
- Eventually summarize the result

POLITECNICO
MILANO 1863

Tool Implementation

Documentation Search Tool

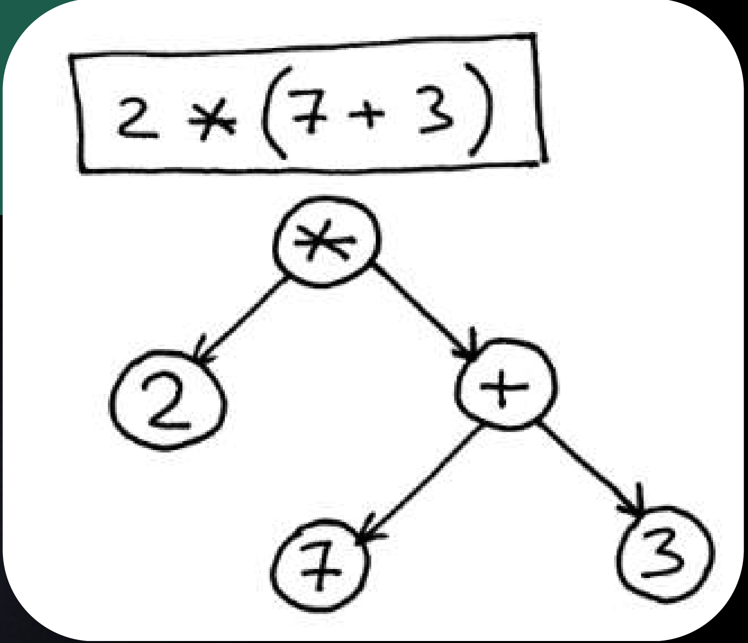
```python
class DocSearchTool(BaseTool):
    name: str = "DocSearch"
    description: str = (
        "Searches the internal project documentation (`api_guide.md`) for a specific class or function. "
        "**This should be your FIRST choice for understanding how to use existing components in this repository.** "
        "It is very fast and provides official usage information. "
        "The input is the name of the symbol you are looking for. "
        "**The tool returns the single most relevant documentation section.**"
    )
```

POLITECNICO
MILANO 1863

Tool Implementation

- Website Search Tool
- Documentation Search Tool
- **Code Symbol Navigation Tool**
- Format Check Tool
- Code Interpreter Tool

- 2 main uses
  - Get functions/classes definitions
  - List all functions-classes in a file
- Leverages **tree-sitter** with Python parser (AST structure)



Tool Implementation

Code Symbol Navigation Tool

```
class CodeSymbolNavigationTool(BaseTool):
    name: str = "CodeSymbolNavigation"
    description: str = (
        "Inspects the structure of Python source code files **without reading the entire file into context.** "
        "This tool is essential for understanding a file's contents before deciding to read or modify it. "
        "It prevents context overflow errors when dealing with large files. It has two modes:\n"
        "1. **List Symbols (recommended first step):** "
        "Provide just a file path to get a summary of all classes and functions in that file. Use this to understand the file's structure.\n"
        "2. **Get Specific Source:** "
        "After listing symbols, provide a file path and a symbol name to get only the source code for that single class or function."
    )
```

Tool Implementation

- Website Search Tool
- Documentation Search Tool
- Code Symbol Navigation Tool
- **Format Check Tool**
- Code Interpreter Tool

Black

- **Black**: Python code formatter
- Detect syntax errors with **compile**
- Return formatted string or error msg

POLITECNICO
MILANO 1863

Tool Implementation

Format Check Tool

```python
class FormatCheckTool(BaseTool):
    name: str = "FormatCheck"
    description: str = (
        "A utility to automatically format Python code using the 'black' standard. "
        "**This is a useful final step to ensure code quality and style consistency before writing the code to a file.** "
        "It takes the raw Python code as a string and returns the formatted code as a string. "
        "This tool will fail if the input code has a syntax error."
    )
```

Tool Implementation

- Website Search Tool
- Documentation Search Tool
- Code Symbol Navigation Tool
- Format Check Tool
- **Code Interpreter Tool**

- Run snippets/scripts (create/modify/test)
- Security:
  - Timeout (20s) for run
  - Execution in temp file + child process

