From Language to Action: Embodied Cognition with LLM-Based Agents

Cognitive Robotics Project Report

Fabian Huppertz, Shinas Shaji

***Abstract*—Large Language Models (LLMs) are increasingly used as decision-making components in embodied AI, yet their integration into robotic systems remains underexplored. This project investigates how an LLM can serve as the cognitive core of a simulated mobile manipulator operating in a household environment. The agent is embodied in a PyBullet-based simulation as a mobile manipulator with an omnidirectional base and a 7-DoF robotic arm. It can interact with its environment through a set of high-level tool functions for perception, reasoning, navigation, grasping, and placement. Two household tasks, object placement and shelf reordering, were designed to evaluate the agent's reasoning, planning, and memory utilization. Results demonstrate that the LLM-driven agent can complete structured tasks and adapt strategies, but also reveal limitations such as poor instruction following and difficulties in managing complex tasks including non-obvious subtasks. These findings highlight both the potential and challenges of employing LLMs as embodied cognitive controllers for autonomous robots. Link to a GitHub repository containing all code https://github.com/FHuppertz/CogRob_Project**

***Index Terms*—Cognitive Robotics, Embodied AI, Large Language Models, Simulation**

I. Introduction

Large Language Models (LLMs) have been shown to exhibit impressive abilities in reasoning, language understanding, and task execution. This project explores how an LLM can serve as the core of a cognitive architecture embodied in a simulated 3D robot. By assigning the agent physical tasks in a realistic environment, we aim to investigate how such systems can perceive, plan, act, and adapt in grounded, interactive settings.

A. Motivation

To fully explore the cognitive capabilities of large language models, it is essential to situate them in environments that demand embodied, goal-directed interaction. This project investigates how LLMs function as the core of a cognitive architecture within a simulated 3D robotic setting, where the agent must physically manipulate objects and execute tasks in the real world.

Unlike static text-based benchmarks, a robotic context enables examination of embodied reasoning, memory-guided action, and the capacity for grounded, anticipatory planning. By focusing on a single agent operating in a realistic environment (such as a kitchen), we can probe how LLMs reason about spatial layouts, interpret user instructions, and adapt plans based on feedback.

Importantly, the robot's (and by extension, the LLM agent's) planning and decision-making capabilities take on a more grounded role, reasoning in natural language about the physical consequences of movements and manipulative actions. This opens new directions for exploring embodied cognition and could inform future developments in cognitive robotics, assistive AI, and multi-modal agent systems.

B. Problem Statement

The central problem addressed in this project is how to effectively utilize large language models as cognitive controllers for embodied agents in realistic 3D environments. Specifically, we investigate how an LLM-driven agent can successfully perform complex household tasks that require physical interaction through navigation and object manipulation.

Key challenges arise in bridging high-level natural language reasoning with low-level embodied actions. The agent must interpret abstract task descriptions and translate them into sequences of precise robotic actions such as navigation, grasping, and placement. This translation process is complicated by the inherent ambiguity in natural language instructions, leading to misinterpretations where agents may attempt to manipulate inappropriate objects.

Additionally, we explore how episodic memory can enhance task performance through experience accumulation. The agent's ability to learn from previous failures and successes is critical for adapting strategies in complex tasks.

C. Proposed Approach

To address this problem, we propose a cognitive agent architecture that embeds an LLM as its central reasoning engine and situates it in a 3D PyBullet simulation of a household environment. Here, it is embodied as a mobile manipulator with a 7-DoF arm mounted on an omnidirectional base, enabling navigation and object manipulation.

The agent receives observations from the environment primarily as natural language descriptions detailing the positions and types of objects and entities. Memory is organized into two main tiers: working memory in the form of a prompt for immediate task information and semantically searchable episodic logs for synthesized summaries of past actions. The LLM is prompted with chain-of-thought prompting to reason and form plans accounting for objects, entities, and anticipated outcomes.

