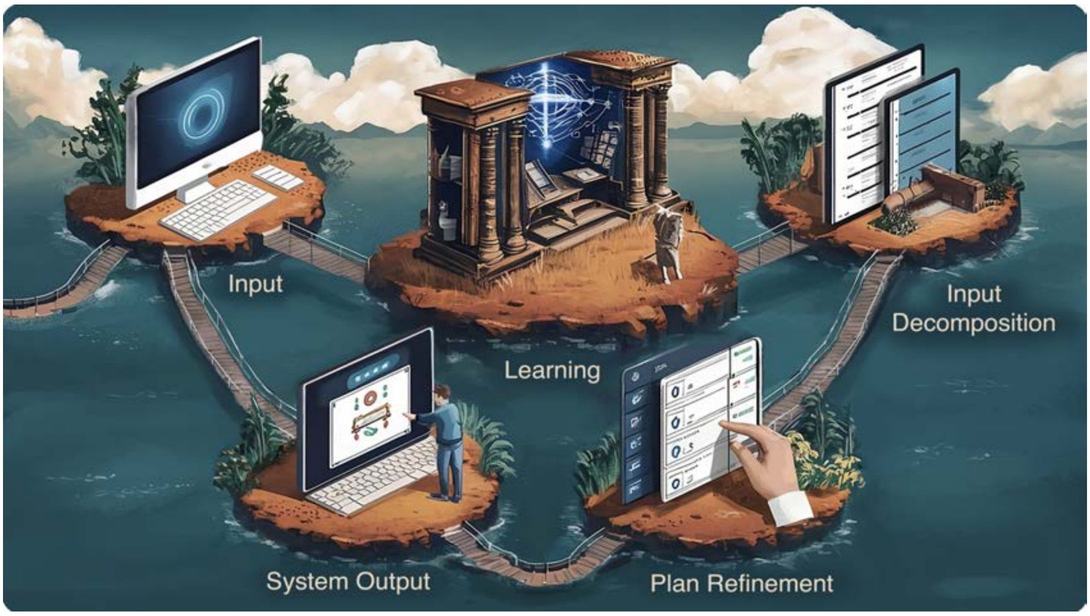
Zurich University of Applied Sciences

School of Engineering
Centre for Artificial Intelligence

BACHELOR'S THESIS

A2C2 - Natural Language-Instructed Autonomous Agent for Computer Control



Authors:
Nobel Gabriel
von Wartburg-Kottler Rebekka

Supervisors:
Prof. Dr. Thilo Stadelmann
Pascal Sager

Submitted on June 7, 2024

Study programme: Computer Science, BSc

Imprint

Project: Bachelor's Thesis
Title: A2C2 - Natural Language-Instructed Autonomous Agent for Computer Control
Author: Nobel Gabriel, von Wartburg-Kottler Rebekka
Date: June 7, 2024
Keywords: A2C2, Artificial Intelligence, Autonomous Agent, Computer Control, Multitask Reasoning, Natural Language Processing, Task Decomposition, Vision Language Model
Copyright: Zurich University of Applied Sciences

Study program:
Computer Science, BSc
Zurich University of Applied Sciences

Supervisor:
Prof. Dr. Thilo Stadelmann
Zurich University of Applied Sciences
Email: stdm@zhaw.ch
Web: stdm.github.io

Supervisor 2:
Pascal Sager
Zurich University of Applied Sciences
Email: sage@zhaw.ch
Web: sagerpascal.github.io

Abstract

Recent advances in artificial intelligence (AI) have boosted progress across various domains, particularly enabling breakthroughs in the discipline of Natural Language-Instructed Autonomous Agents for Computer Control (A2C2s). Due to their capabilities of understanding natural language and executing actions the same way a human would, these agents have the potential to significantly simplify human-machine interaction, reduce resource requirements in business, and empower non-technical users to operate computer systems effortlessly.
This thesis aims to provide an overview of the vast yet fragmented research landscape of A2C2s, enabling further innovation. Through a comprehensive literature review, existing agents and their capabilities were identified, summarized, categorized, and analyzed to extract potentials and challenges.
The result is a detailed taxonomy, likened to an archipelago, encompassing *providing information to the agent*, *refining skills and building knowledge*, *ensuring task comprehensibility*, *debating and refining subtasks* and *interacting with the environment*. Besides reviewing existing work, this thesis offers an analysis of pinnacle agents with foundational A2C2 skills and proposes a novel architecture for a comprehensive A2C2 that leverages the identified strengths. The findings suggest that an A2C2 must be able to decompose and structure user and system input and compare and reason plans in a closed loop. Consequently, state-of-the-art A2C2s utilize large foundation models because of their solid planning and image comprehension capabilities.
Promising progress in AI highlights strengths in general reasoning and image description. Key challenges include specializing these strengths for A2C2s, specifically reducing possible actions, decomposing instructions, and refining plans. Addressing these issues, along with considerations for security, performance, and personalization, is essential for future research.

Keywords: A2C2, Artificial Intelligence, Autonomous Agent, Computer Control, Multitask Reasoning, Natural Language Processing, Task Decomposition, Vision Language Model

Contents

List of Abbreviations

A2C2	Natural Language-Instructed **A**utonomos **A**gent for **C**omputer **C**ontrol *An agent or system that gets natural language as input and can autonomously execute actions on a computer system*
ART	**A**utomatic **R**easoning and **T**ool-use
AI	**A**rtificial **I**ntelligence
computer systems	*personal computers of non-technical users like company laptops*
BC	**B**ehaviour **C**loning
CoT	**C**hain-**o**f-**T**hought
DMR	**D**ense **M**arkup **R**anking
DOM	**D**ocument **O**bject **M**odel
dynamic action inference	*finding all possible actions that can be executed on a GUI observation*
DQN	**D**eep **Q**-**N**etwork
few-shot	*performing a task with only a few prior examples*
GoT	**G**raph **o**f **T**houghts
HTML	**H**yper**T**ext **M**arkup **L**anguage
GUI	**G**raphical **U**ser **I**nterface
IO peripherals	keyboard, mouse, or touchscreen
LAM	**L**arge **A**ction **M**odel
LLM	**L**arge **L**anguage **M**odel
LMM	**L**arge **M**ultimodal **M**odel
LoRA	**Lo**w-**R**ank **A**daptation
MAD	**M**ulti **A**gent **D**ebate
OCR	**O**ptical **C**haracter **R**ecognition
PDDL	**P**lanning **D**omain **D**efinition **L**anguage
RAG	**R**etrieval **A**ugmented **G**eneration
RL	**R**einforcement **L**earning
RLHF	**R**einforcement **L**earning with **H**uman **F**eedback
SQL	**S**tructured **Q**uery **L**anguage
SoM	**S**et-**o**f-**M**ark
SOTA	**S**tate-**O**f-**T**he-**A**rt
TDAG	**T**ask **D**ecomposition and **A**gent **G**eneration
ToT	**T**ree **o**f **T**houghts
VH	**V**iew **H**ierarchy *The equivalent to HTML for android devices*
VLM	**V**ision **L**anguage **M**odel
zero-shot	*performing a task without prior examples*

Dedicated to our partners and children, who have strengthened our backs over the past four years.

Chapter 1

Introduction

Throughout evolution, humans have used tools to survive, and improve their problem-solving skills (Gregersen, 2019). Over time, these tools have become more sophisticated, enabling them to overcome challenges thought impossible (Garza-Herrera, 2024). Artificial intelligence (AI) has been established as a new tool for humans since the 1950s, due to the presentation of the Turing-Test (Turing, 1950) for the development of intelligent machines, and the 1956 Dartmouth Symposium, at which AI was officially introduced. The advancements in AI, from the expert systems of the 1980s and the neural networks of the 1990s to the deep neural networks of recent times, have greatly simplified many human tasks (Haenlein & Kaplan, 2019).

Recently advances in foundation models (see section 2.4) for different data types – *image, text and audio* – have brought AI to the forefront, with the potential to transform many facets of human life in the future (De Angelis et al., 2023; Larsen & Narayan, 2023). These technologies facilitate applications that make everyday life easier (Daley, 2024) such as *movie recommendations* and *voice assistants* but also complex systems with critical applications such as *autonomous driving* and *diagnostics in medicine*.

One of the latest developments is using AI to automate tasks on a personal computer, particularly for non-technical users (henceforth referred to as computer system). Therefore a Natural Language-Instructed Autonomous Agent for Computer Control (henceforth referred to as A2C2) is of fundamental importance. This agent, which operates through keyboard, mouse, or touchscreen interfaces, mimics human interaction on graphical user interfaces (GUIs) and receives instructions in natural language. The potential of this technology lies in simplifying the interaction between humans and machines, reducing resource requirements in various business sectors, and ultimately empowering non-technical users to master the operation of computer systems effortlessly. However, autonomous interaction with computer systems poses complex challenges. While human reasoning offers the advantage of generalizability by gaining independence from domain restrictions, its complexity also presents unsolved hurdles. As shown in previous research (see chapter 4), it is all the more difficult if the agent has to imitate human interaction capabilities, specifically the IO peripherals as action space and the GUI as observation space.

The field of A2C2s is a highly discussed and important topic in AI research, which has already resulted in significant progress (Shaw et al., 2023; Tan et al., 2024; team rabbit research, 2023; L. Zheng et al., 2023). The number of scientific publications and software projects with various approaches is increasing weekly.

Each approach has its strengths, limitations, and areas of application. However, this rapid growth leads to a fragmented landscape in the field of A2C2s.

Problem Statement

The primary **objective** of this thesis is to provide a comprehensive survey to organize the fragmented research landscape in a simple and meaningful way. The resulting survey is intended to be used as a reference point for further research.

In particular, the following questions will be addressed:

- What does the research landscape **look** like, and how can it be **categorized** to gain a clear and valuable overview?
- What are the **strengths** and **potentials** of existing systems, and how can they be **exploited**?
- Which **challenges** are known to be **unsolved** and could lead to breakthroughs in the field if solved?
- What does the look like **proposed system**, based on well-founded knowledge in this field and ideally representing a further step towards A2C2?

The **scope** of this proposal is limited to agents that receive a natural language instruction and a pixel-based representation of the GUI as input and can operate both business and commercial software with a keyboard, mouse, and touchscreen. For the survey part of the thesis, the scope of the input as well as the possible operations is extended to represent the whole research landscape.

Chapter 2

Foundation

This chapter is designed to help understand the basic concepts underlying an A2C2. Furthermore, it provides a real-world example that demonstrates how the agent could simplify the interaction between a user and a computer system.

2.1 Previously on A2C2

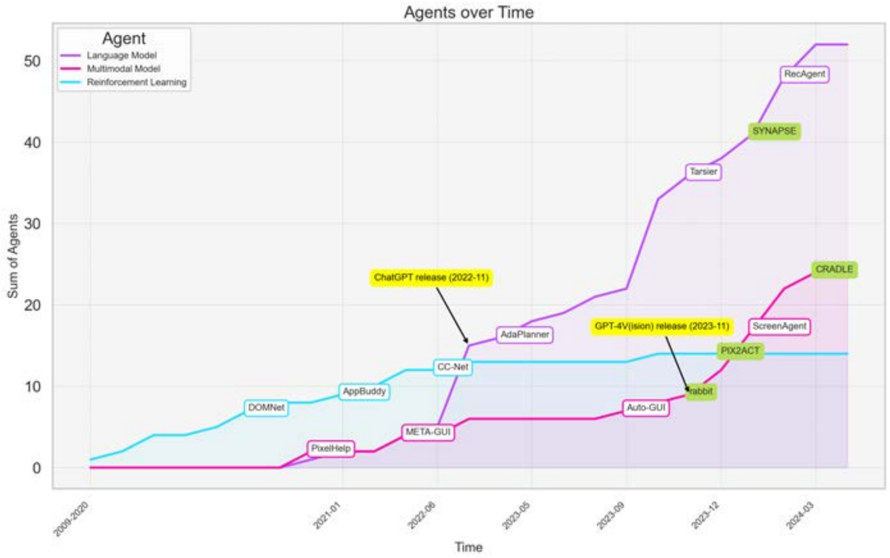


FIGURE 2.1: An illustration of the summed agents for computer control over time, distincted between three approaches *reinforcement learning*, *language model* and *multimodal model*. As reference the publication dates of ChatGPT (openAI, 2022) and GPT-4V(ision) (openAI, 2023) are visually highlighted in yellow and show a correlation to the increase of agents. In addition, the pinnacle methods (Shaw et al., 2023; Tan et al., 2024; team rabbit research, 2023; L. Zheng et al., 2023) presented in chapter 5 highlighted in green, as well as other agents are distributed over the time axis (Humphreys et al., 2022; Jia et al., 2019; Y. Li et al., 2020; Niu et al., 2024; Shvo et al., 2021; H. Sun et al., 2023; L. Sun et al., 2022; L. Wang et al., 2024).

As Figure 2.1 shows, the development of an agent for computer control and its impact by new milestones like ChatGPT (openAI, 2022) and GPT-4V(ision) (openAI, 2023). It can be divided into the following three distinct phases:

Before 2020: Reinforcement learning (RL) (see section 2.2) was widely applied driven by advancements in deep Q-networks (DQN) and proximal policy optimization, as well as its effectiveness in learning from interaction with an environment.

From 2020 until 2023: There was a shift from pure RL to large language models (LLM), as natural language processing empowered by transformers (Vaswani et al., 2017), made great progress and RL showed significant limitations in planning and reasoning.

From 2023 until present: Increasingly versatile foundation models such as vision language models (VLM) have been utilized (see section 2.4). These models possess built-in capabilities to handle various modalities such as text, images, and more, demonstrating great potential in the reasoning and planning of computer system tasks.

2.2 Reinforcement Learning

RL is a versatile area of machine learning that focuses on developing optimal policies for sequential decision problems by optimizing a cumulative future reward. The agent learns to maximize the expected return through its interactions with the environment by identifying the best actions for a given state (Hasselt et al., 2015; Mnih et al., 2013; Sutton et al., 1999).

The basic RL algorithm is modeled as a Markov Decision Process which is a 4-tuple of (S, A, P, R) (Bellman, 1957; van Otterlo & Wiering, 2012) where:

- S is the state space
- A is the action space (or A_s set of actions available from state s)
- $p(s_t + 1|s_t, a_t)$ is the probability that action a for state s in time t leads to s' in time $t + 1$
- $R_a(s, s')$ is the immediate reward after transitioning $s \rightarrow s'$ with action a

The policy – *behavior at a given state* s_t – is typically defined as state-to-action mapping $\pi : S \rightarrow A$ and the objective of the agent is to maximize the cumulative rewards received over time $\mathbb{E}_{s_t}[R_t|s_t]$. The Q-function – *the action-value function* – is defined as $Q_\pi(s_t, a_t) = \mathbb{E}[R_t|s_t]$ and represents the expected cumulative reward an agent can get, taking action a_t in the state s_t following a specific policy and therefore estimates the effectiveness of different actions in a state (Chadi & Mousannif, 2023; Hasselt et al., 2015).

Algorithm 1 shows the Q-learning algorithm (Watkins & Dayan, 1992), one of the basic, classic, and most popular algorithms in the field of RL (Chadi & Mousannif, 2023; Hasselt et al., 2015).

Algorithm 1 Q-Learning Algorithm

```
Initialize:
    $Q(s,a) \leftarrow$ random or 0 values
    Learning rate $\alpha$, Discount factor $\gamma$, Exploration rate $\epsilon \in [0,1]$
for each episode do
    Initialize state $s$
    for each step in episode do
        Choose action $a$ using $\epsilon$-greedy policy
        Take action $a$
        Observe reward $r$ and new state $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
        $s \leftarrow s'$
    end for
end for
```

2.3 Reinforcement Learning with Human Feedback

Reinforcement learning with human feedback (RLHF) is an RL approach integrating human feedback into the learning process to guide the behavior of an agent (Christiano et al., 2017; Ouyang et al., 2022). This presents a significant advantage in environments with rewards that are difficult to define or sparse. During training an agent interacts iteratively with the environment, receiving feedback from humans – *binary, ranking* or *natural language* – on its actions (Christiano et al., 2017; Suay & Chernova, 2011; Warnell et al., 2017), which replaces the reward signal of the environment. The collected feedback is then utilized to update the policy or value function, aiming to maximize human-goal alignment. Human feedback also helps in addressing the exploration-exploitation dilemma (Houthooft et al., 2016; Pathak et al., 2017) by regulating the training direction to human alignment.

However, ensuring the generalization capabilities and robustness of an agent remains a challenge in both basic RL and RLHF. This challenge can be mitigated by having a thoroughly designed feedback collection process, a diverse data set as well as a heterogenous human feedback pool.

2.4 Foundation Models

Foundation models like LLM or VLM are large-scale general-purpose models, designed to generate text, image, audio, and other data types. They accomplish this by training on massive datasets, aiming to demonstrate robust capabilities for unseen tasks (henceforth referred to as zero-shot). (Dosovitskiy et al., 2020; Jiang et al., 2023; J. Li et al., 2023; Lu et al., 2023; G. Team et al., 2023; Touvron et al., 2023; Z. Yang et al., 2023). Their strengths lie in the ability to generalize across domains and data types, the flexibility to adapt to new tasks with a minimal parameter update, and the capability to produce high quality outputs (Schneider et al., 2024). Typically foundation models are pre-trained in an unsupervised manner and can then be fine-tuned by unfreezing parts of their network weights and retraining on domain-specific data (see section 4.2).

2.5 Example of a Working A2C2

In this section, we will run through a real-life example to demonstrate how an A2C2 ideally functions, showcasing the complex tasks such an agent should be able to perform in a computer system to minimize effort for the user. To do this, we have modeled the agent on the functionality of various state-of-the-art (SOTA) agents.

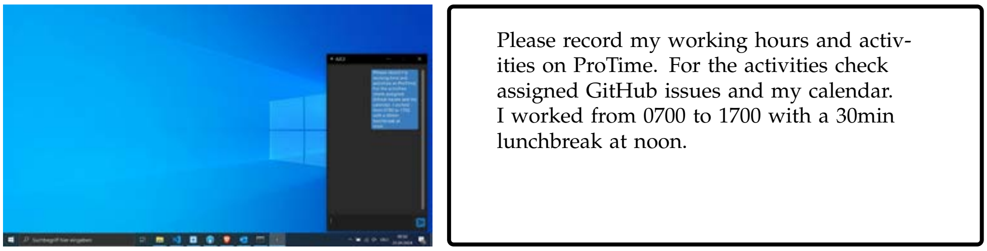
2.5.1 Specification

- **Hila**: A software developer in a large company actively involved in the development and maintenance of multiple applications.
- **ProTime**: Working hours and activity recording tool, based on SAP (All41Group, 2024).
- **Company Guidelines**: All employees must track their working hours and activities daily in ProTime. This includes recording the start and end time of the working day and detailing the projects/issues processed. At the end of each month, a signature of the user is required, to confirm that all entries have been made correctly.
- **A2C2**: A new tool, provided by the company, designed to execute tasks on computer systems via a chat interface.

2.5.2 Task

At the end of a typical working day, Hila receives a message reminding her to report her work hours and activities in ProTime. However, with other urgent tasks to complete and the desire to leave work on time, she is not sure if she will be able to find the time to complete the reporting. Consequently, she relies on the new A2C2 provided by the company and delegates the recording in ProTime to it.

2.5.3 Input



Please record my working hours and activities on ProTime. For the activities check assigned GitHub issues and my calendar.
I worked from 0700 to 1700 with a 30min lunchbreak at noon.

(A) computer system observation

(B) user instruction

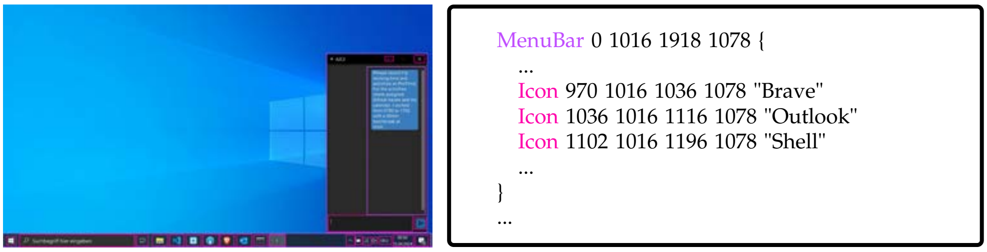
FIGURE 2.2: A computer system observation of a Windows desktop with a A2C2 chat interface open in Figure 2.2 (A) and the corresponding user instruction text in Figure 2.2 (B). Both inputs are necessary for an agent to assess the current state.

The A2C2 receives two inputs that are correlated (see Figure 2.2). The first input Figure 2.2 (A) is a pixel-based representation of the GUI in the form of a screenshot (see subsection 4.1.2). The second input Figure 2.2 (B) is the raw user instruction in natural language (see subsection 4.1.1).

2.5.4 Input Decomposition

As a proficient user of computer systems, Hila can intuitively tell that several applications are involved, namely Calendar, GitHub, and ProTime, and that she currently observes the desktop. With this knowledge, she can formulate a plan that solves the instruction.

Acquiring this level of insight is challenging for an agent. To attain similar insights, the agent needs to decompose the user instruction into manageable subtasks (see section 4.3) and interpret the screenshot of the GUI (see subsection 4.3.1).



```
MenuBar 0 1016 1918 1078 {
   ...
   Icon 970 1016 1036 1078 "Brave"
   Icon 1036 1016 1116 1078 "Outlook"
   Icon 1102 1016 1196 1078 "Shell"
   ...
}
...
```

(A) observation with segmentation

(B) observation in pseudo DOM

FIGURE 2.3: The deconstruction of the observation GUI in two different forms. Figure 2.3 (A) is decomposing the components of a screen with bounding boxes, and Figure 2.3 (B) is translating this decomposition to a pseudo-document object model (DOM) providing a structured representation of the GUI.

In Figure 2.3 a possible deconstruction of the GUI is shown. Atomic components, which cannot be further subdivided, are denoted by pink bounding boxes in Figure 2.3 (A). Elements enclosed by purple bounding boxes, such as – *the menu bar* and *the A2C2 chat interface* – contain multiple atomic components. In addition, the element type and location are identified, and it is determined whether the element can be described textually in Figure 2.3 (B).

1. Open Browser
2. Navigate to github.com
3. Login into GitHub
4. Click on issues
5. Check open issues
6. Check recently closed issues

(A) GitHub subtasks

7. Open Outlook
8. Check today's entries
9. Click on the calendar

(B) calendar subtasks

10. Open ProTime
11. Enter activities
12. Enter time 0700 - 1200
13. Enter time 1230 - 1700
14. Check if signature required

(C) ProTime subtasks

FIGURE 2.4: A possible decomposition of subtasks in three different categories. The blue marked text represents opening web apps which can be a well-known task.

The user instruction also requires to be broken down into less complex instructions. Figure 2.4 shows a possible split into three larger tasks, which could potentially be carried out independently – *check GitHub in Figure 2.4 (A)*, *check calendar in Figure 2.4 (B)* and *make ProTime bookings in Figure 2.4 (C)*. These tasks can further be divided into subtasks. At this stage the agent could consolidate an external memory to increase efficiency and reduce variability. Since opening an application is a repetitive task and likely to be successfully executed before, the subtasks marked blue in Figure 2.4 (A) can be loaded from memory, ensuring a higher success rate. The GUI structure and context can also help in the decomposition of user instructions by providing more insights into the current observation space.

Decomposing an input is not a one-off task, rather it must be repeated after the execution of each subtask. At this point, the subtasks are presented as a list of suggestions lacking order and validation.

2.5.5 Selection

1. Open Browser
2. Navigate to github.com
3. Login into GitHub
4. Click on issues
5. Check open issues
6. Check recently closed issues
7. Open Outlook
8. Check today's entries
9. Click on calendar
10. Open ProTime
11. Enter activities
12. Enter time 0700 - 1200
13. Enter time 1230 - 1700
14. Check if signature required

(A) First plan generated by task decomposition

1. Open Browser
2. Search GitHub in google
3. Click on the first link
4. Click on your profile
5. Click on daily contributions
6. Check today's contributions
7. Open Outlook
8. Check today's entries
9. Click on calendar
10. Open ProTime
11. Enter activities
12. Enter time 0700 - 1200
13. Enter time 1230 - 1700
14. Check if signature required

(B) Second plan generated by task decomposition

FIGURE 2.5: Two plans generated by task decomposition with different subtasks. The first plan is shown in Figure 2.5 (A) and the seconde plan is shown in Figure 2.5 (B). Some subtasks are overlapping and the colored ones highlight different categories of selection.

With the insights from task decomposition, the agent has formulated a plan what subtasks need to be executed and what options the GUI offers. Next, the agent must assess the quality of its plan. This is done by potentially *reasoning* (see subsection 4.4.1) the subtask suggestions *comparing* (see subsection 4.4.2) multiple plans and *validating* (see subsection 4.4.3) whether the information provided in the instruction and observation is sufficient to generate a suitable output.

