
LLM-BASED AGENT ASSISTANTS FOR SUPPORTING KNOWLEDGE-INTENSIVE PROCESSES: A DETAILED SURVEY OF COMPONENTS AND POTENTIAL METHODS

Aline Dobrovsky

Department of Artificial Intelligence and Cybersecurity
University of Klagenfurt
9020 Klagenfurt, Austria
aldobrovsky@edu.aau.at

ToDo Ask Co-Author

Department of Artificial Intelligence and Cybersecurity
University of Klagenfurt
9020 Klagenfurt, Austria
...
...

May 2, 2024

ABSTRACT

TODO

Keywords Large Language Models · Intelligent Agents · ...

1 Introduction

DRAFT: ONLY NOTES

General Intro of idea, notes: Wooldridge and Jennings [1995] in 1995 already describe the vision of a personal digital assistant, that among other activities autonomously fetches and aufbereiten interesting information and performs as assitant in many...information domains...interacts with different systems ... all based on the notion of a software agent Wooldridge [2002] or in 2002 a internet search assistant that autonomously queries different ressources and synthesizes information

INTRO: SUCCESSES OF LLMS

LLMs: towards human-level intelligence (refer to tests/benchmarks) LLMs good at: natural language generation, question answering, summarization interaction, emergent: reasoning and planning (chain of thought prompting -thin step by step) many improvements last few years, remarkably effective

Xi et al. [2023] LLMs have demonstrated powerful capabilities in knowledge acquisition, instruction comprehension, generalization, planning, and reasoning, while displaying effective natural language interactions with human

Valmeeekam et al. [2023] "no exaggeration to say that transformer-based large language models (LLMs) have revolutionized the field of natural language processing (NLP)" original: sequence completion, but claims of emergent capabilities (reasoning, planning)

Bubeck et al. [2023] "GPT-4 can solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, psychology and more, without needing any special prompting. Moreover, in all of these tasks, GPT-4's performance is strikingly close to human-level performance"

INTRO: LIMITS OF LLMS REGARDING REASONING/PLANNING

Problem/issue related to use case: recent data, domain knowledge - hallucinations, LLM on their own: only knowledge based on training data, cannot use "local" specialised data + multi-modality,might need..but..., custom processes, not enough data for training/fine-tuning

Addressed issues Complex tasks may require multi-step reasoning and specific tools LLMs tend to produce hallucinations Formal approaches require complete and accurate formal descriptions Specialised tools require specific formal input formats

Problem/issues of planning and reasoning: complex inputs, requiring advanced reasoning, multi-step questions requiring multi-step reasoning instead of input/output (+ also add that there's RAG but does not change the single step approach), LLM specialised in text completion not planning/no world model besides statistical/word similarity, LLM direct input-output

Shen et al. [2023] challenges: LLMs (not multimodal) limited to text input and text generation, real-world scenarios: complex tasks consisting of multiple sub-tasks, LLMs in specialized...weaker than expert/fine-tuned models Two research directions: multimodal models + models using external tools

Bubeck et al. [2023] LLM limitation: "architecture does not allow for an "inner dialogue" or a "scratchpad", beyond its internal representations, that could enable it to perform multi-step computations or store intermediate results." because of next word prediction only one direction..forward not in training data... planning: needs to know in first steps what last step will be to create consistent answer. Humans can backtrack

LLMs are ill-equipped for naive long-term planning since managing an extensive context over multiple steps is complex and resource-consuming (Silver et al., 2022; Liu et al., 2023) Silver et al. [2023]?? ArXiv, abs/2304.11477 Liu et al. [2023a]

Problem: Want to solve every problem with a LLM: Swiss army knife, From good one tool (complete/predict sentences/words, bloated - can do everything, requiring different modalities, specialized information or tools would be better

INTRO: LLM-BASED AGENT APPROACH

Naming/Definition: LLM-based agents, LLM agents, ...

objective/goal: solve complex tasks, some degree of autonomous behaviour reasoning use to break down complex tasks/queries to simpler steps break down complex goals/tasks/issue into manageable simpler steps/tasks Answering complex queries with LLMs, involving multiple steps, solve a complex task involving several steps multi-step reasoning Employ a LLM as controller for an AI assistant agent Process complex and vague natural language questions Plan sequence of less complex tasks to solve overall task Autonomously use specialized tools without requiring manual queries idea: agents planning which use LLM as controller/main aspect/ core Answer complex questions or solve complex tasks Exploit LLM capabilities with natural language, interaction and reasoning less hallucinations?accuracy? relevance?! Assistant: automated communication with other systems (find documents, edit document content, (use as context within model) Integration, automatization other services (Ai services, databases,..) LLM controller that manages specialised models and offers a natural language interface Task planning, model selection and control, result summarisation Write tasks, planning, select tools, generate prompts/queries from tasks Summarise eg results from elastic search query. References must be included! Autonomous, goal-directed behaviour

Answer queries using local data Find correct and relevant data/content and synthesize Generate explanations for engineers and clients (+ save time) Client: Writes observations, gets feedback of what could be failure and/or if it should be submitted Engineer: existing/past cases with similarities, optionally suggestions about cause (of possible causes described in documents) Professional assistant Connect LLM and information systems Retain intellectual property Specialised models, modularity User interface: NLP, natural, not specialised (like SQL database query)

Use LLM: natural language interface, Input: text (sometimes multimodal) LLM: controller/main controller/brain consists of: a core, mainly with LLM, main coordination, planning, tool usage, memory + user request use LLM as core, computation using prompts to direct/steer behaviour language: understand instructions natural language interface, intuitive interaction reasoning capabilities??

LLM capability? complex tasks -> break into manageable subtasks and reason/execute... to solve overall task, multi-step planning break complex task into subtask and address each on their own, by using tools, operation flow/steps

Add: equip LLMs with an expanded perception space and action space, utilize external tools, external resources combination of LLM with planning, memory and tool usage access to API: information retrieval, database,...eventually: tool to interpret data, produce graphic or similar (eg library, python library like pandas?) recent knowledge not trained on specific knowledge, domain knowledge thinking iteratively specialised tools/plugins

Shen et al. [2023] "LLMs could act as a controller to manage existing AI models to solve complicated AI tasks and language could be a generic interface to empower this" "select models according to their function descriptions available in Hugging Face, execute each subtask with the selected AI model, and summarize the response according to the

execution results" "Language serves as an interface for LLMs" "the key point is how to choose suitable middleware to bridge the connections between LLMs and AI models" task planning, model selection, task execution, response generation

Dagan et al. [2023] despite prompting techniques like Chain of thought (REF), self-consistency (REF) and ReAct(F), limits of context window, hallucinations "LLMs in Planning and Reasoning In contrast to symbolic planners, LLMs have shown promise in adapting to noisy planning and reasoning tasks through various methods. Some general approaches such as Chain-of-Thought (Wei et al., 2022), Self-Consistency (Wang et al., 2023b), and Reasoning via Planning (Hao et al., 2023) augment the context with a reasoning trace that the LLM generates to improve its final prediction." "But symbolic planners require problem and domain descriptions as prerequisites (McDermott, 2000), which hampers their applicability in real-world scenarios where it may be infeasible to achieve these high informational demands."

INTRO: SCOPE OF ARTICLE

What this article's scope is/what it concentrates on and what not For knowledge workers/work? Personal assistants?? relevant information/answers! here approaches with planning AND tool, also some promising approaches not directly using tools but act in simulation environment but nonetheless promising for also this explicit tool use case here: task/goal-oriented agents analyse prompt, extract goal, parameters etc, plan, execute actions and result cover the base...ideas/approaches, not collect every single publication, anyhow already fast paced area potential for practical application but also mention some open issues and potential for further research foremost..of practical use, help support to select reader can identify opportunities for future research by...looking at..less common/never tried/novel combinations many existing... continuously new proposals/papers,... important to...overview, catalogue,...offer comprehensive resource for those... interested in... in order to provide this survey...cover publications...searched for... focus: agent architecture/methods/parts, o be able to ... their advantages/disadvantages techniques differ overview over as many as possible group in appropriate categories full discussion of approaches in the respective papers design: to maximise, find best, find suitable architecture/components,... here: literature research, literature review and critical analysis

Research question: What do we do not know yet? Here: which approaches exist? How can they be classified And then which aspects/designs are appropriate for which tasks with focus ion complex queries like/similar as FA goals and methodology literature analysis

NOT: fine-tuning not finetuning, depends on expected use case: using a trained LLM on very specific domain data, where finetuning would be not or barely possible (effort, cost-effectiveness?, generalizability/catastrophic forgetting), needs specialized data, labelled? also no additional instruction fine-tuning as agent NOT papers with special application areas, like agent for XY, Code commenting etc, where paper not about specifics of agent structure, trading,...playing specific game..?

literature search until: 11.03.2024 updated literature search: 21.04.2024 Methodology: Search: google scholar, <https://dl.acm.org/>, <https://ieeexplore.ieee.org/>, <https://dblp.org/>, following references, Connected papers <https://www.connectedpapers.com/> TODO: mention initial keywords used for search?

TODO: STRUCTURE OUTLINE

2 Related Work

INITIAL FORMULATION

This section introduces, discusses, and presents related literature surveys that discuss characteristics and develop taxonomies for LLM-based agents. It includes nomenclature and what others have identified as the main characteristics and main components of LLM-based agents. Given the novelty of this field, with publications appearing only since 2023, peer-reviewed literature has been complemented by preprints and blog postings. The material has been ordered according to publication date.

The authors of Mialon et al. [2023a] conducted a survey of 'augmented language models' (LLMs), where LLMs, originally trained with the missing token prediction objective, are augmented with additional abilities. These abilities include reasoning, the ability to decompose tasks into subtasks (planning), information retrieval (RAG), and the ability to use tools and call external modules. The objective is to enhance the otherwise limited context of LLMs. The authors discuss prompting and finetuning techniques for reasoning tasks, tool usage and rewards from the environment and humans for reinforcement learning, and they discuss potential issues within each category. The authors of Weng [2023] define 'LLM-Powered Autonomous Agents' as agents that utilise an LLM as their core controller, in conjunction with further components including planning, memory, action, and tools. They describe some illustrative approaches that demonstrate the realisation of these components. Liu et al. [2023b] present a discussion of "LLM-augmented Autonomous Agents," which extend the capabilities of LLM for the purposes of action execution. They further consider

the possibilities of environment interaction and the use of memory. The authors discuss different agent architectures for solo agents, including the use of LLMs for iterative planning with different prompting techniques, and the possibility of orchestrating multiple agents. They compare performances of six different architectures with different LLMs and argue that multiple coordinated agents solve complex tasks better.

Wang et al. [2024a] present a framework for designing 'LLM-based autonomous agents' architectures consisting of four modules. The profile module describes the role of the agent and the possible contents and generation strategies of agent profiles. The memory component can include different memory structures, formats and operations; and the planning and action components. The planning component is distinguished according to whether it employs feedback. Planning comprises single-path reasoning, multi-path reasoning, or an external planner. Feedback can be distinguished into environmental, human, or model-derived feedback. The action component is further divided into four categories: action target, action production, action space, and action impact. The authors outline various techniques that can be employed within the framework's components, and present potential applications and evaluation strategies. Xi et al. [2023] discuss the origin of the 'LLM-based agents' concept and explain the suitability of LLMs as a foundation for agents. They propose a general framework for LLM-based agents with three main components. The brain component includes the agent's knowledge and memory, and it makes decisions via planning or reasoning. The perception component processes environmental inputs and the action component expands the action space of LLMs by allowing the output of text, the use of tools and the execution of embodied actions. They mention three different types of applications: single agent, multi-agent and human-agent cooperation, explain the different, exemplary techniques for each of the components in their typology.

Zhao et al. [2023] survey 'LLM-based AI agents', where the LLM is used as the agent's brain. They introduce an agent classification scheme, consist of three components. The planning component is based on LLM inference and consists of task compositions and self-reflection. The memory component consists of sensory memory as input embedding, the short-term, temporary memory and the long-term stored memory. The third component is tool integration for LLM usage. They further mention general application categories for LLM-based AI agents, including chatbots, games, coding, design, research, collaboration, and general purpose applications. Sumers et al. [2023] are concerned with 'cognitive language agents', propose a framework for 'Cognitive Architectures for Language Agents (CoALA)' and provide a retrospective analysis of this concept in the context of broader AI history. The article describes the structure of a cognitive language agent, which comprises three main components. The information storage contains a working and a long-term memory, the action space offers internal and external actions, and the decision making procedure consists of an interactive loop of planning and action execution.

In Händler [2023] the author introduces a multi-dimensional taxonomy for 'autonomous LLM-powered multi-agent' architectures with focus on the balancing of autonomy and alignment of agents. An agent is defined as an LLM with a specific role and access to external resources, such as data, tools, or models. The author describes the core characteristics of agent architectures, including goal-driven task management, the LLM at the core of agents, agent collaboration, context interaction, and the balancing of autonomy (adaptive and self-organising aspects) and alignment (real-time responsive and user-guided aspects). It then presents a taxonomic classification of some selected representative LLM-powered multi-agent systems. The blog post Wang [2023] discusses the critical components of LLMs as a brain, memory, and external tool and data resources. It explains the principles of LLM-based agents, including the structure and program flow of an agent, and describes prompting techniques used for the part of reasoning, task decomposition, and planning. The article Zhang et al. [2023] focuses on the prompt-based reasoning process known as Chain-of-thoughts(CoT) in language agents and its potential. It outlines the fundamentals of CoT, the evolution of CoT approaches and discuss advantages of CoT and effective use cases. Furthermore, the authors provide an overview of some selected agents approaches using CoT.

The paper Muthusamy et al. [2023] discusses 'LLM-based personal agents' as personal agents in the enterprise. They characterise LLM-based agents as designed to work with complex goals, use a LLM as an orchestrator or brain and have flexible tool access. The agents' task is to identify a sequence of actions that is composed of external tools. The paper goes on to discuss the disadvantages of LLM-based agents in the enterprise and the current shortcomings of this technology. These include the fact that they are brittle to input changes, sensitive to the LLM used, and can get stuck in reasoning loops. Furthermore, they LLMs be slow and expensive to run, and tool usage can emit side effects. The paper Cheng et al. [2024] explores the concept of 'LLM-based intelligent agents'. It offers a comprehensive overview of the current research in this field, including definitions, agent components, single and multiagent approaches, datasets, application scenarios, performance evaluations, and prospects for future applications. The proposed framework for LLM-based agents comprises of five key components. The planning component can be realised by in-context learning methods (via prompting), external methods (like PDDL), and multi-stage methods. The memory component encompasses short-term memory, long-term memory and memory retrieval methods. The rethinking component encompasses different rethinking capabilities, such as in-context learning (prompting), supervised learning, reinforcement learning, and modular coordination. The environment component is divided into possible

general application areas. The action component encompasses tool employment, tool planning and tool creation. The authors also offer a typology for multiagent approaches. Agent relationships can be cooperative, competitive, mixed or hierarchical, planning types can be centralized or decentralised multiagent planning and diverse methods exist to potentially enhance communication efficiency.

The authors of Huang et al. [2024] concentrate on LLM-based agents planning. They provide a detailed analysis of planning abilities, advantages and limitations, and an evaluation of representative methods on four benchmarks. The authors categorise planning in five distinct areas: task decomposition, which involves breaking down a task into smaller subtasks and conduct sequential planning; multi-plan selection, which entails selecting one of several potential plans; external module-aided planning, which leverages external modules to assist in planning; reflection and refinement, which involves analysing and modifying a plan based on feedback; and memory-augmented planning, which utilises memory to enhance the planning process.

DRAFT- THIS SURVEY (without discrimination related work!): This survey introduces a new taxonomy/typology/main components, which is especially for planning very detailed. The fine-grained approach allows for the analysis of techniques for specific components and the ability to identify which component would benefit from which technique, enabling them to be mixed in different ways to those in original approaches. It also includes most recent approaches in this fast evolving field. Noteworthy specially also includes and focuses on/more attention approaches combining LLMs and symbolic AI. Furthermore, the survey makes explicit and discusses diverse assumptions about the planning environment and problem space. It analyses different techniques within components to provide practical guidance for choosing and applying.