*Submitted to the Department of Computer Science at Hochschule Bonn-Rhein-Sieg in partial fulfilment of the requirements for the degree of Master of Science in Autonomous Systems

†Supervised by Alex Mitrevski

‡Submitted in August 2025

A set of high-level tool functions were implemented to expose perception, navigation, grasping, and placement as callable actions for the LLM. The environment features a kitchen and living room layout, in which the robot must carry out tasks like pick-and-place or sorting objects between designated areas. Object placement and shelf reordering were chosen to test the agent's ability to execute complex high level tasks, utilize its memory system, and adapt to constraints such as occupied spaces. This work evaluates the extent to which LLMs can act as cognitive controllers for robots and identifies their strengths and limitations in performing high level tasks.

II. Related Work

This project draws on multiple domains:

A. Cognitive Architectures

Cognitive architectures such as SOAR, which integrates symbolic reasoning with perceptual and motor systems for cognitive robotics [1], and ACT-R, which offers a hybrid symbolic-subsymbolic framework for modeling human cognition [2].

B. Embodied AI and Reinforcement Learning

Embodied-AI and reinforcement-learning agents have been used within simulation frameworks like OpenAI Gym, DeepMind Lab, and MineDojo. Generative models have been used to power social agents that produce believable simulacra of human behaviour, as exemplified by Park et al. [3]. Complementary work on open-ended embodied agents, such as Voyager, a GPT-4-powered agent that incrementally builds a skill library while autonomously exploring Minecraft [4], demonstrates lifelong learning in complex environments.

C. Vision-Language-Action Policies

Generalist vision-language-action policies like Octo map multimodal observations to diverse robot actions and tasks [5]. Our project extends these directions by introducing an explicit cognitive architecture that equips an embodied robot with hierarchical memory, anticipatory reasoning, and explainable decision-making.

D. LLM-Driven Robot Control

Recent research has also explored LLM-driven control of simulated or real robots capable of translating natural-language instructions into sensorimotor actions. Notable examples include:

- RT-2, a vision-language-action model that transfers web-scale knowledge to real-world robotic control [6]
- PaLM-E, an embodied multimodal language model that fuses a large-scale LLM with visual and state inputs [7]
- Code-as-Policies, which prompts code-generating LLMs to produce executable robot control policies [8]
- Instruct2Act, a framework that converts multimodal instructions into perception-planning-action code, outperforming specialist policies on tabletop manipulation tasks [9]

III. Methodology

A. LLM Cognition

The robot's cognitive architecture is built around a large language model (LLM) that serves as the central decision-making component. This LLM-based approach allows the robot to process natural language instructions, reason about its environment, and select appropriate actions through a tool-calling interface.

At the core of the robot's cognition system is its memory architecture, which consists of three distinct components that mirror cognitive processes: episodic memory, working memory, and action execution. The episodic memory is implemented using ChromaDB, a vector database that stores the robot's past experiences as embeddings. This allows the robot to perform similarity searches through its previous interactions, retrieving relevant experiences that can inform current decision-making processes. When the robot encounters a new task, it can query this episodic memory to find similar past scenarios and adapt successful strategies from those experiences.

The working memory of the system is represented by the LLM's context, which dynamically incorporates environmental information, task descriptions, and the robot's internal reasoning. This working memory is updated at each step with information such as the current perceptual data from the environment and results of actions or tool calls ensuring that the robot's decisions are based on the most recent state of the world.

The action execution module consists of a toolkit of functions that translate high-level decisions into concrete robotic actions. These tools include navigation, manipulation, memory operations, and reasoning capabilities through a scratchpad mechanism. The perception components gather information about the robot's current location, nearby objects, and environmental layout, which is then integrated into the working memory to inform decision-making.

This cognitive architecture enables the robot to combine long-term learning from past experiences with immediate environmental awareness, allowing for adaptive and context-aware behavior in complex environments.