Multiple plans can be formulated and proposed from the task deconstruction (see Figure 2.5). These plans may differ in one to many subtasks. The A2C2 should be capable of deciding which plan is more favourable. To accomplish this, it compares both plans with the user instruction in Figure 2.2 (B) determining that plan (A) in Figure 2.5 (A) matches better with subtasks 1 to 6 than plan (B) in Figure 2.5 (B). Consequently, plan (A) in Figure 2.5 (A) is selected for execution.

During the reasoning about subtasks, the agent has to ensure that the sequence and structure are logical. It notices that subtasks 8 and 9 are reversed, and that subtask

11 can only be performed after subtask 13, as entering activities without work hours makes no sense. The agent identifies and corrects this issue either before executing the first subtask or in the event of an error during the execution of subtask 11.

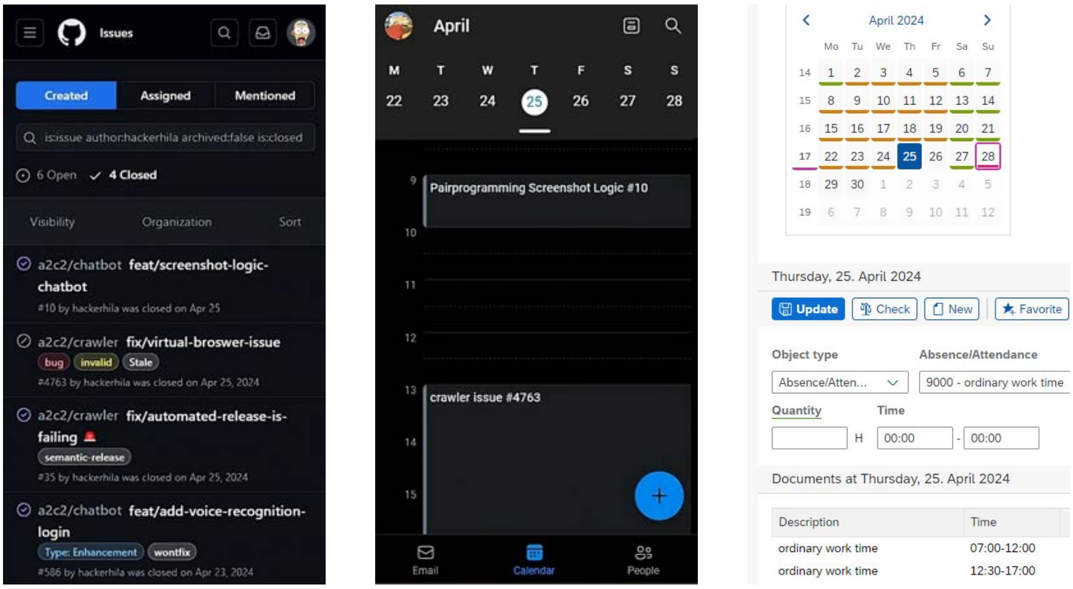
For validation, the agent checks whether all necessary information is available and whether tasks are deemed so critical to require user confirmation. In subtask 3 user credentials are required, which are not provided by the initial instruction. Additionally, subtask 3 and subtask 14 should be marked as critical tasks, necessitating user confirmation before execution.

2.5.6 Output

1. CLICK "browser icon"
2. CLICK "address bar"
3. CLICK "sign in"
4. TYPE "username" "hackerhila"
5. TYPE "password" "S@fePa$$w0rd"
6. CLICK "issues"
7. CLICK "closed"
8. READ

8. CLICK "Outlook"
9. CLICK "calendar"
10. CLICK "today"
11. CLICK "daily view"
12. READ

13. TYPE "win"
14. TYPE "searchfield" "ProTime"
15. CLICK "ProTime"
16. TYPE "from" "0700"
17. TYPE "to" "1200"
18. TYPE "from" "1230"
19. TYPE "to" "1700"
20. ...



is:issue author:hackerhila archived:false is:closed

6 Open ✓ 4 Closed

a2c2/chatbot feat/screenshot-logic-chatbot
#10 by hackerhila was closed on Apr 25

a2c2/crawler fix/virtual-broswer-issue
bug invalid Stale
#4763 by hackerhila was closed on Apr 25, 2024

a2c2/crawler fix/automated-release-is-failing
semantic-release
#35 by hackerhila was closed on Apr 25, 2024

a2c2/chatbot feat/add-voice-recognition-login
Type: Enhancement wontfix
#586 by hackerhila was closed on Apr 23, 2024

Pairprogramming Screenshot Logic #10

crawler issue #4763

April 2024

	Mo	Tu	We	Th	Fr	Sa	Su
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30	1	2	3	4	5
19	6	7	8	9	10	11	12

Thursday, 25. April 2024

Object type: Absence/Atten...
Absence/Attendance: 9000 - ordinary work time
Quantity: H
Time: 00:00 - 00:00

Documents at Thursday, 25. April 2024

Description	Time
ordinary work time	07:00-12:00
ordinary work time	12:30-17:00

(A) check GitHub task with plan and GUI

(B) check calendar task with plan and GUI

(C) make ProTime bookings task with plan and GUI

FIGURE 2.6: The three executed subtasks – *check GitHub in Figure 2.6 (A), check calendar in Figure 2.6 (B), make ProTime bookings in Figure 2.6 (C)* – as well as the GUI representation after executing them.

Figure 2.6 depicts the status after the successful execution of the three subtasks – *check GitHub in Figure 2.6 (A), check calendar in Figure 2.6 (B), make ProTime bookings in Figure 2.6 (C)* – up to *enter activities*, the subtask number 13 (see Figure 2.5) after reasoning. In this example, the agent translates the plan to IO peripheral actions – *CLICK, TYPE, READ* – and can directly execute them on the computer system.

With the context provided by GitHub and the calendar, the agent attempts to execute subtask number 13. After struggling to translate this subtask into IO peripheral actions or receiving an error as feedback, the agent should realize that this specific task encompasses multiple subtasks that need to be decomposed. Additionally, it requires missing information from the user to be executed, such as – *What did Hila do between 10:00 and 12:00?* and *When did Hila work on task #35?*

Armed with new knowledge, the agent should restart the closed loop planning steps with a new starting point and more context.

Chapter 3

Methodology

The objective of this chapter is to detail the methodological approach undertaken to write this thesis and complete the tasks required to answer the questions defined in section 1.

Literature Search and Review: To illustrate the research landscape, we conduct a detailed and well-structured literature review. This analysis can be divided into two phases:

- **Initial Search**: The first step is to search for relevant publications using individual search terms *Action Transformer*, *Computer Control, Web Agent, Multimodal Agents*, *Action Grounding*, and *Task Reasoning* on scientific platforms namely *Google Scholar (Google, 2024)*, *Research Gate (ResearchGate, 2024)*, *IEEE Xplore (IEEE, 2024)* and *arXiv.org (arXiv, 2024)*.

- **Follow-up Search**: Once the first relevant publications are successfully identified, the section on related publications and the corresponding references are used to search for further works.

Summarization: To make the best use of the collected publications and to gain a deeper understanding of the field of an A2C2, we summarize them. This has to be done because the difficult task of categorizing first requires comprehensive knowledge of the research area. The summary template was provided by our supervisor Thilo Stadelmann:

1. Problem tackled/solved
2. Methods/approach used
3. Experimental setup and results
4. Pros when thinking about applying this approach to your problem/task
5. Cons when thinking about applying this approach to your problem/task
6. Useful for working on our problem

Time Management: We restrict the initial literature review to the first eight weeks enabling us to limit the analyzed publication to the most important ones. A publication released or discovered after the initial literature review is included after a thorough inspection of its relevance.

Taxonomy Development: The extensive literature review in combination with organized summaries will help us to group these approaches based on their similarities and extract a taxonomy for the outline of an A2C2. Due to the complexity of the topic, several iterations will be necessary to develop a taxonomy that provides a structured, clear, and meaningful overview of the research landscape (see chapter 4). For each of these components of an A2C2, key publications are identified with a specific focus on their potentials – *what makes this paper particularly special and how could it improve A2C2s?* – and challenges – *what are unsolved problems of this approach in regards to A2C2s?*

Selected Approaches and System Definition: The next step is to present promising agents identified in the literature review that potentially point the way to A2C2 (see chapter 5). Finally, based on this understanding and the insights gained, an architecture is presented that could represent a further step in the development of A2C2s (see section 5.3). This is done by combining the identified strengths of existing agents with novel ideas that come from acquiring knowledge about the research area.

Limitations and Justification: The methodology chosen offers a structured approach to reviewing the literature and answering the problem statement (see section 1). The qualitative nature of this thesis enables an in-depth understanding of existing A2C2s and highlights their potential and limitations. Due to the actuality of the topic, this thesis is limited to the literature available until the time of completing the thesis.

Chapter 4

Taxonomy

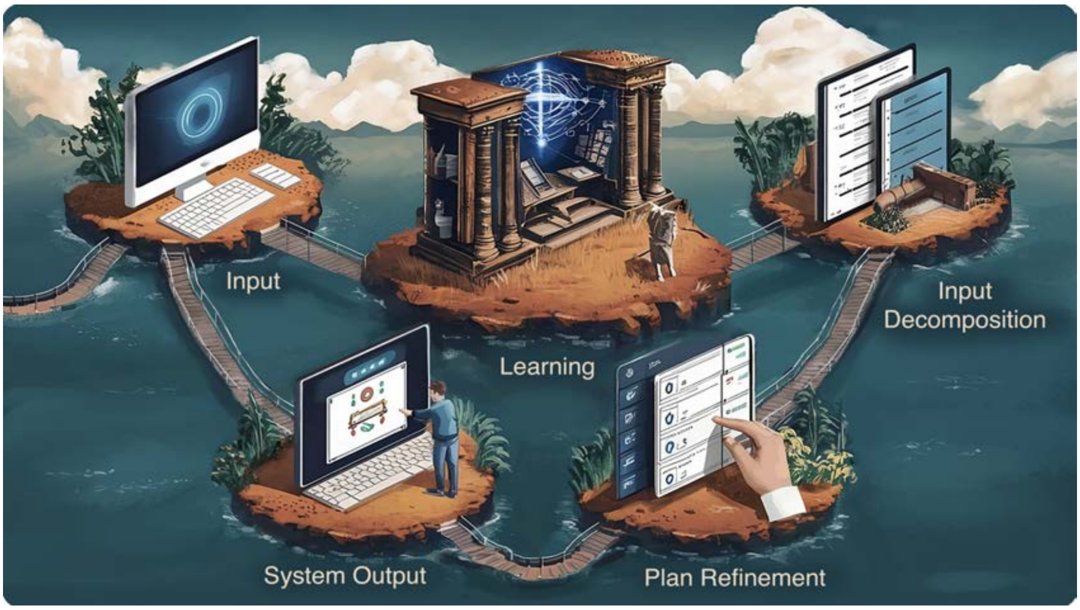


FIGURE 4.1: First, we look at the input, then learning takes its turn,
Next, we sail to input decomposition, where insights we discern.
We cross the seas to plan refinement, with choices firm and stout,
Until at last, the system's output is what it is about.
This figure is generated by Ideogram (2023).

As shown in Figure 4.1 the taxonomy of an A2C2 can be compared with an archipelago where every island has its own strengths and challenges. An agent must be able to navigate from one island to the next to reach the final destination. We therefore offer a comprehensive map of the specialized islands, which has emerged from our research, to make the dangerous passage in the waters of A2C2s to an island hopping vacation. In this chapter, we discuss the individual islands in detail and highlight insights identified during the research ending each section with a short discussion and a structured representation of the identified agents (see Figure 4.5, Figure 4.7, Figure 4.12, Figure 4.14, Figure 4.16). This is done in the same order introduced in section 2.5.

4.1 Input - *providing information to the agent*

Hila sees the desktop view of her computer system and knows what task she can execute.

The **A2C2** receives as input a pixel-based representation of the GUI in the form of a screenshot and the raw user instruction in natural language.

FIGURE 4.2: The input island with regard to what the A2C2 receives as input compared to what Hila sees. Icon of Hila and A2C2 is generated by Ideogram (2023).

The section below provides a description of the input island. The input corresponds to what the agent receives from the user and the computer system to solve a desired task. Figure 4.2 illustrates the comparison of what the agent receives as input and what the user can see. The Input island represents the starting point for an A2C2 in the journey through the archipelago and is divided into the **instruction space** – *what is the user instruction?* – and the **observation space** –*what does the agent perceive?*

4.1.1 Instruction Space

Example 1: Book my working hours for today from 7am to 6pm with 30min lunch at 12pm.
Example 2: Please book my working hours for today. I started work at seven o'clock am and started feeling hungry at around eleven thirty but kept working until noon. Then I went for lunch at my favorite burger joint for thirty minutes. After a delicious meal, I had the strength to continue working until six o'clock in the evening.

(A) User instruction wording

Example 1: Book my working hours for today from 7 to 6 with 30min lunch at 12.
Example 2: Book my working hours. Day: April 20. 2024, Start: 07:00, Lunch:12:00, Resumption: 12:30, End:18:00.

(B) User instruction precision

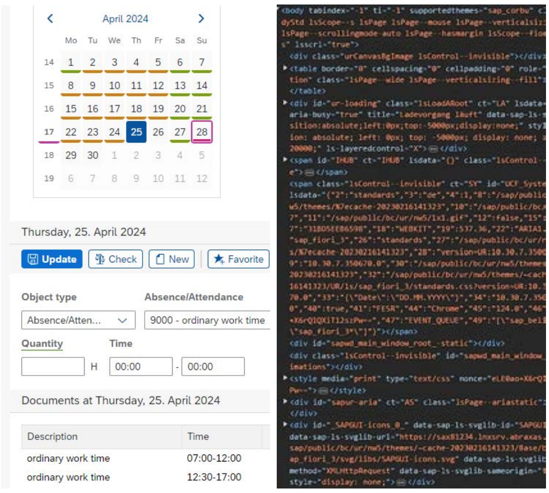
Example 1: Book my working hours. Start: 07:00, Lunch: 12:00, Resumption: 12:30, End:18:00.
Example 2: Book my working hours on the Pro Time app. Day: April 20. 2024, Start: 07:00, Lunch: 12:00, Resumption: 12:30, End: 18:00. I worked from 07:00 - 12:00 on the Issue #714 and from 12:30 on the Issue #182223.

(C) User instruction completeness

FIGURE 4.3: The different user instruction examples for wording in Figure 4.3 (A), precision in Figure 4.3 (B) and completeness in Figure 4.3 (C).

As discussed in the introduction (see chapter 1), the user input is scoped to natural language. Despite this constraint, instructions can vary significantly in wording, precision, and completeness (see Figure 4.3). Depending on how a user instruction is presented to the agent, a task may be more challenging to solve. The more difficult the user instruction is, the more iterations over the agent are typically required for successful execution. This difficulty is increased by *incomplete instructions*, *irrelevant information*, *unstructured text*, and various other factors. The planning process encompasses all the steps outlined in the subsequent taxonomy that can be sequentially executed to obtain an output.

4.1.2 Observation Space



Thursday, 25. April 2024

Object type: Absence/Atten...

Absence/Attendance: 9000 - ordinary work time

Quantity: H

Time: 00:00 - 00:00

Documents at Thursday, 25. April 2024

Description	Time
ordinary work time	07:00-12:00
ordinary work time	12:30-17:00

```
<body tabindex="-1" ti="-1" supportedthemes="sap_corbu" c
dyStd lsScope--s lsPage lsPage--mouse lsPage--verticalsiz
lsPage--scrollingmode-auto lsPage--hasmargin lsScope--fio
s" lsscrl="true">
  <div class="urCanvasBgImage lsControl--invisible"></div>
 ▸<table border="0" cellspacing="0" cellpadding="0" role="
  tion" class="lsPage--wide lsPage--verticalsizing--fill"
  </table>
 ▸<div id="ur-loading" class="lsLoadARoot" ct="LA" lsdata=
  aria-busy="true" title="Ladevorgang läuft" data-sap-ls-
  sition:absolute;left:0px;top:-5000px;display:none;" sty
  ion: absolute; left: 0px; top: -5000px; display: none;
  20000;" ls-layeredcontrol="X"> ⋯ </div>
 ▸<span id="IHUB" ct="IHUB" lsdata="{}" class="lsControl--
  e"> ⋯ </span>
  <span class="lsControl--invisible" ct="SY" id="UCF_Syste
  lsdata="{"2":"standards","3":"de","4":1,"8":"/sap/public
  w5/themes/%7ecache-20230216141323","10":"/sap/public/bc/
  7","11":"/sap/public/bc/ur/nw5/1x1.gif","12":false,"15":
  7":"31BD5EEB6598","18":"WEBKIT","19":537.36,"22":"ARIA1
  "sap_fiori_3","26":"standards","27":"/sap/public/bc/ur/
  s/%7ecache-20230216141323","28":"version=UR:10.30.7.350
  9":"10.30.7.350670.0","30":"/sap/public/bc/ur/nw5/theme
  20230216141323","32":"/sap/public/bc/ur/nw5/themes/~cac
  16141323/UR/ls/sap_fiori_3/standards.css?version=UR:10.
  70.0","33":"{\"Date\":\"DD.MM.YYYY\"}","34":"10.30.7.35
  0","40":true,"41":"FESR","44":"Chrome","45":"124.0","46"
  +X6rQ1QXiT12szPw==","47":"EVENT_QUEUE","49":"[\"sap_bel
  \"sap_fiori_3*\"]"}"></span>
  <div id="sapwd_main_window_root_-static"></div>
  <div class="lsControl--invisible" id="sapwd_main_window_
  imations"></div>
 ▸<style media="print" type="text/css" nonce="eLE0ao+X6rQ
  Pw=="> ⋯ </style>
 ▸<div id="sapur-aria" ct="AS" class="lsPage--ariastatic"
  </div>
 ▸<div id="_SAPGUI-icons_0_" data-sap-ls-svglib-id="SAPGUI
  data-sap-ls-svglib-url="https://sax81234.lnxsrv.abraxas
  sap/public/bc/ur/nw5/themes/~cache-20230216141323/Base/
  ap_fiori_3/svg/libs/SAPGUI-icons.svg" data-sap-ls-svgli
  method="XMLHttpRequest" data-sap-ls-svglib-sameorigin="
  style="display: none;"> ⋯ </div>
```

[Computer System Input]: The screenshot appears to be from a mobile application rather than a website, specifically called "Pro Time Mobile App." It seems to be designed for time management, work schedules, or attendance tracking, as suggested by the calendar interface and terminology used. Here's a list of the GUI components and their potential actions based on standard mobile GUI patterns:
1. Top Navigation Bar:
- Back Navigation Icon (left arrow): Likely returns the user to the previous screen or navigates back one level in the app hierarchy.
- App Title Dropdown (Prolime Mobile App with a downward arrow): Could allow for switching between different sections or functionalities within the app.
- Search Icon: Presumably opens a search input to filter or search for specific information within the app.
- User Initials Icon (GN): Likely leads to a user profile section or account settings.

(A) pixel representation (B) DOM representation (C) textual description

FIGURE 4.4: Inputs of a computer system featuring an SAP time recording application. From left to right: Pixel Representation in Figure 4.4 (A), Structured DOM Representation in Figure 4.4 (B) and Textual Description in Figure 4.4 (C) generated by GPT-4V (OpenAI et al., 2024).

Some agents lack computer system input, limiting them to executing predefined sequences without any interaction possibility with the environment. These agents excel in novel content generation or question answering rather than A2C2s (Y. Du et al., 2023; G. Li et al., 2023; Qian et al., 2023; X. Wang, Wang, et al., 2023; Z. Wu et al., 2024). Qian et al. (2023), for example, introduce ChatDev, a virtual chat-powered software development company that enables the development of a software project and improves traceability but is not able to react or interact with a computer system or human. For this reason, these approaches are not discussed further in this section.

The input of the computer system (see Figure 4.4) serves as the observation space for an agent and is pivotal to defining the action space from one observation to the next. We identify four different categories of observation space, namely **pixel representation** in Figure 4.4 (A), **textual description** in Figure 4.4 (B) and Figure 4.4 (C), **multimodal observation**, and **preprocessed observation**. The categories are listed in ascending order in terms of how structured they are and how complex the process of finding all possible actions that can be executed on an observation (henceforth dynamic action inference) is.

Pixel Representation

Using the pixel representations of the GUI offers generalizability across various domains and environments due to uniform accessibility. Since every single pixel could be needed to execute a subtask, a large action space, due to high resolution, can be expected. For most agents pixel representation is equivalent to screenshots. However, some agents also use videos, given the advantage that IO actions can be tracked and even instructions can be extracted from subtitles (Cheng et al., 2024; D. Gao et al., 2024; Shaw et al., 2023; Tan et al., 2024).

The huge possible action space is a problem that agents with pixel representation input have to handle and is identified as one of the biggest challenges in recent development. This challenge can be tackled by abstracting the pixel representation (Z. He et al., 2020; Hong et al., 2023; Rawles et al., 2023; team rabbit research, 2023; Toyama et al., 2021; Wen et al., 2023). Further considerations can be found in subsection 4.3.1.

Textual Description

Textual descriptions, including interface descriptions (Patil et al., 2023; Schick et al., 2023; Shen et al., 2023; Song, Xiong, et al., 2023) – *APIs, database structure, code* – and hierarchically structured representations (Gur, Nachum, et al., 2023; Gur et al., 2018; Gur, Furuta, et al., 2023; Jia et al., 2019; Kim et al., 2023; E. Liu et al., 2018; Shvo et al., 2021; B. Wang et al., 2022; Yao et al., 2022; L. Zheng et al., 2023) – *DOM, view hierarchy (VH), hypertext markup language (HTML)* – are commonly used as input for the agent. This approach simplifies dynamic action inference because of reduced possibilities and structured grouping but limits the application domain by excluding applications that have no access to textual representation – *desktop applications* – of the GUI.

The hierarchical structure provided by DOM, HTML, or VH significantly improves task solution success rates in contrast to pixel-based observation space. HTML, in particular, can be very successfully inferred (Kim et al., 2023) and outperforms VH due to the nature of the training data of LLMs. Despite this, VH remains popular for mobile agents (B. Wang et al., 2022).

Shen et al. (2023) utilize the Hugging Face (HuggingFace, 2024) model zoo for fulfilling user instructions through a dynamic in-context task model assignment mechanism. This equips an agent with a dynamic toolset, each representing a possible action. However, experimental results demonstrate the effectiveness of this approach is contingent upon the quality, quantity, and uniformity of endpoint descriptions.

The high complexity and overloaded representation of textual descriptions enhance the difficulty of selecting the correct elements for a subtask (Gur, Furuta, et al., 2023; L. Zheng et al., 2023). Therefore, some approaches simplify the textual representation by eliminating unnecessary information. S. Zhou et al. (2023), for example, exploit the less noisy accessibility tree representation, provided in most operation systems and browsers for people with disabilities.

Multimodal Observation

Since pixel representation and textual description are limited, the question arises as to whether a combination can provide the best of two worlds and thus a better interpretation of the observation space (S. Zhou et al., 2023).

With the help of generalized agents or existing VLMs, various works attempt to increase the success rate and performance by pre-training or fine-tuning directly on multimodal inputs. Humphreys et al. (2022) and Kil et al. (2024) show that the textual description via DOM is more significant for completing an instruction, but a further boost in accuracy is possible together with the visual representation.

B. Zheng et al. (2024) prove impressively that, relative to other SOTA models, GPT-4V(ision) (openAI, 2023) has a high success rate on popular datasets like Mind2Web (X. Deng et al., 2023) when focusing only on dynamic action inference. This can be simplified by leveraging textual representation to improve contextual understanding and emphasize on the pixel representation for spatial interpretation of the observation space in VLM (Y. Li et al., 2020).

Preprocessed Environments

Preprocessed human interactive environments are characterized by the assumption that they have been predefined and have a fixed set of actions (L. Chen et al., 2021; Raman et al., 2023; H. Sun et al., 2023; Z. Wang et al., 2023; Zhu et al., 2023). The observation space is formulated clearly and comprehensibly for an agent and varies only by parameters.

Ghost in the Minecraft (Zhu et al., 2023) introduces an agent where the observation space is part of the finite and predefined Minecraft environment. The agent knows which items exist and what the available actions are. Although the agent shows strong capabilities for open world environments and outperforms the ObtainDiamond task by more than 47.5%, the environment provides the agent with structured and interpretable observations and actions.

4.1.3 Discussion

A natural text instruction of the user can become more complex depending on wording, precision, and completeness, which typically results in an A2C2 having to make more iterations due to difficulty in planning. The quality of the instructions cannot be forced and must therefore be taken as a given.

In contrast to the instruction space, the perception of the observation space can be implemented in different ways. There is a trade-off between generalization and simplicity. The more generalizable the observation representation is – *pixel-based* – the more difficult it is to find the right actions for executing a subtask. An A2C2, which functions platform and domain independently, needs a general observation space and therefore handle pixel representation. Additional textual descriptions can be provided for optimization purposes if available, but shall not be necessary.