3 Characteristics and Structure of LLM-Based Agents

DRAFT: ONLY NOTES

Repeat briefly successes of LLMs (transformers, attention mechanism dominant models for natural language processing tasks, by 2018, after attention is all you need paper pretrain large-scale models on background corpora , self-supervised pretraining objective predicting missing words/being successful in this r at first usual to fine-tune these models, but was discovered that few-shot prompting enables them also to fulfil tasks they were not trained upon, and zero-shot...)

Repeat briefly goal/main idea of LLM-based agents (see Intro)

3.1 Goal-Based Intelligent Agents

INITIAL FORMULATION

There is no general consensus on the definition of an agent Wooldridge [2002]. Wooldridge and Jennings [1995] provide a general definition of an agent as a software-based computer system that exhibits the following properties:

- Autonomy: An agent is in control over its actions and internal state and can operate in its environment without human intervention.
- Social Ability: An agent is able to interact with other agents.
- Reactivity: An agent can percept its environment and respond to changes.
- Pro-activeness: An agent exhibits goal-directed behaviour.

Furthermore, Wooldridge and Jennings [1995] also introduce a stronger notion of agents, which incorporates concepts typically associated with human characteristics, such as knowledge, belief, intention, obligation, emotions, mobility, veracity, benevolence and rationality.

In Russell et al. [2010], an agent is defined as a system that perceives its environment through sensors and acts upon that environment through actuators. The behaviour of an agent is described by an agent function mapping a percept sequence to an action. A rational agent must consider the consequences of its actions in order to achieve a desired performance measure. Wooldridge [2002] define agents as systems that are capable of autonomously determining the actions required to achieve a given objective. Moreover, intelligent and autonomous agents need to be able to operate in rapidly changing, unpredictable, or open environments, where the possibility of action failure is present. The environment is represented by states, and the effectoric capability of an agent is defined as the actions that it can perform. A standard agent is a mapping function that takes sequences of environment states to actions where the agents needs to decide for performing actions that satisfy a design objective.

As outlined in Wooldridge and Jennings [1995], agents can be realised through a variety of software engineering models. The conventional methodology is to conceptualise agents as knowledge-based systems utilising the paradigm

of symbolic AI. A deliberative agent requires the explicit representation of a symbolic model of the world. The process of decision-making is based on logical reasoning. However, this approach presents two significant challenges. Firstly, there is the transduction problem, which is about creating an accurate and adequate symbolic description of the world in a reasonable amount of time. Secondly, there is the problem of representation and reasoning, which requires the determination of how to represent complex real-world entities and processes, and the realisation of reasoning in a timely manner.

There are fundamentally distinct agent architectures or structures. Wooldridge [2002] mention reactive architectures, which only respond only to the current state, are less complex and do not need a world model but nonetheless rely on sufficient information about environment. In contrast, belief-desire-intention architectures comprise beliefs about the environment and a belief revision function to update beliefs based on perceptual input, an option generation function of available options, current an idea of possible courses of action, a filter for deliberation, intentions to be used as focus and an action selection function.

Russell et al. [2010] provide an overview over different agent structures. A simple reflex agent has no memory. A model-based reflex agent has an internal state and a history of percepts. A goal-based agent has goal information about desirable situations, search and planning, and reasoning about the means of achieving the goal. A utility-based agent evaluates the desirability of actions or action sequences, and assigns a value to them based on their expected utility or outcome. A learning agent can adapt and improve their performance over time by learning from experiences.

It is necessary to define the properties of the agent's environment in order to determine the applicability of potential solutions or methodologies. The following distinction can be made according to Russell et al. [2010]:

- Fully vs. Partial Observability: In a fully observable environment, the state of the world can be exactly observed during the plan execution, so the agent is able to detect all relevant aspects in order to make a decision.
- Single- vs. Multi-Agent Systems: One or several agents operate in the environment.
- Static vs. Dynamic: Changes in the environment only happen in response to actions of the actor or can also happen on their own.
- Deterministic vs. Stochastic: A state transition $\gamma(s, a) = s'$ will always result in the same, predictable state s' or elements of randomness have to be accounted for.
- Discrete vs. Continuous: There is a finite or continuous number of states and actions. Sometimes this notion also includes if there is an explicit model of duration or concurrency.
- Episodic vs. Sequential processes: Episodes have no dependency between them whereas in sequential environments, actions can potentially influence all future states.
- Known vs. Unknown: In a known environment, the outcome probabilities for all actions are known.

In general, to build a robust agent it should be expected that non-deterministic environments will be encountered, with the possibility of action failure. Wooldridge [2002]. When creating assistants for supporting knowledge-intensive processes in the enterprise, can also expect to deal with an unknown environment, because effects of information retrieval actions may not be determined beforehand.

3.2 LLM-Based Agents Overall Structure

DRAFT: ONLY NOTES + OVERVIEW COMPONENT IMAGE NEEDS TO BE UPDATED

TODO Statistics - distributions across article, possibly interesting statistics: preprints/reviewed, publications dates: only mention, single vs multi-agent, pure LLM vs symbolic (vs Decision transformer??/barely considered), action selection vs generation (programming)?

3.2.1 Agent Core Controller

DRAFT: ONLY NOTES

TODO: brief description that a program component needs to be the "core controller" that connects all the components!

Tasks of the core controller: -Coordination/Brain/Flow, program flow, communication between components/modules, propagate information -storage and application/filling of LLM prompt templates/prompt library (task, additional instructions, context information, demonstrations, roles, output information like format/structure(/attributes/quality criteria/constraints) alignment techniques? (role/profiling, background information) (Role Assignment profile/ specialist role, implicit constraints, induce behaviour) - LLM prompting parameters and model selection (temperature, top-p,

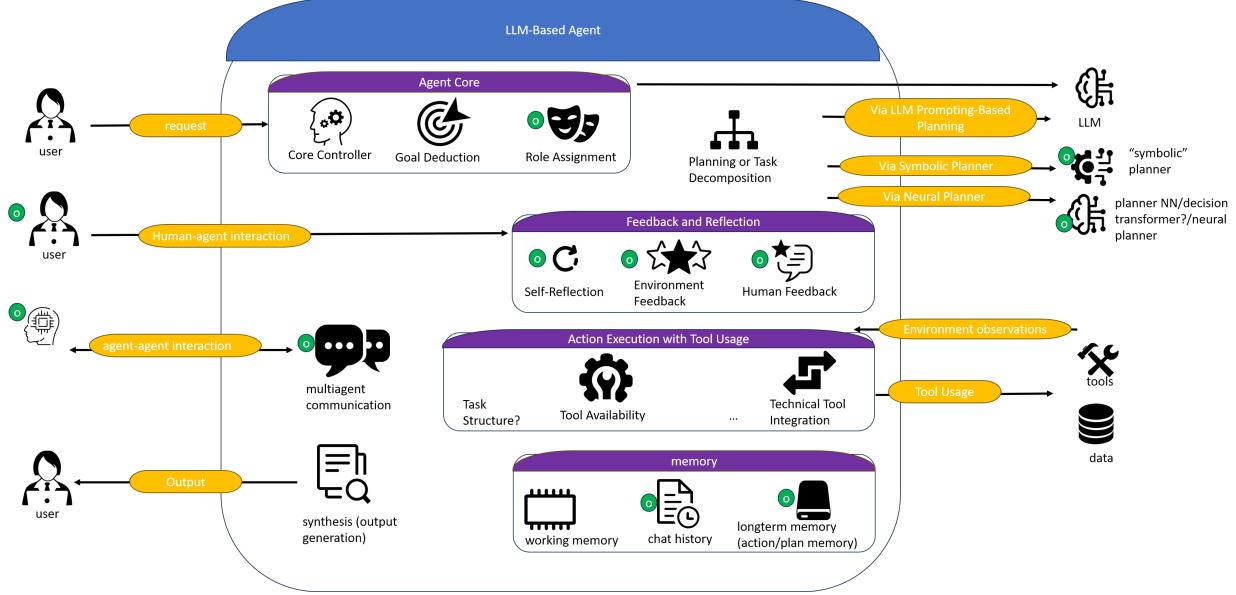


Figure 1: PRELIMINARY PLACEHOLDER: Overview

length) - workflow of reprompting/response handling - Input and Output/user communication handling - memory inclusion/handling - can add procedural knowledge - maybe parse to actions, execute actions (API calls etc), collect results/outputs, store, create tasks, orchestrate/execute, merge/evaluate/synthesise results - decisions about formats, especially for prompting, intermediate results and output (e.g. YAML better than json with LLMs because of tokenization?) - controller tasks and complexity depend on approach - agent control operating procedures - maybe keep track of history/encountered states (in mutliplan approaches keep track which paths already encountered) - human communication, LLM as interface between planner and human user, task human - possible adaption of input: Goal Deduction, user request, task(re-)formulation, What is the actual goal? , extract a planning goal from given a natural language instruction

Inspiration of operating procedures?:

4 Plan Generation

INITIAL FORMULATION

The concept of taking user input as a goal, breaking it down into tasks and solving them step by step is referred to as planning. Planning is a field in AI that deals with devising a plan of action to achieve a defined goal. It is a deliberative process by which an artificial agent reasons about, chooses and organises actions to achieve an objective Ghallab et al. [2016a]. An AI planning system is a central component of an artificial agent, as outlined in Wooldridge and Jennings [1995]. Typically, a factored representation is used, where the world state is represented by a collection of variables, and actions have necessary preconditions and effects related to statesRussell et al. [2010]. In order to plan effectively, it is essential to have a complete and accurate domain representation and a complete, formal problem description. Algorithms to solve planning problems often include forward or backward state-space search, with the use of heuristics, Boolean satisfiability, situation calculus , constraint satisfaction and the refinement of partially ordered plans Russell et al. [2010]. It is not always possible or necessary to create the complete plan before starting execution. Sometimes, a partial plan is sufficient. If the environment is predictable and well modelled or if the actions have a high cost or risk or are not reversible it is advisable to create a complete plan Ghallab et al. [2016b]..

Planning is a form of deliberation. Deliberation is needed when agents need to autonomously perform in diverse environments or diverse tasks to achieve their intended objectives Ghallab et al. [2016b]. This requires a reasoning process which determines the actions to perform, how to execute them and what the result will be. Therefore, agents need some form of predictive model or simulative capabilities Ghallab et al. [2016b]. Ghallab et al. [2016b] describe two important principles of deliberation: hierarchical organization and continual online processing. The paper Ghallab et al. [2016b] outlines two key principles of deliberation: hierarchical organisation and continual online processing, This involves identifying intermediate goals and planning for them, refining planned actions into commands that can be

executed in environment, monitoring the environment and reacting to events, and to compare predicted and observed changes, and possibly searching for recovery actions.

There are several AI planning approaches, each with different assumptions about the planning environment and planning problem (see/compare Russell et al. [2010]). Classical planning assumes a finite number of states and actions, full observability, determinism, and a static environment with a single agent. For partially observable and non-deterministic environments, contingency planning is used, where the plan contains conditional branching based on percepts. In order to reason about partially observable or unobservable environments, belief states can be utilised. In the case of unknown environments or when inaccuracies in the world model are assumed (missing incorrect variables or conditions), online replanning can be employed. This latter approach requires monitoring of action execution in order to replan if necessary. If there are multiple effectors in the world, multiagent planning can be considered.

It would be beneficial to provide a more formal definition of planning here, with an example of classical planning. A planning domain D is a model representing a real-world problem, in order to enable it to be solved by AI planning systems Ghallab et al. [2016a]. D can be based on the standard formalism of a state-transition system (see Ghallab et al. [2016a] p.25 and Cimatti et al. [2003]):

- S : a finite set of states
- A : a finite set of actions
- $\gamma : S \times A \rightarrow S$: a state-transition function
($\gamma(s, a)$ is a partial function because it is only defined if action a is applicable in state s .
If $\exists s' : \gamma(s, a) = s'$, action a is *enabled* or *executable*).
- $S \times A \rightarrow [0, \infty)$: an optional cost function
(It is a partial function that can be used to represent time, monetary cost or similar for a transition.)

A planning problem P in the planning domain D can then be defined as a tuple (D, I, G) , where $I = s_0 \in S$ is the initial state and $G \subseteq S$ is a set of goal states.

A plan is a solution to the planning problem. It describes how to get from the initial state to a goal state by using possible actions. A plan or policy can be defined as follows (Kuter et al. [2005]): Let $A(s) \subseteq A$ be the set of actions a applicable in state s . A plan or policy can be represented as a set of state-action pairs, with each pair describing which action to execute in which state. In classical planning, a plan can be defined by a finite sequence of actions, such as $\langle a_1, a_2, a_3, \dots, a_{n-1}, a_n \rangle$.

Some issues can hinder the applicability of planning to practical real-world problems. One such issue is the transduction problem, which requires a significant amount of effort to create an accurate and adequate symbolic description. Furthermore, the complexity of planning must be considered. In a classical planning problem, plan-existence and plan-length are decidable, with the exemption if function symbols are used, then it is only semi-decidable Ghallab et al. [2004]. The question whether there exists any plan (PlanSAT) for propositionalised problems is in complexity class PSPACE which is more difficult than NP Russell et al. [2010]. However, heuristics can be used to increase efficiency. Although classical planning seems to impose significant restrictions on assumptions about the problem, it can still be utilised in environments that do not fulfil all of these assumptions, provided that the resulting error is deemed tolerable Ghallab et al. [2016a].

A standard encoding language for classical planning problems is PDDL (Planning Domain Definition Language). It descended from the first and highly influential major planning system and language STRIPS (Stanford Research Institute Problem Solver), and the Action Description Language (ADL). The authors of Russell et al. [2010] (pp. 36) provide an example of an air cargo transportation planning problem in PDDL (figures 2 and 3).

```

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK)
     ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2)
     ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
Action(Load(c, p, a),
       PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
       EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
       PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
       EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
       PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
       EFFECT: ¬ At(p, from) ∧ At(p, to))

```

Figure 2: A PDDL description (using predicates) of an air cargo transportation planning problem Russell et al. [2010].

```
[Load(C1, P1, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK),
 Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO)]
```

Figure 3: A possible solution in PDDL as sequence of actions Russell et al. [2010].

4.1 Plan Generation Module

INITIAL FORMULATION

Which methodology should be employed to generate a plan? This section outlines the various methodologies that can be employed to generate a plan. The most popular option is to utilise LLM and apply specific prompting methods developed for reasoning and planning to generate a sequence of tasks as a plan. Another option is to combine an LLM with established symbolic planners. A third option is the use of specially trained neural networks for plan generation.

4.1.1 Pretrained LLM Inference

INITIAL FORMULATION

The LLM module can be used in isolation to generate a plan by decomposing a goal into subtasks. While existing approaches based on prompting are prevalent, few focus on finetuning of planning tasks. Finetuning of pretrained LLMs is not discussed here, but neural networks trained for planning are covered in section 4.1.3.

In prompting, or prompt-based learning, the objective is to provide context information to input that is deemed relevant to solve the task at hand. More formally, a template is employed to modify the original user input x into a textual string prompt x' with unfilled slots. The LLM is tasked with the addition of the unfilled information, resulting in the final string \hat{x} . This string is then used to derive the final output y Liu et al. [2021]. Two distinct types of prompt shape exist: the unfilled slot(s) may be located either in the middle of the text, which is a cloze prompt, or at the end after the entire input, which is a prefix prompt. The choice of this may depend on the model, as described in paragraph 4.1.1.1. The slots are filled by applying the function $f_{\text{fill}}(x', z)$, which searches for possible answers z by calculating the probability of the filled prompts using a pre-trained language model $P(\cdot; \theta)$: $\hat{z} = \text{argmax}_{z \in Z} P(f_{\text{fill}}(x', z); \theta)$ Liu et al. [2021]. The final choice for the slot may be made according to the maximum probability or a sampling-based approach. Furthermore, there can be different answer spaces Z . If Z is an unconstrained space of all tokens, then the filled-in answers z are typically used as the final outputs, as demonstrated in reference Liu et al. [2021]. In the event that the answer space is constrained, whether in the form of a classification task or multiple-choice answers, and particularly when a pre-defined set of actions is available for selection, it is necessary to either directly map the answer to the classes or have the LLM calculate the choices' probabilities.

