GenAI for Software Engineering 2025

In-context learning strategies

Jorge Melegati

In-context learning strategies

- Standard Prompt Engineering ✓
- Chain-of-thought
- Multiple agents

Can you do math on your head?

$7 \; x \; 8 = ?$

$17 \; x \; 8 = ?$

$172 \; x \; 87 = ?$

$$\begin{array}{r} 172 \\ x \;\; 87 \\ \hline 1204 \\ 1376 \\ \hline 14964 \end{array}$$

Chain of thought

Multiplication procedure

- We learn multiplication tables in the school
- But it is impossible to learn for all the numbers
- We learn a procedure to break down the problems into known problems -> values in the multiplication tables

That’s like LLM models!

- They learned several patterns from the training data
- But not all possible problems
- If we are able to break down larger problems into a chain of smaller ones, easier to the model to identify the patterns

Chain-of-thought

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**
Google Research, Brain Team
{jasonwei,dennyzhou}@google.com

Abstract

We explore how generating a *chain of thought*—a series of intermediate reasoning steps—significantly improves the ability of large language models to perform complex reasoning. In particular, we show how such reasoning abilities emerge naturally in sufficiently large language models via a simple method called *chain-of-thought prompting*, where a few chain of thought demonstrations are provided as exemplars in prompting.

Experiments on three large language models show that chain-of-thought prompting improves performance on a range of arithmetic, commonsense, and symbolic reasoning tasks. The empirical gains can be striking. For instance, prompting a PaLM 540B with just eight chain-of-thought exemplars achieves state-of-the-art accuracy on the GSM8K benchmark of math word problems, surpassing even finetuned GPT-3 with a verifier.

Chain-of-thought (CoT)

- A chain of thought is a series of intermediate natural language reasoning steps that lead to the final output.
- Strictly speaking, it is a form of prompt engineering