After the agent has gathered the required input, it needs to assess how to update and preserve acquired knowledge for the long-term. To accomplish this, it proceeds to the island of learning (see section 4.2).

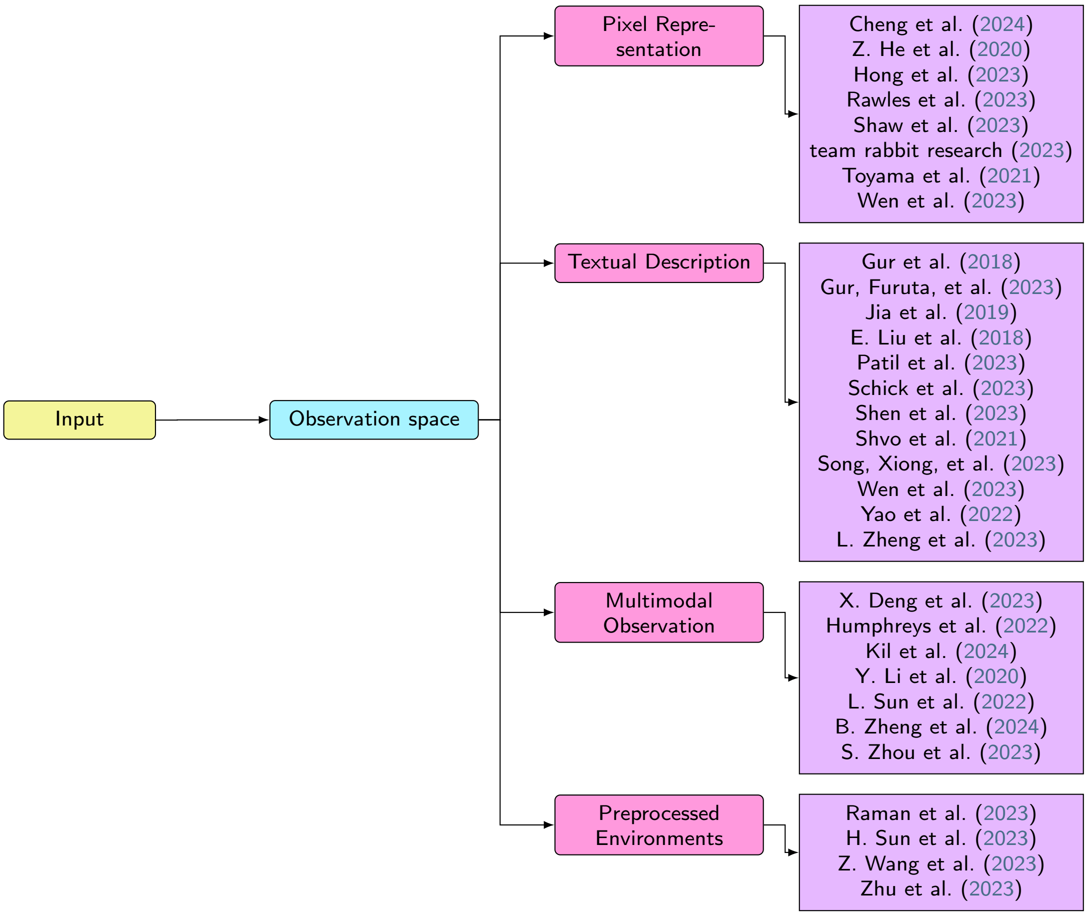


FIGURE 4.5: A compressed overview of agents and their affiliation to the input island.

4.2 Learning - *refining skills and building knowledge*

Through her many years of experience working with computer systems, **Hila** has gained an extensive knowledge of how they work and what tasks can be performed and how they can be solved.

The **A2C2** must first acquire knowledge about computer systems and the tasks that can be solved on them. The methodology he uses for this has a significant impact on efficiency and performance.

FIGURE 4.6: The learning island and how the A2C2 is learning to use computer systems in comparison to Hila.

A good number of A2C2s do not use learning methods and rely on other parts of the taxonomy. These agents operate on the assumption that SOTA models already possess sufficient knowledge and reasoning capabilities to tackle complex instructions without dedicated learning (Wen et al., 2024; Yan et al., 2023; C. Zhang et al., 2023; J. Zhang et al., 2024). For this reason, these approaches are not discussed further in this section. Furthermore, this section does not cover the data collection and pre-training methods of various agents, but tables of different data collections are provided in the Appendix A.

The basis of an A2C2 is a solid knowledge of computer systems and the tasks that can be solved on them. That is why the learning island is crucial for such an agent and the methodology employed significantly impacts sample efficiency and performance. Figure 4.6 shows the learning island and describes how Hila learns to use a computer system compared to an A2C2. This section provides an overview of the two most relevant types of learning: **neural learning** – *how can the model weights be adjusted?* – and **memory** – *how is an external knowledge base built?*. It also discusses **online and offline learning** – *when does the learning happen?* – in the context of A2C2.

4.2.1 Online and Offline Learning

In the context of an A2C2, online learning refers to updating the weights of a model or storing experiences in memory iteratively after the execution of a predefined number of tasks, no matter if they were successful or not. This approach allows a model to adapt dynamically to changes and acquire new knowledge incrementally. In contrast, offline learning means training the agent on a fixed dataset in batches before inference. This typically requires more computational resources upfront and produces a static agent.

4.2.2 Neural Learning

Neural learning includes all training strategies that adjust the weights of a neural network via gradient descent or other update strategies. The relevant neural learning approaches for A2C2s are **RL** and **fine-tuning**. RL refers to pure RL approaches where models learn optimal actions through trial and error interacting with an environment. Fine-tuning includes models that initialize pre-trained weights and are re-trained on domain-specific knowledge, which in turn can also be RL-based.

Reinforcement Learning

Current development in RL has proven successful in various tasks, such as *landing a space shuttle on the moon in gamified environments (Gadgil et al., 2020)* or *autonomously driving vehicles (Grigorescu et al., 2019)*. Therefore efforts have been made to leverage RL for training agents capable of autonomously performing tasks on a computer system.

In the domain of computer control, systems such as AndroidEnv (Toyama et al., 2021) demonstrate the effectiveness of RL, with either textual or pixel-based input. These work in restricted domains like android devices without significant augmentation, but encounter two fundamental difficulties: sparse reward for long trajectories and data efficiency.

Andrychowicz et al. (2017) tackle the issue of sparse rewards with hindsight experience replay, storing experienced episodes $s_0, s_1, ..., s_n$ and training off-policy algorithms like DQN (Mnih et al., 2015) not only on a single sparse goal g but on a set of goals $g \in G$. Training on multiple goals can be easier and yield better results, even if only a single goal is relevant.

Gur et al. (2018) propose curriculum learning and reward augmentation with Q-learning to overcome sparse rewards. Curriculum-DQN simplifies instructions by decomposing complex instructions into an easier subset (see section 4.3), gradually increasing in complexity until the initial instruction can be successfully executed. Reward augmentation involves counting the matching DOM elements between the given state and goal state to compute normalized rewards heuristically.

The issue of data efficiency is addressed through various approaches. Shi et al. (2017) employ behaviour cloning (BC) to supervised pre-train a model on demonstrations. This works very efficiently in known but yields bad results for novel environments. Meta-Trainer (Gur et al., 2022, 2018; Gur et al., 2021) and WebShop (Yao et al., 2022) focus on generating dynamic environments with varying complexity levels to create sufficient demonstration data, rendering a possible data bottleneck negligible. The Meta-Trainer achieves this by training an instructor to recover rule-based or random policies, which can then generate expert-level demonstrations to train an agent. Experiments show good results for simple webtask environments but are limited to 100 DOM elements, which is far from reality – *ProTime (see section 2.5) has 1252 DOM elements*. Another indicator for the limited performance of such agents is that the best model of the WebShop agent has a task succession rate of 29%, while human expert performance is evaluated much higher at 59%.

Decision Transformer (L. Chen et al., 2021) converts RL problems into conditional sequence modeling, enabling the use of masked transformers to obtain optimal actions. Comparison of the proposed architecture and other offline RL architectures shows that Decision Transformer not only performs better on longer context lengths, but is also effective in long-term tasks with sparse rewards.

Fine-Tuning Models

Fine-tuning a model is the process of unfreezing part of the weights of an existing foundation model with pre-trained parameters and adapting them to learn domain-specific knowledge (Howard & Ruder, 2018). This fine-tuning process can in turn use different machine learning strategies. WebGPT (Nakano et al., 2022), for example, learns to interact with the web by RL-tuning and BC-tuning. While RL-tuning can handle dynamic observation spaces well, it tends to be more expensive and time-consuming than other approaches for more complex computer systems.

Instruction tuning and low-rank adaptation (LoRA) are two popular training techniques utilized for fine-tuning existing models (Gur, Furuta, et al., 2023; H. Liu et al., 2023; Patil et al., 2023; Xiang et al., 2023; Zeng et al., 2023). Their strength is the calculated efficiency and data efficiency compared to training a model from scratch and the interpretability provided by the expert data (Chung et al., 2022; Raffel et al., 2019). For instance, Schick et al. (2023) employs Toolformer, which leverages a collection of APIs and their description as training data for instruction-tuning a pre-trained LLM. This leads to an improved capability of selecting the right API for an instruction. Furthermore, both LLM and VLM benefit from fine-tuning (Furuta et al., 2024; Kapoor et al., 2024; W. Wang et al., 2024).

An exciting development highlighted by Lù et al. (2024) is WEBLINX. They illustrate through experiments that smaller, fine-tuned language models outperform even the best zero-shot LLMs such as GPT-4 (OpenAI et al., 2024), in solving web tasks. However, these smaller models exhibit significantly poorer generalization to unseen observation spaces.

4.2.3 Memory

Memory as a training method gained popularity with advances in retrieval augmented generation (RAG) (Cai et al., 2022; Lewis et al., 2020; J. Lin et al., 2023; Mao et al., 2020; Park et al., 2023). While memory may not involve the same kind of weight optimization as traditional neural network training, it focuses on preparing general information, past experiences, and domain-specific knowledge in an efficiently queryable way. This enables an agent to, access a dynamic changeable knowledge base. In literature, there are many ways to build such a long-term memory. The three most relevant ones are **natural language memory**, **embedding memory**, and **symbolic database**.

Natural Language Memory

This type includes all agents that store information as raw text (Shinn et al., 2023; G. Wang et al., 2023). Reflexion (Shinn et al., 2023), stores the experienced tasks and then uses a sliding window technique to retrieve the desired text. The advantage of natural language memory is a, for humans, readable, comprehensible, and easily adaptable knowledge base. However, complex algorithms are needed to retrieve the right information in a growing knowledge base.

Embedding Memory

The majority of agents encode information into an embedding, which is then stored in a vector database (J. Lin et al., 2023; Ng et al., 2023; Park et al., 2023; Qian et al., 2023; Qin, Hu, et al., 2023; L. Wang et al., 2024; L. Zheng et al., 2023; Zhong et al., 2024; Zhu et al., 2023). This enables an agent to retrieve task-relevant knowledge and experiences through embedding similarity – *cosine similarity, euclidean distance* – during input decomposition (see section 4.3). The prerequisites of useful embeddings are a good chunking strategy and a tokenizer and embedding model tailored to the agent.

Inspired by the Davidsonian semantic theory (Davidson, 2001), Modarressi et al. (2023) store the data in a structured list and its vector representation. Each triplet $< t_1, t_2, t_2 >$ defines, very similar to prolog (Warren et al., 2015), the first argument, the relationship and the second argument – $< hila, software\, engineer, large\, company >$. Thanks to this memory, semantic similar triplets representing relations, can be found, leading to more stable and expected output.

Symbolic Database

Some approaches utilize external symbolic databases to store knowledge (Hu et al., 2023; X. Zhou et al., 2023). By exploiting the ability of an LLM to generate structured query language (SQL) an agent can dynamically translate natural language instructions into SQL and *select*, *delete*, or *update* information stored in the symbolic database. Hu et al. (2023) store relevant information of a synthesized dataset into a symbolic database, resulting in a strong improvement over an LLM without memory with 60% better accuracy. This assumes that all the required information is available and can be displayed in a typical tabular form.

4.2.4 Discussion

All training methods can be carried out online and offline, and are not mutually exclusive. Some agents perform fine-tuning and include an experience memory (Cheng et al., 2024; Y. Deng et al., 2024; Wen et al., 2023). If a specific task with low variance is trained, RL and BC may be the right choice – *recurrent task bots*. However, RL is not feasible for an A2C2 due to the large amount of data needed. For more general agents, that intend to operate unseen applications, fine-tuning and/or memory is the right choice because it is more flexible and less resource-intensive. The optimal training method needs extensive research. H. Liu et al. (2022) have shown that optimized fine-tuning can be better and cheaper than memory for classification tasks. In contrast, Ovadia et al. (2024) show that for unseen domains, knowledge injection through memory has an average accuracy gain of over 20% over unsupervised fine-tuning. Unfortunately, these two research experiments are not meaningful enough, as they did not compare two equivalent methods. Both publications feature one highly optimized and another basic implementation. To make a relevant statement, two highly optimized SOTA variants need to be compared.

Now that the A2C2 has explored the possibilities of learning, it needs to know how to utilize this knowledge to plan an instruction and its subtasks. To answer this question, the agent continues to the input decomposition island (see section 4.3).

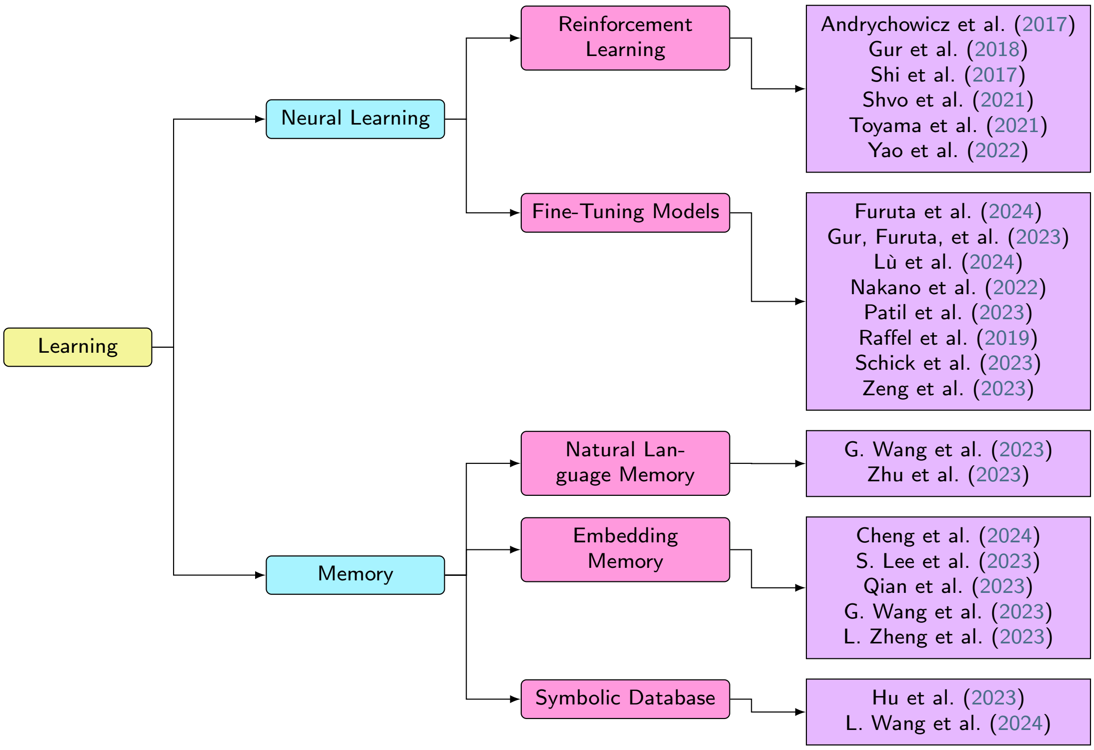


FIGURE 4.7: A compressed overview of agents and their affiliation to the learning island.

4.3 Input Decomposition - *ensuring task comprehensibility*

Hila intuitively knows which sub steps are needed to solve her task and can use this knowledge to formulate a plan.

In order to gain similar insights, the **A2C2** must break down the user instructions into manageable subtasks and interpret the screenshot of the graphical user interface.

FIGURE 4.8: The input decomposition island and how the A2C2 carries out the decomposition and planning of an instruction compared to Hila.

After the A2C2 has learned what information it receives and how it can learn, the next steps are to derive the possible actions from the observation space and to break down complex instructions into subtasks. Figure 4.8 shows the input decomposition island and explains how Hila proceeds with a complex instruction in comparison to an A2C2. This section tackles the two methods to achieve this: **dynamic action inference** – *how can the observation space be simplified?* – and **subtask inference** – *how can a complex instruction be partitioned?*

4.3.1 Dynamic Action Inference

TABLE 4.1: An overview of the individual dynamic action inference methods. The aim is for all columns to contain the smallest possible values. For the visual representation see Appendix B. Tokens were counted with the OpenAI tokenizer as a reference(OpenAI, 2024).

	size (KB)	actions	tokens
raw HTML	224	1331 tags	76485
raw image	530	2400 x 1080 pixels	425
filtered HTML	20	41 tags	6178
processed image	530	91 elements	425
inferred representation	2	48 elements	797

The dynamic action inference describes different methods allowing A2C2s to identify actions (see Table 4.1), localize elements from the observation space – *known as grounding*, and structure them interpretably. An A2C2 needs this action inference to reduce the action space to a manageable size. Static action spaces decrease complexity, simplify, and stabilize the learning process of an agent (Q. Liu et al., 2022) but lack the capability to interact with unseen computer systems. The inference methods can be divided into three categories, namely **textual**, **pixel-based** and **multimodal** inference.

Textual Inference

Textual inference refers to processing DOM or VH input as observation space, making dynamic action inference easier because of their organized structure. These attributes can already be considered partially decomposed since the GUI components are identifiable through their unique representation. The use of such a representation entails limitations due to context length constraints (see Table 4.1 raw HTML tokens) and the presence of task-irrelevant information (see Table 4.1 raw HTML size). This leads to incorrect action suggestions. Hence, existing approaches attempt to minimize the structured representation without losing its benefits to tackle said limitations.

Most methods rely on the ability of LLMs to understand HTML in simulated environments (Tao et al., 2023; N. Xu et al., 2021; L. Zheng et al., 2023). Gur, Furuta, et al. (2023) pay particular attention to real-world websites, showing an increased complexity because of the overloaded representation. They counter this by summarizing the large HTML documents into task-relevant snippets using the self-introduced HTML-T5, an LLM specifically trained to handle long HTML input, to better capture the structure of long HTML documents and thus be better processed by the agent (see Table 4.1 filtered HTML).

To improve the understandability of the observation space B. Wang et al. (2022) map the domain-restricted representation of VH heuristically into HTML using depth-first search. This is done because LLMs are proven to perform better in understanding HTML due to their pre-training dataset. Nonetheless, the HTML needs to be reduced to a compact set of properties – *class*, *text*, *resource_id*, *content_desc*.

Shvo et al. (2021) present an RL-based system that maps VH into a $n\,x\,m$ matrix where n represents a GUI element – *div, button, a* – with m features, including the textual description that specifies its purpose, whether it is clickable or editable, and its relative location in the VH. The relative location helps to capture the spatial correlations across elements. While this simplified VH representation suffices for their benchmarks of four mobile apps – *Shopping list, Alarm clock, Internet settings* – they recognize the potential in including pixel-based representation for capturing salient information that is not provided by the VH.

Pixel-Based Inference

In order to reproduce human behavior when using an application as well as the fact that textual descriptions are not always accessible, an A2C2 must succeed in dynamic action inference based on a pixel-based representation (Bavishi et al., 2023; Cheng et al., 2024; D. Gao et al., 2024; T. J.-J. Li et al., 2021; Shaw et al., 2023; J. Wu et al., 2021; Yan et al., 2023; You et al., 2024; X. Zhang et al., 2021). Contrary to textual description, pixel-based representation is unstructured, and therefore dynamic action inference becomes significantly more difficult. The challenge in pixel-based systems lies not only in the reduction of this extensive action space (see Table 4.1 raw image) but also in bringing it into a structured form. Here the difficulty lies in the meaningful breakdown of a GUI by utilizing computer vision techniques – *edge detection, color detection* and *optical character recognition (OCR)* – for the extraction of logical components from the screenshot (see Table 4.1 processed image).

AutoDroid (Wen et al., 2023) proposes a strategy where HTML is generated from the GUI representation, by exploring an application offline and storing the information in a transition graph. The HTML is then simplified to a reduced number of interactive elements and their properties. Additionally, invisible elements are removed and equivalent elements are merged.

CogAgent (Hong et al., 2023) highlights the problem that especially tiny icons or text are difficult to recognize in the conventional resolution. However, increasing the resolution for better recognizability leads to the known context length constraints. To counteract this, they introduce a high-resolution cross-module – *smaller pre-trained visual encoder with 0.3B parameters using cross-attention with a smaller hidden size* – that is capable of efficiently processing high-resolution image features while minimizing the computing load. This is done by using two different visual decoders that differentiate in hidden size to achieve good performance in VLM and OCR.

An optimization of the approaches above is presented by Song, Bian, et al. (2023). They utilize YOLOv8 (U. Team, 2023), a SOTA deep learning model for object detection and segmentation and explicitly train it on GUI components. To get the textual information PaddleOCR (Y. Du et al., 2020) is applied. The core of this approach is semantic grouping, which gradually divides the recognized elements and texts from the GUI into semantically related blocks. This approach outperforms existing implementations like GPT-4V(ision) (openAI, 2023), by recognizing the semantic groups precisely regardless of the size of the elements.

ScreenAI (Baechler et al., 2024), a VLM specialized in GUI and infographic understanding, uses a detection transformer model to annotate screens by identifying GUI elements like *images*, *graphics*, *buttons*, and *text*. Unlike previous approaches (G. Li et al., 2022), it also predicts the bounding boxes of the GUI elements and has the capabilities of an icon classifier for improved icon recognition. Unrecognizable icons, infographics, and images are fed to the PaLI (X. Chen et al., 2023) image caption model, which provides descriptive and contextual captions. Finally, an OCR engine extracts text content combined with the annotations. Overall, a comprehensive SOTA screen description can be created through this process with a comparatively smaller – *5B parameters* – model. Nevertheless, further research is needed to match the performance of large multimodal models (LMMs).

Multimodal Inference

To benefit from the generalizability of vision components and the structure that comes with textual representation, dynamic action inference can be done on multimodal input. More recent approaches are based on vision transformers, characterized by their transformer architecture, which can process spatial and sequential data and thus show good results in mapping text to images and vice versa (Conde & Turgutlu, 2021; Han et al., 2020; G. Li & Li, 2023; Soselia et al., 2023).

Multiple recent agents utilize the entire or parts of the HTML representation together with a screenshot (Furuta et al., 2024; Kil et al., 2024; Lù et al., 2024). Lù et al. (2024) introduce the Dense Markup Ranking (DMR), which compares HTML elements with past actions to identify the most relevant elements and remove irrelevant ones. This results in a more compact DOM representation that, along with action history and screenshots, becomes faster than X. Deng et al. (2023) and H. He et al. (2024). Similar to Song, Bian, et al. (2023) they use the Dual-View-Contextualized Representation to contextualize each HTML element with its pixel representation neighbors, optimizing further by ensuring semantically related elements are often close to each other.

S. Zhou et al. (2023) explore another approach by using the DOM tree, a screenshot of the website, and in addition its accessibility tree. This three-column representation provides the greatest flexibility to relate to elements on the website and enable textual and pixel-based inference strategies. Following this and inspired by J. Yang et al. (2023), Koh et al. (2024) present their set-of-mark (SoM) implementation. Each interactive element on the website is labeled with a bounding box and an ID. The system takes a screenshot that enables a VLM to reference elements using the labels. Thanks to SoM, small, closely spaced elements can be recognized correctly, which is not always possible with the accessibility tree approach.

4.3.2 Subtask Inference

To reduce complex instructions, the first step is to determine whether the user instruction can be performed in a single step, or whether it is more complex and therefore needs decomposition for successful execution. Afterwards, the instruction gets decomposed into more feasible subtasks with optional experience usage.

Instruction Type

Since the instruction modality is clearly defined in subsection 4.1.1, it is insightful to analyze which types of instructions can occur. This gives a better understanding of what an A2C2 needs to be capable of processing. The vast amount of publications (X. Deng et al., 2023; Y. Deng et al., 2024; Gur, Nachum, et al., 2023; Z. He et al., 2020; Jia et al., 2019; Lù et al., 2024; Reed et al., 2022; S.R.K. Branavan et al., 2010; L. Zheng et al., 2023) are differentiating between single and sequential task instructions.