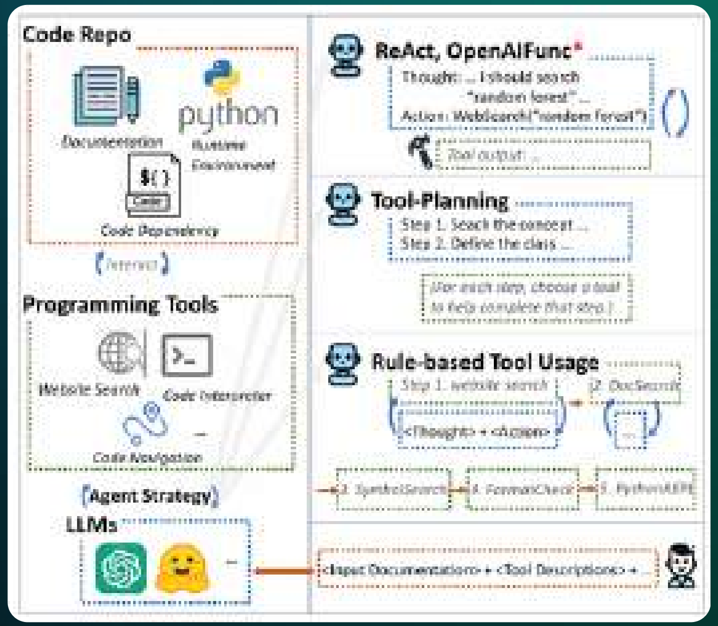
Tool Implementation

Code Interpreter Tool

```python
class CodeInterpreterTool(BaseTool):
    name: str = "CodeInterpreter"
    description: str = (
        "Executes a Python script in the project's root directory. This is your primary tool for all file system modifications and for testing your work."
        "You MUST use this tool to perform any of the following actions:"
        "\n1. **CREATE a new file:** Provide a script that opens a new file path in write mode ('w'). "
        "Example: `with open('path/to/new_file.py', 'w') as f: f.write('# New code')`"
        "\n2. **MODIFY an existing file:** Provide a script that reads the file, modifies the content, and writes it back. "
        "Example: `with open('path/to/file.py', 'r') as f: c = f.read() \n # ... modify c ... \n with open('path/to/file.py', 'w') as f: f.write(c)`"
        "\n3. **TEST code:** Provide a script that runs `pytest` or other checks to verify your changes. "
        "Example: `import pytest; pytest.main(['tests/test_file.py'])`"
        "\nThe tool returns the script's stdout and stderr, which you should use to confirm that your action was successful."
    )
```

POLITECNICO
MILANO 1863

Experiment Replication



1

LLM SETUP
- Small Quantized Model
- Large Models through API

2

TOOL IMPLEMENTATION
- WebSiteSearchTool
- DocumentationSearchTool
- CodeSymbolNavigationTool
- FormatCheckTool
- CodeInterpreterTool

3

AGENT INTEGRATION
- ReAct
- ToolPlanning
- Rule-based
- Tool-calling

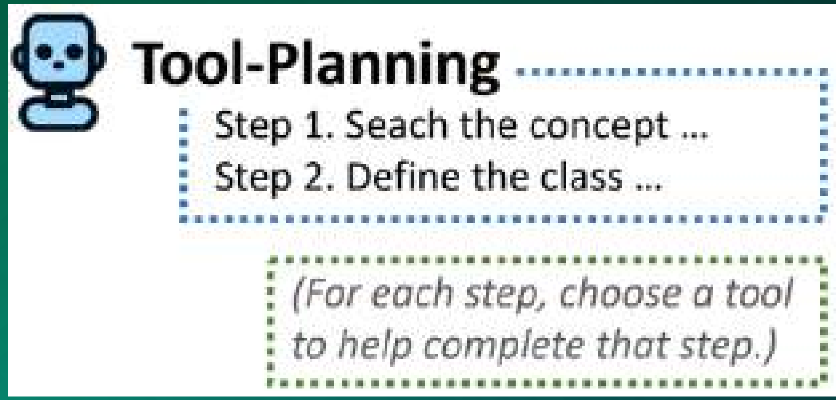
4

SYSTEM EVALUATION
- Repository-level testing
- Function-level testing

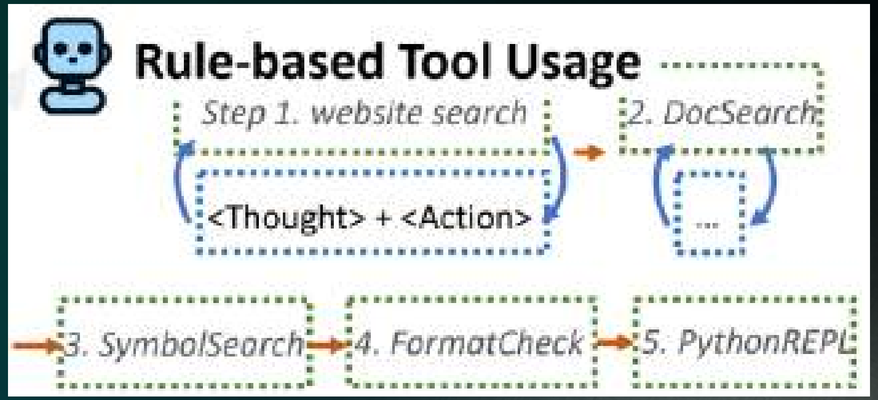
CodeAgent

Agent strategies for tools' usage (LLM as brain; tools as hands)

ToolPlanning



Rule-Based Tool



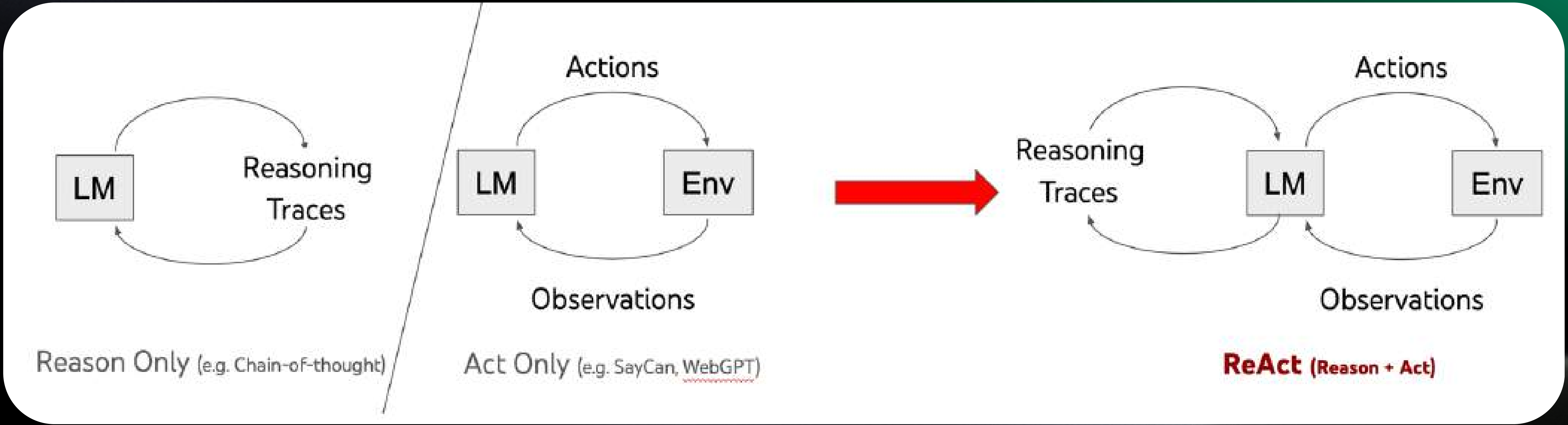
ReAct

Function Calling

OpenAI

Agent Integration

- **ReAct**: select and invoke suitable tools according to actions
  AgentExecutor: Though => Action => Observation => Repeat => End/Test (*assert*)