Prompt-based learning is particularly useful when there is little or no labelled data for specific model fine-tuning. It is also efficient because no parameters are updated and no catastrophic forgetting can occur. Prompt-based learning is also applicable in zero-shot settings, avoiding the need for defined demonstrations Liu et al. [2021].

However, in order to formulate effective prompts, prompt engineering is required. Prompt engineering can be described as the "process of creating a prompting function $f_{\text{prompt}}(x)$ that results in the most effective performance on the downstream task" Liu et al. [2021]. There are specific prompt formulations for reasoning and planning tasks, such as Chain of Thought or ReAct, which will be explained in the following paragraphs.

In addition, there are general approaches to enhance prompting Liu et al. [2021]. Prompt ensembling is a technique where different permutations or orders of the prompt sentences are chosen, with the masked positions altered. Another approach is prompt augmentation or demonstration learning. This involves the provision of a few-shot demonstration of an answered problem. Prompt composition is another technique where the overall prompt is defined based on composing several subprompts. Finally, prompt decomposition is a technique whereby complex prompts are broken down into subprompts, assuming that these are easier to answer separately. This approach is also applied in task decomposition in planning.

It is important to note that other factors can influence the efficacy of prompts, which may not be immediately apparent. For instance, Jin et al. [2024] investigated the impact of reasoning step length on LLM performance and found that simply increasing the number of reasoning steps in prompts can significantly enhance LLM reasoning abilities across multiple datasets. In few-shot settings, even the use of incorrect rationales as demonstrations is beneficial if they exhibit the requisite length of inference. In zero-shot settings, the transition from "let's think step by step" to "let's think step by step, you must think more steps" also resulted in improvements. Nevertheless, the advantage of increasing the number of steps appears to be task-dependent. The greater the complexity of the tasks, the greater the benefit of increasing the number of steps.

In addition, a code-style prompt structure can be implemented instead of relying on natural language prompts. As stated in Sun et al. [2023a], this approach reduces ambiguity and misinterpretation in the prompt, which in turn leads to a reduction in LLM hallucinations in plan generation and refinement. The authors utilise manually designed, pythonic-style coded prompts. Ablation studies demonstrated a substantial decline in performance when textual prompts were not converted to code style prompts. Code style prompts were also applied by Singh et al. [2023] to directly prompt the LLM with program-like, pythonic code specifications of the available actions and objects in an environment. Import statements are employed to define available actions and parameters and sequences of actions operating on objects are described in function definitions. The goal of an action is still described in natural language. The approach demonstrated a notable decline in performance when natural language was employed instead of a programming language structure within the prompts.

4.1.1.1 LLM Backbone Model Choice INITIAL FORMULATION

The question of which LLM to choose as the backbone for an agent when the LLM is used for plan generation must be addressed. Prior to selecting a specific model, it is advisable to select the general agent architecture, which should be the most suitable composition for the task at hand. Once the general architecture has been selected, the specific model can be chosen based on the task and the agent architecture.

There are different LLM architectures, and the fundamental differences between them are briefly explained in the following paragraphs.

- **Encoder-decoder** architectures (**Seq2seq** models) have two transformer block stacks, an encoder and a decoder. In the encoder, input is encoded into latent representations using multi-head self-attention layers. At the output, cross-attention? is used to generate the target output sequence. Encoder decoders are particularly useful for tasks requiring contextual understanding in conjunction with generation. Well known examples are T5 and BART.
- In the **causal decoder** architecture (also **autoregressive** language models) there is no encoder, only a decoder stack with a unidirectional attention mask. Therefore, each input token only attends to the previous tokens, including itself, and the predicted tokens only depend on the previous ones. The goal of pre-training is to predict the next word by estimating the probability distribution of a text given the previous tokens. Causal decoders are strong in text generation tasks. The most prominent examples are the GPT series models.
- The architecture of the **prefix decoder** is similar to that of the causal decoder, but the masking mechanism allows bidirectional attention over input tokens. The output tokens are also generated autoregressively.
- The **encoder-only** architecture (**autoencoder**) is used where only the contextual encoding of the input sequence is relevant, and not the autoregressive generation. Instead, the output is a sequence of embeddings. The goal of pre-training is to correct corrupted data, where input parts are masked and the model has to fill in the blanks. A prominent example is BERT and all its successors.
- The **Mixture-of-Experts** (MoE) approach can be applied to any architecture. Here, the dense feedforward parts are replaced by an MoE layer containing different 'experts' (neural networks) and a router or gate network that decides which expert to route which token to. This allows sparser computation in training and inference, because instead of activating the whole dense layer per token, only the subset of selected experts is activated.

It is considered critical to adapt the prompting method to the architecture of the chosen model, as the training objective is important in determining the applicability of the modes to particular prompting tasks Liu et al. [2021]. For example, prefix prompts should be used with autoregressive models, whereas cloze prompts could be used with bidirectional architectures.

In summary, it may be advisable to first work out the most probably suitable LLM architecture for the task and the overall agent architecture. For example, if the LLM is to do all the plan generation, perhaps a different model needs to be chosen than if the model is to translate textual information into PDDL and call an external planner. If it is possible to compare several models, ablation studies of the overall agent architecture with different LLM models can be conducted to find the most appropriate LLM.

This section includes evaluations that compared multiple LLMs on planning or very similar tasks. The following provides a brief overview of the evaluation studies. The results and comparisons are presented in a concise summary in a table, which facilitates the identification of differences and enables the assessment of the suitability of different models as agent backbones, particularly for plan generation. [[Furthermore, if the approaches examined in this work included comparisons of diverse LLMs as backbone models, the results are also included in the table. The aforementioned approaches are described in their respective sections and not replicated here.]]

Name	Creator/Vendor and License	Architecture and Parameters	Tokens	Evaluation Remarks
GPT-4	OpenAI			\cite{ma_agentboard_2024} leads by wide margin in multi-round action selection \cite{li_api-bank_2023} best in planning, tool retrieval and tool call \cite{nan_evaluating_2023} SQL query planning and execution best performance with wide margin \cite{valmeeekam_planbench_2023} blocksword task plan generation best by margin with 1/3 useful plans \cite{guo_ptc_2023} powerpoint editing tasks very good (best) \cite{wu_smartplay_2023} CoT gameplaying best and close to human baseline
GPT-3.5-Turbo	OpenAI			\cite{ma_agentboard_2024} Turbo/Turbo-16k brauchbar \cite{li_api-bank_2023} good in tool retrieval and call, not in planning
GPT-3				\cite{nan_evaluating_2023} SQL query planning and execution second best performance \cite{valmeeekam_planbench_2023} blocksword task plan generation bad performance
ChatGPT		200B?		\cite{guo_ptc_2023} powerpoint editing tasks very good \cite{ruan_tptu_2023} planning, tool order and usage overall very good one of the best
Text-Davinci-003	OpenAI / Ouyang et al., 2022?			\cite{ma_agentboard_2024} brauchbar \cite{li_api-bank_2023} not good at planning, tool retrieval and tool call \cite{pallagani_understanding_2023} classical planning in PDDL overall poor performances \cite{guo_ptc_2023} powerpoint editing tasks very good \cite{wu_smartplay_2023} CoT gameplaying ganz gute performance \cite{pallagani_understanding_2023} classical planning in PDDL best of overall poor performances
Code-Davinci				\cite{wu_smartplay_2023} CoT gameplaying ganz gute performance
Claude				\cite{ruan_tptu_2023} planning, tool order and usage overall very good one of the best
Claude2	Anthropic			\cite{ma_agentboard_2024} brauchbar
DeepSeek-67b	DeepSeek-AI et al., 2024?			\cite{ma_agentboard_2024} brauchbar
Lemur-70b	Xu et al., 2023b?			\cite{ma_agentboard_2024} 70b, weniger gut especially in tool usage \cite{ma_agentboard_2024} 34b/13b, weniger gut especially in tool usage
Roziere et al., 2023?				\cite{nan_evaluating_2023} 7b/13b/34b, SQL query planning and execution barely useful
CodeLlama	Touvron et al., 2023?			\cite{guo_ptc_2023} powerpoint editing tasks rather not good \cite{wu_smartplay_2023} 13b CoT gameplaying mäßige performance
Llama				\cite{ma_agentboard_2024} 70b/13b, nicht gut und eher unbrauchbar bei tool usage tasks \cite{nan_evaluating_2023} 7b/13b, SQL query planning and execution barely useful \cite{guo_ptc_2023} powerpoint editing tasks not good \cite{wu_smartplay_2023} 13b CoT gameplaying mäßige performance
Llama2				\cite{ma_agentboard_2024} 7b, nicht gut und eher unbrauchbar bei tool usage tasks \cite{nan_evaluating_2023} 7b, SQL query planning and execution barely useful
Mistral	Jiang et al., 2023?			\cite{ma_agentboard_2024} 13b, nicht gut und eher unbrauchbar bei tool usage tasks \cite{guo_ptc_2023} v1.5 powerpoint editing tasks rather not good \cite{wu_smartplay_2023} 13b CoT gameplaining mäßige performance
Vicuna	Chiang et al., 2023?			\cite{guo_ptc_2023} powerpoint editing tasks not good \cite{wu_smartplay_2023} 13b CoT gameplaining mäßige performance
Alpaca				\cite{li_api-bank_2023} 7b, unbrauchbar at planning, tool retrieval and tool call
ChatGLM-6B				\cite{li_api-bank_2023} 6b, unbrauchbar at planning, tool retrieval and tool call
CodeT5				\cite{pallagani_understanding_2023} classical planning in PDDL overall poor performances (whereas a finetuned CodeT5 excels)
CodeGen				\cite{pallagani_understanding_2023} classical planning in PDDL overall poor performances
WizardLM				\cite{guo_ptc_2023} powerpoint editing tasks rather not good
Baichuan				\cite{guo_ptc_2023} powerpoint editing tasks not good
Baichuan-2				\cite{guo_ptc_2023} powerpoint editing tasks not good
Bard				\cite{wu_smartplay_2023} CoT gameplaining ganz gute performance
InternLM	Shanghai AI Lab	120B		\cite{ruan_tptu_2023} planning, tool order and usage overall quite good
Ziya-13B	IDEA			\cite{ruan_tptu_2023} planning, tool order and usage overall mittelmäßig
ChatGLM-130B	Tsinghua University			\cite{ruan_tptu_2023} planning, tool order and usage overall mittelmäßig sehr durchwachsen
Chinese-Alpaca-Plus-33B				\cite{ruan_tptu_2023} planning, tool order and usage overall bad

Figure 4: PRELIMINARY DRAFT - Will be colour-coded to identify best potential, LLM comparison summary of different benchmark articles overview

- Ma et al. [2024] evaluated the performance of various LLM models across a range of tasks in four distinct partially observable, multi-round environments of embodied AI, games, web tasks and tool usage. The LLM's task was to determine the particular best subsequent action.

- In order to evaluate LLM API tool call abilities, Li et al. [2023a] conducted a study in three distinct settings: tool call based on user queries, tool calls + tool retrieval (search), tool call + retrieval + planning. Only GPT-4 could succeed in the last category.
- In their study, Nan et al. [2023] evaluated the reasoning and action of LLM agents in answering database questions. To do so, the LLM agents were required to interact with an SQL interpreter and generate multiple queries. The study employed two different settings. In the sequential setting, the LLM agents acted in a linear sequence of interaction planning, tool employment, and information synthesis. In the iterative setting, the agents alternated between interaction planning and tool employment.
- Pallagani et al. [2023] tried to use LLM reasoning capabilities on plain PDDL domain and problem files of six classical planning problems to devise a plan i.e. find a sequence of actions to achieve the defined goal. Using the plain files with prompting and without finetuning resulted in overall poor performance, whereas LLMs pretrained on code showed slightly better performance and argue that maybe encoder-decoder architectures might be better than decoder only for such tasks
- In their study, Pallagani et al. [2023] sought to utilise the reasoning capabilities of LLM on plain PDDL domain and problem files of six classical planning problems. The LLM objective was to devise a plan, namely a sequence of actions that would achieve the defined goal. They tested an approach only based on prompting and a finetuning approach. Solely relying on prompting resulted in overall poor performance. LLMs pretrained on code demonstrated slightly better performance. Furthermore, the authors argue that encoder-decoder architectures may be more suitable for such tasks than decoder-only architectures.
- The authors of Valmeekam et al. [2023] investigated the claims that LLMs have the ability to reason, particularly in terms of planning and sequential decision-making. This involves creating a policy that leads, by executing it, to a goal state. The article examines the plan generation capabilities of LLMs in three different settings and on simple blocksworld tasks. In the setting where LLMs were required to autonomously generate and validate simple plans on blocksworld, they demonstrated very limited planning abilities, with only 3% of the plans being executable without error and reaching the goal. This was improved in a later repeated experiment with GPT-4, which outperformed all other models and reached more than 34% in plan generation. In a further setting, LLMs were required to develop a plan draft as a heuristic, which was then provided to the local search planner LPG [9]. This approach utilised the flawed generated plan as a seed and could iteratively repair the flaws to identify a correct plan. All the plans generated by this approach were valid. In the final setting, the LLMs draft plan was presented to a human, who could utilise it as inspiration. A greater proportion of individuals who were provided with plan suggestions were able to generate a correct plan than those who were not.
- Guo et al. [2023] created a PowerPoint task completion benchmark with multturn, multi-model instructions to test the ability of LLM systems to create and edit PowerPoint files. The primary challenges and sources of error for the LLMs were the accumulation of errors during the multi-turn session, the lengthy processing of long PPT templates, and the difficulty of multi-modality perception. In general, the results indicated that open-source LLMs exhibited lower performance than closed-source models, and smaller models demonstrated inferior performance compared to larger models. Additionally, it was found that CoT ?? and ToT ??, in conjunction with access to the dialogue history, exhibited a slight improvement in the performance of GPT-4, while ToT did not outperform the relatively simple CoT, despite requiring a significantly greater number of tokens.
- Wu et al. [2023a] tested the ability of LLM to decide the next action in six different games, including Rock-Paper-Scissors, Tower of Hanoi and Minecraft. The researchers employed a direct prompting approach, initially querying “What is the next action to take, let’s think step by step.”. They also provided a manual, the history, and the current observation as context. Subsequently, they queried “Choose the best executable action from the list of all actions. Write the exact chosen action.” to map the chosen action to one of the environment actions.
- In the study Ruan et al. [2023], the authors compared LLM performance on task planning and tool usage in tasks such as SQL query formulation and Python code generation with 12 available tools. Two distinct approaches were tested: One approach is that of the one-step agent, which instead of attempting to solve the entire problem in one go, breaks it down into a sequence of subtasks and generates a plan in an offline manner. By contrast, the sequential agent method involves resolving the current subtasks and generating the next task in an iterative manner. Furthermore, they examined different partial tool usage capabilities, including tool order planning, subtask description generation and the planning of tool and subtask pairs.

4.1.1.2 Prompt Augmentation for Reasoning TODO: INTRO

if giving in prompt more context information in form of demonstrations: prompt-based learning ... In prompt-based learning, the provision of additional contextual information in the form of demonstrations is

4.1.1.2.1 Zero-Shot Planning Prompts INITIAL FORMULATION

Analogous to a standard zero-shot prompt, the zero-shot planning prompt merely contains the task with the directive to create an answer or a plan, without any additional demonstrations of, for example, an exemplary plan with subtasks. In zero-shot planning, no task-specific training data is required, thus avoiding the manual effort of crafting demonstrations. Conversely, there is no possibility of conditioning the LLM on how the intermediate steps, for example in planning, should ideally look like or if the output should exhibit a certain structure demonstrated by examples. Consequently, in zero-shot, the model is more reliant on pretraining data, as no new data is incorporated into the prompt for in-context learning. Zero-shot approaches are adaptable to a wide range of tasks, as no task-specific knowledge is required, and they consume fewer tokens in inference than few-shot prompts, as only the task and some directives are prompted.