CoT example

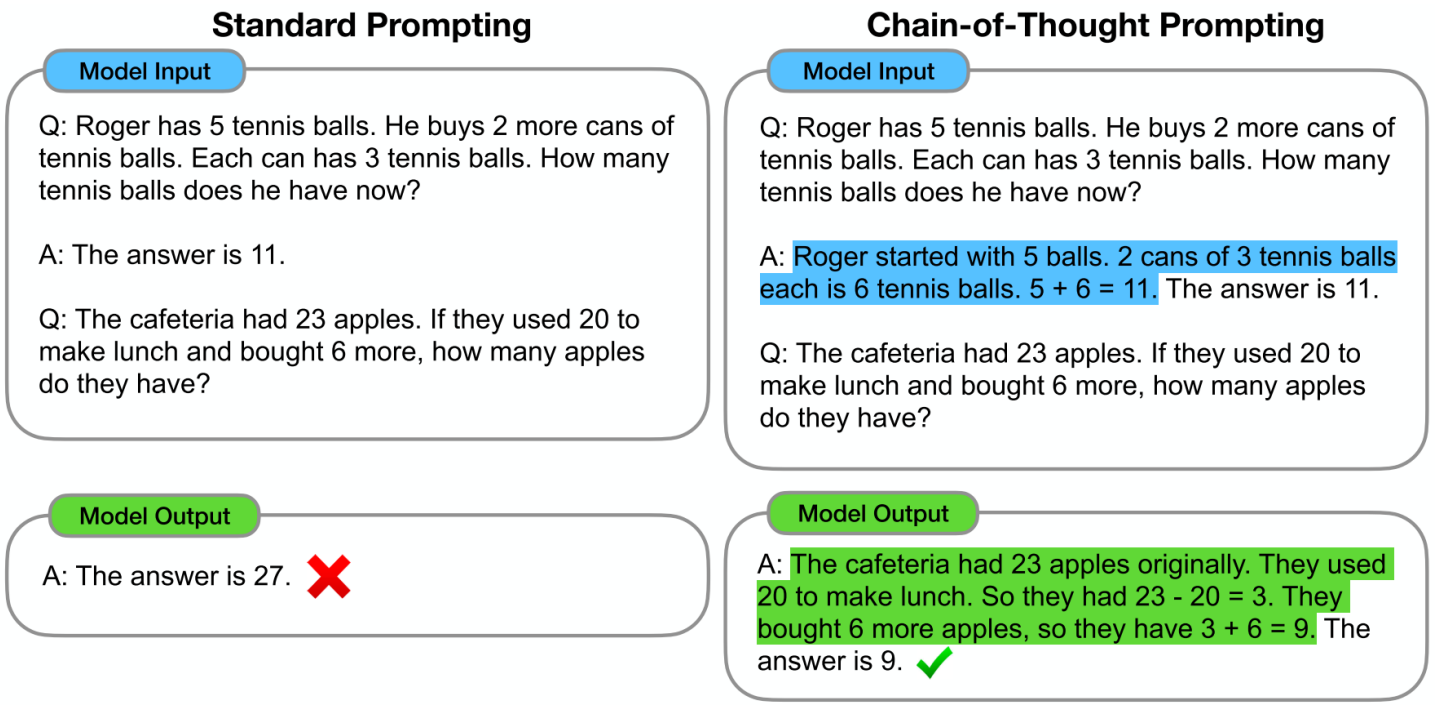


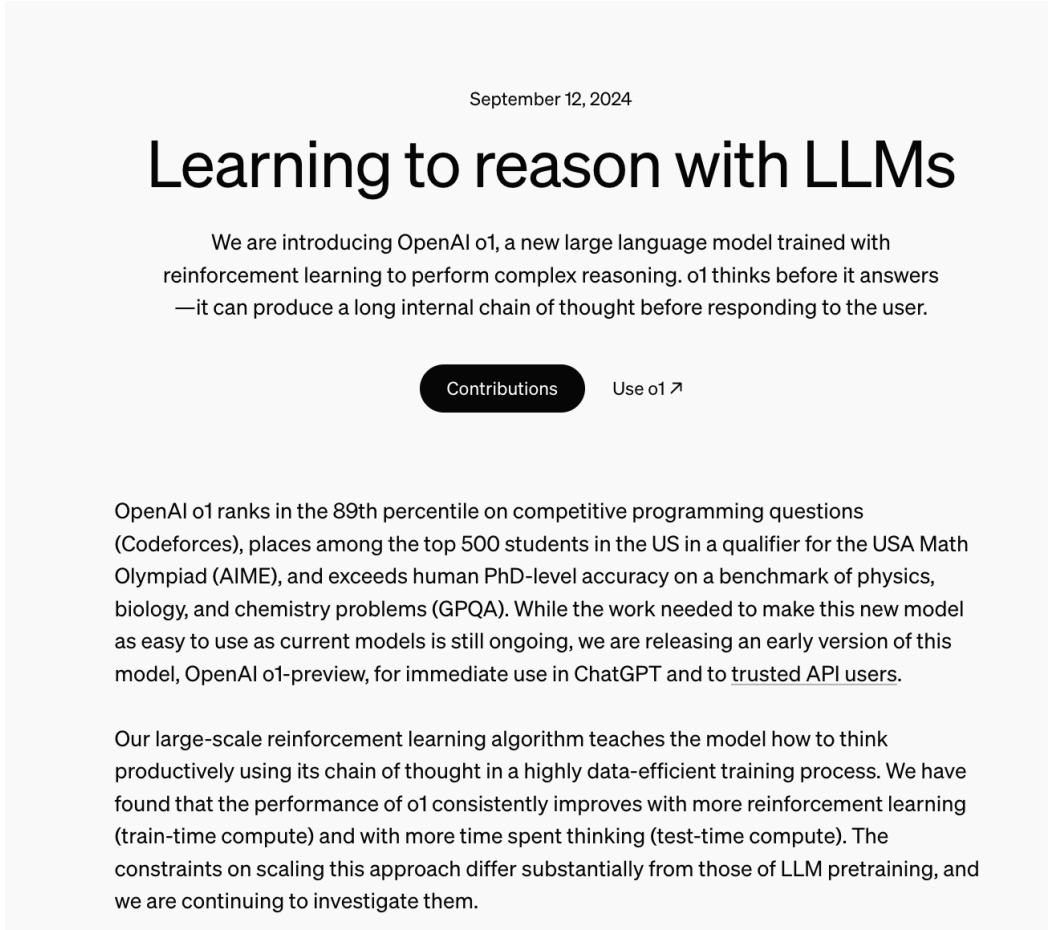
Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Source: Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models” (2022)

Advantages

- Increase the explainability of the answer
  - How it arrived at that answer
- Provide steps to debug when the answer is wrong
- Allow the model to better access relevant knowledge acquired during training