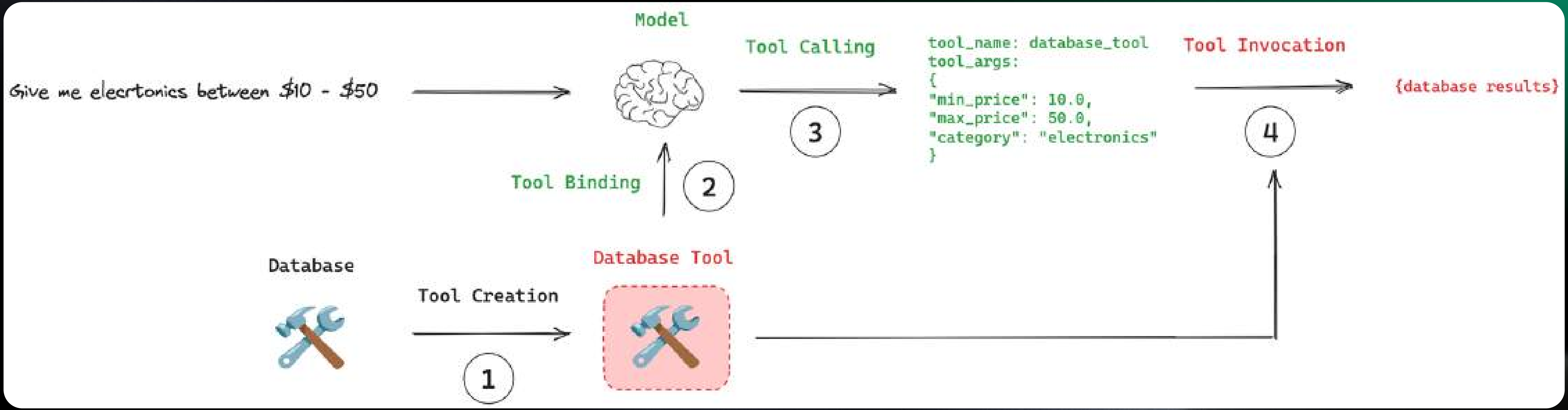


POLITECNICO MILANO 1863

Source: https://react-lm.github.io/

Agent Integration

- **Tool-calling:** *structured* way to *call tools* (for modern models)
  We've to be careful with **max_iterations** and **time** after each task



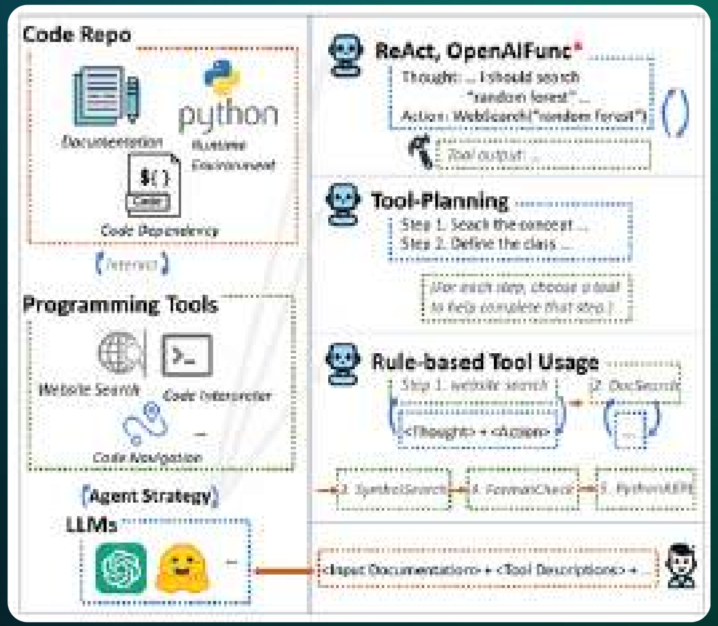
POLITECNICO
MILANO 1863

Source: *https://python.langchain.com/docs/concepts/tool_calling/*

6. System Evaluation

Repo-level: CodeAgentBench & MiniTransformers
Function-level: HumanEval

Experiment Replication



2

TOOL IMPLEMENTATION

- WebSiteSearchTool
- DocumentationSearchTool
- CodeSymbolNavigationTool
- FormatCheckTool
- CodeInterpreterTool

4

SYSTEM EVALUATION

- Repository-level testing
- Function-level testing

1

LLM SETUP

- Small Quantized Model
- Large Models through API

3

AGENT INTEGRATION

- ReAct
- ToolPlanning
- Rule-based
- Tool-calling

POLITECNICO
MILANO 1863

Literature Review

- Evaluation benchmarks

  - **Function-level** benchmarks
    - HumanEval, MBPP
  - **Repository-level** benchmarks
    - SWE-bench
    - CodeAgentBench

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

Source: Evaluating Large Language Models Trained on Code (arXiv:2504.10046)

POLITECNICO
MILANO 1863

CodeAgentBench

- Codebase

Environment: **numpy-ml** (classes + contextual information)

- High-star GitHub repositories
- Topics: machine learning, data structure, database, information extraction, networking
- 192 classes and 1431 functions

	path	content
0	numpy_ml/__init__.py	[# noqa\n, """Common ML and ML-adjacent algori...
1	numpy_ml/gmm/gmm.py	["""A Gaussian mixture model class"""\n, impor...
2	numpy_ml/gmm/__init__.py	[from .gmm import *\n]
3	numpy_ml/nonparametric/kernel_regression.py	[from ..utils.kernels import KernelInitializer...
4	numpy_ml/nonparametric/gp.py	[import warnings\n, import numpy as np\n, from...