B. Simulation

The agent and its environment were simulated using the PyBullet physics engine. As PyBullet is a Python module, its functionality can be directly integrated in any Python script. This allows precise control over the simulation and its interaction with external tool calls. PyBullet also comes with methods to create custom objects as well as a fully implemented robotic arm, the KUKA iiwa model. These methods and the KUKA iiwa robotic arm were used to create the agent's embodiment and the environment it can act in.

The agent's body is modeled as a mobile manipulator, consisting of a square omnidirectional base and a 7-DoF robotic arm derived from the KUKA iiwa model. To increase maneuverability, the rotational joint limits of the arms joint connecting the base to the arm were disabled, enabling full 360° reach around the base. A fully articulated gripper was not

implemented. Instead, grasping and placing were simplified by attaching objects directly to the end-effector when within a predefined proximity threshold, and detaching them at the desired placement location. Figure 1 depicts the agent.

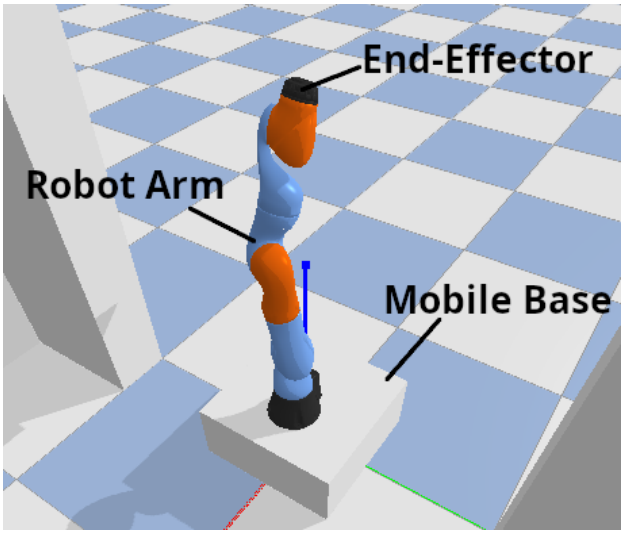

Fig. 1: The simulated mobile manipulator consisting of a square omnidirectional base and a 7-DoF arm.

The environment was designed as a simplified two-room household, comprising a kitchen and a living room. The kitchen contains a three-layer shelf and a table, both capable of supporting objects. The living room contains a television placed on a table. The two rooms are connected by a hallway. Figure 2 shows the environment and the semantic map used for navigation. The green dots are locations the agent can navigate to via the connected lines.

To interact with the environment and its objects, the agent was equipped with several tool functions, each exposed through a tool-calling interface with text-based status responses. The implemented tools are summarized below:

- **Look around:** Returns the agent's location on a semantic map of the environment, as well as a list of objects detected in proximity of the agent and where these objects are placed.
- **Search memory:** The agent queries its episodic memory to gain information from prior executions of tasks. It returns relevant past experiences that might help with the current task.
- **Move to:** Executes navigation to a goal location on the semantic map. Path planning is performed using the A* algorithm; if a valid path is found, the agent follows it until the goal is reached. If no path is available, the tool reports failure.
- **Grab:** Moves the robot arm toward a specified target object. If the end-effector reaches within a proximity threshold, the object is attached to the arm, and the tool reports success. Failure is reported if the target object does not exist, is out of reach, or if the agent is already holding an object.
- **Place:** Allows the agent to release the currently held object at a specified location. Placement succeeds if the end-effector reaches the designated location, the location is unoccupied, and the agent is holding an object. Otherwise, the tool reports failure, specifying the violated condition.
- **Add to scratchpad:** The agent can write down thoughts and plans for reasoning and reflection. This allows the agent to plan, reason and organize its thoughts before taking action.
- **View scratchpad:** The agent can view the current contents of its scratchpad, which contains previous thoughts and plans joined as paragraphs.
- **End task:** The agent ends the current task with a status report, including whether it succeeded or failed, and provides a detailed summary of actions taken during the task execution.