CoT is now part of training processes



September 12, 2024

Learning to reason with LLMs

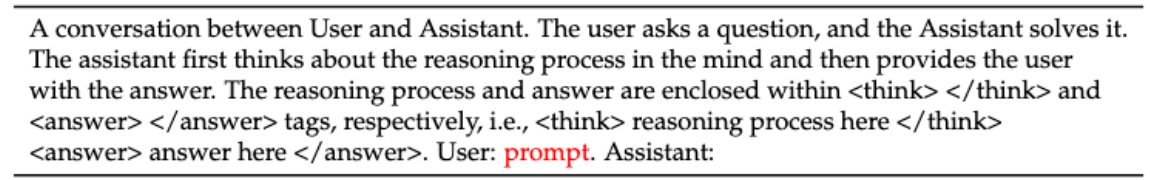
We are introducing OpenAI o1, a new large language model trained with reinforcement learning to perform complex reasoning. o1 thinks before it answers —it can produce a long internal chain of thought before responding to the user.

Contributions Use o1 ↗

OpenAI o1 ranks in the 89th percentile on competitive programming questions (Codeforces), places among the top 500 students in the US in a qualifier for the USA Math Olympiad (AIME), and exceeds human PhD-level accuracy on a benchmark of physics, biology, and chemistry problems (GPQA). While the work needed to make this new model as easy to use as current models is still ongoing, we are releasing an early version of this model, OpenAI o1-preview, for immediate use in ChatGPT and to trusted API users.

Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining, and we are continuing to investigate them.

Source: https://openai.com/index/learning-to-reason-with-llms/



A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>. User: prompt. Assistant:

Table 1 | Template for DeepSeek-R1-Zero. prompt will be replaced with the specific reasoning question during training.

Source: DeepSeek-AI et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning https://arxiv.org/abs/2501.12948

The so-called “reasoning models”

CoT in DeepSeek

- Let’s check DeepSeek-R1’s CoT

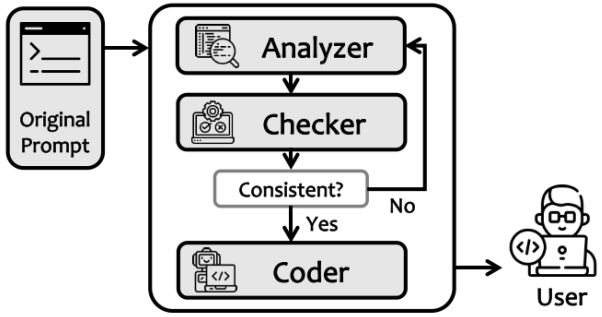
- First, install ollama (https://ollama.com/)
  - It allows to locally run open source LLMs