FIGURE 4.9: A single task instruction example: Open the application ProTime.

Single task instructions (see Figure 4.9) are characterized by their singular focus on a solitary observation, typically straightforward. Such instructions are designed to require only a limited number of actions to accomplish the intended task.

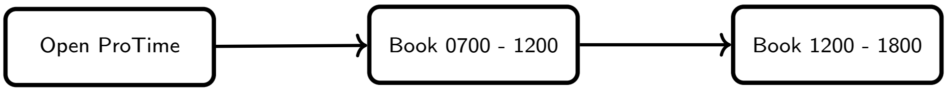


FIGURE 4.10: A sequential task instruction example: Open the application ProTime. Book my working hours from 0700 - 1200. Book the working hours from 1230 - 1800.

Sequential task instructions (see Figure 4.10) encompass a series of single tasks distributed across various observations within a single application. With the most amount of data collected in this category, we can conclude that the highest percentage of user instructions can be assigned to this category (Shen et al., 2023).

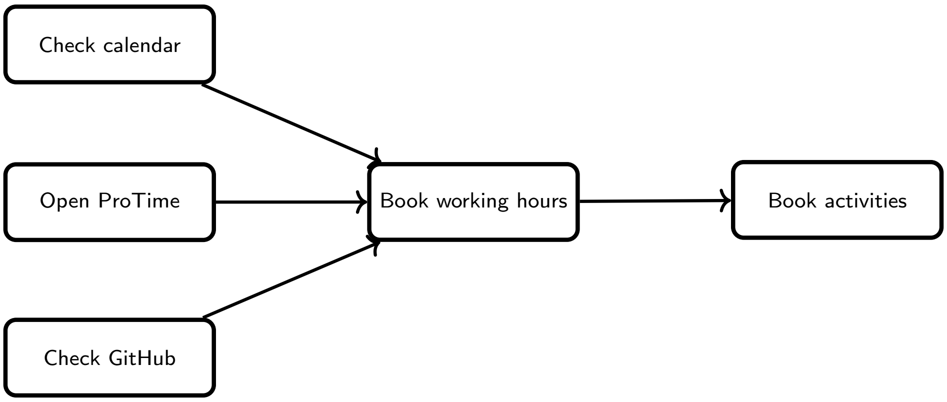


FIGURE 4.11: A graph task instruction example (see section 2.5): Please record my working hours and activities on ProTime. For the activities check assigned GitHub issues and my calendar. I worked from 0700 to 1700 with a 30min lunch break at noon.

Graph Task as illustrated in Figure 4.11 is introduced by HuggingGPT (Shen et al., 2023), covering many real world applications. This category is a useful addition, as it allows different subtasks to be handled separately, improving performance and expanding planning options. Graph tasks involve sequential tasks, with the addition of parallelism, which means that multiple single tasks do not depend on each other – *check calendar, check GitHub, open ProTime*. Dependent instructions can be merged at a later point in time – *book working hours, book activities*.

In a generalized application domain, we propose a second dimension to the above-mentioned task types, the **single-/multiapplication task** was not found in the available literature. This dimension determines whether a task operates on multiple applications or if the goal of the user is achievable using a single application. In current literature and existing benchmarks, it is commonly assumed that the agent starts with the right observations and a user instruction that is bound to it. Figure 4.11 shows three different applications, calendar, GitHub, and ProTime. For awareness of complete observation space switching – *launch a new application* – it is crucial to determine whether the agent is dealing with a single or multi-application task.

Decomposition Process

If a user instruction is not assignable to the type of single task, planning and executing an instruction can be challenging for the agent. Therefore, some recent approaches are dedicated to simplifying the instruction by decomposing it into more feasible subtasks. These methods are directly integrated into the planning process of the agent (Ding, 2024; Gur, Furuta, et al., 2023; Huang et al., 2024; G. Li et al., 2023; Niu et al., 2024; Qian et al., 2023; Ruan et al., 2023; Z. Wang et al., 2023). The decomposition is limited to agents that generate text via LLMs. Other agents can use a similar process internally, without being fully comprehensible by humans.

L. Wang et al. (2023) address limitations in few-shot chain-of-thoughts (CoT) prompting (Wei et al., 2022) and zero-shot CoT prompting (Kojima et al., 2022), namely calculation errors, missing step errors, and semantic misunderstanding. They are tackling these problems with plan and solve prompting. The plan, as the first step, represents the decomposition of the instruction. The LLM receives the user instruction as well as a prompt template on what to do. In contrast to the conventional step-by-step approach, they claim that the problem must first be understood before a plan can be developed – *Let's first understand the problem and devise a plan to solve the problem*. As a result, it has been proven that calculation errors and missing step errors can be reduced by 2% to 3%, but semantic misunderstanding remains due to the difficulty of correct prompting.

Y. Wu et al. (2023) build on the concept of hierarchical task scheduling and develop a three-level framework Plan, Eliminate, and Track, where the Plan module performs task decomposition, inspired by the human ability to organize complex tasks into higher-level subtasks and the contextual prompting techniques of LLMs. Based on the task description, the module is asked the question – *What are the middle steps of this description?* – and instructed to generate a list of subtasks. Despite the good performance, evaluation errors occur due to the variance in token generation – *synonyms do not achieve full accuracy on the ground truth*.

AssistGUI (D. Gao et al., 2024) introduces a planner based on LLM that receives an instruction video in addition to the user instruction and creates a hierarchical task tree $p = [p_1, p_2, ..., p_N]$, where each p_i corresponds to a list of subtasks. This is done by creating hierarchical steps based on the video subtitles from the observation space and then adapting them according to the user instruction. Despite promising results, flaws such as the usage of redundant operations in the task tree remain.

Some approaches allocate the subtasks to different agents (Khot et al., 2023; Y. Wang, Wu, et al., 2024). TDAG (Y. Wang, Wu, et al., 2024) focuses on the problem of unchangeable subtasks during runtime. This means that the failure of a subtask results in the failure of the entire complex instruction. To counteract this problem a main agent decomposes the user instruction and assigns a specific subtask to each subagent which increases the success rate by an average of 5%. In addition, the subtasks are dynamically adaptable and therefore have a chance to recover from errors. Nevertheless, the optimized performance of the TDAG comes at the cost of increased resource consumption and slower inference speed, which must be taken into account, especially in time-critical and resource-constrained environments.

Experience Usage

As an optimization of the direct decomposition of user instructions, the additional integration of past experiences is mentioned in various approaches. These empirical values can be gained from the short-term experience of an agent (Qin, Hu, et al., 2023; R. Zhou et al., 2024) or by working with a long-term memory module (Y. Deng et al., 2024; S. Lee et al., 2023; Z. Wu et al., 2024; Zhu et al., 2023). In the beginning of A2C2, hierarchical reinforcement learning approaches have transformed from manual specification (E. Liu et al., 2018; Parr & Russell, 1997) of subtasks to combining task decomposition with experience (Andreas et al., 2016) to fully relying on experience (Daniel et al., 2016; Marzari et al., 2021). In more recent times foundation model-based agents have experienced a similar evolution.

RestGPT (Song, Xiong, et al., 2023) presents a method based on the cooperation of a planner and an API selector. This collaboration can be described as an iterative decomposition of the user instruction into subtasks and their corresponding APIs. In each step t, the planner uses its experience to generate a high-level natural language plan based on the user instruction, previous natural language plans $(p_1, p_2, ..., p_{t-1})$ and the execution results $(r_1, r_2, ..., r_{t-1})$. Despite impressive results with a success rate of > 70% for their baseline, the success rate decreases drastically if the planner loses sight of the goal after several rounds of execution.

AgentSims (J. Lin et al., 2023) a simulated environment evaluates LLMs and their task performance capabilities. The task decomposition is described as a planning system that processes the generated subtasks based on information from a vector database. Embedded experiences are stored and retrieved to expand knowledge about specific or recurrent events.

Inspired by how humans decompose complex instructions into subtasks, MobileGPT (S. Lee et al., 2023) uses the Explore-Select-Derive process.

- **Explore**: In this phase, relevant subtasks are generated for as many screens as possible in a pre-emptive, offline manner. This subtasks are stored in memory to be used during the select phase.
- **Select**: Based on the user instruction and the subtasks associated with the current screen, a list of potential subtasks is retrieved from memory. An LLM is then queried to determine which subtask should be executed to fulfill the user instruction.
- **Derive**: The LLM is prompted with low-level actions in an iterative approach to determine the specific action required to execute the subtask. This process continues until the subtask is fulfilled and the next subtask can be selected. Once all subtasks have been completed, they, along with the user instruction, are saved in memory.

The Explore-Select-Derive approach exceeds the success rate compared to the GPT-4 baseline by 11%, and the normally high latency when using VLMs can be reduced by querying the memory.

In RecMind (Y. Wang, Jiang, et al., 2024), task decomposition divides the individual steps into thoughts, actions, and observations. This extends prompting strategies through the self-inspiring approach, which encourages the agent to generate and refine its thoughts and solutions instead of reacting to directly given user instructions. The agent is supported by a memory system that collects individual user data — *personalized memory* – as well as general information – *world knowledge* – that is loaded via external tools such as databases or search engines. The additional knowledge improves to decomposition of an instruction but leads to limitations imposed by the context length of LLMs.

4.3.3 Discussion

Many approaches in dynamic action inference use the VH or are tested in simulated environments, due to the minimized complexity of these environments. This leads to an illusion of reality, as these approaches work very well on selected benchmarks but show disappointing results for real-world applications.

Although the actions to be derived from the task are easier to determine with textual inference of the observation space, the use of computer vision proves to be necessary. This approach can be used system-independently and the rapid development of VLMs already shows promising results. Combining the use of screen and text could pave the way for future improvements in A2C2.

It can be stated that task decomposition is becoming increasingly important, especially in newer systems (Niu et al., 2024; Z. Wu et al., 2024). Dynamic solutions for subtask inference show good results due to their adaptability, especially in real-life systems. The use of experience for the decomposition of tasks can be seen as the most promising approach. Both personalized, user-related knowledge and knowledge from past instructions can optimize the accuracy of task decomposition. A sensible balance must be made between the amount of knowledge used and the restriction imposed by the context lengths of LLMs.

Having learned how to derive actions from the observation space and how complex instructions can be decomposed into subtasks, an A2C2 still lacks the knowledge to refine a plan from the received subtasks. This knowledge can be obtained on the plan refinement island (see section 4.4).

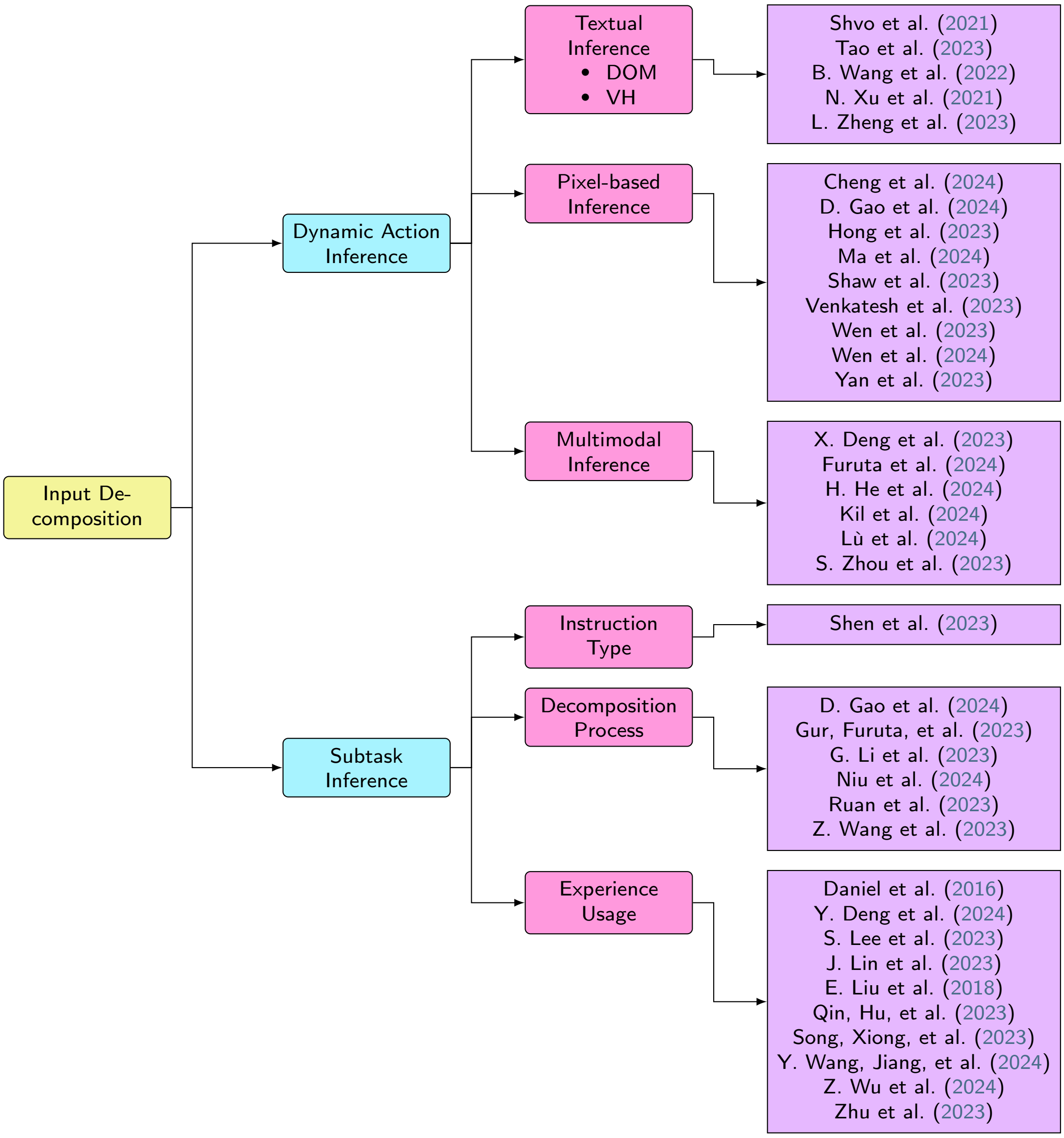


FIGURE 4.12: A compressed overview of agents and their affiliation to the input decomposition island.

4.4 Plan Refinement - *debating and refining subtasks*

Hila intuitively knows which sub steps are needed to solve her task and can use this knowledge to formulate a plan.

In order to gain similar insights, the **A2C2** must break down the user instructions into manageable subtasks and interpret the screenshot of the graphical user interface.

FIGURE 4.13: The plan refinement island and how the A2C2 is determining iteratively the correct plan compared to how Hila is verifying her plan.

Creating a meaningful plan from the subtasks is a core component of an A2C2. Figure 4.13 shows the plan refinement island and how an A2C2 verifies its plan compared to how Hila proceeds.

After the agent has successfully reached the plan refinement island, possibilities for analyzing the decomposed instruction and converting it into a polished plan are presented. Therefore, this section is divided into three parts: **open loop reasoning** – *how good is the plan in itself?* – **multiagent reasoning** – *how good is the plan if compared* – and **closed loop reasoning** – *what if an unexpected behaviour occurs?* An agent may incorporate several of these categories.

4.4.1 Open Loop Reasoning

The basis for sequential reasoning was established with CoT (Wei et al., 2022). This prompting strategy uses a series of intermediate reasoning steps to take the complexity out of a – *arithmetic, common sense, symbolic* – instruction, offering not only an increase of the success rate – *2% for arithmetic instructions* – but also an optimization in robustness compared to standard prompting methods. It is noteworthy that CoT becomes increasingly important as the parameter size of a model grows. Since CoT, there has been a whole field of publications dedicated to surpassing it or creating a variation of it adapted for specific open loop reasoning use cases (Y. Deng et al., 2024; S. Gao et al., 2024; Kim et al., 2023; Sel et al., 2023; L. Wang et al., 2023; Y. Wang, Jiang, et al., 2024; Z. Wang et al., 2024; Yao, Yu, et al., 2023; Yao, Zhao, et al., 2023; J. Zhang et al., 2024; P. Zhou et al., 2024).

Kim et al. (2023) formulate a prompting scheme that recursively criticizes and improves the output by including a critique – *Review your previous answer and find problems with your answer* – and improvement – *Based on the problems you found, improve your answer* – step during reasoning. Such a self-critique ability leads to a proactive reduction of errors and can be done multiple times before providing the user with the final plan (Y. Bai et al., 2022; Ganguli et al., 2023; Saunders et al., 2022).

LLM+P (B. Liu et al., 2023) utilizes an LLM to convert an instruction to a problem in the planning domain definition language (PDDL). After the conversion, an external planner solves the problem more efficiently than an LLM, and the solved plan is then translated back into natural language. This leverages the success of tasks like Tower of Hanoi (Hinz et al., 2013) and robotic applications from not working to over 85%, showing that an LLM, even if prompted to behave like a planner, cannot handle the amount of states and preconditions of a real-world task. However, for the correct translation in PDDL, the task must be specified completely, which is rarely the case with user instructions.

Graph of Thoughts (GoT) (Besta et al., 2023) introduces a framework that transforms prompting to a complex thought network capable of retrieving the most important information and enhancing the reasoning process with a feedback loop. Building on self-consistency (X. Wang, Wei, et al., 2023) – *generating multiple CoTs* – and Tree of Thoughts (ToT) (Yao, Yu, et al., 2023) – *structuring thoughts as a tree* – GoT tries to imitate a human reasoning process, that is not limited to follow a single plan or making multiple separate plans but can refine a task through complex thoughts. The graph $G = (V, E)$ contains a set of vertices V – *solution to a subtask* – and edges $E \in VxV$ – *subtasks* $(v_x, v_y, ..., v_n)$ *result in subtask* v_r *with edges* $(e_{xr}, e_{y_r}, ..., e_{nr})$ – and uses a modular approach for scoring, validating and selecting the optimal subtasks. This framework results in a 62% increase in quality over ToT and brings a great speed reduction.

4.4.2 Multiagent Reasoning

One big improvement from CoT to GoT is the ability to evaluate and compare multiple plans. This multi-plan capability has been done on a single agent, which requires fine granular experience and memory management, as it tends to fixate on one solution or hallucinate (Ji et al., 2022). To tackle this issue, and have separate entities reasoning about a user instruction, multi-agent reasoning has been tried, enabling research in techniques like majority votes and debates (P.-L. Chen & Chang, 2023; Y. Du et al., 2023; T. Liang et al., 2023; Qian et al., 2023; L. Wang et al., 2024; Q. Wu et al., 2023), which are often encountered in the real world – *swiss parliament, open-source community*.

MAD (T. Liang et al., 2023) addresses the problem of sticking to an answer defined as degeneration-of-thought by simulating a debate between multiple agents and a judge who manages it and decides the final solution. In contrast to self-reflection, where the average disagreement remains fairly constant over several reasoning iterations – *between 15% - 25%* – MAD starts with a very high disagreement between agents of over 75% and reduces it to below 50% as the debate progresses. The framework is designed to have a meta prompt that defines the number of agents, the number of debate iterations, and other configurations. Then the agents and a judge are instantiated and a debate round starts. One debate round consists of the agents speaking in a fixed order after each other to argue and contradict the others. After the maximum iterations or if the judge decides that the correct solution has been obtained the debate is stopped, improving the success rate of CoT by 12%.

ChatDev (Qian et al., 2023) introduces a virtual software technology company, that can solve complex software engineering tasks, through multiple agents – *CEO, Programmer, Designer, Tester* – and distinct phases – *designing, coding, testing, documenting*. The agents iteratively discuss and refine task requirements, in a process called memory stream, until they agree. While the system significantly reduces development time – *avg. of 409.84s per task* – and cost – *avg. of $0.2967 per job*, it struggles with project complexity and code quality due to limitations in autonomously determining implementation specifications and the inherent randomness of LLM-generated code.

4.4.3 Closed Loop Reasoning

One problem that is not addressed in the systems, mentioned above, is the interaction between the plan and the environment, as well as the interaction between the plan and a user. After a plan is finalized and the first subtask is executed, undesired behaviors can occur – *errors in an application*, *execution of critical subtasks, nothing happening at all* – which cause the plan to fail. In these situations, an A2C2 has to be able to recover autonomously and include the feedback in the continuous plan refinement loop (P.-L. Chen & Chang, 2023; Lù et al., 2024; Raman et al., 2023; Shinn et al., 2023; Yao, Zhao, et al., 2023; R. Zhou et al., 2024).

ReAct (Yao, Zhao, et al., 2023) has introduced a prompting method that combines plan reasoning with action generation. This can be achieved through a combination of thought – *what does my environment offer, what do I need to be able to execute the subtask?* – and act – *how do I execute the subtask?* – prompts in an iterative reasoning loop. This allows the agent to be more controllable, human-aligned, and flexible for new instructions. However, this method only works better than open loop approaches if fine-tuned.

Reflexion (Shinn et al., 2023) proposes an alternative approach to autonomous thinking that uses verbal reinforcement to learn from previous failures by reusing internal or external feedback as context for the next attempt. Specifically, the agent consists of an internal feedback evaluator – *formulates the problem of a subtask* –, a self-reflection component – *explains why the subtask failed* – an actor – *executes new subtasks* – and a short-term memory – *executed subtask knowledge for the current instruction* – as well as a long-term memory – *general knowledge about multiple instructions and domains*. This approach is a complement to CoT or ReAct and increases their performance in several benchmarks.

ART (Paranjape et al., 2023) pursues the idea that a human in the loop can intervene in the execution of an instruction by assisting the agent with an example or by introducing new requirements and information. This is done by allowing the user to add correction subtasks to the current instruction, which are taken into account during the next execution step. It achieves a significant improvement over few-shot prompting.

4.4.4 Discussion

The ability to analyze critically is very important and brings a great increase in performance for an A2C2. To work on computer systems, the agent requires a closed loop reasoning approach, able to receive not only environmental feedback but also human feedback. Human in the loop is one of these approaches, that do not yet exist widely, as this restricts the autonomy of an A2C2. However, this could improve human alignment. To keep as much autonomy as possible human in the loop could be modified to a system, where the human is represented by another agent, whereby the real human only needs to serve as a control and intervention instance. Promising approaches for this island are those that enable modular plan refinement, as these can be flexibly exchanged and made dependent on instructions and domains. (H. Zhang et al., 2024; P. Zhou et al., 2024)

In the meantime, the A2C2 has visited four islands and acquired a broad knowledge of how to successfully execute a user instruction. However, it still lacks the knowledge of how to translate the plan it has received into executable actions (see section 4.5).

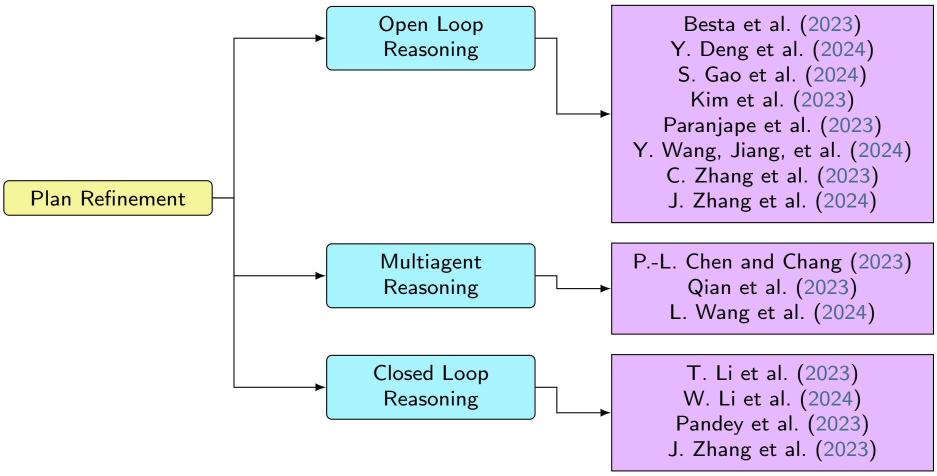


FIGURE 4.14: A compressed overview over agents and their affiliation to the plan refinement island.

4.5 System Output - *interacting with the environment*

Hila executes her plan with her keyboard and mouse. If it is not working, she makes the necessary changes and tries out the new plan.

The **A2C2** can translate the plan into IO peripheral actions and execute them directly on the computer system. In the case of an error message or if information from the user is missing, the **A2C2** repeats the iterative planning mechanism with a new starting point and more knowledge.

FIGURE 4.15: The system output island and how the A2C2 is correcting his plan if the execution fails compared to the strategy of Hila.

Some methods are based only on generating text. In contrast to the generation of executable actions, these methods focus on interacting with the user and telling them what to do. As the generated texts cannot be utilized for computer control, these techniques are not further considered (G. Li et al., 2023; Raman et al., 2023).