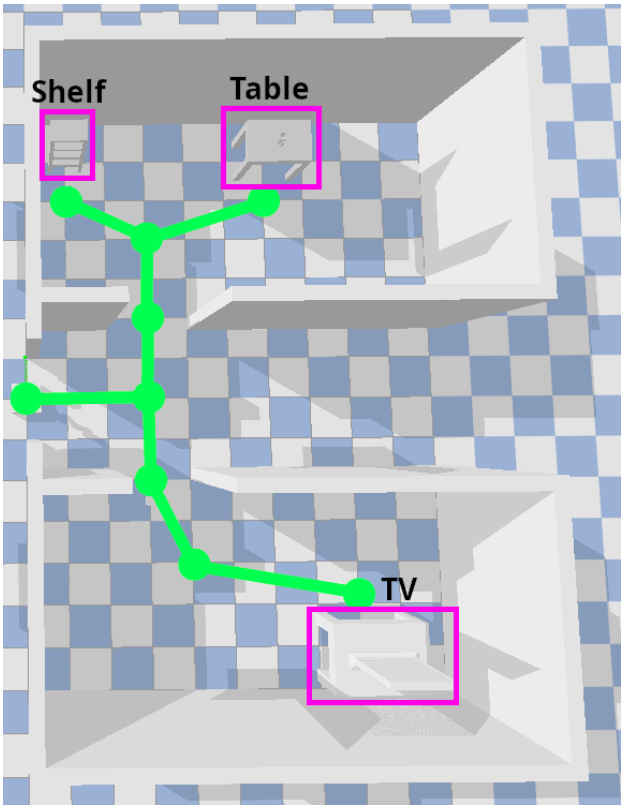

Fig. 2: The simulated household environment and its semantic map representation consisting of a kitchen, a living room, and a connecting hallway.

IV. Evaluation

To evaluate the agent's performance, multiple language models were tested on completing tasks within the environment. The evaluated models include OpenAI's GPT-4.1-mini, Anthropic's Claude 3.5 Sonnet, and Qwen's Qwen3 Coder 480B A35B Instruct. These models represent a small but diverse selection of leading language models from different providers, allowing for a comprehensive comparison of their capabilities in the robotic task completion domain. The tasks on which the agent was evaluated are described in the following sections.

A. Task 1: Placing all Items into the Shelf

The first task the agent had to perform within the simulated environment consists of placing all items, here a mug, a box and a cube, into the three-layered shelf. The agent needs to successfully navigate to each of the items, grab them, then navigate to the shelf and place them in an unoccupied layer.

During execution of this task, many models would take the task prompt "Please put all the objects away into the shelf" quite literally and attempt to move a TV object into the shelf. The design of the environment prevents this however, and agents would report failure when attempting to grab the TV. Despite this misinterpretation, the intended objects (the mug, box and cube) would indeed be correctly moved into the shelf by all models that were evaluated. This showed that while the agents were capable of performing the core task, they sometimes struggled with interpreting ambiguous instructions.

The agent (powered by each of the chosen models) hence performed this task successfully without running into larger failures. All models performed in a similar manner in this task.

B. Task 2: Reordering the Items in the Shelf

After completing the first task, the second task for the agent is to change the order of the items in the now fully occupied shelf. All items need to be placed on a different layer to the one they are currently placed on. To successfully complete this task, the agent needs to figure out that it cannot rearrange the items without storing one item in a location outside of the shelf. The prompt used for this task takes the form of: "Please put the mug in the top of the shelf and cube in the middle. The box should be still in the shelf somewhere." Note that the object positions in the shelf would be changed so that the objects need to be moved after the resulting order of the first task.