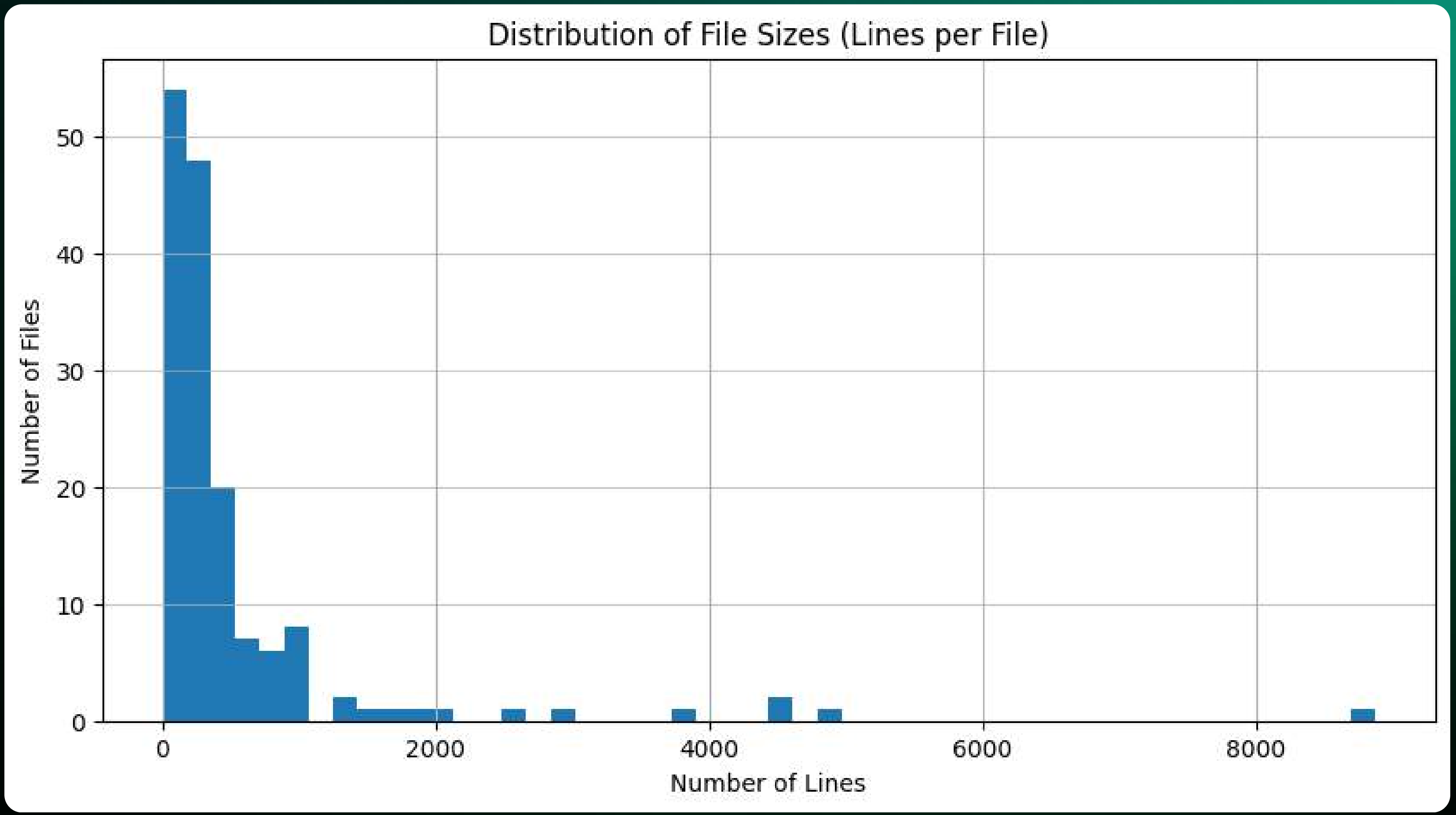
-> 156 entries

POLITECNICO
MILANO 1863

CodeAgentBench

- Codebase
  - Heavily left-skewed distribution
  - External Library Dependencies
    - *numpy* in 126 files
    - *time* in 40 files
    - *torch* in 37 files