A specific prompt to enhance reasoning and planning capabilities without the need for demonstrations is zero-shot chain of thought (CoT), introduced by Kojima et al. [2023]. The objective is to decompose the task into a series of explicit intermediate reasoning steps. This is essentially almost equal to a zero-shot prompting query to an LLM. The prompt template is just the addition of the phrase "Let's think step by step" to the original question. This results in the generation of an intermediate output comprising a series of reasoning steps. Subsequently, the LLM is queried a second time with the initial input prompt and the corresponding output and with "Therefore, the answer is," accompanied by optional information such as the desired output format. The authors demonstrated that the incorporation of these sentences resulted in a notable enhancement in the zero-shot LLM accuracy performance on a range of benchmark reasoning tasks, utilising GPT-3, InstructGPT and PaLM. Furthermore, the listing of intermediate steps seems to provide insight into the reasoning process. It was demonstrated that Zero-Shot-CoT was unable to outperform Few-shot-CoT when the latter was provided with carefully crafted and task-specific step-by-step examples. However, Few-shot-CoT necessitates human engineering and is highly sensitive to examples. The results also demonstrated that CoT was not effective when applied to models with a small size. CoT proved effective on GPT-3 and InstructGPT, with a model size of 170B, and on PaLM, with a model size of 540B. The remaining tested models, all having less than 13 billion parameters (with the exception of PALM, which had 63 billion parameters) were found to be ineffective when using CoT. The authors of Zhang et al. [2023] complement that CoT is effective for LLMs with more than 20 billion parameters and if the LLM encompasses pretraining knowledge relating to the task.

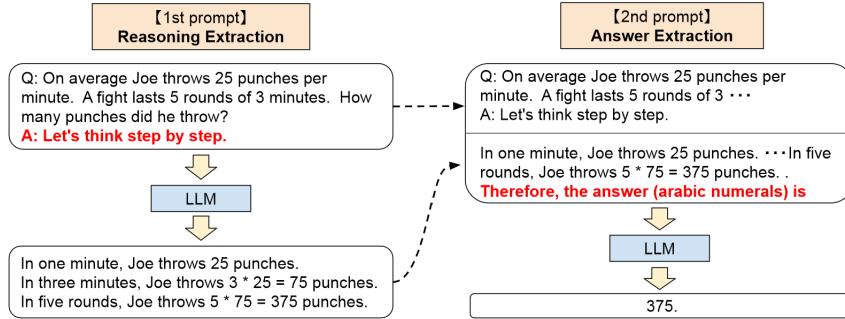


Figure 5: Example of a zero-shot CoT prompt Kojima et al. [2023]

In their approach, Wang et al. [2023a], the authors adapted the concept of CoT and divided the reasoning process into two stages: the generation of the steps of a plan and the subsequent execution of the steps. In order to achieve this, the following sentence was included in the prompt: "Let's first understand the problem and devise a plan to solve the problem. Then, let's carry out the plan and solve the problem step by step." The second step is analogous to the CoT approach, whereby the final answer is prompted for, for instance, in the context of mathematical reasoning: "Therefore, the answer (arabic numerals) is". It is also possible to extend the prompt instructions as follows: "extract relevant variables and their corresponding numerals" and "calculate intermediate results (pay attention to calculation and commonsense)". The results of the study indicate that Plan&Solve prompting outperforms Zero-shot CoT across a range of reasoning test problems. This approach can be extended with self-consistency (SC) ??, to reduce randomness by generating multiple sample results and selecting the majority vote. According to the authors, applying SC significantly enhances performance.

4.1.1.2.2 Few-Shot Planning Prompts INITIAL FORMULATION

The concept of few-shot learning is to enhance in-context learning by providing new data that is not included in the pre-training data of the model. This is achieved without the need to directly train or finetune the model by changing the weights. In-context learning can be applied on a task-by-task basis, as the demonstrations can be tailored to specific tasks. This necessitates either the manual creation of suitable examples or an approach of automatically generating those examples. Few-shot prompting requires a greater number of tokens, as the examples and the original task are appended and then prompted to the model. The provision of numerous demonstrations may consume a significant proportion of the context window size and may also result in an increase in inference time. Consequently, it is not always possible to utilise this approach when extensive demonstrations or a large number of examples are required.

Zhao et al. [2021] conducted research with the objective of improving the few-shot performance of language models. The study demonstrated that three components of a prompt, format, set of training examples and permutation (ordering) of the examples exert a significant influence on the accuracy of GPT-3 and GPT-2. The research identified three biases that can negatively influence performance. The majority label bias implies a greater probability of predicting responses that appear frequently within the provided examples. The recency bias implies a greater probability of predicting responses that are similar to those that appear at the end of the prompt. Furthermore, the common token bias implies a higher probability of predicting answers that are frequent in the pre-training data. The authors employed content-free test prompts to measure the possible bias and subsequently applied a weight matrix and a bias vector to calibrate the output distribution of the original probabilities. This resulted in a consistently improved and stabilised accuracy.

In Wei et al. [2022] the authors introduced the concept of chain of thought prompting and demonstrated its efficacy in enhancing the ability of large language models (LLMs) to perform complex reasoning tasks. The authors describe chain of thought as a "series of intermediate natural language reasoning steps that lead to the final output". This approach is inspired by the human thought process of tackling complex reasoning tasks as a multi-step problem. In practice, each query is enhanced with a few examples of chain of thought intermediate reasoning steps for similar tasks and responses. It should be noted that CoT reasoning only increased performance with large model scales. For instance, for arithmetic

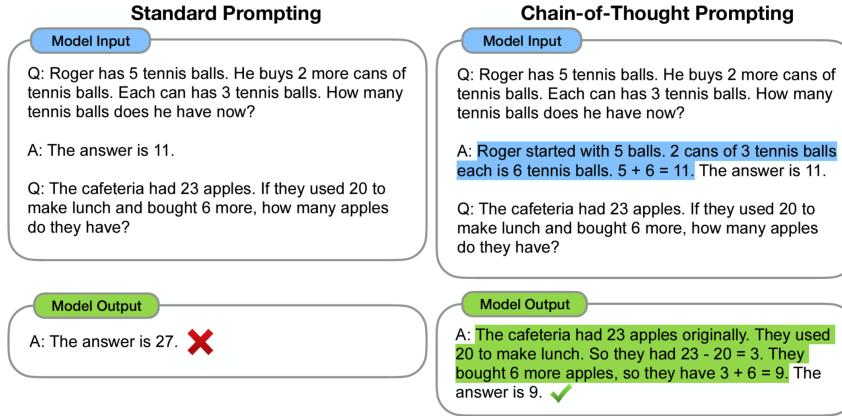


Figure 6: Chain-of-thought prompting example Wei et al. [2022]

reasoning and symbolic reasoning, the required model parameter size is at least 100B. This makes the application of CoT expensive. Furthermore, they found that in smaller models, the production of illogical chains of thought can even lead to lower performance than standard prompting.

In order to enhance the efficacy of CoT, it is beneficial to ensure that the demonstrations align with the actual task. Accordingly, Song et al. [2023] employ task-specific in-context example retrieval. Prior to utilising this approach, it is necessary to have a training set comprising pairs of instructions and examples. The authors utilise a pre-trained BERT-base model to assess the pairwise similarity between the test example and training examples, employing the Euclidean distance between the BERT embeddings of the corresponding instructions. Subsequently, the K most similar examples can be selected for the purpose of augmenting the prompt.

TOTO: ADD Least-to-most prompting <https://doi.org/10.48550/arXiv.2205.10625> Poster presentation at ICLR 2023 without link/doi??

Demonstrations can also contain cues to actively request follow-up questions and gather more information within the stepwise reasoning process, thereby enhancing performance on compositional reasoning. In the study Press et al. [2023], the researchers were concerned with the question of how to measure and increase the ability of compositional reasoning. This means that, in order to reach an overall solution, the answers to the subproblems must be composed.

They found that an increased model size alone does not lead to an improvement in the ability to perform compositional reasoning. Conversely, it has been demonstrated that elicitive prompting, such as CoT, enhances the capacity to perform compositional reasoning by encouraging explicit reasoning. The authors propose the self-ask approach to enhance CoT, which involves prompting the LLM to ask self-questions. In order to achieve this, they include within the demonstrations an initial question, followed by an intermediate question and, where appropriate, a series of follow-up questions and corresponding answers. It is possible that intermediate questions could be answered by the LLM itself, but it is also possible that a tool such as a search engine could be employed.

Question: Who lived longer, Theodor Haecker or Harry Vaughan Watkins?

Are follow up questions needed here: Yes. Follow up: How old was Theodor Haecker when he died? Intermediate answer: Harry Vaughan Watkins was born in 1868 and died in 1940, so he lived for 72 years. Follow up: How old was Harry Vaughan Watkins when he died? Intermediate answer: Harry Vaughan Watkins died at the age of 72. So the final answer is: Harry Vaughan Watkins

The concept of a chain of thought can be expanded to encompass not only valid reasoning demonstrations, but also invalid ones. In Chia et al. [2023], the authors introduced the concept of Contrastive Chain-of-Thought Prompting, which involves providing examples of valid and invalid reasoning, the latter with the idea to demonstrate which mistakes to avoid. To obviate the necessity of manually constructing negative examples, contrastive demonstrations are constructed automatically from existing valid reasoning chains. Invalid examples can be constructed by invalidating valid reasoning, for example by destroying coherence by changing the ordering of steps, or creating object and language incoherence by shuffling object or word positions or introducing irrelevant objects. In their experiments with GPT-3.5-Turbo, the provision of positive and negative demonstrations led to a significant improvement in CoT. The approach is even more effective when combined with self-consistency (see ??).

Question : <demonstration question>

Explanation: <correct demonstration explanation>

Wrong Explanation: <incorrect demonstration explanation>

Question: <actual question>

It is possible to enhance the prompts in simple ways, thus adding specific reasoning capabilities. An illustrative example is the work of Zhang et al. [2024], which demonstrates that a simple prompt can be used to instruct an LLM to conduct a mathematical proof of contradiction. An explanation of the proof is provided in the prompt, which is followed by demonstrations of some applications of the proof.

<Instruction> Proof by contradiction in logic and mathematics is a proof that determines the truth of a statement by assuming its opposite and showing that it leads to a contradiction.

A limitation of few-shot planning prompts is their lack of generalizability, as explained in Gao et al. [2024a]. Demonstrations are made of solutions specific to a given task and do not easily apply to other cases. Furthermore, if several demonstrations are included in a prompt and employ different solution strategies to solve the same task, this can "confuse" the LLM. Consequently, the objective is to derive general strategies from task examples and to "construct generalizable and consistent few-shot prompts for various tasks automatically". To identify strategies for a task, the authors employ LLMs in four distinct roles. The role of strategy generator formulates a pool of task-solving strategies based on task examples:

Task:

{task definition}

Some examples of the task are as follows:

{task examples}

Let's understand the task and write a strategy that consists of a sequence of subtasks to solve the task.

- Include all necessary subtasks.
- All subtasks are easy to solve.
- Subtasks are in an appropriate order.
- Do not include specific information in the examples.
- Make sure the strategy is general and concise. The result must be a numbered list in the following format:
 1. First subtask
 2. Second subtask

The strategy executor applies the strategies to task examples, obtains results, and computes the respective accuracies. In the event that an insufficient number of effective strategies are identified, the strategy optimizer is tasked with refining strategies with low accuracy. Otherwise, the strategy evaluator is tasked with constructing strategy-based

few-shot prompts and assessing the constructed prompts on a validation set. This approach automatically constructs

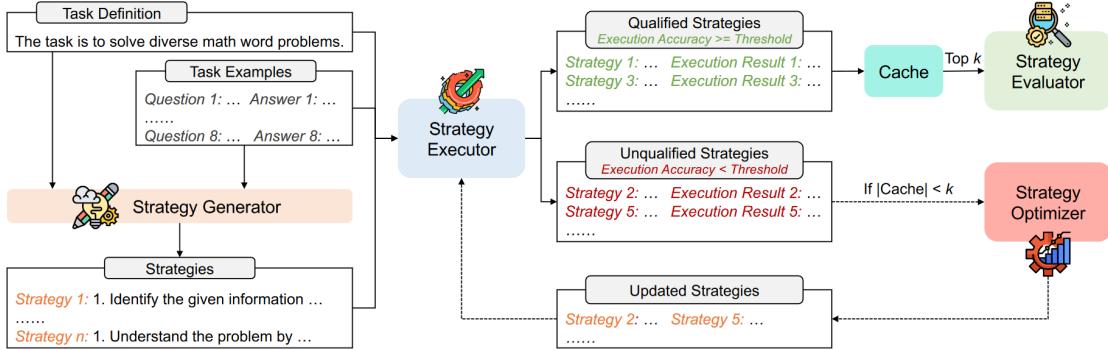


Figure 7: StrategyLLM Gao et al. [2024a]

few-shot prompts and needs no human annotated examples. But it needs more tokens than CoT because generated prompts seem to be often longer than human-written CoT prompts. This approach demonstrated superior performance

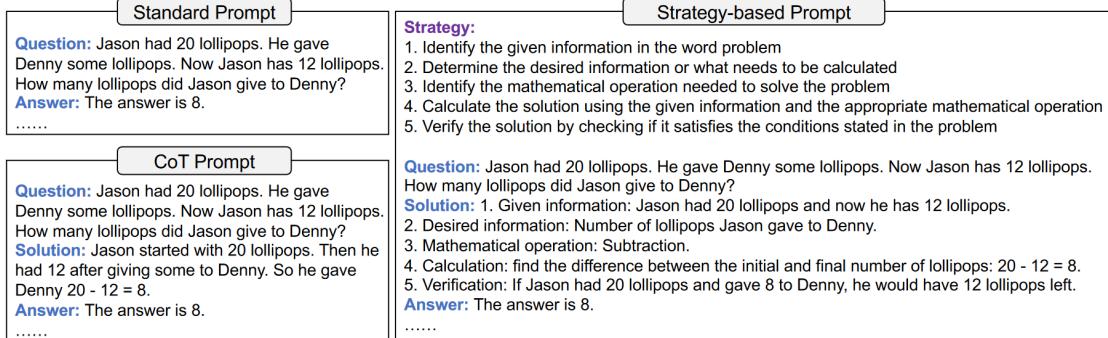


Figure 8: StrategyLLM-StrategyBasedPrompt.png Gao et al. [2024a]

compared to CoT with self-consistency (which necessitates human annotation of solutions) in mathematical reasoning, common sense reasoning, algorithmic reasoning, and particularly in symbolic reasoning. The results were obtained using GPT-3.5-Turbo and GPT-4. GPT-4 yielded superior outcomes. It is notable that there is limited transferability of strategies between different LLMs.