To translate the received plan into executable actions, the A2C2 requires the ability to interact with its environment. The most human-like approach for it encompasses the utilization of **IO peripherals**. This technique is characterized by considerable generalizability for various application domains. Other methods are based on the generation of **executable code** or selecting suitable **tools**. Figure 4.15 shows the system output island and compares how an A2C2 and Hila interact with the environment.

4.5.1 IO Peripherals

The most intuitive option of system outputs is via IO peripherals. They enable a high degree of generalizability and can be used across computer systems. The category of IO peripheral output not only includes agents that can control IO peripherals directly but also takes agents into account that output IO actions in the form of natural text – $CLICK(x, y)$ *or* $CLICK(boundingbox)$, which then can be transformed via a module to executable actions. This output category is the most common for RL-based agents (Gur et al., 2018; Humphreys et al., 2022; Jia et al., 2019; E. Liu et al., 2018; Shaw et al., 2023; Shi et al., 2017; Shvo et al., 2021; Toyama et al., 2021; Yao et al., 2022) and in recent development an increasing amount of LLM based agents utilize IO peripheral output because of its strength of being able to react flexibly to changes on the GUI (Cheng et al., 2024; X. Deng et al., 2023; Ding, 2024; H. He et al., 2024; Kil et al., 2024; Kim et al., 2023; Rawles et al., 2023; Song, Bian, et al., 2023; team rabbit research, 2023; Wen et al., 2023; Xing et al., 2024; Z. Zhang & Zhang, 2023; B. Zheng et al., 2024; Zhu et al., 2023). However, it faces the challenge of dealing with the enormous action space described in subsection 4.3.1. Therefore the IO peripheral output only becomes truly efficient when it is paired with a smart policy for dynamic action inference, as this is a prerequisite for reducing the complexity of the action space.

AndroidEnv (Toyama et al., 2021), proves that the basic action types, such as touchscreen movements can be reduced to a core set representing the entire action type space if related parameters – *coordinates* – are provided. An action contains its type and additional parameters that are necessary for execution.

$$\begin{gathered} ActionType \in \{TOUCH, LIFT, REPEAT\} \\ tap = \{LIFT, TOUCH, LIFT\} \\ swipe = \{LIFT, TOUCH, TOUCH, LIFT\} \end{gathered}$$

Such abstractions lead to smaller action spaces, which in turn leads to a reduction in overall complexity.

Some publications try to achieve good performance of an agent without reducing the action space complexity (Humphreys et al., 2022; Y. Li et al., 2020; Niu et al., 2024; Shaw et al., 2023). This is attempted because the dynamic action inference is a complex and not yet solved task itself (see chapter 6).

4.5.2 Executable Code

Another way in which executable actions can be made available by the agent is the generation of executable code. This particular category is made possible exclusively and impressively by SOTA LLMs like GPT-4 (X. Du et al., 2023) and Gemini (G. Team et al., 2023), thanks to their extensive pre-training on open-source code repositories. Most works in this area focus on generating python code (M. Chen et al., 2021; J. Liang et al., 2022; Surís et al., 2023; X. Wang, Wang, et al., 2023), which is widely used in machine learning and artificial intelligence (Thaker & Shukla, 2020). Generating code has the advantage that it can execute actions independently of tasks and domains.

H. Sun et al. (2023) demonstrate the stability improvement for solving complex tasks when executable code is not only used as output but also in the entire refinement process of an agent. However, generating executable code does not solve nor tackle the challenges of limited context-length or dynamic action inference. In addition, there is no guarantee of whether the most performing libraries or the one preferred by the user is used in the code generation.

Gur, Furuta, et al. (2023) present an approach that addresses the complexity of real-world websites with a modular concept, where Flan-U-PaLM (Chung et al., 2022) is used for synthesizing python programs, which interact directly with websites by executing selenium code to simulate browser actions. They are based on summaries and plans provided by an upstream module. This approach enables effective web automation but faces the challenge of incorporating feedback about errors in the generated code.

4.5.3 Tool Usage

Another system output involves the usage of existing tools (Anantha et al., 2023; Kong et al., 2023; Y. Liang et al., 2024; Mazumder & Riva, 2020; Patil et al., 2023; Qin, Hu, et al., 2023; Qin, Liang, et al., 2023; Ruan et al., 2023; Schick et al., 2023; Tang et al., 2023; R. Yang et al., 2023), distinguishing itself from executable code by predefining endpoints and their parameters. This enables reduced complexity through executable and verified endpoints but limits flexibility. A problem with tool usage as output is determining which tool to apply and what the required parameters are (Patil et al., 2023). This is addressed by either implementing a language model-based retrieval system or by heuristically deciding the most suitable endpoint.

Patil et al. (2023) demonstrate that current LLMs, combined with retrieval systems, can accurately determine the most fitting endpoint and its parameters. Several consecutive works (Kong et al., 2023; Y. Liang et al., 2024; Qin, Liang, et al., 2023; Ruan et al., 2023) optimize the retrieval system to maximize the accuracy of endpoint predictions. Schick et al. (2023) enable dynamic tool use by employing a self-supervised learning approach that generates tokens based on the best-fitting endpoint.

Mazumder and Riva (2020) propose generating a concept-level API to heuristically determine actions to take. This approach is much more dynamic than predefined endpoints but reintroduces the complexity of dynamic action inference.

Noteworthy is ToolAlpaca an approach by Tang et al. (2023), which fine-tunes small language models on generalized tools, resulting in smaller expert language models for the fine-tuned tool domain.

4.5.4 Discussion

The generation of executable code provides more accurate and reliable results thanks to the pre-training of LLMs, but they do not solve the problem of context length limitations. In addition, most of these approaches generate python code because it is the most popular language for machine learning (Sultonov, 2023). However, this could lead to a reduction of quality, as Buscemi (2023) has shown that julia has the best success rate when generated by ChatGPT 3.5 (openAI, 2022).

The use of tools can decrease the complexity due to the predefined endpoints and their parameters. In addition, LLMs can determine these endpoints and parameters using a retrieval system. However, this leads to a limitation of flexibility in both approaches. Even if the use of IO peripherals faces the challenge of an extremely large action space, it still represents the most human-like and, above all, the most generalizable form of system output. In correlation with the increasing importance of dynamic action inference (see subsection 4.3.1) and subtask inference (see subsection 4.3.2), it can also be shown that IO peripherals are integrated in many recent developments. This leads to the conclusion that accurate IO peripheral output can only be obtained through optimal decomposition, which could, eliminate the limitations of executable code or tool usage.

The A2C2 has now successfully navigated to all of the five islands – *Input (see section 4.1), Learning (see section 4.2), Input Decomposition (see section 4.3), Plan Refinement (see section 4.4), System Output (see section 4.5)* – and filled its backpack with a wealth of knowledge about how to successfully fulfill a user instruction and what the remaining unsolved challenges are. If it ever lacks knowledge in its task as a Natural Language-Instructed Autonomous Agent for Computer Control, it will remember the visits to the islands and retrieve the missing knowledge. In chapter 5 we present some of the most promising agents as well as our architecture of an A2C2.

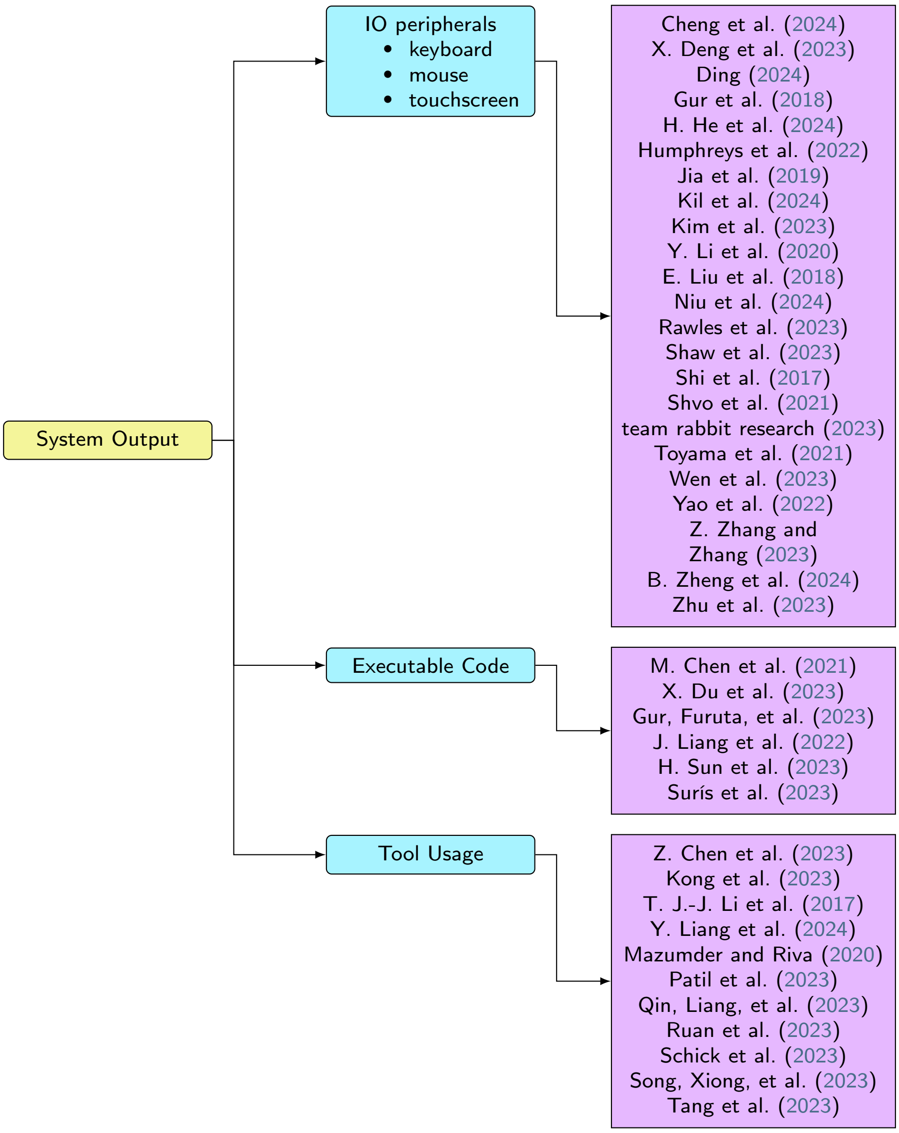


FIGURE 4.16: A compressed overview of publications and their affiliation to the system output.

Chapter 5

Identifying the Pinnacle Methods

With a wide insight into all the specialized islands an A2C2 has to navigate through, the following chapter will present the most innovative systems across all islands based on the criteria listed in section 5.1. The highlight is an own architecture of an A2C2 in section 5.3, which draws inspiration from the most promising approaches from each island and is intended to serve as a foundation for a possible practical implementation in the future.

5.1 Criteria

To identify the most promising agents, we have defined criteria guiding our selection process. It should be noted, however, that the selection of these criteria was made at our discretion following the extensive literature review and with the categorization of the taxonomy in mind. The four criteria chosen are actuality – *how up to date is the agent?*, degree of functionality – *has it been tested in production?*, uniqueness – *is the basis of the agent unique?* – and accessibility – *how available is the agent for further research?*. The methods identified and their allocation to the defined criteria are shown in Table 5.1.

TABLE 5.1: The methods identified based on the specified criteria where ✔ means fulfilled and ✘ means not fulfilled.

	actuality ↑	functionality	uniqueness	accessibility
Rabbit team rabbit research	2023-03-12	✔	✔	✘
PIX2ACT Shaw et al.	2023-06-12	✘	✘	✔
SYNAPSE L. Zheng et al.	2024-01-19	✘	✘	✔
CRADLE Tan et al.	2024-03-07	✔	✔	✔

5.2 Methods

Each of the following sections is dedicated to an agent identified based on the defined criteria (see section 5.1). We focus on illustrating how an agent navigates through the islands that are described in chapter 4 and their strengths and weaknesses.

5.2.1 Rabbit by team rabbit research (2023)

Input: The input to the Large Action Model (LAM) of rabbit are voice instructions. It is worth mentioning that, unlike known voice-controlled assistants, sound instruction can only be given by holding a physical button for security reasons. The observation space is a virtual environment hosted on the rabbit OS cloud and is passed to the agent as a pixel-based representation.

Learning: Rabbit leverages multiple ways of learning its LAM. Employing neural networks and symbolic algorithms, the neuro-symbolic model is learning by demonstration – *behavioral cloning* – and uses collected long-term memory to achieve user personalization and better interpret the intended instruction.

Input Decomposition and Plan Refinement: The authors claim that the zero-shot capability of LLMs as well as symbolic algorithms can be combined to model the structure of applications without a temporary textual representation. This model is designed to optimize the selected action through regular, minimalist, stable, and explainable improvements. Combined with numerous imitations of demonstrations, this should achieve a generalizability that can cope with unknown complex instructions.

System Output: The system output is not clearly defined, but since human imitation of demonstrations plays a major role, it can be assumed that the system works with IO peripheral output. How these actions are represented is not shown.

In summary, rabbit is a very exciting approach, as it is productively available and offers its ecosystem with a hardware device, teach mode, and very well-integrated features. The need for imitation or BC makes sense because the complexity of instructions can be reduced most efficiently by users and if an instruction can be learned through individual demonstrations, this can be an exciting way to realize an A2C2. However, the generalizability must be viewed critically, as there are currently only eight different, very specific, functionalities available – *search, music, rideshare, food, vision, generative AI, note-taking, translation*. Due to the large community and many interested parties and the available teach mode, it is questionable why this list is not growing exponentially if the neuro-symbolic model works so well.

5.2.2 PIX2ACT by Shaw et al. (2023)

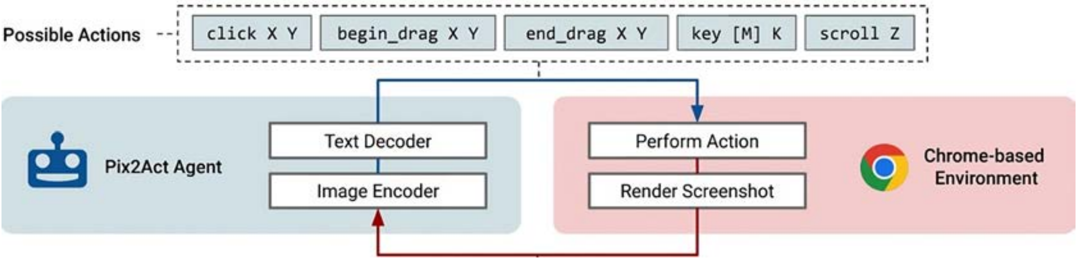


FIGURE 5.1: An overview of the PIX2ACT taken from the original publication of Shaw et al. (2023), showing the possible actions and the agents workflow.

The PIX2ACT agent is presented below using the taxonomy defined in chapter 4. An overview taken from the original publication of Shaw et al. (2023) of the PIX2ACT agent is shown in Figure 5.1.

Input: The input of the PIX2ACT agent consists of the pixel-based observation of the current state in the form of a screenshot, scaled to extract the maximum number of fixed-size patches within the sequence length limit, as well as the instruction in form of natural language.

Learning: Through a combination of human demonstrations and an adaptation of Monte Carlo tree search (Świechowski et al., 2021) integrated with a neural network to estimate state values, new expert trajectories can be generated for training. The value function uses the PIX2STRUCT (K. Lee et al., 2023) architecture, predicting state values instead of actions. Successful episodes from the tree search policy are used to iteratively improve the model through standard supervised learning.

Input Decomposition: PIX2ACT uses the PIX2STRUCT model based on image transformer encoders and text transformer decoders, pre-trained to parse screenshots into possible actions.

Plan Refinement: Due to the RL nature of the agent, the plan refinement is made by a next action prediction. This is a black box and strongly depends on training data and strategy.

System Output: IO peripheral actions are encoded as text tokens and predicted autoregressively by the Transformer decoder. Beam search is used over tokens to output the k-best actions, producing a set of top-k actions for a given state with their approximate probabilities. These probabilities can be adjusted by a length normalization factor.

In summary, PIX2ACT works very well on simpler benchmarks such as MiniWob++ (Shi et al., 2017) and WebShop (Yao et al., 2022) and outperformed at the time of publication all other approaches that only work with pixel-based observations as input. Compared to human solving of these benchmarks and agents that have the DOM available, PIX2ACT performs on par. It needs to be tested on more difficult benchmarks or online to evaluate real-world capabilities.

5.2.3 SYNAPSE by L. Zheng et al. (2023)

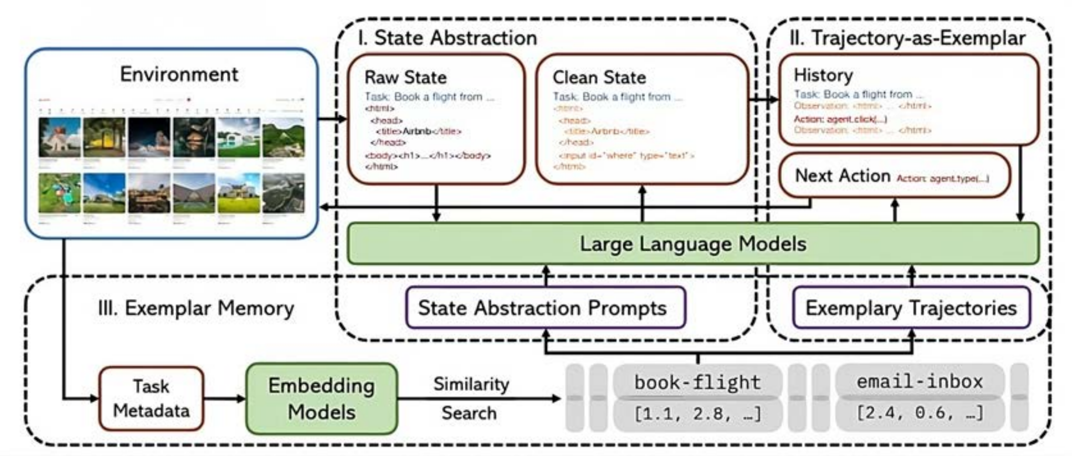


FIGURE 5.2: An overview of the SYNAPSE framework taken from the original publication of L. Zheng et al. (2023) consists of three key component: state abstraction, trajectory-as-exemplar(TaE) prompting, and exemplar memory.

With SYNAPSE, L. Zheng et al. (2023) present an agent that builds upon three key components *state abstraction*, *trajectory-as-exemplar (TaE) prompting* and *exemplar memory* to solve computer control tasks (see Figure 5.2). Nevertheless, SYNAPSE can be presented according to our taxonomy.

Input: SYNAPSE receives as input the state of the computer in textual representation in the form of HTML.

Learning: The agent utilizes the few-shot learning capabilities of LLMs to extract relevant information from the raw states and to obtain a cleaned observation for subsequent action generation.

Input Decomposition: To reduce the length of the individual states, SYNAPSE applies explicit and implicit abstractions depending on the complexity of the states. In explicit abstraction scenarios, state-observation pairs are passed to the LLM as few-shot exemplars together with the raw state to obtain a cleaned observation. While for implicit abstraction scenarios, task descriptions and state analysis codes are used as few-shot exemplars. These exemplars are combined with the current task, whereupon code is generated that also returns a cleaned-up observation.

Plan Selection: SYNAPSE uses trajectory-as-example (TaE) prompting in the plan refinement phase. Complete trajectories are used to prompt the LLM to generate actions, formatted as $<task, observation, action, ..., observation, action>$. The LLM is fed with successful trajectories, followed by the current one to generate the next action. This process is repeated until the task is completed or the maximum number of steps is reached. To find relevant trajectories as few-shot exemplars, embedding models and similarity searches in the vector database are used.

System Output: The actions generated from the plan refinement phase are executed directly. Depending on the benchmark, the actions can be natural text or pseudocode – $agent.type(ls)$, $agent.press(enter)$.

In summary, the three-part approach of SYNAPSE addresses current problems of computer agents namely *the limited context length of LLMs, the unexplored exemplar structure,* and *task-specific exemplars*, successfully solving a more challenging book-flight task of MiniWoB++ (Shi et al., 2017) and also achieving a 56% improvement over SOTA in-context learning methods in Mind2Web (X. Deng et al., 2023). It should be noted that SYNAPSE currently works mainly on text. A promising approach for the future is the integration of multimodal understanding to successfully solve more complex tasks and improve generalizability.

5.2.4 CRADLE by Tan et al. (2024)

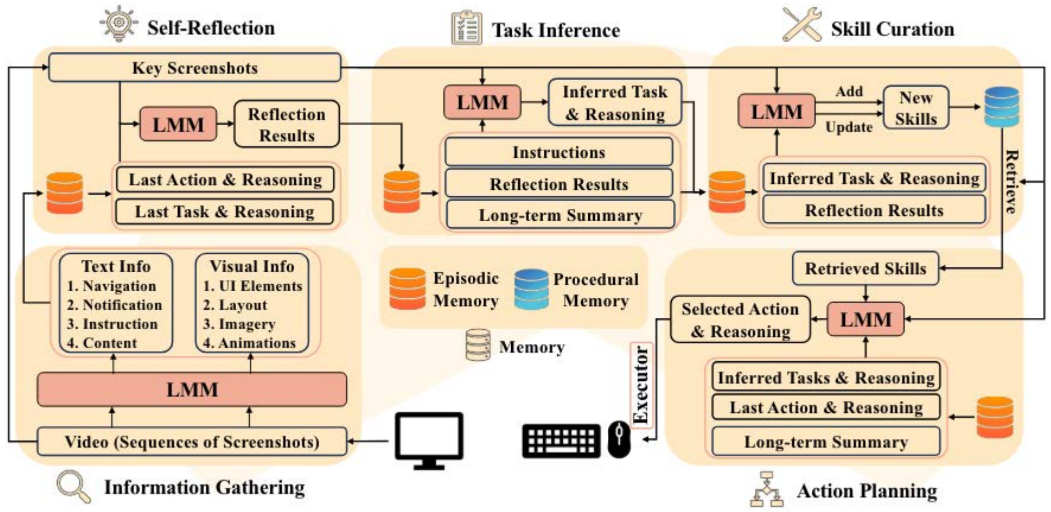


FIGURE 5.3: An overview of the Cradle framework taken from the original publication of Tan et al. (2024). Cradle takes video from the computer screen as input and outputs computer keyboard and mouse control determined through inner reasoning.

Input: The input for the CRADLE agent is the video sequence recorded since the last action, which contains both visual information and the instruction.

Learning: The learning mechanism is called skill curation and consists of a process that extracts relevant and successful skills from the episodic – *experiences including screenshots, LMM outputs* – memory and stores them in a procedural – *long-term, structured* – memory so that they can be reused. These procedural memory entries can also be updated if necessary and consist of semantic code functions for IO control.

Input Decomposition: Decomposing the input is done by the task inference component which utilizes a long-term summary of past experiences, the last reflection results, and the input to prompt an LMM on what next actions should be done.

Plan Refinement: The validation and decision as to which action was successful is carried out by an LMM. If errors occur during the action, new knowledge is gained via a self-reflection module, which is incorporated into the next task inference step to create a closed-loop system that can correct errors autonomously.

System output: As system output, semantic code functions that encapsulate IO instructions – $move\_forward(duration) : key\_hold('W', duration)$ – are output, which in turn can be executed by the agent to control IO peripherals.

In summary, CRADLE is closely aligned with the archipelago introduced in chapter 4. Successful experiments on a functionally complex video game demonstrate great potential for the application of computer control. The major challenge lies in seamlessly integrating the numerous individual components of the CRADLE framework and ensuring they harmonize with each other. Figure 5.3, taken from the original publication by Tan et al. (2024), shows an overview of this framework. After the integration of computer control, it would be very insightful to compare CRADLE with other mentioned agents.