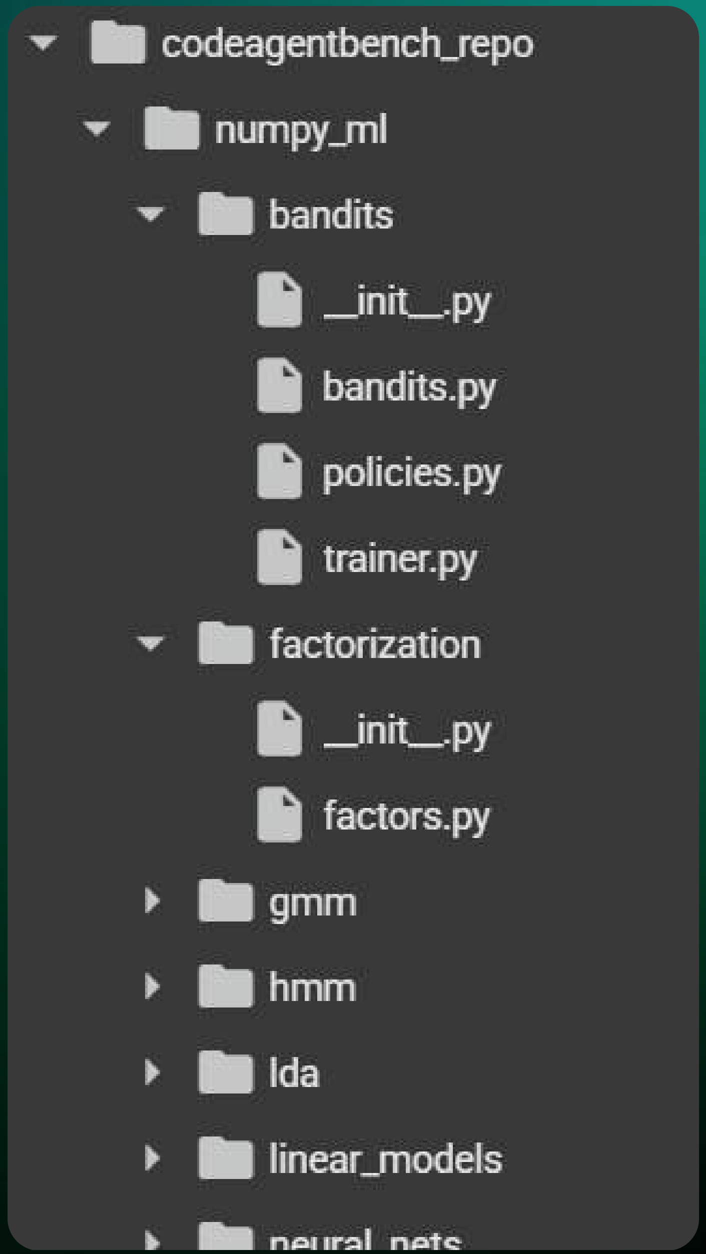
  - Internal Module Dependencies
    numpy_ml.neural_nets.activations in 58 files



POLITECNICO
MILANO 1863

CodeAgentBench

- Repository structure reconstruction
  - **tree-sitter** for parsing
  - creation of directories and files



......

CodeAgentBench

- **Tasks**

  Task defined in each entry with metadata

  - 51 class implementation tasks
  - 6 function-level tasks

The final CODEAGENTBENCH contains 101 samples, and for each task, LLMs are provided with documentation containing the requirements needed to be implemented, along with a set of tools we designed, as well as full access permissions to code files in the repository.

	title	class_annotation	comment	class_name	class_link	test_file_path
0	BallTree	numpy_ml.utils.d...	"BallTree"\n\n**...	numpy_ml.utils.d...	numpy_ml/utils/d...	numpy_ml/tests/t...
1	BatchNorm2D	numpy_ml.neural_...	"BatchNorm2D"\n\...	numpy_ml.neural_...	numpy_ml/neural_...	numpy_ml/tests/t...
2	RandomForest	numpy_ml.trees.R...	"RandomForest"\n...	numpy_ml.trees.R...	numpy_ml/trees/r...	numpy_ml/tests/t...
3	MLENGram	numpy_ml.ngram.M...	"MLENGram"\n\n**...	numpy_ml.ngram.M...	numpy_ml/ngram/n...	numpy_ml/tests/t...
4	BidirectionalLSTM	numpy_ml.neural_...	"BidirectionalLS...	numpy_ml.neural_...	numpy_ml/neural_...	numpy_ml/tests/t...

-> 57 entries

MiniTransformers

Gemini

- Repository-level benchmark

  Environment: core **miniformer** repository with additional sub-packages
  - Replicate the essential structure of large repositories
  - Generated by Gemini 2.5 Pro 06-05
  - 6 classes and 23 functions

	path	content
0	main.py	[# Main entry point for agent tasks. Initially...
1	README.md	[# Miniformer\n, A minimal, educational librar...
2	requirements.txt	[numpy\n, torch\n, scipy\n, pydantic]
3	miniformer/__init__.py	[from .models.block import TransformerBlock]
4	miniformer/config.py	[from pydantic import BaseModel, Field\n, from...