TODO: RELEVANT? Raman et al. [2023] 22.10.2023 planning few-shot?? Planning LLM: demonstration set: high-level example task and respective plan, cosine similarity qith query task, generate actions for task Translation LLM: Sentence-BERT LLM, embed free form actions (nat language) to semantically similar actions in agent's repertoire. step by step: append action, further autoregressive steps

TODO include example from robotic action plan, eg inner monologue Huang et al. [2022a] Example of example plan:

Task: Put objects in their corresponding bowls

```

Scene: Visible objects are ["red block", "green block", "red bowl", "green bowl"]
Scene: Occluded objects are []
Robot action: robot.pick_place("red block", "red bowl")
Successful action: False
  
```

```

Scene: Visible objects are ["red block", "green block", "red bowl", "green bowl"]
Scene: Occluded objects are []
Robot action: robot.pick_place("red block", "red bowl")
Successful action: True
  
```

```
Scene: Visible objects are ["red block", "green block", "red bowl", "green bowl"]
Scene: Occluded objects are []
Robot action: robot.pick_place("green block", "green bowl")
Successful action: True
```

```
Scene: Visible objects are ["red block", "green block", "red bowl", "green bowl"]
Scene: Occluded objects are []
Robot action: robot.stop()
STOP
```

4.1.1.3 Branching Task Sequences TODO: BRIEF INTRO

4.1.1.3.1 Single Path TODO: INTRO Task: several steps, one path (eg linearly ordered)

TODO: REFER TO THE PROMPTING APPROACHES AND TO APPLICATIONS MENTIONED IN OPEN-LOOP PLANNING

short repetition/reference to CoT: Kojima et al. [2023] and all the improvements, self-ask Press et al. [2023], contrastive CoT Chia et al. [2023]

+ refer to the open-loop planning and some replanning approaches!

4.1.1.3.2 Multi Path with Discriminator DRAFT: ONLY NOTES

TODO INTRO structure, generation + selection/ranking, choices, structures: multiple parallel paths/tree-like/graph, idea: several possible ways to solve problem, can explore more potential solution, select most promising or try several, but needs more computations/more resources, how to evaluate/rank tasks/which to choose? - needs discriminator

APPROACH1: PARALLEL PATHS; TRAINED PLAN RANKER

Logeswaran et al. [2022] 27.05.2022 Few-shot Subgoal Planning with Language Models, multi-path! LLMS can infer detailed subgoal sequences, only need few training examples and no finetuning ALFRED benchmark plan execution: pre-trained low-level policy available and state representation is provided (both assumed). feedback from environment: task success/failure. Use feedback to train a ranking model that re-ranks LLM prediction subgoal inference: semantic parsing. text instruction to sequence of subgoals, predict sequence of subgoals in-context learning Lm predicts several subgoal sequences, re-rank to a ranked list of subgoal sequence, execute next subgoal from highest ranking plan observations: update state representation environment feedback: train supervised plan ranking model based on information from environment/environment feedback/execution of plans and success/failure outcomes of the plans in the environment: construct labelled dataset of instructions with corresponding plans and agent states, significant improvement

Multiple CoTs (CoT-SC): multiple parallel CoTs, select 1, here Self-consistency ?? ref: many prompting approaches can be combined without changes with self-consistency! Wang et al. [2023b] 07.03.2023 title: SELF-CONSISTENCY IMPROVES CHAIN OF THOUGHT REASONING IN LANGUAGE MODELS increased temperature setting, diverse outputs/trajectories increased performance in arithmetic and common-sense reasoning tasks not greedy decoding. samples several reasoning paths from language model via CoT, trying to find most consistent answer: majority vote, significantly improved CoT? "replace the naive greedy decoding used in chain-of-thought prompting" sample diverse set of reasoning paths, select most consistent answer "boosts the performance of chain-of-thought prompting with a striking margin on a range of popular arithmetic and commonsense reasoning benchmarks" "intuition that complex reasoning tasks typically admit multiple reasoning paths" "determine the optimal answer by marginalizing out the sampled reasoning paths to find the most consistent answer" idea: if multiple different ways of thinking lead to answer, more confidence that its correct "The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the "greedy decode" in CoT prompting by sampling from the language model's decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set" directly applicable "One should note that self-consistency can be applied only to problems where the final answer is from a fixed answer set, but in principle this approach can be extended to open-text generation problems if a good metric of consistency can be defined between multiple generations, e.g., whether two answers agree or contradict each other." "find that self-consistency robustly improves reasoning accuracy for every language model" UL2, GPT-3 code-davinci001 and code-davinci-002, LaMDA-137B, PaLM-540B best GPT-3 code-davinci-002 and PaLM-540B "incurs more computation cost", can check how many paths necessary

APPROACH2: TREE + TREE SEARCH

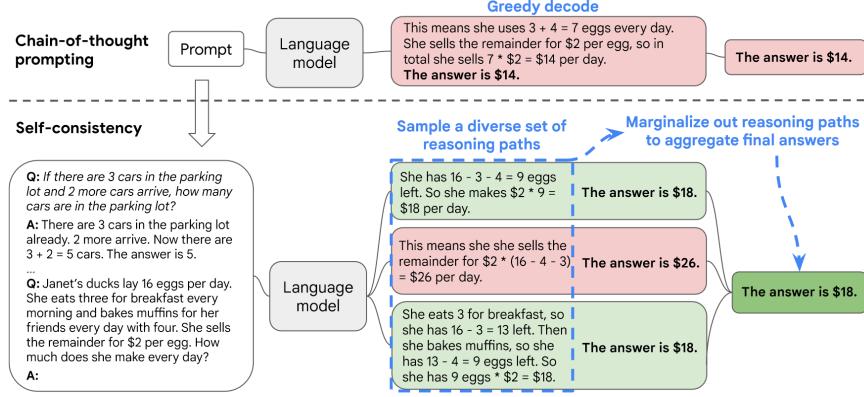


Figure 9: SelfConsistency Wang et al. [2023b]

Tree of thought Long [2023]14.05.2023 Large Language Model Guided Tree-of-Thought explores the solution space through a tree-like thought process, trial and error, backtracking modules: prompter agent, checker module, memory module, ToT controller multi-round conversation with the LLM memory module: allows to save history and backtracking tested: only on Sudoku Puzzle? prompter: encourage LLM o produce intermediate solution, of correct stored in memory

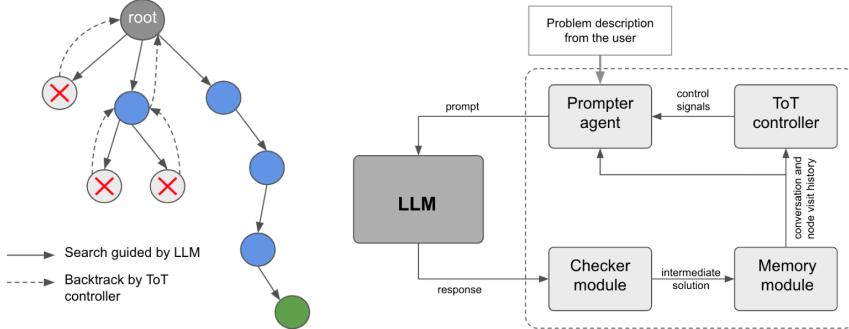


Figure 10: TreeOfThought Long [2023]

prompter agent based on memory: next step if invalid, prompt LLM to consider again controller monitors whether to continue search from current node or backtrack tree search, with LLM as heuristic Checker module: problem dependent, check validity of intermediate solution – assumption that it can be effectively checked!! Memory module: history of conversation between LLM and prompter agent + optional information ToT controller: steers tree search, valid solution: explore children, otherwise backtrack, can alternatively use policy network to determine a backtracking policy Prompter agent: templates for prompting LLM, generic template:

For the given problem: [problem description], we have come up with a partial solution: [partial solution]

problem description and partial solution summary: from memory gpt-3.5-turbo

STATES, ACTIONS, SEARCH, TREE; MCTS Hao et al. [2023] 23.10.2023 RAP Reasoning via Planning, Reasoning with Language Model is Planning with World Model LLM lack internal world model to predict the world state, therefore hard to simulate long-term outcomes of actions deliberate planning: explore alternative reasoning paths, anticipate future states and rewards, iteratively refining reasoning steps LLM as: world model, reasoning agent, build a reasoning tree planning algorithm: based on MCTS LLaMA-33B build reasoning tree world model: future outcomes, backpropagate estimated future rewards 2/4/6-step problems of Blocksworld, significantly more better than CoT state description eg variable values prompt to LLM to generate the next state compared to CoT: augment reasoning steps with predicted world states rewards for applying actions to states (reasoning steps), different possibilities, eg likelihood of action (produced by LLM/probability), confidence of state (LLM), self-evaluation by LLM “Is this reasoning step correct?”, Task-specific heuristics based on world model, can use planning algorithms, here MCTS demonstrations for action likelihood:

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions

Pick up a block

...

I have the following restrictions on my actions:

I can only pick up or unstack one block at a time. ...

[STATEMENT] As initial conditions I have that, the red block is clear, the yellow block is clear, the ...

My plan is as follows:

[PLAN]

unstack the yellow block from on top of the orange block put down the yellow block pick up the orange ...

[PLAN END]

next state prediction:

```
<actions>
<restrictions>
After being given an initial state and an action, give the new state after performing the action.
[SCENARIO 1]
[STATE 0] I have that, the white block is clear, the cyan block is clear, ...
[ACTION] Pick up the brown block.
[CHANGE] The hand was empty and is now holding the brown block, ...
[STATE 1] I have that, the white block is clear, the cyan block is clear, ...
[SCENARIO 2]
...
[SCENARIO 3]
[STATE 0] <state>
[ACTION] <action>
[CHANGE]
```

Tree of Thoughts (ToT): generate from given position and possibly backtrack, scored thoughts Yao et al. [2023a] 03.12.2023 Tree of Thoughts: Deliberate Problem Solving with Large Language Models tree structure node: thought-action pair search with BFS or DFS improvements in specific tasks (Game of 24, crosswords, creative writing) can explore multiple possibilities of reasoning at each step "ToT allows LMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices" enhances problem solving on Game of 24, Creative Writing, and Mini Crosswords significantly outperformed CoT thought: coherent language sequence as

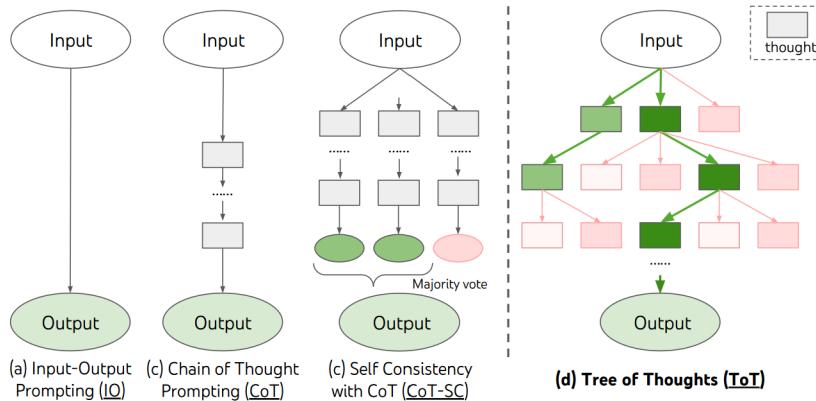


Figure 11: ToTComparison Yao et al. [2023a]

intermediate step , explore different intermediate thoughts implementation of search heuristics via LM self-evaluation

and deliberation LLM. generate and evaluate thoughts, BFS/DFS tree exploration with lookahead and backtracking multiple reasoning paths, problem = search over tree, node = state, representing partial solution generate next thought for next step: Sample thought eg from CoT prompt or propose thoughts sequentially (propose prompt) state evaluator: evaluate progress towards solving the problem (heuristic for search algorithm), two strategies: Value of each state independently (value prompt about scalar value of state, needs ability of LLM to be able to lookahead in this situation or reason via commonsense logic/knowledge inherent), or: vote across states/compare different states and vote for most promising one outperforms CoT and CoT-SC by a very large margin examples of evaluation prompts

```
vote_prompt = '''Given an instruction and several choices, decide which choice is most promising. Analyze
''',
score_prompt = '''Analyze the following passage, then at the last line conclude "Thus the coherency score
'''
```

STATES, ACTIONS, SEARCH Zhou et al. [2023a] 05.12.2023 LANGUAGE AGENT TREE SEARCH UNIFIES REASONING ACTING AND PLANNING IN LANGUAGE MODELS LATS (Language Agent Tree Search), a general framework inspired by monte carlo tree search LLM: agent, value function, optimizer external feedback from environment crucial? self-reflection external memory expand ReAct, outperforms ReAct introduce an LM-based Monte Carlo tree search variant to deliberately construct the best trajectory from sampled actions, using heuristics from LM external feedback and self-reflection: learn from experiences mitigate: autoregressive sampling like in CoT neglecting alternatives Usually MCTS requires environment model ReAct basis: receive observation , take action a following policy base on instruction+ggf few-shot examples ReAct actions: permissible actions and reasoning traces (thoughts) policy: $\pi(a_t|x, o_1 \dots o_{t-1}, a_1 \dots a_{t-1})$ sample n actions, assumption: for complex decision making tasks probably several correct trajectories/reasoning paths + mitigate stochastic text generation + greater exploration MCTS: tree with nodes = states, of input and action and observations sequence state: $s = [x, a_1 \dots a_t, o_1 \dots o_t]$ model-free operations: selection, expansion, evaluation, simulation, backpropagation, and reflection, selection: UCT algorithm expansion: sample n actions, give to environment and get feedback as observation, add child nodes evaluation: LLM reason about state, attribute scalar value for correctness of trajectory simulate: until terminal state is reached, backpropagate value, self-reflection, eg unsuccessful terminal node about errors in reasoning or acting Observe performance gain of using MCTS over variants like DFS higher computational cost for tasks where performance is more important than efficiency (eg programming?) OWN ANALYSIS: valuable if actions take short time and are fully reversible! because several actions are actually executed! , sometimes no roll-back or high costs!

APPROACH3: GRAPH STRUCTURE

Besta et al. [2023] 24.11.2023 Graph of Thoughts: Solving Elaborate Problems with Large Language Models model LLM output information as graph: information/thoughts=vertices, edges: dependencies between vertices networks of thoughts, feedback loops assumption: closer to human thinking could merge chains of reasoning model reasoning process as graph, networked reasoning generalizes CoT and ToT, and also no model updates modular architecture to implement GoT: fine-grained control over individual thoughts and extendable (eg with other prompting ideas?) where tasks can be decomposed into subtasks and solutions merged, GoT seems to outperform CoT and ToT by a large margin and reduce costs compared to ToT can aggregate and generate thought aggregate: several thoughts leading to

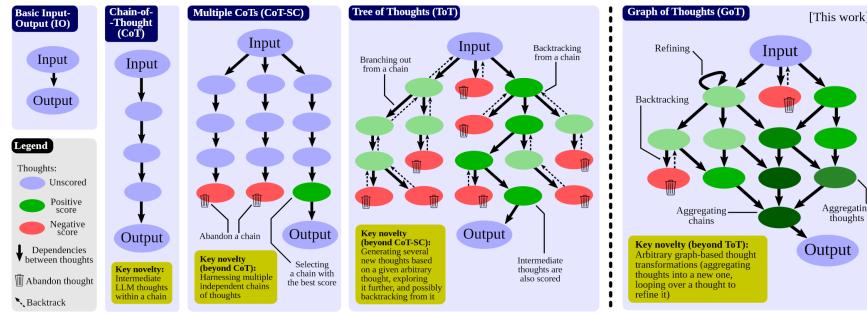


Figure 12: Besta et al. [2023] GoT compared to other single and multiple path prompting strategies

one/incoming to ne, generate: vertex from which several thoughts emerge vertex: (partial) solution, different vertices can model different aspects of the reasoning process can also loop over a thought and enhance it, can merge best thoughts, can aggregate whole paths of reasoning thoughts are scored with a function, can also use function to rank thoughts Architecture: prompter (prepares message), parser (extracts information from LLM reply), scoring module, controller (of reasoning process, decides about progression), controller contains: graph of operations(graph decomposition of task)

and graph of reasoning state /history of reasoning process) examples: good for tasks that can be split and the results

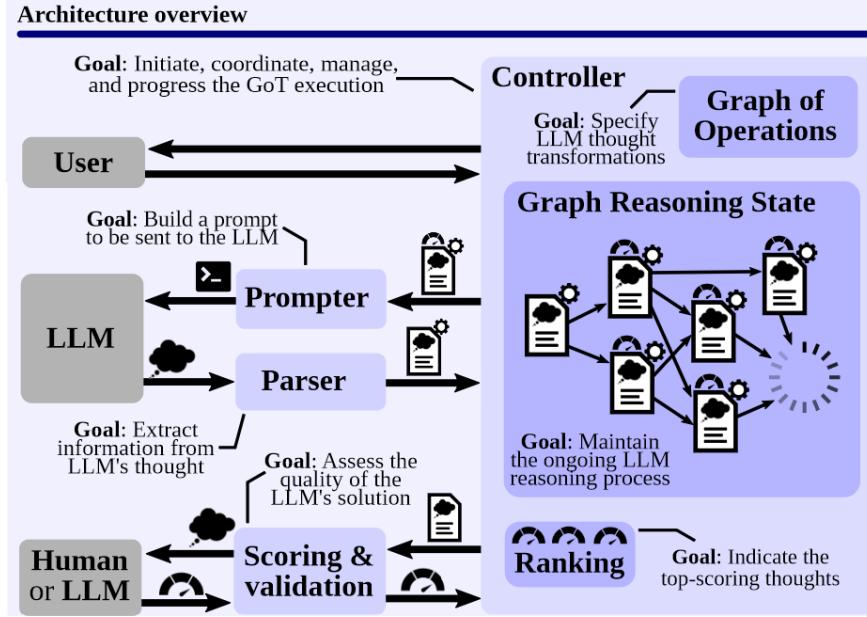


Figure 13: Besta et al. [2023] GoT general architecture overview

merged, like sorting, set operations , keyword counting, document merging GPT-3.5 (also tried Llama-2, but was always worse) additional info: graph-based structure (also: aggregations, self-loops) TODO graphic of the paper comparing different multi-path methods and mentioning the key novelties self-refine, eliminate nodes can follow several paths reduction in error rates for some tasks (sorting, set operations) cut costs?!