5.3 Our A2C2

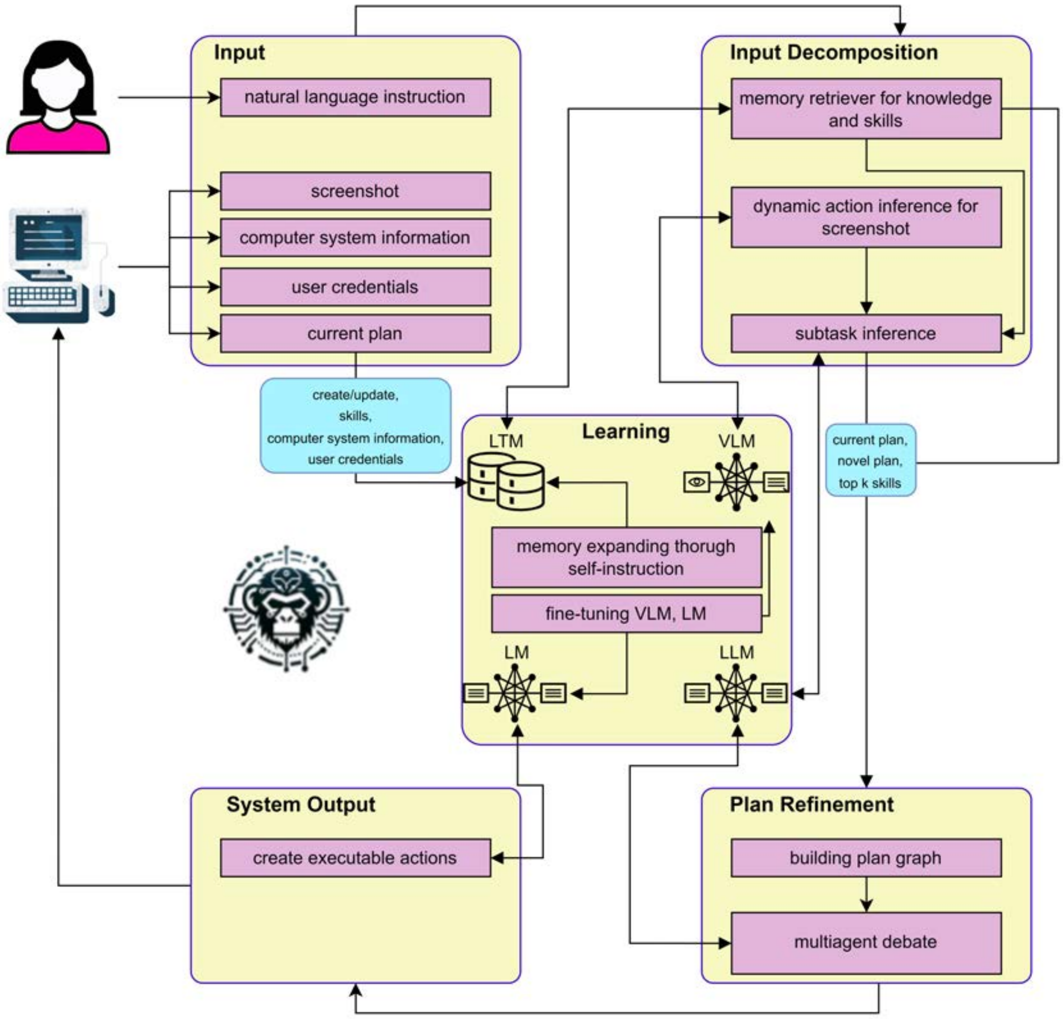


FIGURE 5.4: System overview over our A2C2 implementation where LTM stands for long-term memory, LM is the specialized language model, LLM is the orchestrator, and VLM the model for dynamic action inference. Icons for the computer system, LTM, LM, and VLM are generated by Ideogram (2023).

After presenting the pinnacle methods in the previous section 5.2, this section is dedicated to our architecture of an A2C2. The objective of this section is to exploit the strengths of existing methods in combination with our own ideas and present a solid agent specification to facilitate an upcoming technical implementation. To stay on the course of our archipelago, we structure our approach using the same taxonomy from chapter 4, so that comparisons between A2C2s are straightforward. Figure 5.4 shows a system overview of our A2C2 implementation, which is described in detail below.

Input: From the knowledge acquired in subsection 4.1.1, it is clear that the user instruction is given as natural language. Our agent receives a screenshot of the current observation to obtain a GUI representation. Additionally, the agent is provided with computer system information – *OS information*, *list of installed apps*, *currently running apps*, *default apps* – to simplify reasoning and planning and personalize the user experience. To avoid reloading this information for the same user each time, the agent is provided with user credentials, enabling user management.

Learning: The insights in section 4.2 helped us to realize that a smaller fine-tuned model can work better on specialized tasks than a large foundation model. In addition, access to past experiences is very important to align repetitive tasks and optimize the performance of an agent. These findings have led to the integration of the following components:

- A VLM, trained for dynamic action inference so that it has grounding capabilities. It receives the screenshot as input and generates a textual representation similar to the example in section 2.5.

- A language model, fine-tuned to generate actions from the current observation state and an instruction.

- Two long-term memories should store existing information about a user (Z. Wu et al., 2024) and past successful experiences Tan et al. (2024) (henceforth skills).

All components should be capable of being iteratively extended/fine-tuned so that the agent can improve naturally. A further learning capability consists of extracting information from the internet, allowing the agent to learn skills through self-instruction in the background. This self-instruction works by exploring applications and generating instructions with known ground truths from the gathered knowledge.

Input Decomposition: The importance of input decomposition in the context of an A2C2 was demonstrated in section 4.3. The decomposition module in our A2C2 needs to break complex user instruction into simpler subtasks. Firstly the VLM is tasked to transform the screenshot input into a structured and accurately grounded representation (Baechler et al., 2024). This description in combination with user and computer system information and the most similar skills retrieved from the long-term memory are fed into an orchestrator LLM, that decomposes the instruction into as small as possible subtasks.

Plan Refinement: Our plan refinement (see section 4.4) component receives several possible plans from the input decomposition. In a multitask instruction, one of the plans is always the current plan being executed. Furthermore, the best plans loaded from memory and a novel plan with the current information created by the specialized language model are provided. Each plan has several subtasks, which are, inspired by GoT (Besta et al., 2023), tracked in a knowledge graph. If a subtask appears in multiple plans or is a child of a successfully executed subtask, it is given more weight. Once this graph has been created, subtasks that are on the same level should be compared and checked for validity, criticality, and completeness using a debating system inspired by MAD (Y. Du et al., 2023), where several agents debate which one is the most suitable and a decision is made with the help of the orchestrator as a judge. Finally, the subtasks that are picked by the orchestrator are added to the current plan and passed on to the system output. If a subtask is critical or invalid, the subtask is set with a flag that expects user feedback. As no subtask defines the end of an instruction, the judge should also verify whether the objective has been reached or a termination criterion has occurred. If a subtask is successfully executed it is saved as a new skill in the corresponding memory.

System Output: To keep the system output as generalizable as the rest of our agent, we were reassured to execute IO actions (see subsection 4.5.1). Therefore, the subtasks are mapped with the help of the specialized language model to corresponding actions. An action consists of the action type – *CLICK*, *TYPE*, *READ*, *SCROLL*, a representation of the element – *bounding box*, *xy coordinates*, *pixel-representation*, which can be obtained from the dynamic action inference, and action type specific parameters – *number of clicks*, *text*, *wheel rotations*, which are used to define the action more precisely.

In summary, our A2C2 should combine the analyzed strengths into a smart ecosystem. To measure the comprehensibility and maturity of the A2C2, the components are presented independently. In a second version, all components could be combined in a deep neural network, but this would make it more difficult to monitor and test the individual components. In the course of the bachelor thesis, a simplified prototype of our A2C2 was developed to draw additional technical conclusions and outlooks (see https://github.com/Yingrjimsch/a2c2).

Chapter 6

Conclusions

This chapter summarizes our findings on the field of Natural Language-Instructed Autonomous Agents for Computer Control providing an outlook for further considerations and research topics in this field.

6.1 Summary

Our objective was to provide a survey of the research landscape of A2C2s by categorizing existing agents and outlining their strengths and limitations.

There are numerous ways to categorize the research landscape of A2C2s. The challenge lies in ensuring that as many agents as possible are represented by the categorization while keeping it easy to understand and sequentially readable. We decided to structure our categories based on human task-solving strategies – *how would Hila go about solving the task?*, described with the analogy of an archipelago. The advantage of this categorization is the reduction of complexity through a familiar pattern for readers, resulting in a map that enables them to explore the islands and select the most suitable findings for their requirements. Other categorization options include domain-specific categories – *web*, *mobile*, *desktop*, model-specific categories – *RL*, *LLM*, *VLM*, skill-specific categories – *GUI interpretation*, *reasoning*, *simplifying instructions* – and more. Each approach provides unique insights and perspectives into the field of A2C2s. However, they do not fulfill both mentioned criteria.

Another objective was to come up with a new architecture that integrates the identified strengths and our acquired ideas. Through this architecture and a simplified prototype, we have gained insights into the current strengths and limitations of the different islands of an A2C2. To demonstrate them in a comprehensible way, the example described in section 2.5 – *reporting work hours and activities on ProTime with information gathered from the calendar and GitHub* – is reused and the following sections are structured identically to the islands introduced in chapter 4.

6.1.1 Input

After receiving an instruction, first an A2C2 has to retrieve the observation space. Therefore, it takes screenshots of the computer system and passes them to the next component, together with optional information. This works very well, thanks to increasingly open operating systems. The developed prototype has shown that the information about installed and open applications, needs to be filtered to remove those that cannot be operated by the user.

6.1.2 Learning

The agent needs to know how to manage and retrieve long-term memory such as the most similar skills to an instruction, and information about ProTime and Hila. In addition, the agent should periodically fine-tune its specialized models to improve their native quality.

Strenghts

Vector search works great for retrieving data from big knowledge bases like user manuals. It is even easier to retrieve data from structured and self-managed data like past experiences, as this data can be supplemented with metadata that simplifies selection.

Fine-tuning is widely used and almost every newly released model includes a corresponding guide. Additionally, most models can be fine-tuned with a reasonable amount of computer resources and examples. With the establishment of machine learning operation systems the automation for periodical fine-tuning and quality monitoring becomes less sophisticated.

Limitations

RL is not suitable for A2C2 as a very large amount of training data is needed to learn instructions. If the required data is available, the agent can operate learned domains or applications well but still fails to generalize to other domains or applications.

Datasets in the research area of A2C2 are mostly mobile or web-focused. This leads to a deficiency in data for desktop applications. Most datasets and benchmarks contain simple instructions and few representative instructions on how a user interacts with a computer system (see section 6.2.1).

VLMs suffer from the same limitation regarding datasets. However, this limitation will soon be solved by an increasing number of A2C2 approaches and open-source VLMs.

Hallucination – *the generation of logical-sounding but factually incorrect or useless responses* – is a limitation that will always accompany generative AI systems. While it can be reduced and controlled to some extent, it will not disappear completely (see section 6.2.3).

6.1.3 Input Decomposition

During input decomposition, the agent must be able to analyze a screenshot, to identify and localize the individual elements on the desktop of Hila. In addition, it must be able to break down the provided instruction into manageable subtasks.

Strenghts

Screen description can already be achieved by the capabilities of SOTA VLMs. This description is limited to *what is visible on the screen* and lacks the knowledge of *where an element is visible on the screen*.

Instruction decomposition works very well thanks to rapid improvements of LLMs, enabling natural decomposition capabilities. The three applications *GitHub, Calendar,* and *Protime* can be separated correctly and used for individual further decomposition, which is currently very important to assess the quality of subtasks, ensure explainability, and identify critical actions. Insights gained from the prototype show, that generating a multitask plan yields better results than predict the next subtask.

Limitations

Dynamic action inference remains a major challenge for the agent. Specifically, the exact localization of elements on the screen works poorly. Although LMMs can recognize elements partially correctly, an increasing complexity of the screen decreases said ability. This limitation could be optimized by either providing a segmentation in advance or by using expert VLMs like ScreenAI (Baechler et al., 2024) in combination with a large amount of data (see section 6.2.2).

Interpretation of GUI elements – *what is the purpose of this element?* – is a remaining limitation. Although SOTA LMMs can identify elements on a screen, they struggle to fully understand how they are related to each other and what they trigger. This is due to the great diversity in appearance and ways of labeling. Eliminating this limitation proves difficult as it would require the unification of the elements. We also noticed that when using multiple screens, there is a duplication that needs to be taken into account.

Handling context becomes a limitation as soon as LLMs are provided with more extensive knowledge. If an instruction requires multiple knowledge sources and a experience history the maximum context length is reached quickly. This limitation can be stretched with external memory, fine-tuning, or in-context learning. However, not only the context size but also the ability to interpret a large amount of knowledge simultaneously and draw the expected conclusions is challenging for current agents. Research such as Mamba (Gu & Dao, 2023) and Infini-attention (Munkhdalai et al., 2024) attempt to solve the context length limitation and show promising results. However, evaluating whether these approaches can extract information in a large amount of relevant context is difficult.

6.1.4 Plan Refinement

During the plan refinement component, an agent needs to compare and interpret the plans provided by the input decomposition and reduce them to a final plan. It must therefore understand whether the subtasks are ready to be executed.

Strenghts

Reasoning capabilities of SOTA LLMs are very advanced (Kambhampati, 2024). It does not matter whether open-/closed-loop or multiagent reasoning, all these methods work in the context of computer control. As described in section 4.4, a closed-loop reasoning approach is necessary for multistep reasoning and can be improved through multiple agents.

Limitations

Performance is one of the biggest limitations not only in plan refinement but for A2C2s in general. For users to be able to work productively with such an agent, the response time must be reduced to a minimum. Multiple sources show that a closed-loop system must respond in under 300ms. If constant informative feedback is given, the user can wait up to a maximum of 10 seconds without disengaging from the agent (Doherty & Sorenson, 2015; educative, 2024; Nielson, 1993). These limits apply to one subtask execution, nevertheless, tests on the prototype and analytics (artificialanalysis, 2024) have shown that the throughput of an LLM API call is not yet very far, and if a multistep reasoning calls several endpoints, the response time adds up very fast.

Nesting depth of subtasks is another limitation in plan refinement. The uncertainty of how many subtasks an instruction has and whether these subtasks correspond to actions or whether they consist of additional subtasks makes it difficult to reason about the status of single subtasks and the end of an instruction. The implemented prototype has shown, that if subtasks are defined as actions it is harder for LLMs to reason and compare them because they lose the semantical meaning of a natural language subtask.

6.1.5 System Output

The plan defined in the plan selection phase gets executed in the system output component. Therefore, the agent must possess the skill to translate subtasks into executable actions.

Strenghts

IO peripheral control is possible on almost all computer systems. In concrete terms, this means that the GUI elements can be controlled directly through the agent using integrated libraries such as *pyautogui*.

Definition of action types and required parameters – *coordinates, number of clicks, text* – can be defined manually, mitigating the complexity of learning how IO peripherals work – *CLICK*, *TYPE*, *SCROLL*.

Limitations

Grounding is a complex challenge for existing agents based on dynamic action inference, and restricts the quality of action execution. The execution of the correct element can only be achieved if the agent is given the valid positions of these elements. Therefore, this limitation can be solved only by addressing the problem in dynamic action inference.

Screen interpretation after execution – *the determination by the agent when a previous execution has been completed* – proves to be a challenge in the area of system output. Possible solutions are the repeated creation of screenshots to determine the current observation space or the insertion of a fixed time buffer that waits for the execution of actions. However, both approaches harbor the risk of being cost-intensive or harming performance.

6.2 Outlook

Finally, this last chapter provides an outlook to show the potential of exciting research questions derived from the thesis and the identified limitations. This section is categorized into **low hanging fruits** – *achievable by the end of 2024*, **intermediate goals** – *achievable in the next three years*, and **long-term visions** – *not solvable with current technology*. All predictions were based on our current knowledge of the development of the area.

6.2.1 Low Hanging Fruits

In this section, we present potentials that have not yet been sufficiently researched or that can easily be further experimented with.

Security

Even if all the limitations of an A2C2 mentioned in section 6.1 have been overcome, one of the most important aspects is still underrepresented in current literature. The security and privacy of user data are decisive factors towards a productive A2C2. As mentioned in section 5.3, the observation space of an agent should be pixel-based. However, a user does not want to constantly stream their desktop view to an external system without any guarantee of privacy, which leads to the need for either local processing or encryption of the transmitted data. Furthermore, it is elementary that critical tasks – *purchase, password entry* – can only be performed after user confirmation. This raises the challenge of how to detect such critical actions. Ultimately, inadequate detection of critical actions as well as prompt injection carry the risk of potential exploitation by an intruder, resulting in loss of sensitive data. It is, therefore, crucial to consider these aspects in future development steps by proactively asking for user feedback if necessary, and the underlying generative models.

Personalization

Turning an A2C2 into a personalized agent fully tailored to the needs of a user requires not only success rates when executing instructions but also a personalized way of working. Such personalization can enhance performance, quality, and user experience. Options like *saving favorite applications, customizing workflows,* or *adapting to the language pattern of a user* can significantly improve the user experience. Personalization features such as *storing computer system specifications, tracking installed applications,* or *maintaining a detailed history of all instructions* can boost performance by providing quicker access to necessary information and the possibility of preloading relevant knowledge from external data sources. They come with the sacrifice of anonymity and the critical question of data ownership.

Datacollection pipeline

A very important topic, which is not dealt with extensively in this thesis, is how an agent obtains its training data. This topic is very large due to many completely different and great approaches. To work out the best way to realize such a data collection pipeline, it would be interesting not only to know existing benchmarks and datasets (see Appendix A) but also to get a detailed summary of the different approaches. A follow-up survey or an extended version of this thesis could be useful to gather this information. In our opinion, a great way to build such a pipeline contains a scraper, which interacts with different applications, annotates elements if possible – *in the web and mobile domain*, and processes the data so it can be used for fine-tuning. In addition, an LLM can instruct an agent with tasks, triggering it to execute instructions and at the same time learn new skills and knowledge about specific domains. This should be done on a virtual machine encapsulated by a user to not interfere with their workflow. Another approach could be to leverage human imitation (team rabbit research, 2023) and utterance masking (W. Li et al., 2024) to stay human-aligned but still achieve generalization of action parameters.

6.2.2 Intermediate Goals

This section describes the limitations of the current state of A2C2 research, which will be overcome in the next three years.

Performance

As described in the previous section 6.2.1, an A2C2 must be capable of performing critical actions such as processing screenshots or storing credentials locally or in encrypted form. In addition, at least one model is available for inference. These components of require substantial computing time and must therefore be optimized to be viable. This speed-up is an important area of research for A2C2s. Internal experiments with the prototype featuring multistep reasoning and multi-agent debate have shown, that a single instruction can quickly accumulate over ten inference calls, thereby extending the time until execution and resulting in an unsatisfactory user experience. One optimization can be achieved by minimizing these calls whilst still successfully executing instructions. Another performance optimization could involve staying in the embedding space during the reasoning and refining to bypass decoding efforts, coming with the risk of interpretation losses.

Grounding

As detailed in subsection 4.3.1, grounding pixel-based representations is a major unsolved limitation. This can be tackled by upscaling current LMMs to the size of LLMs, equipped with large amounts of grounding-specific data. This data can be merged from existing datasets and enhanced by creating a data collection pipeline (see section 6.2.1). With the fast-paced advancement of LLMs in mind, this limitation will most likely be solved as soon as big tech companies are seriously trying to achieve A2C2. The only hurdle is that the appearance of GUIs is continuously changing with the current preferences of society, necessitating iterative retraining.

6.2.3 Long-Term Visions

Long-term visions refer to limitations, which, in our opinion, are not solvable with the current technology. Therefore it is not possible to estimate a date for solving these problems, nonetheless, these problems can be heavily researched.

Hallucination reduction

As described in section 6.1.2, there is always a risk of hallucinations in generative AI systems. Eliminating this proves impossible with deep neural networks, as a certain degree of unpredictability always remains in such black-box systems.

Chapter 7

Indeces

7.1 References

All41Group. (2024). Zeiterfassung mit SAP leicht gemacht. [Online]. Available: https://www.all-for-one.ch/de/themen-impulse/hr-employee-experience/protime-zeiterfassung-mit-sap/.

Anantha, R., Bandyopadhyay, B., Kashi, A., Mahinder, S., Hill, A. W., & Chappidi, S. (2023). ProTIP: Progressive Tool Retrieval Improves Planning arxiv:2312.10332.

Andreas, J., Klein, D., & Levine, S. (2016). Modular Multitask Reinforcement Learning with Policy Sketches. *International Conference on Machine Learning*, *abs/1611.01796*.

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. (2017). Hindsight Experience Replay. *Neural Information Processing Systems*, 5048–5058.

artificialanalysis. (2024). LLM Leaderboard - Compare GPT-4o, Llama 3, Mistral, Gemini & other models | Artificial Analysis. [Online]. Available: https://artificialanalysis.ai/leaderboards/models.

arXiv. (2024). arXiv.org e-Print archive. [Online]. Available: https://arxiv.org/.

Baechler, G., Sunkara, S., Wang, M., Zubach, F., Mansoor, H., Etter, V., Cărbune, V., Lin, J., Chen, J., & Sharma, A. (2024). ScreenAI: A Vision-Language Model for UI and Infographics Understanding arxiv:2402.04615.

Bai, C., Zang, X., Xu, Y., Sunkara, S., Rastogi, A., Chen, J., & Arcas, B. A. Y. (2021). UIBert: Learning Generic Multimodal Representations for UI Understanding. *International Joint Conference on Artificial Intelligence*, 1705–1712. doi:10.24963/ijcai.2021/235.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., … Kaplan, J. (2022). Constitutional AI: Harmlessness from AI Feedback arxiv:2212.08073.

Bavishi, R., Elsen, E., Hawthorne, C., Nye, M., Odena, A., Somani, A., & Taşırlar, S. (2023). Introducing our Multimodal Models. [Online]. Available: https://www.adept.ai/blog/fuyu-8b.

Bellman, R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, *6*(5), 679–684.

Besta, M., Blach, N., Kubíček, A., Gerstenberger, R., Gianinazzi, L., Gajda, J., Lehmann, T., Podstawski, M., Niewiadomski, H., Nyczyk, P., & Hoefler, T. (2023). Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *AAAI*

Conference on Artificial Intelligence, 17682–17690. doi:10.1609/aaai.v38i16.29720.

Burns, A., Arsan, D., Agrawal, S., Kumar, R., Saenko, K., & Plummer, B. A. (2022). A Dataset for Interactive Vision-Language Navigation with Unknown Command Feasibility. *European Conference on Computer Vision*, 312–328. doi:10.1007/978-3-031-20074-8_18.

Buscemi, A. (2023). A Comparative Study of Code Generation using ChatGPT 3.5 across 10 Programming Languages arxiv:2308.04477.

Cai, D., Wang, Y., Liu, L., & Shi, S. (2022). Recent Advances in Retrieval-Augmented Text Generation. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 3417–3419. doi:10.1145/3477495.3532682.

Chadi, M.-A., & Mousannif, H. (2023). Understanding Reinforcement Learning Algorithms: The Progress from Basic Q-learning to Proximal Policy Optimization arxiv:2304.00026.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., & Mordatch, I. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling. *NeurIPS*, 15084–15097.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating Large Language Models Trained on Code arxiv:2107.03374.

Chen, P.-L., & Chang, C.-S. (2023). InterAct: Exploring the Potentials of ChatGPT as a Cooperative Agent arxiv:2308.01552.

Chen, Q., Pitawela, D., Zhao, C., Zhou, G., Chen, H.-T., & Wu, Q. (2024). WebVLN: Vision-and-Language Navigation on Websites. *AAAI*, 1165–1173. doi:10.1609/aaai.v38i2.27878.

Chen, X., Wang, X., Changpinyo, S., Piergiovanni, A. J., Padlewski, P., Salz, D., Goodman, S., Grycner, A., Mustafa, B., Beyer, L., Kolesnikov, A., Puigcerver, J., Ding, N., Rong, K., Akbari, H., Mishra, G., Xue, L., Thapliyal, A., Bradbury, J., … Soricut, R. (2023). PaLI: A Jointly-Scaled Multilingual Language-Image Model arxiv:2209.06794.

Chen, Z., Zhou, K., Zhang, B., Gong, Z., Zhao, X., & Wen, J.-R. (2023). ChatCoT: Tool-Augmented Chain-of-Thought Reasoning on Chat-based Large Language Models. *EMNLP*, 14777–14790. doi:10.18653/v1/2023.findings-emnlp.985.

Cheng, K., Sun, Q., Chu, Y., Xu, F., Li, Y., Zhang, J., & Wu, Z. (2024). SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents arxiv:2401.10935.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep Reinforcement Learning from Human Preferences. *Neural Information Processing Systems*, *abs/1706.03741*.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., … Wei, J. (2022). Scaling Instruction-Finetuned Language Models arxiv:2210.11416.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). Training Verifiers to Solve Math Word Problems arxiv:2110.14168.

Conde, M. V., & Turgutlu, K. (2021). CLIP-Art: Contrastive Pre-training for Fine-Grained Art Classification. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 3951–3955. doi:10.1109/cvprw53098.2021.00444.

Daley, S. (2024). 54 Artificial Intelligence Examples Shaking Up Business Across Industries. [Online]. Available: https://builtin.com/artificial-intelligence/examples-ai-in-industry.

Daniel, C., Neumann, G., Kroemer, O., & Peters, J. (2016). Hierarchical Relative Entropy Policy Search. *Journal of Machine Learning Research*, *17*(93), 1–50.

Davidson, D. (2001). The Logical Form of Action Sentences. In *Essays on Actions and Events*. Oxford University Press. doi:10.1093/0199246270.003.0006.

De Angelis, L., Baglivo, F., Arzilli, G., Privitera, G. P., Ferragina, P., Tozzi, A. E., & Rizzo, C. (2023). ChatGPT and the rise of large language models: The new AI-driven infodemic threat in public health. *Frontiers in Public Health*, *11*, 1166120. doi:10.3389/fpubh.2023.1166120.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., & Su, Y. (2023). Mind2Web: Towards a Generalist Agent for the Web. *NeurIPS*.