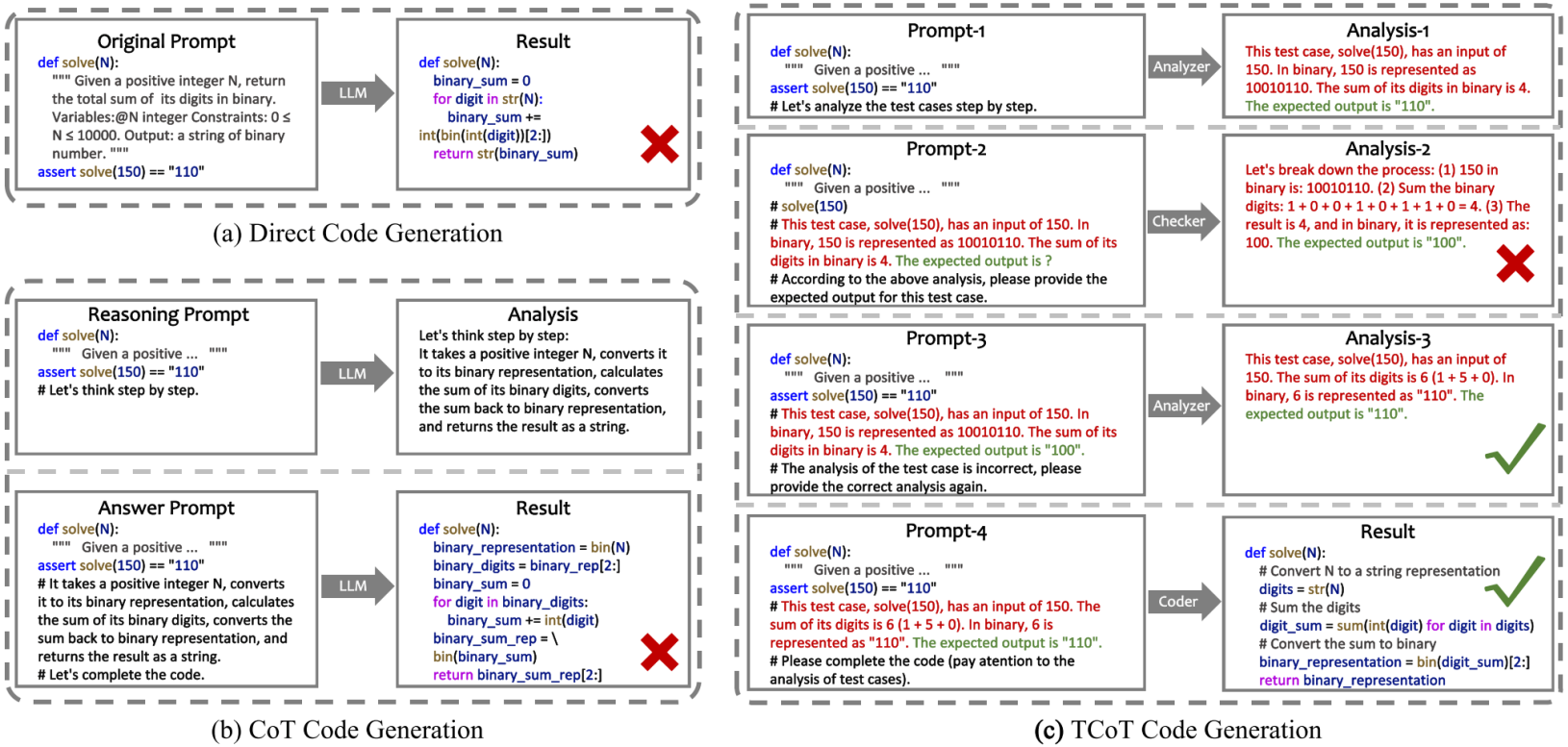
- Then, run the smallest DeepSeek-R1 version (1.5b parameters)
  - ollama run deepseek-r1:1.5b

- Now, ask something and check the text in the <think> tags

TCoT: the test-case-driven CoT technique

Goal: generate code based on the tests

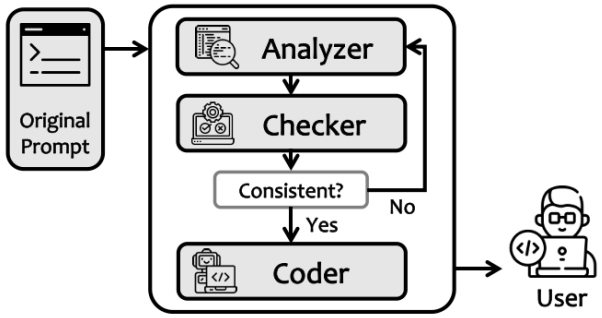


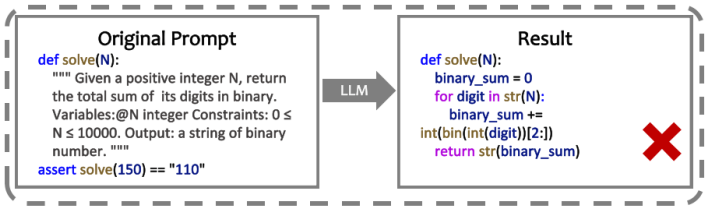


Source: Tian and Chen, “Test-Case-Driven Programming Understanding in Large Language Models for Better Code Generation” (2023)

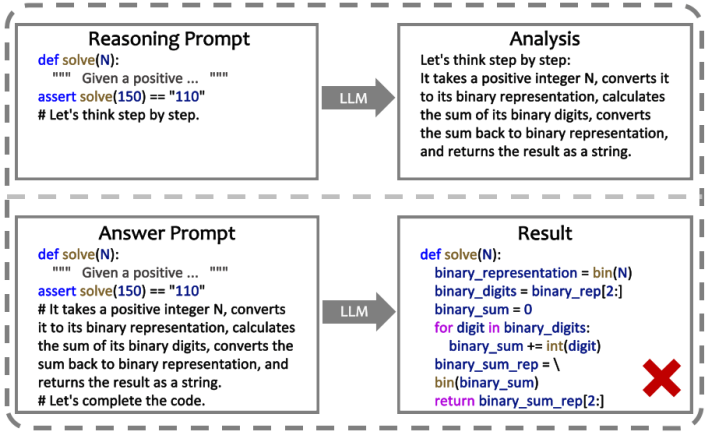
TCoT: the test-case-driven CoT technique