STATES, ACTIONS, SEARCH: encoded in decomposition Sel et al. [2023] Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models multi path methods like ToT outperformed single path methods but need more queries on goal: avoid too many model queries: monetary, latency, energy consumption –reduce the query counts employed by contemporary multi-query reasoning methods instead of of [PROBLEM, SOLUTION] or [PROBLEM, SUCCESSIVE STEPS TO SOLUTION], present a new structure that covers [PROBLEM, SEARCH PROCESS, SOLUTION] in-context examples patterned after search algorithms, notably depth-first search (DFS) and breadth-first search (BFS), tasks reminiscent of tree search problems key components of tree search algorithms: decomposition of a problem into sub problems including interrelations between subtasks and the complexity of subtasks solution proposal to subproblems gauging promise of a subproblem/potential/promising routes backtracking AoT surpasses other single prompt methods like CoT-/SC prompting and competitive to ToT, but significantly less queries than ToT, but longer input than CoT Disadvantage: authors gave comprehensive demonstration (1 demonstration, but like tree search, decomposing into subproblems and backtracking) of exactly the task the model was supposed to solve and examples how to solve the task by search similar procedure

GENERAL: DISCRIMINATOR Chen et al. [2024] When is Tree Search Useful for LLM Planning? It Depends on the Discriminator multi-step LLM planning iterative generation/generator: at each step, search for possible next action, generate language representations discrimination/discriminator: predict outcomes of actions (rewards, correctness) planning: incorporate outcome into problem-solving process, strategy to find best action sequence planning methods tested: re-ranking, iterative correction, tree-search "te discriminators with different levels of accuracy. The simulation experiments exhibit a strong correlation between discrimination accuracy and overall task performance among all three types of planning methods. Then, in a non-oracle setting, we closely investigate the LLM-based discriminators and show how environmental observations can effectively improve them" "Advanced planning methods, i.e., iterative correction and tree search, demand highly accurate discriminators ($\geq 90\%$ accuracy) to achieve decent improvements over the simpler method, re-ranking." "compared to the other two methods, tree search is at least 10–20 times slower but leads to negligible performance gains" compare: -generate and rerank (generator generates complete action sequences, discriminator scores them) -iterative correction (generator provides complete plan/action sequence to discriminator, discriminator provides feedback, several rounds of correction possible, here: also scoring and revision o best plan, zero-shot instruction for plan revision) -tree search (generator proposes new steps for current best partial plan, discriminator evaluates steps and updates tree, here MCTS like implementation) generator: CodeLlama-13BInstruct discriminators:

CodeLlama-7B-Instruct and CodeLlama-13B-Instruct, closed: GPT-3.5-Turbo (OpenAI, 2022) and GPT-4-Turbo (OpenAI, 2023), and (3) fine-tuned LLMs: CodeLlama-7B-Instruct-FT and CodeLlama-13B-Instruct-FT Observation from experiments: "Advanced planning methods demand highly accurate discriminators." "Advanced planning methods may not adequately balance accuracy and efficiency. By calculating the average inference time per example (Figure 3), we find that our implementation of tree search is at least 10–20 times slower than the other two planning methods" "Monte-Carlo tree search can be unstable, especially in the early stages" LLMs as discriminators: "closed source LLMs exhibit stronger discrimination abilities, with GPT-4 achieving the best performance", next finetuned CodeLlama-13B-FT and GPT-3.5-Turbo, but CodeLlama-13B/7B at least not that far behind propose to add program executability checks into feedback (score 0 if not executable), notable improvement with such environment feedback observe more than 50% discrimination errors (eg high ranking of not best choice etc) "For these reasons, iterative correction and tree search cannot gain decent improvement over reranking with the same LLM-based discriminator." Me: seems that decisive criterion in these tree based approach is a good discriminator 8heuristics/ranking), not the tree search method itself!!

4.1.2 Planning via Symbolic Planner

DRAFT: ONLY NOTES

INTRO NOTES: avoid the usual huge drawback of symbolic approach in model building, theoretical advantages: completeness, interpretability, stability?... But: translation..errors... then with only small error no planning, translation of vague text into symbolic model, potentially ambiguity, when having constraints, all respected but also all have to be declared explicitly

mention? Elaboration Tolerance McCarthy98 ->small change in problem -> small change in representation???

How to combine LLM+symbolic approach? - is taken as translation task: translate natural language into PDDL, call solver, What to translate into PDDL: Goal (need domain + problem file), problem file, all

SIMPLEST APPROACH: ONLY TRANSLATE GOAL TO PDDL Xie et al. [2023] 10.02.2023 Translating Natural Language to Planning Goals with Large-Language Models "recent work has also shown that LLMs are unable to perform accurate reasoning nor solve planning problems," more effective at translation than at planning "central question is whether LLMs are able to translate goals specified in natural language to a structured planning language" LLM as interface between planner and human user GPT-3.5 for our experiments, specifically code-davinci-002 for all Blocksworld tasks and text-davinci-003 for ALFRED-L "variants show that LLMs are much better suited towards translation rather than planning" under-specified goals in natural language ability to generate goals depends on domain (hard in numerical or physical reasoning) and sensitive to prompts extract a planning goal from given a natural language instruction udr effective classical planners translation from natural language to PDDL, needs: contextual information, prior knowledge + needs correct syntax of PDDL, and compatibility with domain and problem information Blocksworld, ALFRED prompt must consist of: Domain PDDL, One-Shot Problem PDDL, One-shot Language Instruction, One-shot Goal PDDL, Test Problem PDDL, Test Language Instruction ONLY GENERATES THE GOAL!!! very sensitive to prompt, eg if Blocksworld positions are described from top to bottom instead of bottom to top partial goal specification: LLM able to apply commonsense reasoning to fill gaps (with daily/common objects and relations) tests to test understanding of the domain showed that probably LLM not able to understand physical world semantics of predicates works mainly through goodl and many example prompts and therefore linguistic capabilities

CREATE MOST OF PDDL PROBLEM FILE/ TRANSLATE GOAL AND STATES Dagan et al. [2023] 11.08.2023 planning via neuro-symbolic+ online replanning context window vs complex plans with multi-step reasoning planning: effects of actions, states, goal state symbolic planners: optimal solution BUT need planning problem complete and accurate description LLMs: noisy observations, uncertainty LLM Dynamic Planner (LLM-DP): a neurosymbolic framework, LLM+traditional planner ALFWORLD LLMs: hallucinations (incorrect or spurious information) + dependent on prompt formulation LLM iterated invocations: high computational costs symbolic planners: if domain and problem description existent, efficient in optimal plan search but descriptions hard to get, sometimes not existent LLM. understands actions and impact on environment, planner finds solution LLM-DP: high level process LLM observations + natural language instructions parse into PDDL, generates predicates for objects (semantic and pragmatic inference), sample possible predicates: multiple plans possible. Action selector decides about action or review problem or as questions tracking of world state W and beliefs B about predicates in environment W World all known information beliefs: set of possible valid predicates, can be true or false With W and B construct different planning problem PDDL files with: objects observed, current state and goal LLM samples from B, Known world state + options for predicate to LLM, LLM completes the state random sampling vs LLM sampling: same average accuracy! use traditional solver (here BFS(f))(Lipovetzky et al., 2014, plan = sequence of actions (symbolic) (=classical planning) LLM: translates task description into goal state, sample beliefs to generate plausible world states Assumptions: -Need PDDL domain file: available actions with pre- and post-conditions and existing predicates -perfect observations -causal environment (static, inertial?, only changed by agent) -valid actions always succeed gpt-3.5-turbo-0613 LLM model with a temperature of

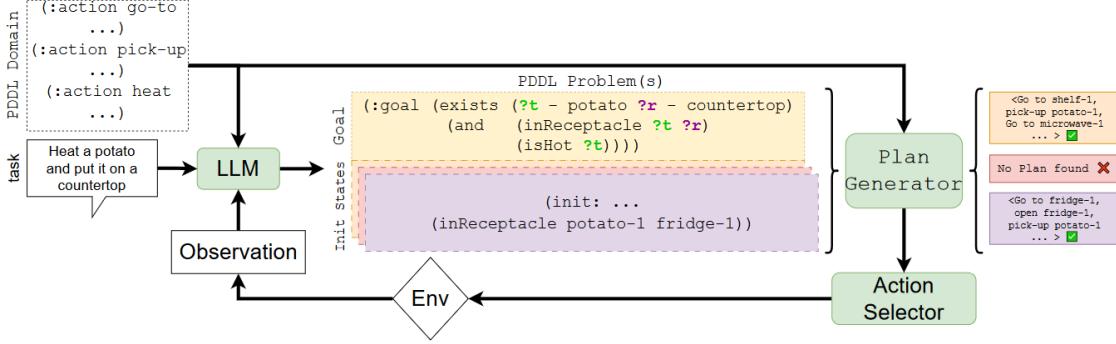


Figure 14: LLM-DP Dagan et al. [2023]

0 compared to ReAct significantly higher average accuracy and most significantly/extremely less LLM tokens used (LLM-Dp 633k vs on ReAct implementation 9.16M) Action selector: planner's output is set of plans. Select action from the plans. here: select actions from shortest plan if no valid plans found: could use self-reflection to update beliefs, or human-agent interaction Observations: symbolic action effects lead to change of state representation + observation to update state and beliefs, new information: trigger re-planning process One of the few-shot examples for generating the goal

```
Your task is to: put a clean plate in microwave.
(:goal (exists (?t - plate ?r - microwave)
  (and (inReceptacle ?t ?r)
    (isClean ?t) )))
```

excerpt from predicate definition for generating the goal

```
(define (domain alfred)
  (:predicates
    (isReceptacle ?o - object) ; true if the object is a receptacle
    (atReceptacleLocation ?r - object) ; true if the robot is at the receptacle location
    ...)
```

TRANSLATE WHOLE PROBLEM FILE TO PDDL transform planning problem to PDDL planning problem file Liu et al. [2023a] 27.09.2023 LLM+P: Empowering Large Language Models with Optimal Planning Proficiency long-horizon robot planning problems vs classical planners: efficient search algorithms issue: need formal problem description input: natural language description of planning problem, returns correct or optimal plan in natural language convert into PDDL, use classical planner, translate back LLM+P in robotic domains can often find optimal solution vs pure LLM which most times fail to produce feasible plans Why not use specialised tools like symbolic planners , they are available, connect LLM to planner as tool not altering LLM classical planning problem, plan = sequence of actions, PDDL, standardized encoding of classical planning problems assumption: domain rules are available, needed: already existing PDDL planning domain file! "we assume that for each problem domain, a human expert can provide a domain description (i.e. action preconditions and effects)" assumption: LLM may be bad at planning but good at translating natural language input to PDDL formal format, like a machine translation task GPT-4, even without prompt engineering almost useable use in-context learning, demonstrations, give an example of natural language problem description and its PDDL representation, then GPT-4 able to directly translate to PDDL (in article they use only simple blocksworld and very comprehensively formulated problem description!) LLM alone: lacks reasoning about preconditions, cannot keep track of properties, large number of LLM calls in long-horizon problems LLM+P can produce optimal plan (due to solver!), but needs demonstration to produce valid PDDL file

GENERATE PDDL DOMAIN AND PROBLEM FILE !!! NOT ONLY translate goal to PDDL requiring handcrafted PDDL domain model, but also generate domain model: Guan et al. [2023] problems with pure LLM planning: limited correctness of plans, strong reliance on environment feedback, inefficiency in using human feedback here: construct explicit world/domain model in PDDL, can then use domain-independent planners "involves maintaining an explicit world model instead of directly mapping user prompts to plans" LLMs as interface between PDDL + sources of feedback (PDDL validators, humans) human involvement: at beginning in supporting domain model generation, not needed later for feedback on planning GPT-4 LLM may overlook physical plausibility of actions, not effective in long-term dependencies LLMs still not reliable planners (to produce correct and executable plans) improvement by

An Example PDDL Problem File Written by GPT-4 with Context

Context (shaded):

```
An example planning problem is: You have 5 blocks. b2 is on top of b5. b5 is on top of b1. b1 is on top of b4. b3 is on top of b2. b4 is on the table. b3 is clear. Your arm is empty. Your goal is to move the blocks. b4 should be on top of b3. The problem PDDL is:
(:objects b1 b2 b3 b4 b5)
(:init (arm-empty) (on b1 b4) (on
b2 b5) (on b3 b2)
(on-table b4) (on b5 b1)
(clear b3))
(:goal (and (on b4 b3)))
```

Prompt: Description of Problem (P1) + Context
+ Provide me with the problem PDDL file that describes the planning problem directly without further explanations.