Deng, Y., Zhang, X., Zhang, W., Yuan, Y., Ng, S.-K., & Chua, T.-S. (2024). On the Multi-turn Instruction Following for Conversational Web Agents arxiv:2402.15057.

Ding, T. (2024). MobileAgent: Enhancing mobile control via human-machine interaction and SOP integration arxiv:2401.04124.

Doherty, R. A., & Sorenson, P. (2015). Keeping Users in the Flow: Mapping System Responsiveness with User Experience. *Procedia Manufacturing*, *3*, 4384–4391. doi:10.1016/j.promfg.2015.07.436.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations*, *abs/2010.11929*.

Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., Del Verme, M., Marty, T., Boisvert, L., Thakkar, M., Cappart, Q., Vazquez, D., Chapados, N., & Lacoste, A. (2024). WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks? arxiv:2403.07718.

Du, X., Liu, M., Wang, K., Wang, H., Liu, J., Chen, Y., Feng, J., Sha, C., Peng, X., & Lou, Y. (2023). ClassEval: A Manually-Crafted Benchmark for Evaluating LLMs on Class-level Code Generation arxiv:2308.01861.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., & Mordatch, I. (2023). Improving Factuality and Reasoning in Language Models through Multiagent Debate arxiv:2305.14325.

Du, Y., Li, C., Guo, R., Yin, X., Liu, W., Zhou, J., Bai, Y., Yu, Z., Yang, Y., Dang, Q., & Wang, H. (2020). PP-OCR: A Practical Ultra Lightweight OCR System arxiv:2009.09941.

educative. (2024). What are response times in UI? [Online]. Available: https://www.educative.io/answers/what-are-response-times-in-ui.

Furuta, H., Lee, K.-H., Nachum, O., Matsuo, Y., Faust, A., Gu, S. S., & Gur, I. (2024). Multimodal Web Navigation with Instruction-Finetuned Foundation Models arxiv:2305.11854.

Gadgil, S., Xin, Y., & Xu, C. (2020). Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning. *2020 SoutheastCon*. doi:10.1109/southeastcon44009.2020.9368267.

Ganguli, D., Askell, A., Schiefer, N., Liao, T. I., Lukošiūtė, K., Chen, A., Goldie, A., Mirhoseini, A., Olsson, C., Hernandez, D., Drain, D., Li, D., Tran-Johnson, E., Perez, E., Kernion, J., Kerr, J., Mueller, J., Landau, J., Ndousse, K., . . . Kaplan, J. (2023). The Capacity for Moral Self-Correction in Large Language Models arxiv:2302.07459.

Gao, D., Ji, L., Bai, Z., Ouyang, M., Li, P., Mao, D., Wu, Q., Zhang, W., Wang, P., Guo, X., Wang, H., Zhou, L., & Shou, M. Z. (2024). ASSISTGUI: Task-Oriented Desktop Graphical User Interface Automation arxiv:2312.13108.

Gao, S., Dwivedi-Yu, J., Yu, P., Tan, X. E., Pasunuru, R., Golovneva, O., Sinha, K., Celikyilmaz, A., Bosselut, A., & Wang, T. (2024). Efficient Tool Use with Chain-of-Abstraction Reasoning arxiv:2401.17464.

Garza-Herrera, R. (2024). Humans use tools: From handcrafted tools to artificial intelligence. *Journal of Vascular Surgery. Venous and Lymphatic Disorders*, *12*(2), 101705. doi:10.1016/j.jvsv.2023.101705.

Gehman, S., Gururangan, S., Sap, M., Choi, Y., & Smith, N. A. (2020). RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models. *Findings*, 3356–3369. doi:10.18653/v1/2020.findings-emnlp.301.

Google. (2024). Google Scholar. [Online]. Available: https://scholar.google.com/.

Gregersen, E. (2019). History of Technology Timeline. [Online]. Available: https://www.britannica.com/story/history-of-technology-timeline.

Grigorescu, S., Trasnea, B., Cocias, T. T., & Macesanu, G. (2019). A survey of deep learning techniques for autonomous driving. *J. Field Robotics*, *37*, 362–386. doi:10.1002/rob.21918.

Gu, A., & Dao, T. (2023). Mamba: Linear-Time Sequence Modeling with Selective State Spaces arxiv:2312.00752.

Gur, I., Jaques, N., Miao, Y., Choi, J., Tiwari, M., Lee, H., & Faust, A. (2022). Environment Generation for Zero-Shot Compositional Reinforcement Learning. *Neural Information Processing Systems*, 4157–4169.

Gur, I., Nachum, O., Miao, Y., Safdari, M., Huang, A., Chowdhery, A., Narang, S., Fiedel, N., & Faust, A. (2023). Understanding HTML with Large Language Models. *EMNLP*, 2803–2821. doi:10.18653/v1/2023.findings-emnlp.185.

Gur, I., Rückert, U., Faust, A., & Hakkani-Tür, D. Z. (2018). Learning to Navigate the Web. *International Conference on Learning Representations*, *abs/1812.09195*.

Gur, I., Furuta, H., Huang, A., Safdari, M., Matsuo, Y., Eck, D., & Faust, A. (2023). A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis arxiv:2307.12856.

Gur, I., Jaques, N., Malta, K., Tiwari, M., Lee, H., & Faust, A. (2021). Adversarial Environment Generation for Learning to Navigate the Web arxiv:2103.01991.

Haenlein, M., & Kaplan, A. M. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California Management Review*, *61*, 14–5. [Online]. Available: https://api.semanticscholar.org/CorpusID:199866730.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Z., Zhang, Y., & Tao, D. (2020). A Survey on Vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PP*, 1. doi:10.1109/tpami.2022.3152247.

Hasselt, H. V., Guez, A., & Silver, D. (2015). Deep Reinforcement Learning with Double Q-Learning. *AAAI Conference on Artificial Intelligence*, 2094–2100. doi:10.1609/aaai.v30i1.10295.

He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., & Yu, D. (2024). WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models arxiv:2401.13919.

He, Z., Sunkara, S., Zang, X., Xu, Y., Liu, L., Wichers, N., Schubiner, G., Lee, R. B., & Chen, J. (2020). ActionBert: Leveraging User Actions for Semantic Understanding of User Interfaces. *AAAI Conference on Artificial Intelligence*, 5931–5938. doi:10.1609/aaai.v35i7.16741.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2020). Measuring Massive Multitask Language Understanding. *International Conference on Learning Representations*, *abs/2009.03300*.

Hinz, A. M., Klavžar, S., Milutinović, U., & Petr, C. (2013). *The tower of Hanoi-Myths and maths*. Springer.

Hong, W., Wang, W., Lv, Q., Xu, J., Yu, W., Ji, J., Wang, Y., Wang, Z., Zhang, Y., Li, J., Xu, B., Dong, Y., Ding, M., & Tang, J. (2023). CogAgent: A Visual Language Model for GUI Agents arxiv:2312.08914.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F. D., & Abbeel, P. (2016). VIME: Variational Information Maximizing Exploration. *Neural Information Processing Systems*, *29*, 1109–1117.

Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. doi:10.18653/v1/p18-1031.

Hu, C., Fu, J., Du, C., Luo, S., Zhao, J., & Zhao, H. (2023). ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory arxiv:2306.03901.

Huang, S., Zhong, W., Lu, J., Zhu, Q., Gao, J., Liu, W., Hou, Y., Zeng, X., Wang, Y., Shang, L., Jiang, X., Xu, R., & Liu, Q. (2024). Planning, Creation, Usage: Benchmarking LLMs for Comprehensive Tool Utilization in Real-World Complex Scenarios arxiv:2401.17167.

HuggingFace. (2024). Hugging Face – The AI community building the future. [Online]. Available: https://huggingface.co/.

Humphreys, P., Raposo, D., Pohlen, T., Thornton, G., Chhaparia, R., Muldal, A., Abramson, J., Georgiev, P., Goldin, A., Santoro, A., & Lillicrap, T. (2022). A data-driven approach for learning to control computers. *International Conference on Machine Learning*, 9466–9482.

Ideogram, A. I. (2023). Ideogram: Image Generation for Everyone. [Online]. Available: https://ideogram.ai/.

IEEE. (2024). IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, D., Dai, W., Madotto, A., & Fung, P. (2022). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, *55*, 1–38. doi:10.1145/3571730.

Jia, S., Kiros, J., & Ba, J. (2019). DOM-Q-NET: Grounded RL on Structured Language. *International Conference on Learning Representations*, *abs/1902.07257*.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023). Mistral 7B arxiv:2310.06825.

Kambhampati, S. (2024). Can Large Language Models Reason and Plan? *Annals of the New York Academy of Sciences*. doi:10.1111/nyas.15125.

Kapoor, R., Butala, Y. P., Russak, M., Koh, J. Y., Kamble, K., Alshikh, W., & Salakhutdinov, R. (2024). OmniACT: A Dataset and Benchmark for Enabling Multimodal Generalist Autonomous Agents for Desktop and Web arxiv:2402.17553.

Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., & Sabharwal, A. (2023). Decomposed Prompting: A Modular Approach for Solving Complex Tasks. *ICLR*.

Kil, J., Song, C. H., Zheng, B., Deng, X., Su, Y., & Chao, W.-L. (2024). Dual-View Visual Contextualization for Web Navigation arxiv:2402.04476.

Kim, G., Baldi, P., & McAleer, S. (2023). Language Models can Solve Computer Tasks. *NeurIPS*.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M. C., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., & Fried, D. (2024). VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks arxiv:2401.13649.

Kojima, T., Gu, S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large Language Models are Zero-Shot Reasoners. *Neural Information Processing Systems*, *abs/2205.11916*.

Kong, Y., Ruan, J., Chen, Y., Zhang, B., Bao, T., Shi, S., Du, G., Hu, X., Mao, H., Li, Z., Zeng, X., & Zhao, R. (2023). TPTU-v2: Boosting Task Planning and Tool Usage of Large Language Model-based Agents in Real-world Systems arxiv:2311.11315.

Lai, H., Liu, X., Iong, I. L., Yao, S., Chen, Y., Shen, P., Yu, H., Zhang, H., Zhang, X., Dong, Y., & Tang, J. (2024). AutoWebGLM: Bootstrap And Reinforce A Large Language Model-based Web Navigating Agent arxiv:2404.03648.

Larsen, B., & Narayan, J. (2023). Generative AI: A game-changer that society and industry need to be ready for. [Online]. Available: https://www.weforum.org/agenda/2023/01/davos23-generative-ai-a-game-changer-industries-and-society-code-developers/#:~:text=URL%3A%20https%3A%2F%2Fwww.weforum.org%2Fagenda%2F2023%2F01%2Fdavos23.

Lee, K., Joshi, M., Turc, I. R., Hu, H., Liu, F., Eisenschlos, J. M., Khandelwal, U., Shaw, P., Chang, M.-W., & Toutanova, K. (2023). Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding. *ICML*, 18893–18912.

Lee, M., Liang, P., & Yang, Q. (2022). CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities. *International Conference on Human Factors in Computing Systems*. doi:10.1145/3491102.3502030.

Lee, S., Choi, J., Lee, J., Choi, H., Ko, S. Y., Oh, S., & Shin, I. (2023). Explore, Select, Derive, and Recall: Augmenting LLM with Human-like Memory for Mobile Task Automation arxiv:2312.03003.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Neural Information Processing Systems*, *abs/2005.11401*.

Li, G., Baechler, G., Tragut, M., & Li, Y. (2022). Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale. *International Conference on Human Factors in Computing Systems*. doi:10.1145/3491102.3502042.

Li, G., Hammoud, H., Itani, H., Khizbullin, D., & Ghanem, B. (2023). CAMEL: Communicative Agents for Mind Exploration of Large Language Model Society. *Neural Information Processing Systems*.

Li, G., & Li, Y. (2023). Spotlight: Mobile UI Understanding using Vision-Language Models with a Focus. *ICLR*.

Li, J., Li, D., Savarese, S., & Hoi, S. C. H. (2023). BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. *ICML*, 19730–19742.

Li, M., Song, F., Bowen, Y., Yu, H., Li, Z., Huang, F., & Li, Y. (2023). API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. *Conference on Empirical Methods in Natural Language Processing*, 3102–3116. doi:10.18653/v1/2023.emnlp-main.187.

Li, T., Li, G., Deng, Z., Wang, B., & Li, Y. (2023). A Zero-Shot Language Agent for Computer Control with Structured Reflection. *EMNLP*, 11261–11274. doi:10.18653/v1/2023.findings-emnlp.753.

Li, T. J.-J., Popowski, L., Mitchell, T. M., & Myers, B. (2021). Screen2Vec: Semantic Embedding of GUI Screens and GUI Components. *International Conference on Human Factors in Computing Systems*. doi:10.1145/3411764.3445049.

Li, T. J.-J., Azaria, A., & Myers, B. A. (2017). SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 6038–6049. doi:10.1145/3025453.3025483.

Li, W., Hsu, F.-L., Bishop, W., Campbell-Ajala, F., Lin, M., & Riva, O. (2024). UINav: A Practical Approach to Train On-Device Automation Agents arxiv:2312.10170.

Li, Y., He, J., Zhou, X., Zhang, Y., & Baldridge, J. (2020). Mapping Natural Language Instructions to Mobile UI Action Sequences. *Annual Meeting of the Association for Computational Linguistics*, *abs/2005.03776*. doi:10.18653/v1/2020.acl-main.729.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P. R., & Zeng, A. (2022). Code as Policies: Language Model Programs for Embodied Control. *IEEE International Conference on Robotics and Automation*, 9493–9500. doi:10.1109/icra48891.2023.10160591.

Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Tu, Z., & Shi, S. (2023). Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate arxiv:2305.19118.

Liang, Y., Wu, C., Song, T., Wu, W., Xia, Y., Liu, Y., Ou, Y., Lu, S., Ji, L., Mao, S., Wang, Y., Shou, L., Gong, M., & Duan, N. (2024). TaskMatrix.AI: Completing Tasks by Connecting Foundation Models with Millions of APIs. *Intelligent Computing*, *3*. doi:10.34133/icomputing.0063.

Lin, H., Wang, Z., Ma, J., & Liang, Y. (2023). MCU: A Task-centric Framework for Open-ended Agent Evaluation in Minecraft arxiv:2310.08367.

Lin, J., Zhao, H., Zhang, A., Wu, Y., Ping, H., & Chen, Q. (2023). AgentSims: An Open-Source Sandbox for Large Language Model Evaluation arxiv:2308.04026.

Lin, S. C., Hilton, J., & Evans, O. (2021). TruthfulQA: Measuring How Models Mimic Human Falsehoods. *Annual Meeting of the Association for Computational Linguistics*, 3214–3252. doi:10.18653/v1/2022.acl-long.229.

Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., Biswas, J., & Stone, P. (2023). LLM+P: Empowering Large Language Models with Optimal Planning Proficiency arxiv:2304.11477.

Liu, E., Guu, K., Pasupat, P., Shi, T., & Liang, P. (2018). Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration. *International Conference on Learning Representations*, *abs/1802.08802*.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., & Raffel, C. (2022). Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. *NeurIPS*.

Liu, H., Sferrazza, C., & Abbeel, P. (2023). Chain of Hindsight Aligns Language Models with Feedback arxiv:2302.02676.

Liu, J., Song, Y., Lin, B. Y., Lam, W., Neubig, G., Li, Y., & Yue, X. (2024). VisualWebBench: How Far Have Multimodal LLMs Evolved in Web Page Understanding and Grounding? arxiv:2404.05955.

Liu, Q., Guo, Y., Deng, L., Liu, H., Li, D., & Sun, H. (2022). Reducing Action Space: Reference-Model-Assisted Deep Reinforcement Learning for Inverter-based Volt-Var Control arxiv:2210.07360.

Lu, J., Clark, C., Lee, S., Zhang, Z., Khosla, S., Marten, R., Hoiem, D., & Kembhavi, A. (2023). Unified-IO 2: Scaling Autoregressive Multimodal Models with Vision, Language, Audio, and Action arxiv:2312.17172.

Lù, X. H., Kasner, Z., & Reddy, S. (2024). WebLINX: Real-World Website Navigation with Multi-Turn Dialogue arxiv:2402.05930.

Ma, X., Zhang, Z., & Zhao, H. (2024). Comprehensive Cognitive LLM Agent for Smartphone GUI Automation arxiv:2402.11941.

Mao, Y., He, P., Liu, X., Shen, Y., Gao, J., Han, J., & Chen, W. (2020). Generation-Augmented Retrieval for Open-Domain Question Answering. *Annual Meeting of the Association for Computational Linguistics*, 4089–4100. doi:10.18653/v1/2021.acl-long.316.

Marino, K., Rastegari, M., Farhadi, A., & Mottaghi, R. (2019). OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge. *Computer Vision and Pattern Recognition*, 3190–3199. doi:10.1109/cvpr.2019.00331.

Marzari, L., Pore, A., Dall'Alba, D., Aragon-Camarasa, G., Farinelli, A., & Fiorini, P. (2021). Towards Hierarchical Task Decomposition using Deep Reinforcement Learning for Pick and Place Subtasks. *International Conference on Advanced Robotics*, 640–645. doi:10.1109/icar53236.2021.9659344.

Mazumder, S., & Riva, O. (2020). FLIN: A Flexible Natural Language Interface for Web Navigation. *North American Chapter of the Association for Computational Linguistics*, *abs/2010.12844*. doi:10.18653/v1/2021.naacl-main.222.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning arxiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. doi:10.1038/nature14236.

Modarressi, A., Imani, A., Fayyaz, M., & Schütze, H. (2023). RET-LLM: Towards a General Read-Write Memory for Large Language Models arxiv:2305.14322.

Munkhdalai, T., Faruqui, M., & Gopal, S. (2024). Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention arxiv:2404.07143.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., & Schulman, J. (2022). WebGPT: Browser-assisted question-answering with human feedback arxiv:2112.09332.

Ng, Y., Miyashita, D., Hoshi, Y., Morioka, Y., Torii, O., Kodama, T., & Deguchi, J. (2023). SimplyRetrieve: A Private and Lightweight Retrieval-Centric Generative AI Tool arxiv:2308.03983.

Nielson, J. (1993). Response Time Limits: Article by Jakob Nielsen. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/.

Niu, R., Li, J., Wang, S., Fu, Y., Hu, X., Leng, X., Kong, H., Chang, Y., & Wang, Q. (2024). ScreenAgent: A Vision Language Model-driven Computer Control Agent arxiv:2402.07945.

OpenAI. (2024). OpenAI Tokenizer. [Online]. Available: https://platform.openai.com/tokenizer.

openAI. (2022). Introducing ChatGPT. [Online]. Available: https://openai.com/index/chatgpt/.

openAI. (2023). New models and developer products announced at DevDay. [Online]. Available: https://openai.com/index/new-models-and-developer-products-announced-at-devday/.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., … Zoph, B. (2024). GPT-4 Technical Report arxiv:2303.08774.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L. E., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. J. (2022). Training language models to follow instructions with human feedback. *Neural Information Processing Systems*, *abs/2203.02155*.

Ovadia, O., Brief, M., Mishaeli, M., & Elisha, O. (2024). Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs arxiv:2312.05934.

Pandey, R., Watkins, A., Shrestha, A., & Subedi, S. (2023). Tarsier. [Online]. Available: https://github.com/reworkd/tarsier.

Paranjape, B., Lundberg, S., Singh, S., Hajishirzi, H., Zettlemoyer, L., & Ribeiro, M. T. (2023). ART: Automatic multi-step reasoning and tool-use for large language models arxiv:2303.09014.

Park, J., O'Brien, J. C., Cai, C. J., Morris, M., Liang, P., & Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *ACM Symposium on User Interface Software and Technology*. doi:10.1145/3586183.3606763.

Parr, R., & Russell, S. (1997). Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Information Processing Systems*, *10*.

Pasupat, P., Jiang, T., Liu, E., Guu, K., & Liang, P. (2018). Mapping natural language commands to web elements. *Conference on Empirical Methods in Natural Language Processing*, 4970–4976. doi:10.18653/v1/d18-1540.

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 488–489. doi:10.1109/cvprw.2017.70.

Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large Language Model Connected with Massive APIs arxiv:2305.15334.

Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). Communicative Agents for Software Development arxiv:2307.07924.

Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Huang, Y., Xiao, C., Han, C., Fung, Y. R., Su, Y., Wang, H., Qian, C., Tian, R., Zhu, K., Liang, S., Shen, X., Xu, B., … Sun, M. (2023). Tool Learning with Foundation Models arxiv:2304.08354.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., & Sun, M. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs arxiv:2307.16789.

Raffel, C., Shazeer, N. M., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of machine learning research*, *21*, 140:1–140:67.

Raman, S. S., Cohen, V., Paulius, D., Idrees, I., Rosen, E., Mooney, R., & Tellex, S. (2023). CAPE: Corrective Actions from Precondition Errors using Large Language Models arxiv:2211.09935.

Rawles, C., Li, A., Rodriguez, D., Riva, O., & Lillicrap, T. (2023). Android in the Wild: A Large-Scale Dataset for Android Device Control arxiv:2307.10088.

Reed, S. E., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., & Freitas, N. d. (2022). A Generalist Agent. *Trans. Mach. Learn. Res.*, *2022*.

ResearchGate. (2024). ResearchGate | Find and share research. [Online]. Available: https://www.researchgate.net/.

Ruan, J., Chen, Y., Zhang, B., Xu, Z., Bao, T., Du, G., Shi, S., Mao, H., Li, Z., Zeng, X., & Zhao, R. (2023). TPTU: Large Language Model-based AI Agents for Task Planning and Tool Usage arxiv:2308.03427.

Saunders, W., Yeh, C., Wu, J., Bills, S., Ouyang, L., Ward, J., & Leike, J. (2022). Self-critiquing models for assisting human evaluators arxiv:2206.05802.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. *NeurIPS*.

Schneider, J., Meske, C., & Kuss, P. (2024). Foundation Models. *Business & Information Systems Engineering*. doi:10.1007/s12599-024-00851-0.

Sel, B., Al-Tawaha, A., Khattar, V., Jia, R., & Jin, M. (2023). Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models arxiv:2308.10379.

Shaw, P., Joshi, M., Cohan, J., Berant, J., Pasupat, P., Hu, H., Khandelwal, U., Lee, K., & Toutanova, K. (2023). From Pixels to UI Actions: Learning to Follow Instructions via Graphical User Interfaces. *NeurIPS*.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., & Zhuang, Y. (2023). HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. *NeurIPS*.

Shi, T., Karpathy, A., Fan, L., Hernandez, J., & Liang, P. (2017). World of Bits: An Open-Domain Platform for Web-Based Agents. *Proceedings of the 34th International Conference on Machine Learning*, *70*, 3135–3144.

Shinn, N., Cassano, F., Labash, B., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Neural Information Processing Systems*.

Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., & Hausknecht, M. J. (2020). ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. *International Conference on Learning Representations*, *abs/2010.03768*.

Shvo, M., Hu, Z., Icarte, R. T., Mohomed, I., Jepson, A., & McIlraith, S. A. (2021). AppBuddy: Learning to Accomplish Tasks in Mobile Apps via Reinforcement Learning. *Canadian Conference on AI*, *abs/2106.00133*. doi:10.21428/594757db.e57f0d1e.

Song, Y., Xiong, W., Zhu, D., Wu, W., Qian, H., Song, M., Huang, H., Li, C., Wang, K., Yao, R., Tian, Y., & Li, S. (2023). RestGPT: Connecting Large Language Models with Real-World RESTful APIs arxiv:2306.06624.

Song, Y., Bian, Y., Tang, Y., & Cai, Z. (2023). Navigating Interfaces with AI for Enhanced User Interaction arxiv:2312.11190.

Soselia, D., Saifullah, K., & Zhou, T. (2023). Learning UI-to-Code Reverse Generator Using Visual Critic Without Rendering arxiv:2305.14637.

S.R.K. Branavan, Luke S. Zettlemoyer, & Regina Barzilay. (2010). Reading Between the Lines: Learning to Map High-level Instructions to Commands. [Online]. Available: https://aclanthology.org/P10-1129.pdf.

Suay, H. B., & Chernova, S. (2011). Effect of human guidance and state space size on Interactive Reinforcement Learning. *2011 RO-MAN*, 1–6. doi:10.1109/ROMAN.2011.6005223.

Sultonov, S. (2023). Importance of Python Programming Language in Machine Learning. *International Bulletin of Engineering and Technology*, *3*(9), 28–30.

Sun, H., Zhuang, Y., Kong, L., Dai, B., & Zhang, C. (2023). AdaPlanner: Adaptive Planning from Feedback with Language Models. *NeurIPS*.

Sun, L., Chen, X., Chen, L., Dai, T., Zhu, Z., & Yu, K. (2022). META-GUI: Towards Multi-modal Conversational Agents on Mobile GUI. *EMNLP*, 6699–6712.