-> 22 entries

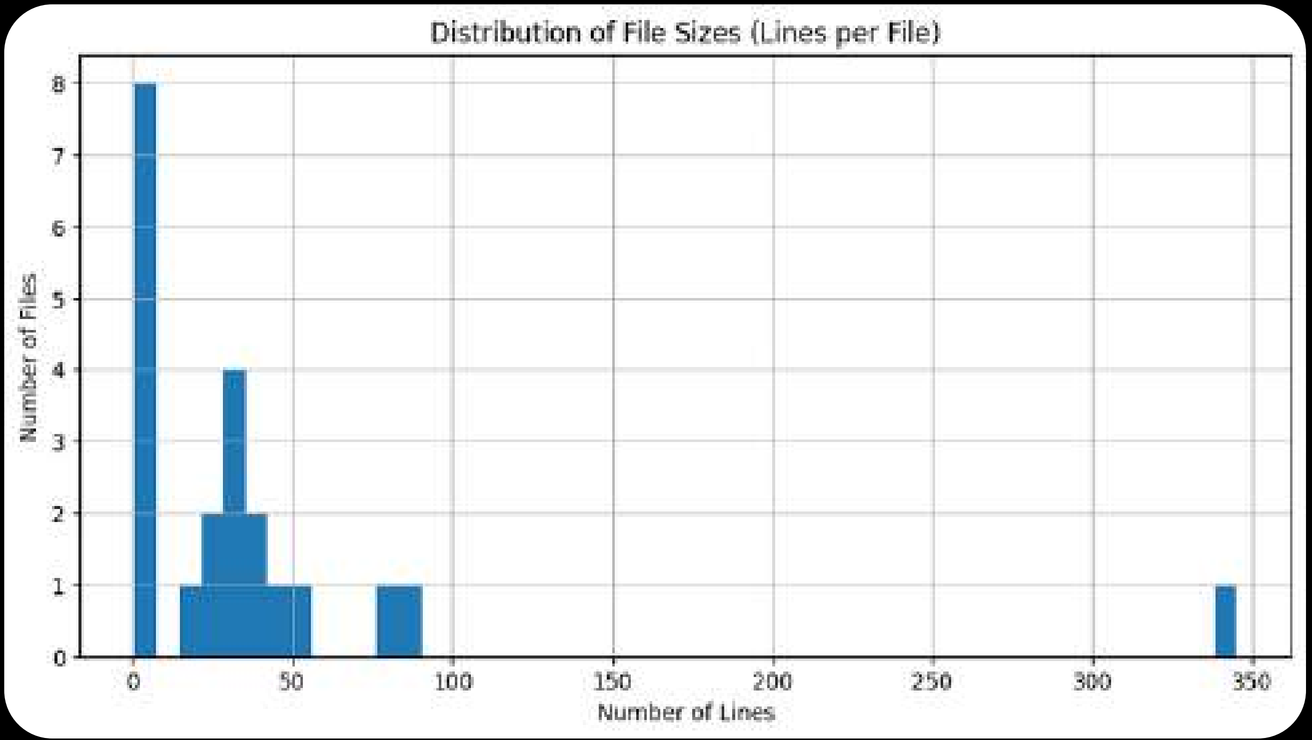
MiniTransformers

- Codebase

Statistically representative of real-world projects

- Generation guided by statistics of CodeAgentBench codebase
- Replicate file size distribution in scaled version

Lines per File	count	mean	min	25%	50%	75%	max
CodeAgentBench	156.00	542.49	2.00	113.50	265.50	512.50	8866.00
MiniTransformers	22.00	41.27	1.00	2.00	29.00	40.50	345.00



POLITECNICO
MILANO 1863

MiniTransformers

- Codebase
  - Core components to build and use transformers
  - External Library Dependencies
    - *torch* in 7 files
    - *pytest* in 5 files
    - *numpy* in 4 files

File Path: `miniformer/layers/attention.py`

Content Snippet (first 500 characters):

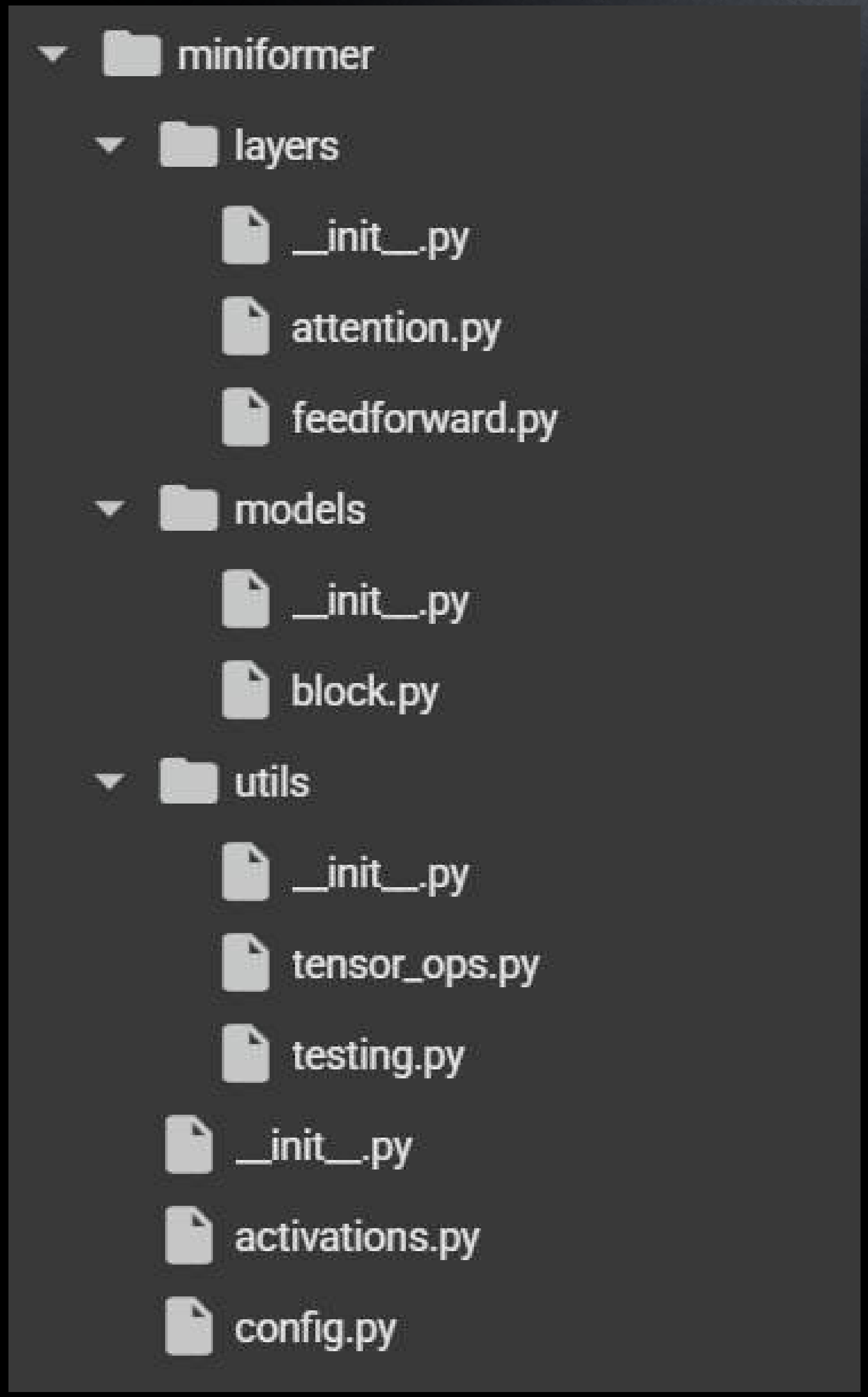
```
import torch

import torch.nn as nn

import torch.nn.functional as F

import math


class CausalSelfAttention(nn.Module):

    """A vanilla multi-head masked self-attention layer with a projection at the end."""

    def __init__(self, config):

        super().__init__()

        assert config.n_embd % config.n_head == 0

        # Key, query, value projections for all heads, but in a batch

        self.c_attn = nn.Linear(config.n_embd, 3 * config.n_embd)

        # Output projection

...
```

	path	line_count	class_count	function_count
15	tests/test_integration.py	345	0	0
6	miniformer/layers/attention.py	89	2	2
21	tests/test_created_files.py	79	0	3