GPT-4 (the generated problem PDDL):

```
(:objects b1 b2 b3 b4 b5 )
(:init (arm-empty) (on-table b1)
(on b2 b1) (on b3 b4)
(on b4 b2) (on b5 b3) (clear
b5))
(:goal (and (on b1 b2) (on b3 b5)
(on b4 b1)))
```

Planner:

```
(unstack b5 b3) (putdown b5)
(unstack b3 b4) (stack b3 b5)
(unstack b4 b2) (putdown b4)
(unstack b2 b1) (putdown b2)
(pickup b1) (stack b1 b2) (pickup
b4) (stack b4 b1)
```

Figure 15: LLM+P Generated Problem Liu et al. [2023a]

feedback: need simulator or fully online, during executing the plan model-based paradigm instead actions + brief natural language description, LLMS: extract symbolic representation to PDDL action model human feedback: translate PDDL in natural language, get natural language feedback, translate back to PDDL GPT-4 was able to generate high-level domain models translate user instruction to PDDL goal specification can also use in other way: PDDL model to validate plan generated by LLM, provide corrective feedback (preconditions), PDDL model= "simulator" skill library, skill and short description, low-level control policies of skills available generate sequence of high-level skills PDDL: standard encoding language for classical planning problems Prompt to translate the domain:

```
<Instructions for the PDDL generation task>
You are defining the preconditions and effects (represented in PDDL format) of an AI agent's actions.
<One or two examples from other domains for illustrating the input and output formats>
Here are two examples from the classical BlocksWorld domain for demonstrating the output format. ...
<A natural language description of the domain>
Domain information: The AI agent here is a household robot that can navigate to various large and normal
<A natural language description of the action>
Action: This action enables the robot to toggle small appliances (like humidifiers and light bulbs) which
<The dynamically updated list of predicates>
You can create and define new predicates, but you may also reuse the following predicates: 1. (robot-at
Parameters:
```

iterate over generation to identify preconditions feedback sources: PDDL model validation tools (e.g., the one in VAL [18], human domain experts Generating plans using PDDL model: 2 options: 1) Classical planner with LLM-acquired PDDL model, eg Fast Downward 2) LLM modulo planner backprompted by VAL using LLM-acquired PDDL model; iteratively refine LLM plan through feedback from "simulator" and re-prompting "GPT-4 can produce high-quality PDDL models with significantly fewer errors when compared to GPT-3.5-Turbo" LLMs lack reasoning skills of causal

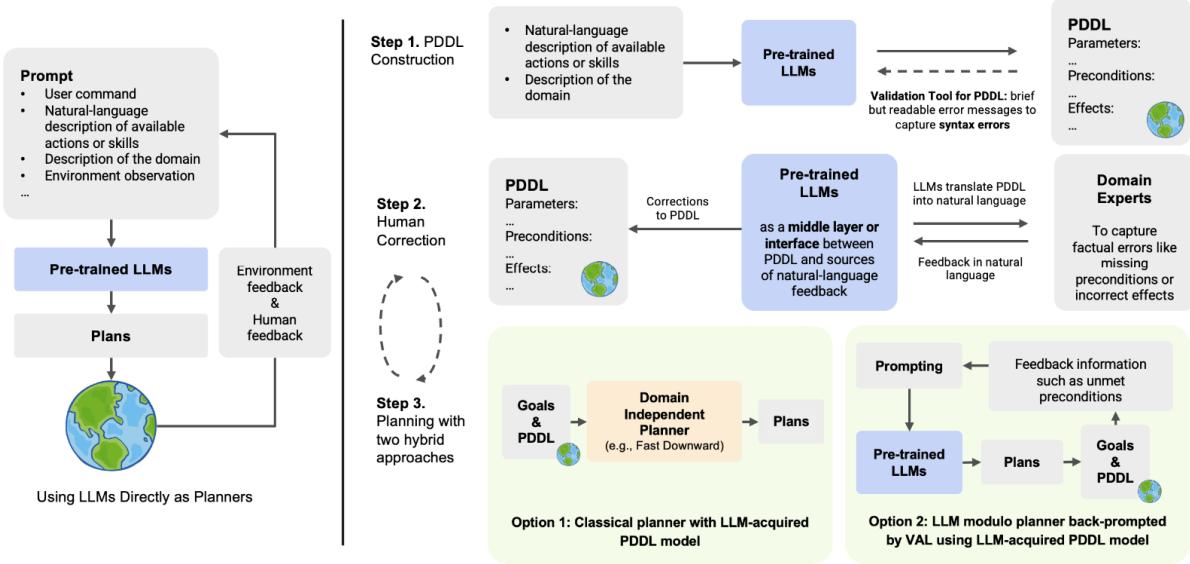


Figure 16: PDDLT ranslationFramework Guan et al. [2023]

relationships when constructing PDDL also feedback/correction dependent on GPT-4, 3-5 not that useful Comparison of success rates in household and logistics domain: Only LLM planner barely able to find executable plans, LLM backprompted by VAL using LLM-acquired PDDL model significantly better, but approach of Fast Downward with LLM-acquired PDDL model finds feasible plan in 95%-100% successful advantages of this approach: safety concerns of LLM plans? Human can correct domain model, improved explainability construct PDDL action by action because of restricted context window size

TODO: Check: Romero e. a. Synergistic Integration of Large Language Models and Cognitive Architectures for Robust AI: An Exploratory Analysis 2308.09830v3

4.1.3 Planning via Neural Planner

DRAFT: ONLY NOTES

Gur et al. [2023] WebAgent: introduce own model HTML-T5 "HTML-T5, a domain-expert pre-trained language model, for task planning and conditional HTML summarization" captures syntax and semantics of HTML pages, "introduce self-experience supervision, where the domain-expert language models are finetuned with self-generated demonstrations" specialized model for HTML, finetuned with environmental feedback to remove critical failures finetuning for planning and summarization web automation good example for planning interaction with pages for task fulfilment often requires long-horizon planning, depending on use case, they: real-estate search 20 steps per episode describe that with supervised experience, programming and summarization errors decreased, but planning especially long-horizon remains difficult domain-expert language model + self-experience data finetuned for planning, predicts next subinstruction given instruction history and current HTML data Eval: surely perfect for the use case web automation, where page structure must be understood dynamically

TODO: ADD? SwiftSage Lin et al. [2023], adds offline imitation learning, BUT: needs oracle paths!! general idea of a Decision Transformer Chen et al., 2021 <https://arxiv.org/abs/2106.01345> Combine language model for action generation and RL model for choosing/evaluation: CALM Yao et al., 2020 <https://arxiv.org/abs/2010.02903>

4.1.4 Planning Approaches

DRAFT: ONLY NOTES

4.1.4.1 Classical Open-Loop Planning TODO INTRO: offline (complete, immutable plan) also called open-loop systems pre-determined plans, do not incorporate (online) feedback, no adaptation simpler? computationally cheaper?

can lead to suboptimal or failed plan without feedback no feedback, so plan and output only dependent on initial input and plan, not considering observations/environment feedback, entire plan predetermined, static

TODO Short repetition CoT etc

TODO ADD ReWOO:Decoupling Reasoning from Observations for Efficient Augmented Language Models, Xu, 23.05.2023 Xu et al. [2023]

Shen et al. [2023] 25.05.23 HuggingGPT, Offline Plan Generation AI tasks, different domains, modalities models for specific tasks how to handle complicated task? LLM as controller for different AI models ChatGPT LLM: task planning, select model (according to function description in HuggingFace) to execute subtask summarize response complex AI/ML tasks consisting of multiple subtasks, requiring different specialized models, expert models (e.g. finetuned to specific tasks) Task planning: user request to intention and decompose into subtasks model selection: Hugging Face model based

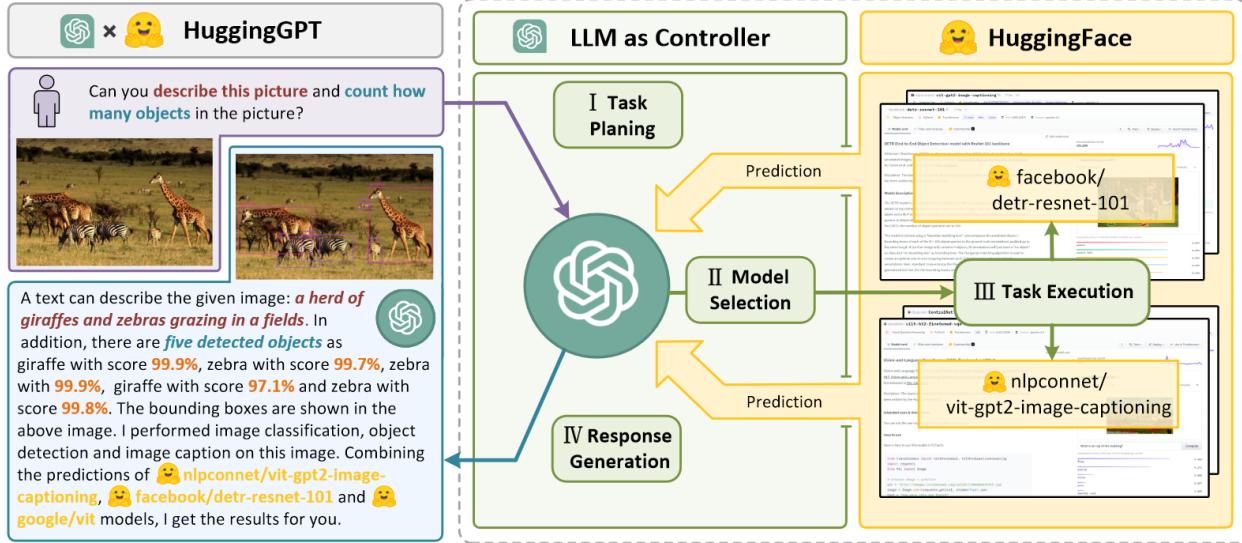


Figure 17: HuggingGPT Shen et al. [2023]

ion their description task execution: invoke and execute model, results to LLM response generation: integrate all models answers translate task into intention and decompose into several ML subtasks, parse into task list, determine task format: `[{"task": "task", "id": task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": dep}}`: dependency with id of previous task Prompt asks the LLM to generate task list in specific format and provides examples of task lists (demonstrations) to queries execution order, resource dependencies among tasks JSON format for all formal specifications, templates global planning, not iterative iterative: more queries, danger of ending in endless loop globally: relies on correctness and executability of each step clear pipeline with 4 steps par/excerpt from the planning prompt:

```
#1 Task Planning Stage - The AI assistant performs task parsing on user input, generating a list of ta
```

type of task, id for dependencies, dep dependencies/prerequisite tasks, args: text/image/audio resources for task execution example demonstration:

```
Can you tell me how many objects in e1.jpg? [{"task": "object-detection", "id": 0, "dep": [-1], "args": {}}
```

GPT models: gpt-3.5-turbo, text-davinci-003 and gpt-4 variants, OpenAI API GPT-3.5 showed better planning capabilities (eg dependencies of tasks) than Alpaca-7b and Vicuna-7b stable outputs: decoding temperature to 0 JSON format: set the logit_bias to 0.2 on the format constraints (e.g., "" and "") needs reliability of LLM, multiple interactions with LLM, limit of maximum token length (eg to incorporate all model descriptions), instability (reliability of LLMS)

TODO ADD: Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models which is really similar to HugginGPT! Lu et al. [2023]

Gao et al. [2024b] Efficient Tool Use with Chain-of-Abstraction Reasoning, Offline planning, see also actions Chain-of-Abstraction CoA/planning with abstract chains: decode reasoning chains with placeholders, then call domain tools: reify reasoning chain/fill in specific knowledge, eg use calculator on placeholder variables LLM fine-tuned to generate

abstract multi-step reasoning chain with placeholders on a user question. Call external tools to reify step with domain specific knowledge and fill in placeholders generate chain of reasoning once. Then add knowledge (other approaches rather interleaving) experiments show seems especially beneficial with long reasoning chains

4.1.4.2 Iterative Task Generation TODO INTRO: recursive prompting, re-prompting closed-loop systems environment feedback, monitoring, adjustment/adaptation/refine flexibility when to update plan? after each action execution and receive environment feedback? iterative, thought-action-observation interweaving (e.g. ReAct) also called autoregressive chaining? add LLM generated output to next input/to context window

can either change only one single action based on the feedback, refinement, integrate information for better action grounding, then generate future actions based on this updated context or refining the entire plan (see next section), potentially changing up to all future actions, env feedback directly used to revise entire plan

Ahn et al. [2022] 16.08.2022 SayCan -see also tool availability via descriptions LLM not physically grounded in environment, cannot observe physical consequences of actions LLM: high-level semantic knowledge, complex, temporally extended goals/instructions robot: "hands and eyes", low-level skills, combine: need feasible and contextually appropriate actions subtask decomposition: only reasonable if: context knowledge about robot capabilities, current state and information about environment LLM interpret instruction + likelihood evaluate how a skill can help progress towards goal + affordance function) how likely it will succeed: LLM "aware" of capabilities + explainable reasoning/choosing of steps/actions Affordance functions: learned by RL grounding via affordances: leads to doubled performance on robotic tasks set of skills and affordance function for probability of completing skill successfully from state s LLM should propose optimal skill multiply LLM probability of skill being useful for high-level instruction with affordance function of successful execution probability break down high-level instruction into sequence of available low-level skills set of skills, skill: policy (e.g. learned), value function (TD/RL), short language description (e.g.g "pick up the can") algorithm SayCan, TODO: as algorithm not image condition policies on language: pre-trained large sentence

Algorithm 1 SayCan

Given: A high level instruction i , state s_0 , and a set of skills Π and their language descriptions ℓ_Π

```

1:  $n = 0, \pi = \emptyset$ 
2: while  $\ell_{\pi_{n-1}} \neq \text{"done"}$  do
3:    $C = \emptyset$ 
4:   for  $\pi \in \Pi$  and  $\ell_\pi \in \ell_\Pi$  do
5:      $p_\pi^{\text{LLM}} = p(\ell_\pi | i, \ell_{\pi_{n-1}}, \dots, \ell_{\pi_0})$                                  $\triangleright$  Evaluate scoring of LLM
6:      $p_\pi^{\text{affordance}} = p(c_\pi | s_n, \ell_\pi)$                                  $\triangleright$  Evaluate affordance function
7:      $p_\pi^{\text{combined}} = p_\pi^{\text{affordance}} p_\pi^{\text{LLM}}$ 
8:      $C = C \cup p_\pi^{\text{combined}}$ 
9:   end for
10:   $\pi_n = \arg \max_{\pi \in \Pi} C$ 
11:  Execute  $\pi_n(s_n)$  in the environment, updating state  $s_{n+1}$ 
12:   $n = n + 1$ 
13: end while

```

Figure 18: SayCan algorithm Ahn et al. [2022]

encoder language model. generate embeddings by passing in Text descriptions of each skill - this is input to policy and value function LLMs: PaLM 540B can additionally add CoT reasoning to provide an explanation of a task for reasoning Iterative planning: prompt engineering: used 17 examples. Already with 4 provided examples food used explicit numbers between steps improved performance, separate line for each step ("n") spelling errors problematic demo prompt is structured like a dialogue example (1):

Robot: Hi there, I'm a robot operating in an office kitchen.

Robot: You can ask me to do various tasks and I'll tell you the sequence of actions I would do to accomplish

Human: How would you move the water bottle from the table to the counter? Robot: 1. find a water bottle

including CoT example:

Robot: Hi there, I'm a robot operating in an office kitchen.

You can ask me to do various tasks and I'll tell you the sequence of actions I would do to accomplish

The following objects are in the scene: 7up, apple, tea, multigrain chips, kettle chips, jalapeno chip

The following locations are in the scene: close counter, far counter, table, you, trash, bowl.