Goal: generate code based on the tests

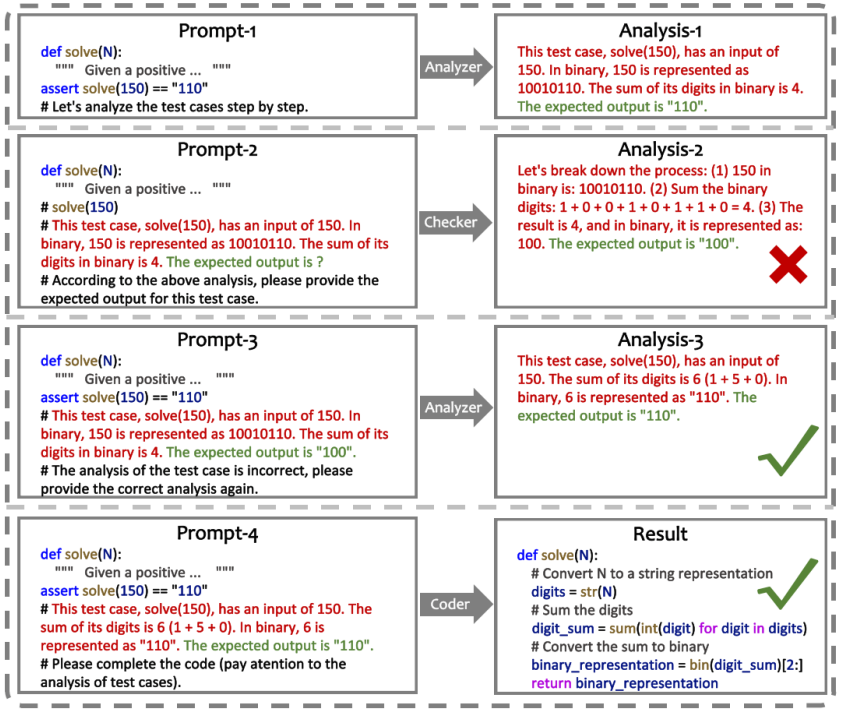




(a) Direct Code Generation



(b) CoT Code Generation

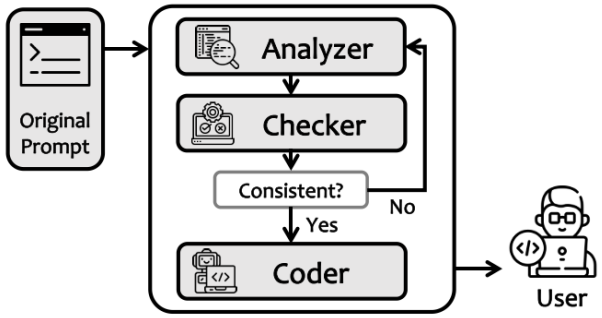


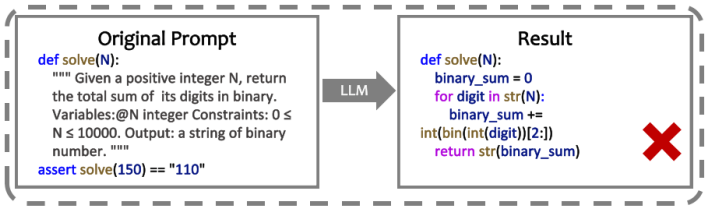
(c) TCoT Code Generation

Source: Tian and Chen, “Test-Case-Driven Programming Understanding in Large Language Models for Better Code Generation” (2023)