POLITECNICO
MILANO 1863

MiniTransformers

- Repository reconstruction
  - Simulation of realistic file system
  - **tree-sitter** for parsing
  - creation of directories and files



POLITECNICO
MILANO 1863

MiniTransformers

- Generation of API Guide
  - Generate markdown document for source files
  - **tree-sitter** for parsing
  - function and class definitions

Miniformer API Guide

This guide details the core library components.

File: `miniformer/activations.py`

`gelu_exact(x)`
Gaussian Error Linear Unit (GELU) activation function using SciPy's erf for NumPy arrays.

`get_activation(name: str)`
Returns the activation function corresponding to the name.

File: `miniformer/config.py`

`TransformerConfig()`
Configuration for a Miniformer model.

File: `miniformer/models/block.py`

`TransformerBlock()`
A single block of a transformer model. It consists of a multi-head self-attention layer followed by a feed-forward network. Layer normalization and residual connections are applied.

File: `miniformer/layers/attention.py`

`CausalSelfAttention()`
A vanilla multi-head masked self-attention layer with a projection at the end.

`MultiHeadSelfAttention()`
Alias for CausalSelfAttention for clearer naming conventions.

File: `miniformer/layers/feedforward.py`

`FeedForward()`
A position-wise feed-forward network.

POLITECNICO
MILANO 1863

MiniTransformers

- Tasks

15 programming tasks, metadata generated by Python's **ast** module

- Prompt written in documentation-style format
- From simple file modifications to code refactoring and file creation
- Support executable test suite verification

	title	class_annotation	comment	class_name	class_link	test_file_path	task_id
0	Transformer...	miniformer....	"Add Bias C...	miniformer....	miniformer/...	tests/test_...	miniformer-01
1	to_numpy	miniformer....	"Verify Ten...	N/A	miniformer/...	tests/test_...	miniformer-02
2	Transformer...	miniformer....	"Implement ...	miniformer....	miniformer/...	tests/test_...	miniformer-03
3	get_activation	miniformer....	"Add Swish ...	N/A	miniformer/...	tests/test_...	miniformer-04
4	PositionalE...	miniformer....	"Create Pos...	miniformer....	miniformer/...	tests/test_...	miniformer-05

-> 15 entries

MiniTransformers

- Tasks

```
"Add Swish Activation Support"

function miniformer.activations.get_activation(name)

Extend the activation function factory to include support for 'swish'.

-[ Notes ]-
The Swish activation function, defined as `f(x) = x * sigmoid(x)`, is a smooth, non-monotonic function that often matches

Implementation Steps:
1.  Implement a new Python function `swish(x)` that computes the activation. It should use `torch.sigmoid`.
2.  Modify the `get_activation` function to return a reference to your `swish` function when the input `name` is 'swish'.
```

MiniTransformers

- Tests
  - Support executable test suite verification

```python
import pytest
import torch
from miniformer.activations import get_activation

def test_swish_activation():
    # This test is expected to fail until Task #4 is completed.
    try:
        swish = get_activation('swish')
        x = torch.tensor([1.0, 2.0, -1.0])
        expected = x * torch.sigmoid(x)
        assert torch.allclose(swish(x), expected)
    except ValueError:
        pytest.fail("Activation 'swish' is not registered in get_activation.")
```

Evaluation

REPOSITORY-LEVEL TESTING: MINITRANSFORMERS, PASS@1

AgentStrategy	DeepSeekV3	GPT-4.1-mini	GPT-4.1-nano	Gemini 2.5 Flash
NoAgent	0.333	0.333	0.333	0.533
ReAct	0.533	0.800	0.400	0.667
Tool-calling	-	0.867	0.467	0.733
Rule-based	0.600	0.667	0.400	0.667
Tool-Planning	0.467	0.733	0.333	0.600

- Low success rate in NoAgent: recurrent errors on missing imports and failing function calls
- Performance of GPT-4.1 mini boosted by agent strategy
- GPT-4.1 nano may be limited by the small model size
- DeepSeekV3 could need different approach or few-shot learning
- Gemini 2.5 Flash may share affinity with model used to generate benchmark
- Results may slightly differ among executions for non determinism

HumanEval

- Function-level code generation benchmark
  - Most popular Python code generation benchmark nowadays
  - 164 handwritten programming problems with about 8 test units for each task

	task_id	prompt	entry_point	canonical_solution	test
HumanEval/0	HumanEval/0	from typing import List\n\n\ndef has_close_ele...	has_close_elements	for idx, elem in enumerate(numbers):\n ...	\n\nMETADATA = {\n 'author': 'jt',\n 'da...
HumanEval/1	HumanEval/1	from typing import List\n\n\ndef separate_pare...	separate_paren_groups	result = []\n current_string = []\n ...	\n\nMETADATA = {\n 'author': 'jt',\n 'da...
HumanEval/2	HumanEval/2	\n\ndef truncate_number(number: float) -> floa...	truncate_number	return number % 1.0\n	\n\nMETADATA = {\n 'author': 'jt',\n 'da...