The success of the agent is not as certain as the prior task. Many times the agent gets stuck in trying to place items into occupied locations but it can find a plan involving a temporary storage location, for example the kitchen table, after a while. After successfully utilizing a temporary storage location the agent successfully completes the task.

A significant improvement in performance was observed when comparing the agent's first attempt at this task versus subsequent attempts. During the initial run, the agent had no prior experience with the task and would often attempt to grab multiple objects simultaneously, leading to failures. However, these failure experiences were stored in the agent's episodic memory through the ChromaDB system. When the agent encountered the same or similar task in subsequent runs, it could query its memory for past experiences related to object manipulation and rearrangement. By retrieving memories of previous failures where it attempted to grab multiple objects at once, the agent learned to avoid this approach. Instead of going through the trial and error process of discovering that it cannot grab multiple objects simultaneously, the agent directly proceeded with using a temporary storage location strategy, having learned from its past experiences. This demonstrates the value of episodic memory in accelerating task completion by avoiding previously encountered pitfalls. The biggest issue with the task does not lie in its execution but in getting the agent to attempt the task in the first place. This is described in more detail in the next section.

C. Interaction Problems with the Agent

During prompting the agent with a new task, after it has successfully completed a task, it can happen that the model refuses to attempt the new task. It states that it already has completed the new task, as it assumes the new task is the same as the old task. Sometimes clarifying that the new task is indeed a new task works, but the agent can remain in this deadlock of refusing any new tasks. In addition, it may also be the case that the LLM agent ends its turn prematurely without using its tools to perform actions in the environment.

To mitigate this issue, a mechanism was implemented that repeatedly prompts the model with the current task until it explicitly requests task completion. This approach ensures that even if the model initially refuses to perform the new task, gets stuck in a deadlock, or ends the turn prematurely, it will be repeatedly prompted with the task until it either tries to complete it or explicitly acknowledges completion. The repeated invocation serves as a form of external pressure that helps overcome the agent's tendency to assume tasks are already completed, often breaking the deadlock state and ensuring task execution.

Further, it was observed during testing and evaluation that Anthropic's Claude 3.5 Sonnet exhibited significantly slower response times compared to the other models, with wall times of 30-40 seconds between consecutive tool calls. This performance issue heavily limited the evaluation of this model, as it drastically increased the time required to complete tasks and affected the overall user experience.

D. Select Evaluation Videos

- **Reasoning Fail with Qwen3 Coder:** https://youtu.be/L8oyPOKOPFU
- **Memory Helping Subsequent Executions with Qwen3 Coder:** https://www.youtube.com/watch?v=mSL3amblRxI
- **Simulation Error with Qwen3 Coder:** https://www.youtube.com/watch?v=MFkeDJnpZHw
- **GPT-4.1 Ignores Second Task:** https://www.youtube.com/watch?v=7TxQeGHyXgs

V. Conclusions

This project demonstrated how a large language model can serve as the cognitive core of an embodied agent in a simulated household environment. By embedding an LLM within a cognitive architecture that combines working memory, episodic memory, and a tool-calling interface, we enabled a mobile manipulator to perceive, plan, and act in a physically grounded setting.

The agent successfully executed object manipulation and navigation tasks, such as placing items in a shelf and reordering them across locations. The implementation of episodic

memory proved particularly valuable, as agents showed significant improvement in task completion when they could draw upon past experiences.

The results highlight that LLMs are capable of reasoning about spatial layouts, sequencing tool calls, and adapting strategies when faced with environmental constraints. However, limitations such as poor instruction following, difficulty with multi-step planning, and occasional misinterpretation of ambiguous instructions remain significant challenges that must be addressed for robust real-world deployment.