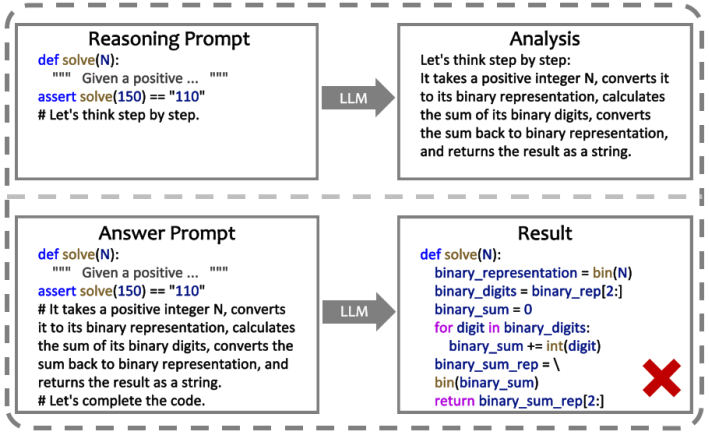
TCoT: the test-case-driven CoT technique

Goal: generate code based on the tests

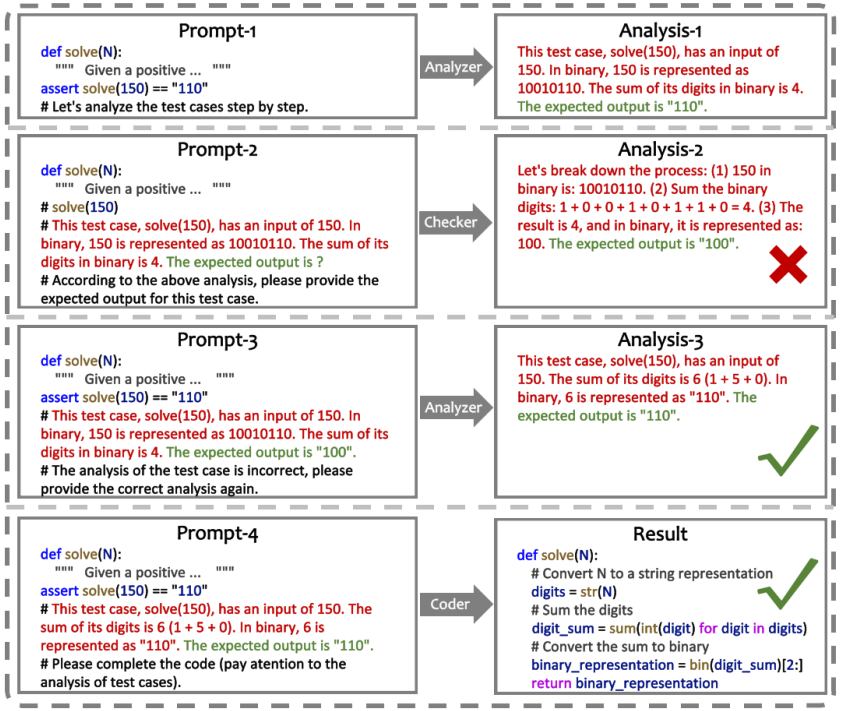




(a) Direct Code Generation



(b) CoT Code Generation



(c) TCoT Code Generation

Source: Tian and Chen, “Test-Case-Driven Programming Understanding in Large Language Models for Better Code Generation” (2023)

ThinkRepair: automated program repair with CoT

- Let’s check the paper (Section 3):
  - https://dl.acm.org/doi/10.1145/3650212.3680359

Docker image

- For the next exercises, we will use a Docker image

- Run the following in a terminal

export OPENAI_API_KEY=<key>

docker run –e OPENAI_API_KEY –d melegati/thinkrepair

- Once the container is running, connect your IDE to it

Multiple Agents: Remember the Persona pattern?

- Intent: To make the LLM output to take a certain point of view

- Fundamental contextual statements:
  - “Act as persona X”
  - “Provide outputs that persona X would create”

Problem

- Many SE activities are performed by the collaboration of different roles
- Example: prioritization of user stories involves developers, product owners, tech leaders, etc.
- Each role brings a specific knowledge to the table (and has specific goals)
- But how to replicate that with the models?