HumanEval

- Function-level code generation benchmark

```python
from typing import List


def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """

------------


METADATA = {
    'author': 'jt',
    'dataset': 'test'
}


def check(candidate):
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
    assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
    assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
    assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True
    assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
    assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False
```

Evaluation

FUNCTION-LEVEL TESTING: HUMANEVAL, PASS@1

AgentStrategy	DeepSeekV3	GPT-4.1-mini	GPT-4.1-nano	Gemini 2.5 Flash
NoAgent	0.781	0.805	0.701	0.793
Tool-calling	-	0.848	0.768	0.842
ReAct	0.817	0.829	0.756	0.829
Rule-based	0.823	0.835	0.762	0.829
Tool-Planning	0.811	0.823	0.744	0.817

- Documentation reading and code symbol navigation tools disabled
- HumanEval tasks easier to handle
- GPT-4.1 mini results the best
- Remarkable performance of all models
- Results may slightly differ among executions for non determinism

Replication Summary

Key Features:

- Open source and proprietary LLMs as base model
- **Custom tools** implementation
- **Agent strategies** implementation
- Extensive **unit** and **integration testing**
- Evaluation from multiple approaches
- Validity of framework and tools assessed

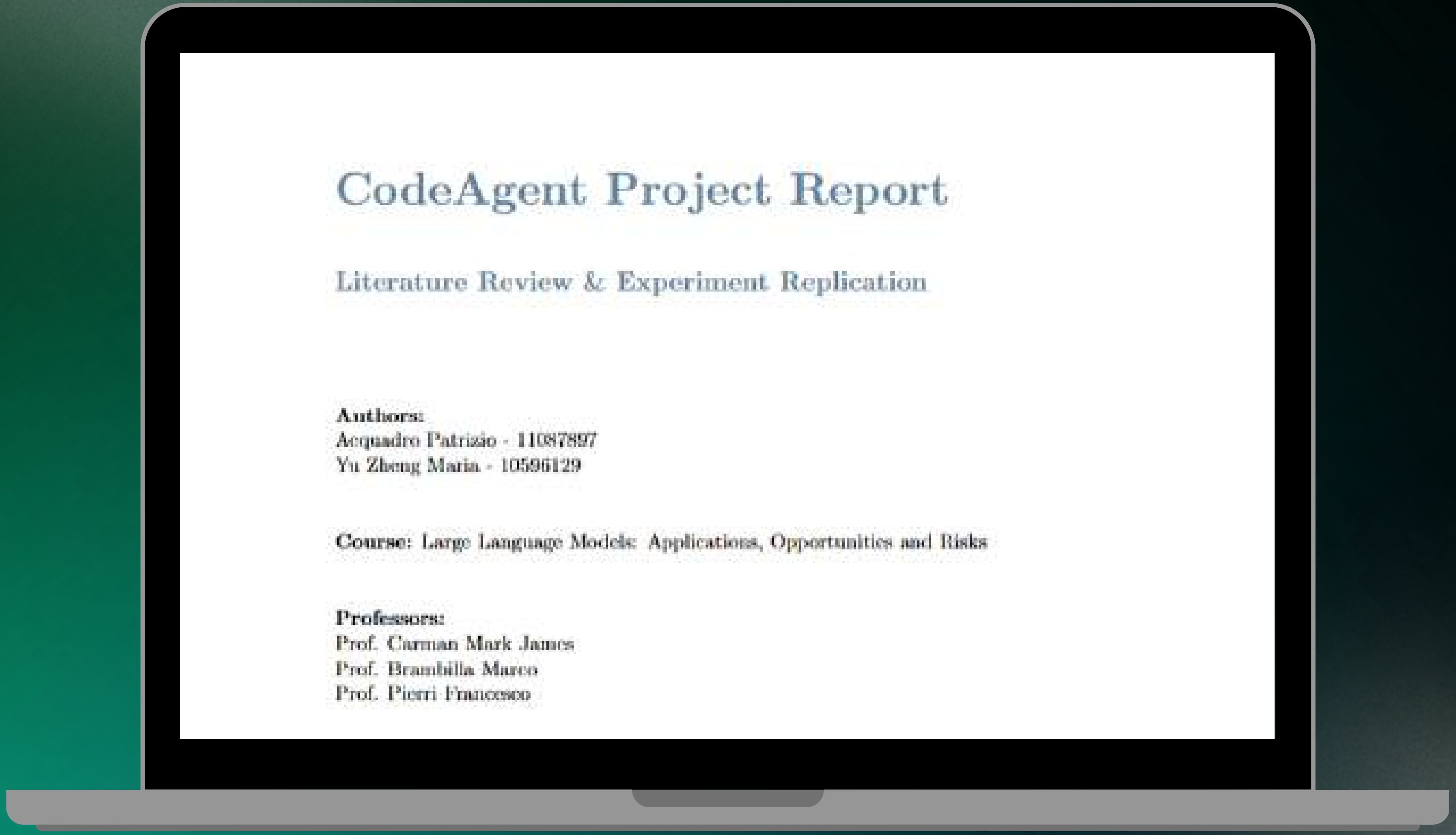
Major Challenges

- Local computational resources
- Agent stuck in self-doubt loops
- Undefined Function-calling behaviour

Solutions

- Model quantization
- Adopt more effective models
- Consider few-shot learning

POLITECNICO
MILANO 1863


CodeAgent Project Report

Literature Review & Experiment Replication

Authors:
Acquadro Patrizio - 11087897
Yu Zheng Maria - 10596129

Course: Large Language Models: Applications, Opportunities and Risks

Professors:
Prof. Carman Mark James
Prof. Brambilla Marco
Prof. Pierri Francesco

Conclusion

Successful replication of **CodeAgent framework**, by implementing the entire suite of tools and the agent strategies

Creation of **MiniTransformer** benchmark

Future perspectives

Multi-agent systems (AgentCoder)
Integration of knowledge graphs (CodeRAG)
Explainability for LLMs

POLITECNICO
MILANO 1863

POLITECNICO
MILANO 1863

Thank You!

patrizio.acquadro@mail.polimi.it
zhengmaria.yu@mail.polimi.it

GitHub repository