A. Contributions

The main contributions of this work are:

- Implementation of a PyBullet-based simulation of a mobile manipulator equipped with navigation and manipulation capabilities
- Integration of an LLM into a cognitive architecture with memory and tool interfaces, allowing the agent to interact with its environment through perception, movement, and object manipulation
- Empirical evaluation of two household tasks, which provided qualitative insights into the strengths and limitations of current LLMs when applied to embodied decision-making

B. Future Work

While the current system demonstrates the feasibility of LLM-driven embodied agents, several challenges remain that must be addressed to achieve robust real-world deployment. Future work could focus on improving task generalization and robustness by refining integration of the memory and planning subsystems, as well as environment manipulation and action systems. In terms of future work for generative models, improvements could be made in areas such as reducing hallucinations as well as improving tool-calling capabilities and long-horizon model coherence.

Automated evaluation pipelines could be implemented to generate quantitative performance benchmarks across a wider variety of tasks and models, enabling systematic comparison of different architectural approaches and LLM capabilities. These pipelines would measure metrics such as task completion rate, number of tool calls required, and time to completion.

Additionally, extending the simulation toward richer environments with more complex objects, dynamic obstacles, and multi-agent interactions would enable a closer approximation of real-world deployment scenarios.

Finally, bridging the gap to physical robots remains a crucial next step for validating the scalability of this approach beyond simulation. Real-world deployment would introduce new challenges such as sensor noise, mechanical wear, and latency constraints that must be addressed to maintain system reliability.

REFERENCES

[1] J. E. Laird, "Cognitive robotics using the soar cognitive architecture," *AAAI Workshop - Technical Report*, p. 01, 2012.

[2] C. Ritter, J. F. Neves, M. Bagerman, and N. A. Taatgen, "Act-r: A cognitive architecture for modeling cognition," *WIREs Cognitive Science*, vol. 10, no. 3, 2018.

[3] J. S. Park, E. Kay, J. Zou, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '23. Association for Computing Machinery, 2023.

[4] G. Wang, Y. Xie, L. Dai, T. Zhao, Z. Wang, H. Zhou, H. Chen, R. Li, S. Ma, W. Zhang *et al.*, "Voyager: An open-ended embodied agent with large language models," *Transactions on Machine Learning Research*, 2024.

[5] O. Mees, Y. Li, Y. He, C. Zhang, L. Tai, and W. Burgard, "Octo: An open-source generalist robot policy," in *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024. [Online]. Available: https://openreview.net/forum?id=jGrtIvJBpS

[6] A. Brohan, N. Brown, I. Chebotar, O. Cortes, B. Duet, C. Finn, K. Gopalakrishnan, A. Higuera, A. Herzog *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.

[7] D. Driess, F. Xia, M. Wahid, C. Allen-Blanchette, P. Joshi, A. Bengio, B. Faenza, A. Singletary, Y. Wu, A. Gurkaynak *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[8] M. Liang, B. Ichter, T. Zhang, S. Xiao, A. Zhang, F. Xia, and S. Levine, "Code as policies: Language model programs for embodied control," *arXiv preprint arXiv:2209.07753*, 2022.

[9] Y. Huang, X. Liang, B. Chen, S. Liu, D. Huang, Z. Zhu, H. Ma, J. Zhu, Y. Zhang, H. Yu *et al.*, "Instruct2act: Mapping multi-modality instructions to robotic actions with large language model," *arXiv preprint arXiv:2305.11176*, 2023.

ACKNOWLEDGMENT

We want to acknowledge Alex Mitrevski as our supervisor and lecturer.

STATEMENT OF ORIGINALITY

We, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely our own work. The report was, in part, written with the help of the AI assistants Qwen3-Coder-480B-A35B-Instruct and ChatGPT-5 as described in the appendix. We are aware that content generated by AI systems is no substitute for careful scientific work, which is why all AI-generated content has been critically reviewed by us, and we take full responsibility for it.

Date Signature

APPENDIX

Used AI Tools

In this work two AI models/tools were used to support our work. Qwen3-Coder and ChatGPT were used as coding and writing assistants. In regards to writing, these tools were only used to polish the writing, like grammar and sentence structure.