Multiple agents

- Idea: combine several instances of LLMs, each instructed for a specific subtask, to perform a task through discussion

- Advantage: use prompt engineering techniques for subtasks has a greater chance of success
  - Facilitates the access to knowledge (like CoT)

Example: user stories prioritization

- Paper:
  - https://doi.org/10.1007/978-3-031-78386-9_20

- Original repository:
  - https://github.com/GPT-Laboratory/multiagent-prioritization

Early Results of an AI Multiagent System for Requirements Elicitation and Analysis

Malik Abdul Sami[1(✉)], Muhammad Waseem[2], Zheying Zhang[1], Zeeshan Rasheed[1], Kari Systä[1], and Pekka Abrahamsson[1]

[1] Tampere University, Tampere, Finland
{malik.sami,zheying.zhang,zeeshan.rasheed,
kari.systa,pekka.abrahamsson}@tuni.fi
[2] University of Jyväskylä, Jyväskylä, Finland
muhammad.m.waseem@jyu.fi

Abstract. In agile software development, user stories capture requirements from the user's perspective, emphasizing their needs and each feature's value. Writing concise and quality user stories is necessary for guiding software development. Alongside user story generation, prioritizing these requirements ensures that the most important features are developed first, maximizing project value. This study explores the use of Large Language Models (LLMs) to automate the process of user story generation, quality assessment, and prioritization. We implemented a multi-agent system using Generative Pre-trained Transformers (GPT), specifically GPT-3.5 and GPT-4o, to generate and prioritize user stories from the initial project description. Our experiments on a real-world project demonstrate that GPT-3.5 handled user story generation well, achieving a higher semantic similarity score comnpared to the GPT-4o. Both models showed consistent performance in prioritizing requirements, effectively identifying the core features of the application. These early results indicate that LLMs have significant potential for automating requirements analysis, particularly generating and prioritizing user stories.

Keywords: Software Engineering · Requirements Engineering · Requirements prioritization · Generative AI · Large language model(s)

Process

Roles:

- Product Owner (PO)
- Quality Assurance specialist (QA)
- Developers (DEV)

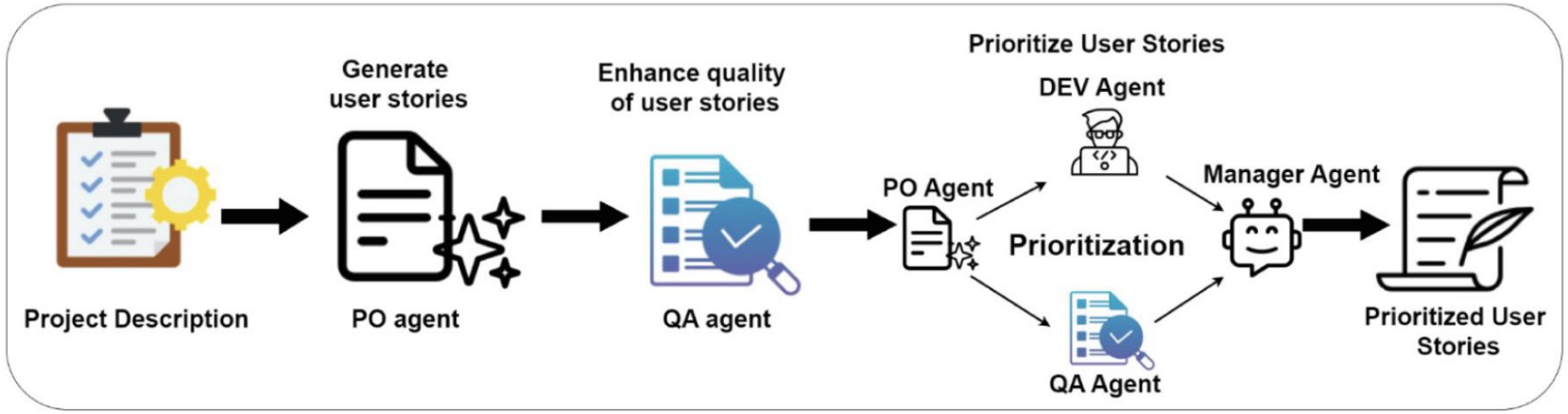


Fig. 1. Process Model of LLM Agents for User Story Generation, Quality Check, and Prioritization.

Docker image

- For the next exercises, we will use a Docker image

- Create a file for the environment variables

export API-KEY1= <key>
export API-KEY2= <key>
export API-KEY3= <key>

- Run the following in a terminal

docker run –env-file=<env-file> -p 8000:8000 –d melegati/multiagent-prioritization

- Once the container is running, open in the browser http://localhost:8000

Exercise

- Connect to container and open the code the folder
  - /multiagent-prioritization

- Try to improve the prompts for the roles in the file helpers.py:
  - PO: method construct_product_owner_prompt
  - Dev: method construct_senior_developer_prompt
  - QA: method construct_senior_qa_prompt