Surís, D., Menon, S., & Vondrick, C. (2023). ViperGPT: Visual Inference via Python Execution for Reasoning. *ICCV*, 11854–11864. doi:10.1109/iccv51070.2023.01092.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*, *12*.

Świechowski, M., Godlewski, K., Sawicki, B., & Ma'ndziuk, J. (2021). Monte Carlo Tree Search: A review of recent modifications and applications. *Artificial Intelligence Review*, *56*, 2497–2562. doi:10.1007/s10462-022-10228-y.

Tan, W., Ding, Z., Zhang, W., Li, B., Zhou, B., Yue, J., Xia, H., Jiang, J., Zheng, L., Xu, X., Bi, Y., Gu, P., Wang, X., Karlsson, B. F., An, B., & Lu, Z. (2024). Towards General Computer Control: A Multimodal Agent for Red Dead Redemption II as a Case Study arxiv:2403.03186.

Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., Cao, B., & Sun, L. (2023). ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases arxiv:2306.05301.

Tao, H., T V, S., Shlapentokh-Rothman, M., & Hoiem, D. (2023). WebWISE: Web Interface Control and Sequential Exploration with Large Language Models arxiv:2310.16042.

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Petrov, S., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., … Vinyals, O. (2023). Gemini: A Family of Highly Capable Multimodal Models arxiv:2312.11805.

Team, U. (2023). Wir stellen vor: Ultralytics YOLOv8. [Online]. Available: https://www.ultralytics.com/de/blog/introducing-ultralytics-yolov8.

team rabbit research, r. r. (2023). Learning human actions on computer applications. [Online]. Available: https://rabbit.tech/research.

Thaker, N., & Shukla, A. (2020). Python as Multi Paradigm Programming Language. *International Journal of Computer Applications*, *177*, 38–42. doi:10.5120/ijca2020919775.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., … Scialom, T. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models arxiv:2307.09288.

Toyama, D., Hamel, P., Gergely, A., Comanici, G., Glaese, A., Ahmed, Z., Jackson, T., Mourad, S., & Precup, D. (2021). AndroidEnv: A Reinforcement Learning Platform for Android arxiv:2105.13231.

Turing, A. M. (1950). I.—Computing Machinery and Intelligence. *Mind*, *59*(236), 433–460. doi:10.1093/mind/LIX.236.433.

van Otterlo, M., & Wiering, M. (2012). Reinforcement Learning and Markov Decision Processes. In *Reinforcement Learning: State-of-the-Art* (pp. 3–42). Springer Berlin Heidelberg. doi:10.1007/978-3-642-27645-3_1.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *Neural Information Processing Systems*, 5998–6008.

Venkatesh, S. G., Talukdar, P., & Narayanan, S. (2023). UGIF: UI Grounded Instruction Following arxiv:2211.07615.

Wang, B., Li, G., & Li, Y. (2022). Enabling Conversational Interaction with Mobile UI using Large Language Models. *International Conference on Human Factors in Computing Systems*. doi:10.1145/3544548.3580895.

Wang, B., Li, G., Zhou, X., Chen, Z., Grossman, T., & Li, Y. (2021). Screen2Words: Automatic Mobile UI Summarization with Multimodal Learning. *ACM Symposium on User Interface Software and Technology*. doi:10.1145/3472749.3474765.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models arxiv:2305.16291.

Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., & Lim, E.-P. (2023). Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. *ACL*, 2609–2634. doi:10.18653/v1/2023.acl-long.147.

Wang, L., Zhang, J., Yang, H., Chen, Z., Tang, J., Zhang, Z., Chen, X., Lin, Y., Song, R., Zhao, W. X., Xu, J., Dou, Z., Wang, J., & Wen, J.-R. (2024). User Behavior Simulation with Large Language Model based Agents arxiv:2306.02552.

Wang, W., Lv, Q., Yu, W., Hong, W., Qi, J., Wang, Y., Ji, J., Yang, Z., Zhao, L., Song, X., Xu, J., Xu, B., Li, J., Dong, Y., Ding, M., & Tang, J. (2024). CogVLM: Visual Expert for Pretrained Language Models arxiv:2311.03079.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ICLR*.

Wang, X., Wang, Z., Liu, J., Chen, Y., Yuan, L., Peng, H., & Ji, H. (2023). MINT: Evaluating LLMs in Multi-turn Interaction with Tools and Language Feedback arxiv:2309.10691.

Wang, Y., Jiang, Z., Chen, Z., Yang, F., Zhou, Y., Cho, E., Fan, X., Huang, X., Lu, Y., & Yang, Y. (2024). RecMind: Large Language Model Powered Agent For Recommendation arxiv:2308.14296.

Wang, Y., Wu, Z., Yao, J., & Su, J. (2024). TDAG: A Multi-Agent Framework based on Dynamic Task Decomposition and Agent Generation arxiv:2402.10178.

Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., & Liang, Y. (2023). Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents arxiv:2302.01560.

Wang, Z., Zhang, H., Li, C.-L., Eisenschlos, J. M., Perot, V., Wang, Z., Miculicich, L., Fujii, Y., Shang, J., Lee, C.-Y., & Pfister, T. (2024). Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding arxiv:2401.04398.

Warnell, G., Waytowich, N. R., Lawhern, V. J., & Stone, P. (2017). Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces. *AAAI Conference on Artificial Intelligence*, *abs/1709.10163*. doi:10.1609/aaai.v32i1.11485.

Warren, D., Pereira, L., & Fernando, N. (2015). Prolog the language and its implementation.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3), 279–292. doi:10.1007/BF00992698.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Xia, F., Le, Q., & Zhou, D. (2022). Chain of Thought Prompting Elicits Reasoning in Large Language Models. *Neural Information Processing Systems*, *abs/2201.11903*.

Wen, H., Li, Y., Liu, G., Zhao, S., Yu, T., Li, T. J.-J., Jiang, S., Liu, Y., Zhang, Y., & Liu, Y. (2023). AutoDroid: LLM-powered Task Automation in Android. *ACM/IEEE International Conference on Mobile Computing and Networking*, 543–557. doi:10.1145/3636534.3649379.

Wen, H., Wang, H., Liu, J., & Li, Y. (2024). DroidBot-GPT: GPT-powered UI Automation for Android arxiv:2304.07061.

Wu, J., Zhang, X., Nichols, J., & Bigham, J. P. (2021). Screen Parsing: Towards Reverse Engineering of UI Models from Screenshots. *ACM Symposium on User Interface Software and Technology*. doi:10.1145/3472749.3474763.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation arxiv:2308.08155.

Wu, Y., Min, S. Y., Bisk, Y., Salakhutdinov, R., Azaria, A., Li, Y., Mitchell, T., & Prabhumoye, S. (2023). Plan, Eliminate, and Track – Language Models are Good Teachers for Embodied Agents arxiv:2305.02412.

Wu, Z., Han, C., Ding, Z., Weng, Z., Liu, Z., Yao, S., Yu, T., & Kong, L. (2024). OS-Copilot: Towards Generalist Computer Agents with Self-Improvement arxiv:2402.07456.

Xiang, J., Tao, T., Gu, Y., Shu, T., Wang, Z., Yang, Z., & Hu, Z. (2023). Language Models Meet World Models: Embodied Experiences Enhance Language Models. *NeurIPS*.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., & Yu, T. (2024). OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments arxiv:2404.07972.

Xing, M., Zhang, R., Xue, H., Chen, Q., Yang, F., & Xiao, Z. (2024). Understanding the Weakness of Large Language Model Agents within a Complex Android Environment arxiv:2402.06596.

Xu, N., Masling, S., Du, M., Campagna, G., Heck, L., Landay, J., & Lam, M. (2021). Grounding Open-Domain Instructions to Automate Web Support Tasks. *North American Chapter of the Association for Computational Linguistics*, 1022–1032. doi:10.18653/v1/2021.naacl-main.80.

Xu, Q., Hong, F., Li, B., Hu, C., Chen, Z., & Zhang, J. (2023). On the Tool Manipulation Capability of Open-source Large Language Models arxiv:2305.16504.

Yan, A., Yang, Z., Zhu, W., Lin, K., Li, L., Wang, J., Yang, J., Zhong, Y., McAuley, J., Gao, J., Liu, Z., & Wang, L. (2023). GPT-4V in Wonderland: Large Multimodal Models for Zero-Shot Smartphone GUI Navigation arxiv:2311.07562.

Yang, J., Zhang, H., Li, F., Zou, X., Li, C., & Gao, J. (2023). Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V arxiv:2310.11441.

Yang, R., Song, L., Li, Y., Zhao, S., Ge, Y., Li, X., & Shan, Y. (2023). GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction. *NeurIPS*.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., & Manning, C. D. (2018). HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. *Conference on Empirical Methods in Natural Language Processing*, 2369–2380. doi:10.18653/v1/d18-1259.

Yang, Z., Li, L., Lin, K., Wang, J., Lin, C.-C., Liu, Z., & Wang, L. (2023). The Dawn of LMMs: Preliminary Explorations with GPT-4V(ision) arxiv:2309.17421.

Yao, S., Chen, H., Yang, J., & Narasimhan, K. (2022). WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. *NeurIPS*.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2023). Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *NeurIPS*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR*.

You, K., Zhang, H., Schoop, E., Weers, F., Swearngin, A., Nichols, J., Yang, Y., & Gan, Z. (2024). Ferret-UI: Grounded Mobile UI Understanding with Multimodal LLMs arxiv:2404.05719.

Zeng, A., Liu, M., Lu, R., Wang, B., Liu, X., Dong, Y., & Tang, J. (2023). AgentTuning: Enabling Generalized Agent Abilities for LLMs arxiv:2310.12823.

Zhang, C., Yang, Z., Liu, J., Han, Y., Chen, X., Huang, Z., Fu, B., & Yu, G. (2023). AppAgent: Multimodal Agents as Smartphone Users arxiv:2312.13771.

Zhang, D., Xu, H., Zhao, Z., Chen, L., Cao, R., & Yu, K. (2024). Mobile-Env: An Evaluation Platform and Benchmark for LLM-GUI Interaction arxiv:2305.08144.

Zhang, H., Du, W., Shan, J., Zhou, Q., Du, Y., Tenenbaum, J. B., Shu, T., & Gan, C. (2024). Building Cooperative Embodied Agents Modularly with Large Language Models arxiv:2307.02485.

Zhang, J., Zhang, J., Pertsch, K., Liu, Z., Ren, X., Chang, M., Sun, S.-H., & Lim, J. J. (2023). Bootstrap Your Own Skills: Learning to Solve New Tasks with Large Language Model Guidance. *CoRL*, 302–325.

Zhang, J., Wu, J., Teng, Y., Liao, M., Xu, N., Xiao, X., Wei, Z., & Tang, D. (2024). Android in the Zoo: Chain-of-Action-Thought for GUI Agents arxiv:2403.02713.

Zhang, X., de Greef, L., Swearngin, A., White, S., Murray, K., Yu, L., Shan, Q., Nichols, J., Wu, J., Fleizach, C., Everitt, A., & Bigham, J. P. (2021). Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. doi:10.1145/3411764.3445186.

Zhang, Z., & Zhang, A. (2023). You Only Look at Screens: Multimodal Chain-of-Action Agents arxiv:2309.11436.

Zheng, B., Gou, B., Kil, J., Sun, H., & Su, Y. (2024). GPT-4V(ision) is a Generalist Web Agent, if Grounded arxiv:2401.01614.

Zheng, L., Huang, Z., Xue, Z., Wang, X., An, B., & Yan, S. (2024). AgentStudio: A Toolkit for Building General Virtual Agents arxiv:2403.17918.

Zheng, L., Wang, R., Wang, X., & An, B. (2023). Synapse: Trajectory-as-Exemplar Prompting with Memory for Computer Control arxiv:2306.07863.

Zhong, W., Guo, L., Gao, Q., Ye, H., & Wang, Y. (2024). MemoryBank: Enhancing Large Language Models with Long-Term Memory. *AAAI*, 19724–19731. doi:10.1609/aaai.v38i17.29946.

Zhou, P., Pujara, J., Ren, X., Chen, X., Cheng, H.-T., Le, Q. V., Chi, E. H., Zhou, D., Mishra, S., & Zheng, H. S. (2024). Self-Discover: Large Language Models Self-Compose Reasoning Structures arxiv:2402.03620.

Zhou, R., Yang, Y., Wen, M., Wen, Y., Wang, W., Xi, C., Xu, G., Yu, Y., & Zhang, W. (2024). TRAD: Enhancing LLM Agents with Step-Wise Thought Retrieval and Aligned Decision arxiv:2403.06221.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., & Neubig, G. (2023). WebArena: A Realistic Web Environment for Building Autonomous Agents arxiv:2307.13854.

Zhou, X., Li, G., & Liu, Z. (2023). LLM As DBA arxiv:2308.05481.

Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., Qiao, Y., Zhang, Z., & Dai, J. (2023). Ghost in the Minecraft: Generally

Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory arxiv:2305.17144.

7.2 List of Figures

7.3 List of Tables

Appendix A

Data Collections

In this appendix, various data collections are presented, and organized into three tables based on the specific areas of an A2C2 they target. Table A.1 lists data collections specialized in LLM reasoning, while Table A.2 focuses on data collections designed for dynamic action inference for screens. Lastly, Table A.3 includes data collections utilizing screen representations and actions as output. Decoupling an A2C2 allows for separate evaluation of its different components on specialized benchmarks, enhancing the flexibility of analysis.

TABLE A.1: Summary of data collections specialized in reasoning. ① Dataset, ② Benchmark, ③ Environment.

Name	Type	Detail	Reference
CoAuthor	①	Text	M. Lee et al. (2022)
Constitutional AI	①	Text	Y. Bai et al. (2022)
GSM8K	①	Text	Cobbe et al. (2021)
MMLU	①	Text	Hendrycks et al. (2020)
Real Toxicity Prompts	①	Text	Gehman et al. (2020)
API-Bank	②	Tools	M. Li et al. (2023)
OK-VQA	① ②	Image	Marino et al. (2019)
TruthfulQA	① ②	Text	S. C. Lin et al. (2021)
HotPotQA	② ③	Text	Z. Yang et al. (2018)

TABLE A.2: Summary of data collections specialized in dynamic action inference for screens. ① Dataset, ② Benchmark.

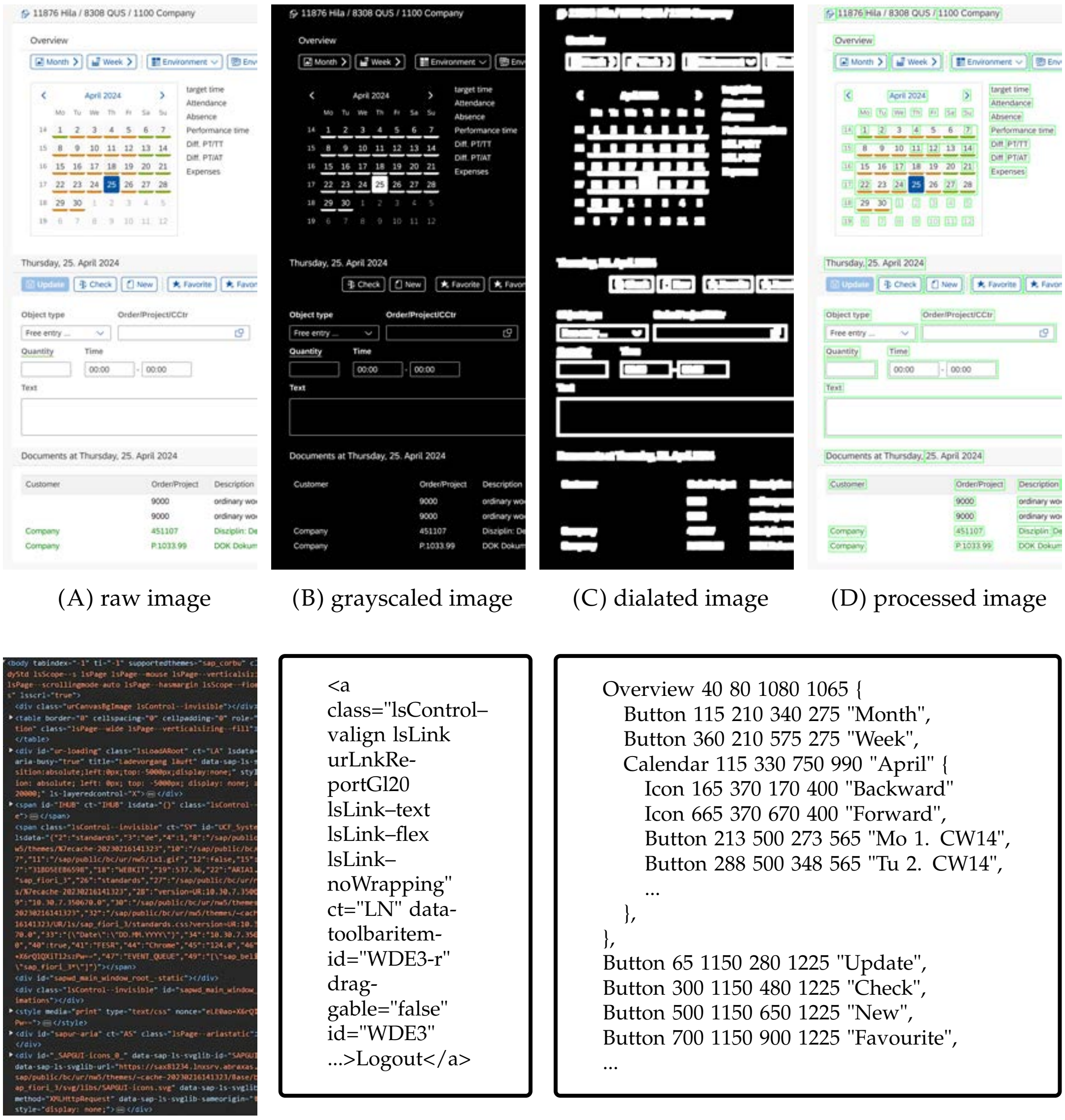
Name	Type	Detail	Reference
PhraseNode	①	DOM, Screen	Pasupat et al. (2018)
UIBert	①	DOM, Screen	C. Bai et al. (2021)
RicoSCA	①	VH, Screen	Y. Li et al. (2020)
ScreenAI	①	Text, Screen	Baechler et al. (2024)
VisualWebBench	②	Web	J. Liu et al. (2024)
Screen2Words	① ②	Screen	B. Wang et al. (2021)

TABLE A.3: Summary of data collections for full A2C2. ① Dataset, ② Benchmark, ③ Environment.

Name	Type	Detail	Reference
RUSS	①	DOM, Screen	N. Xu et al. (2021)
WebVLN-v1	①	HTML, Screen	Q. Chen et al. (2024)
AITW	①	Text, Screen	Rawles et al. (2023)
PixelHelp	①	Text, Screen	Y. Li et al. (2020)
META-GUI	①	VH, Screen	L. Sun et al. (2022)
MoTiF	①	VH, Screen	Burns et al. (2022)
UGIF	①	VH, Screen	Venkatesh et al. (2023)
ScreenAgent	①	Desktop	Niu et al. (2024)
SkillForgeChain	②	Text	H. Lin et al. (2023)
ToolBench	②	Tool	Q. Xu et al. (2023)
Androidenv	③	Android	Toyama et al. (2021)
MobileEnv	③	Mobile	D. Zhang et al. (2024)
ALFWorld	③	Text	Shridhar et al. (2020)
MiniWoB	③	Web	Shi et al. (2017)
MiniWoB++	③	Web	E. Liu et al. (2018)
WebShop	③	Web	Yao et al. (2022)
AutoWebBench	③	Web	Lai et al. (2024)
OmniACT	① ②	Desktop	Kapoor et al. (2024)
DroidTask	① ②	VH, Screen	Wen et al. (2023)
AgentStudio	① ③	Desktop	L. Zheng et al. (2024)
Mind2Web	① ③	DOM, Screen	X. Deng et al. (2023)
OSWorld	② ③	Desktop	Xie et al. (2024)
MT-Mind2Web	② ③	HTML	Y. Deng et al. (2024)
BrowserGym	② ③	HTML, Screen, Accessibility Tree	Drouin et al. (2024)
WorkArena	② ③	HTML, Screen, Accessibility tree	Drouin et al. (2024)
AssistGUI	② ③	Screen, Text	D. Gao et al. (2024)
WebArena	② ③	Web	S. Zhou et al. (2023)

Appendix B

Dynamic Action Inference



(A) raw image

(B) grayscaled image

(C) dialated image

(D) processed image

```
<body tabindex="-1" ti="-1" supportedthemes="sap_corbu" c
dyStd lsScope--s lsPage lsPage--mouse lsPage--verticalsi
lsPage--scrollingmode-auto lsPage--hasmargin lsScope--flo
s" lsscrl="true">
  <div class="urCanvasBgImage lsControl--invisible"></div
▸ <table border="0" cellspacing="0" cellpadding="0" role="
  tion" class="lsPage--wide lsPage--verticalsizing--fill"
  </table>
▸ <div id="ur-loading" class="lsLoadARoot" ct="LA" lsdata=
  aria-busy="true" title="Ladevorgang läuft" data-sap-ls-
  sition:absolute;left:0px;top:-5000px;display:none;" sty
  ion: absolute; left: 0px; top: -5000px; display: none;
  20000;" ls-layeredcontrol="X"> ⋯ </div>
▸ <span id="IHUB" ct="IHUB" lsdata="{}" class="lsControl-
  e"> ⋯ </span>
  <span class="lsControl--invisible" ct="SY" id="UCF_Syste
  lsdata="{"2":"standards","3":"de","4":1,"8":"/sap/publi
  w5/themes/%7ecache-20230216141323","10":"/sap/public/bc
  7","11":"/sap/public/bc/ur/nw5/1x1.gif","12":false,"15"
  7":"31BD5EEB6598","18":"WEBKIT","19":537.36,"22":"ARIA1
  "sap_fiori_3","26":"standards","27":"/sap/public/bc/ur/
  s/%7ecache-20230216141323","28":"version=UR:10.30.7.350
  9":"10.30.7.350670.0","30":"/sap/public/bc/ur/nw5/theme
  20230216141323","32":"/sap/public/bc/ur/nw5/themes/~cac
  16141323/UR/ls/sap_fiori_3/standards.css?version=UR:10.
  70.0","33":"{\"Date\":\"DD.MM.YYYY\"}","34":"10.30.7.35
  0","40":true,"41":"FESR","44":"Chrome","45":"124.0","46
  +X6rQ1QXiT12szPw==","47":"EVENT_QUEUE","49":"[\"sap_bel
  \"sap_fiori_3*\"]"}"></span>
  <div id="sapwd_main_window_root_-static"></div>
  <div class="lsControl--invisible" id="sapwd_main_window
  imations"></div>
▸ <style media="print" type="text/css" nonce="eLE0ao+X6rQ
  Pw=="> ⋯ </style>
▸ <div id="sapur-aria" ct="AS" class="lsPage--ariastatic"
  </div>
▸ <div id="_SAPGUI-icons_0_" data-sap-ls-svglib-id="SAPGU
  data-sap-ls-svglib-url="https://sax81234.lnxsrv.abraxas
  sap/public/bc/ur/nw5/themes/~cache-20230216141323/Base/
  ap_fiori_3/svg/libs/SAPGUI-icons.svg" data-sap-ls-svgli
  method="XMLHttpRequest" data-sap-ls-svglib-sameorigin="
  style="display: none;"> ⋯ </div>
```

(E) raw HTML

```
<a
class="lsControl–
valign lsLink
urLnkRe-
portGl20
lsLink–text
lsLink–flex
lsLink–
noWrapping"
ct="LN" data-
toolbaritem-
id="WDE3-r"
drag-
gable="false"
id="WDE3"
...>Logout</a>
```

(F) a filtered HTML tag

```
Overview 40 80 1080 1065 {
   Button 115 210 340 275 "Month",
   Button 360 210 575 275 "Week",
   Calendar 115 330 750 990 "April" {
      Icon 165 370 170 400 "Backward"
      Icon 665 370 670 400 "Forward",
      Button 213 500 273 565 "Mo 1. CW14",
      Button 288 500 348 565 "Tu 2. CW14",
      ...
   },
},
Button 65 1150 280 1225 "Update",
Button 300 1150 480 1225 "Check",
Button 500 1150 650 1225 "New",
Button 700 1150 900 1225 "Favourite",
...
```

(G) inferred representation

FIGURE B.1: The first row Figure B.1 (A) to Figure B.1 (D) shows dynamic action inference from raw image until a filtered image with computer vision approaches. The second row Figure B.1 (E) and Figure B.1 (F) shows dynamic action inference from an HTML representation reducing tags to interactable tags. Figure B.1 (G) represents the inferred representation achievable by advanced vision language processing methods. For the tabular representation see Table 4.1.