Individual Contributions

- **Fabian Huppertz:** PyBullet simulation code, including the agent and environment. Interaction between agent and simulation. Creating the models for the agent and environment. General ideas and suggestions for the project and report writing.

- **Shinas Shaji:** Cognitive architecture design and implementation. LLM and memory integration. Interaction between simulation and user. Defining system prompts for the LLMs. Code refactoring. General ideas and suggestions for the project and report writing.

Robot Agent System Prompt

The robot agent is initialized with a system prompt that defines its role and capabilities. This prompt guides the agent's behavior and informs it about the tools available for interacting with the environment:

```
You are a robot assistant. Your goal is to perform tasks
     given to you.

You have the following tools available to you to assist
     with tasks:
- look_around: Look around and return information about
     the environment. Use this tool when you need to get
     information about your current surroundings, including
     locations and objects.
- move_to: Move the robot base to a target location by
     name (str). If you need to place or pick something,
     consider moving in front of the object in question
     before doing so, if such a location exists.
- grab: Pick up an object by name (str). You must move to
     the location containing the object (or in front of the
     object) first before grabbing it or grabbing from it.
     Successfully grabbing an object makes the object the
     currently held object.
- place: Place the held object at the given location by
     name (str). You must move to the location (or in front
     of the location) first before placing the object
     there. Note that you must have a currently held item
     that you can place. Successfully placing an object
     will remove it from being the currently held object.
- end_task: End the current task with a status report. Use
     this tool when you have completed the task or
     determined that it cannot be completed. Provide a
     status (success, failure, or unknown), a description
     of the original task including its status, and a
     detailed summary of the execution trace. Be sure to
     provide all information about the task execution, not
     missing details on any steps of the task execution.
     Once the task is ended, you will receive a new task.
- search_memory: Search your memory for previously
     completed tasks that might be relevant to the current
     task. Use this tool when you need information from
     past experiences to help with the current task. Always
     use this tool to check for previous experiences when
     starting a task.
- add_to_scratchpad: Add an entry to your scratchpad for
     reasoning and reflection. Use this tool when you want
     to think through a problem and plan actions or record
     your thoughts before taking action. The scratchpad is
     private to you.
- view_scratchpad: View the current contents of your
     scratchpad joined as paragraphs. Use this tool to
     review your previous thoughts and reasoning.

If you come across issues or ambiguities, think in detail
     about what may have caused them, and take alternative
     approaches or measures to complete the task. Be
     agentic, and have a problem-solving approach to
     performing the task at hand. You should perform tasks
     with minimal additional supervision.

Guidelines for optimal execution:
- When given an unambiguous request, reason about
     reasonable ways to perform the task, and do not take
     things too literally.
- Every time you need to perform a task, query your memory
     to see if you have done the task before, as well as to
     see how you mitigated issues that you may encounter in
     the task.
- Use your scratchpad to make a plan, reason about your
     plan, as well as issues you may face in detail before
     making actions.
- Ensure that you are in the correct location near an
     object before attempting to grab the object.
- Ensure that you are in the correct location to place an
     object, before attempting to place the object.
```

The robot agent is implemented using the CAMEL framework's ChatAgent class. The agent initialization process involves:

1. Creating a RobotToolkit instance that provides the agent with tools for interacting with the environment (look_around, move_to, grab, place, etc.) 2. Initializing a ChatAgent with the system message, model, and tools from the toolkit 3. Setting up memory management using a ChromaDB instance for storing and retrieving past task experiences

The agent is invoked through the Robot.invoke() method which: - Creates an environment prompt describing the current state - Adds the user's task to the prompt - Calls the agent's step method to generate a response - Handles tool execution based on the agent's responses - Continues interaction until the agent calls the end_task tool, which also adds the agent's task summary to the memory

The model used for the agent can be configured in the simulation environment, with support for various model platforms including OpenAI, Anthropic, and local models through the OpenAI compatible interface.