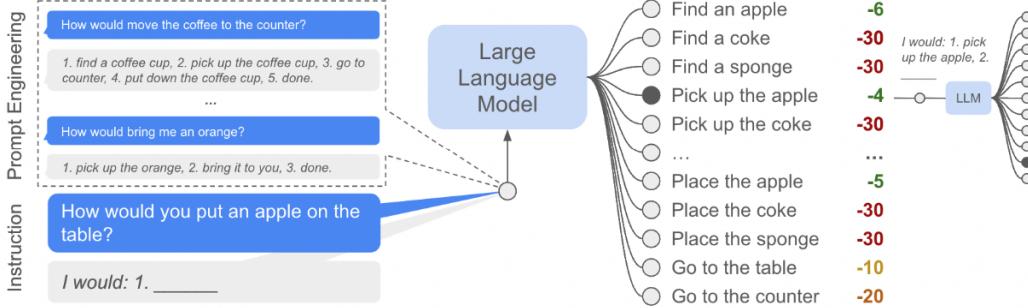


Figure 19: SayCan Scoring LLM, prompt-engineered examples, output: probability of each skill being selected, interatively plan next step: add selected skill to query and query LLM again Ahn et al. [2022]

Human: Bring me something that isn't a fruit

Explanation: The user has asked for something food that isn't an fruit, I will bring an energy bar.
Robot: 1. find an energy bar, 2. pick up the energy bar, 3. bring it to you, 4. put down the energy ba

ReAct: Thought - Action - Observation (=outcome of action) <https://www.promptingguide.ai/techniques/react> ReAct Yao et al. [2023b] ReAct: Synergizing Reasoning and Acting in Language Models Thought-action-observation: combine reasoning and action iterative executor, thoughts+actions, can lead to large history before: rather either reasoning (CoT) or acting, here: together/interleaved "generate both reasoning traces and task-specific actions in an interleaved manner" track and update action plans, handle exceptions interface with environment and gather additional information help overcome hallucination and error propagation in CoT CoT not grounded in external world, CoT internal own model combine reasoning and acting dynamic reasoning, create and adjust high-level plans, interact with environment eg Wikipedia, additional information prompt-based paradigm ReAct in few-shot learning setup interaction setup: "At time step t , an agent receives an observation $o_t \in O$ from the environment and takes an action $a_t \in A$ following some policy $\pi(a_t|c_t)$, where $c_t = (o_1, a_1, \dots, o_{t-1}, a_{t-1}, o_t)$ is the context to the agent. Learning a policy is challenging when the mapping $c_t \rightarrow a_t$ is highly implicit and requires extensive computation." idea: augment action space by language space, action in language space: thought, reasoning trace, thought no effect in environment but reasons and comprises information, reasoning over current context, and updates current context $A = A \cup L$, where L is the space of language. An action $\hat{a}_t \in L$, $c_{t+1} = (c_t, \hat{a}_t)$ PaLM-540B, GPT-3(text-davinci-002, greedy decoding) in-context examples: "human trajectory of actions, thoughts, and environment observations to solve a task instance" trajectory: multiple thought-action-observation steps thoughts not need to occur at every step simple design, use six demonstrations, reasoning process more human interpretable less hallucinations BUT: special reasoning error with ReAct, repetitively generating previous thoughts and actions, and in general increased error rate, successfully retrieving information via search is critical human-in-the-loop possible! can remove some hallucination and add hints Part of the demonstration, as example:

Question What is the elevation range for the area that the eastern sector of the Colorado orogeny extends?
Thought 1 I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny
Action 1 Search[Colorado orogeny]
Observation 1 The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding states.
Thought 2 It does not mention the eastern sector. So I need to look up eastern sector. Action 2 Lookup
...
Action 5 Finish[1,800 to 7,000 ft]

noa Nakajima [2023] intelligent task manager infinite loop pull task from list, execute, enhance outcomes, generate new tasks based on initial objective and outcome of previous task Pulls the first task from the task list. Sends the task to the execution agent, which uses OpenAI's API to complete the task based on the context. Enriches the result and stores it in Chroma/Weaviate. Creates new tasks and reprioritizes the task list based on the objective and the result of the previous task. BabyBeeAGI: enhancements/vendors: dependent tasks, tools like web search

Wang et al. [2023c] 19.10.2023 VOYAGER: An Open-Ended Embodied Agent with Large Language Models lifelong learning agent in Minecraft GPT-4 develops skills: temporally extended, interpretable, compositional three key components: automatic curriculum that maximizes exploration, skill library, iterative prompting mechanism 1) an automatic curriculum that maximizes exploration, description to program, generated by GPT 3.5 text-embedding-ada-002 [51] API for text embedding query library: GPT-3.5 uses pretrained knowledge to add context to a task, then

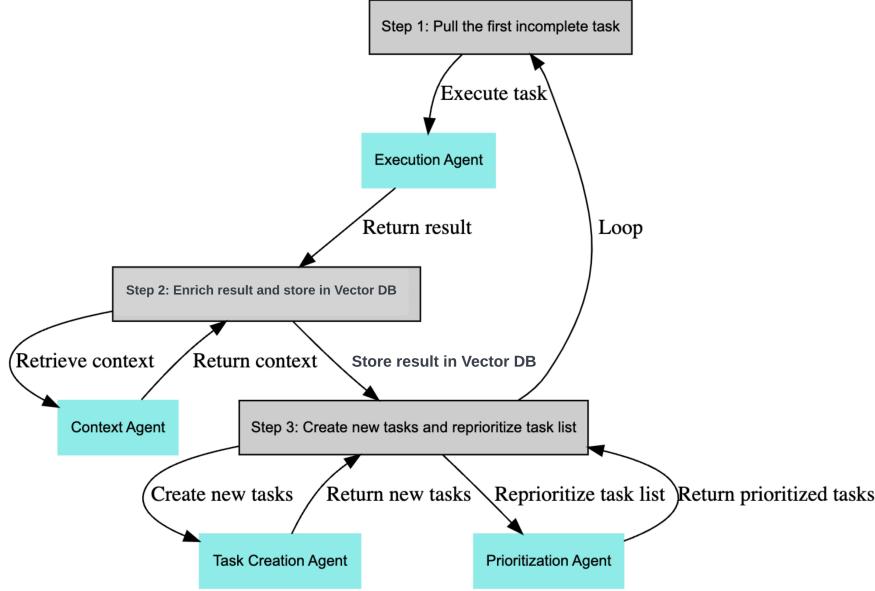


Figure 20: BabyAGI noa

embedding of task used to retrieve relevant skills from library Skill /code generation input to prompt: Guidelines for code generation, APIs and relevant skills from library, generated code from last round with feedback, agent current state in game, CoT prompting for reasoning before code generation 3) a new iterative prompting mechanism that incorporates environment feedback, execution errors, and self-verification for program improvement. code as the action space instead of low-level motor commands 8can represent then temporally extended and compositional actions) -execute a generated program and obtain observations from minecraft environment or error trace of code interpreter - use for GPT-4 prompt feedback for code refinement - refine until self-verification module confirms task completion (then program added to skill library), GPT-4 agent: current state+task, let check whether program achieves task Skill library and feedbacks significant in ablation studies, also very huge difference to using GPT-3.5 for code generation, self-verification most important feedback type is possible to use feedback from humans instead of self-verification module high costs for GPT-4 calls

Du et al. [2024] AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls not directly planning, see actions tools just assumption 16,000 APIs from Rapid API: subset of the APIs can resolve the queries

Add ChatCot? Chen et al. [2023]

4.1.4.3 Online Replanning plan with online re-planning, online planning: integrate planner in action loop of agent, monior execution of plan and revise planning when needed

INTRO- Why beneficial: Touvron et al., 2023? correlation between subtasks and original task? errors in single steps lead to overall failure without replanning iterative/interleaved: environment feedback: better fault tolerance, can lead to longer more complicated trajectories, deviating, repetitions

Inner Monologue: ADD? employs replanning, but the actual process of replanning when receiving feedback is not mentioned! Huang et al. [2022a]

Sun et al. [2023a] 26.05.2023 LLM has two roles: planner and refiner generates initial planning policy for solving the task observation after each action, may modify the actions according to the feedback in beginning high level planning policy generating entire plan, later action-generation policy o a given plan ALFWorld ttext based virtual household environment, MiniWoB++ simulation environment of computer tasks (web tasks) see also Environment feedback REFERENCE in-plan feedback: use information from observation to adapt context knowledge for tasks out-of-plan-feedback: generate revised plan from current step onwards initially provide one exper sample of solution per task part of the refinement prompt:

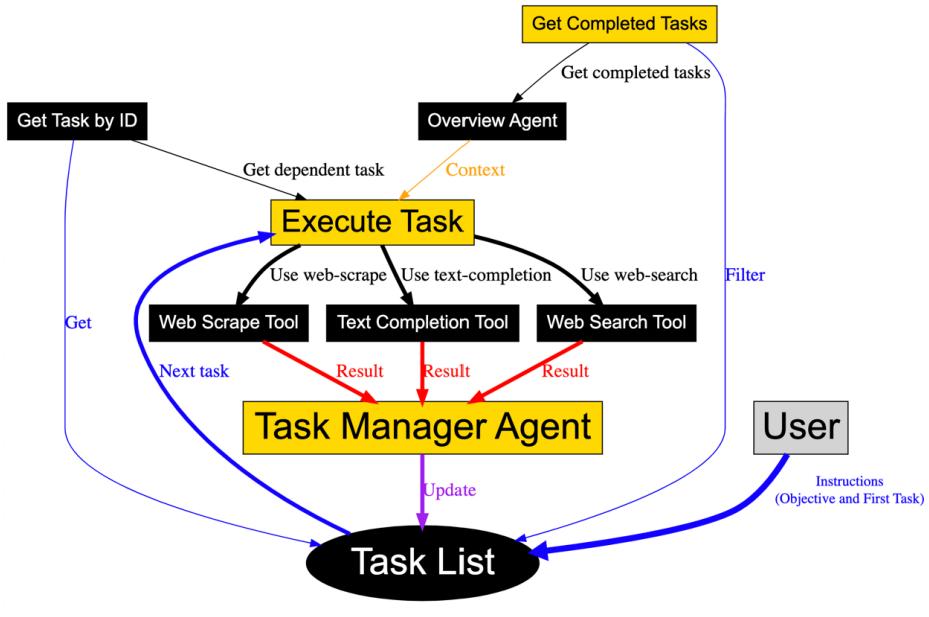


Figure 21: BabyBeeAGI Nakajima [2023]

```
# Task: <task> You have generated code of solution() to solve the task: <solution_func>
However, you executed the solution() function and get an error message: [Error message] <feedback>
Let's think step by step. You must output the revised solution function based on the error message. You
ProgPrompt: Replan = rewrite the program code Singh et al. [2023] 28.08.2023
```

recursive decomposition/hierarchy Prasad et al. [2023] 08.11.2023 recursive subtask decomposition. decomposition as needed: if task is not executable As-Needed Decomposition and Planning for complex Tasks (ADAPT), a recursive algorithm that further decomposes sub-tasks when necessary, if task cannot be executed it is assumed to be too complex and is then decomposed into subplan improved success rate compared to previous approaches with GPT-3.5 planner module and executor module controller ADAPT: two options: try to execute input task directly, else call planner to decompose the task, then recursively call this/controller theoretically always a complete plan, only at different hierarchy levels? needs termination criterium because of recursiveness only one "round" of decomposing task like in ReAct? may ot be sufficient, for complex tasks further decomposition can increase performance, the more complex the more it can improve planner includes AND and OR logic: linearly tasks all must, or alternatives

(nothing special) Raman et al. [2023] 22.10.2023 planning few-shot?? Planning LLM: demonstration set: high-level example task and respective plan, cosine similarity with query task, generate actions for task step by step: append action, further autoregressive steps corrective prompt whole plan? + corrective actions in case of precondition violation (re-planning)

Wang et al. [2023d] 29.10.2023 Interactive Planning with Large Language Models (goal: for open world tasks, long-term tasks, parallel subgoals) DEPS: Describe, Explain, Plan and Select initial LLM generated plan, integrate description of plan execution process, self-explanation of feedback of failures trainable goal selector (rank parallel candidate sub-goals on estimated number of steps) LLM-based planner with domain specific knowledge for the environments (Minecraft, ALFWORLD, Tabletop environments) challenges: sub-task dependencies and state-dependent task feasibility controller fails to complete subgoal: descriptor that summarizes current situation - to LLM planner for adjusting plan, LLM as explainer to locate errors in previous plan, planner then refines plan based on descriptor and explainer planner: decomposes high level goal into short-horizon subgoals. goal-preconditioned policy generates action based on current state and sub-goal LLM as zero-shot planner: goal/task, decompose into sequence of subgoals as initial plan action: controller, execute the provided sub-goals sequentially through a goal-conditioned policy action fails: descriptor: summary of current state end execution outcome of most recent goal. LLM: explainer/self-explanation to locate error, re-plan current task/generate revised plan (to original) prompt-template: python like code not only environment feedback that execution failed, but "describe, explain and plan" structured prompt: goals with preconditions and effects,

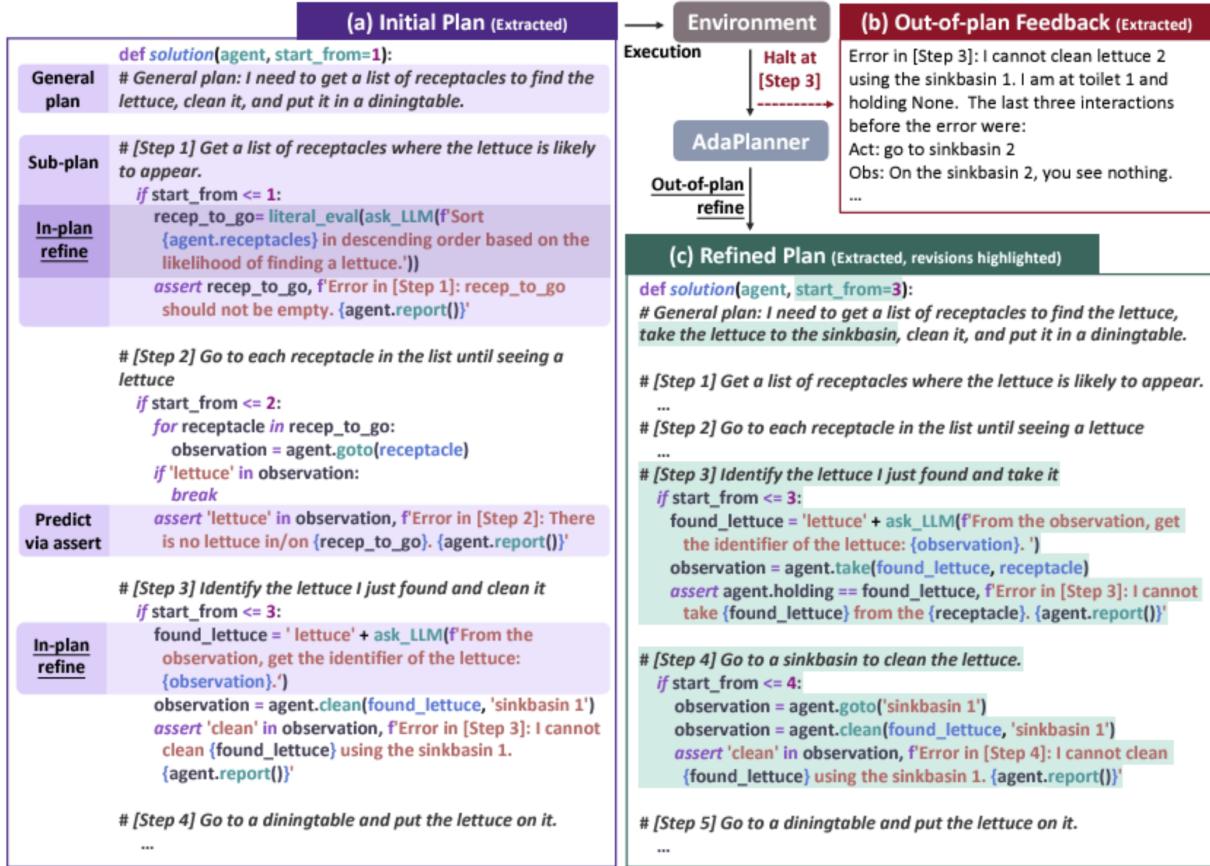


Figure 22: AdaPlanner ...Sun et al. [2023a] p.5

interactive dialogue format descriptor for feedback: human feedback/human description or pre-trained vision-language model, or returned information from (simulation) environment/available symbolic information explainer: few-shot CoT LLMS: code-davinci-02 text-davinci-03 gpt-3.5-turbo LLM planner: Codex, ChatGPT, GPT3, GPT-4 as planners, LLaMA2 + own fine-tuned LLaMA2 maximum token limits: limits maximum number of re-planning rounds assumption: pre-trained LLM already cover /dad exposure to minecraft-related corpus prompt for explainer explainer:

Here are some actions that the agent fails to perform in Minecraft. Please give the explanation of act

```
### Failed Action: mine({'iron_ore':1}, null); # step 5: mine 1 iron_ore without tool Current Inventory
```

Goal parser: pre-defined controller skill set. KKM Sentence-Bert model to select most similar skill to current goal
Example of DEPS full prompt process:

```
User: How to obtain 1 stone_sword in Minecraft step-by-step?  
Planner: The code for obtaining stone_sword is as follows:  
def obtain_1_stone_sword(inventory = {}):  
mine({'log':3}, null); # step 1: mine 3 log without tool  
craft({'planks':12}, {'log':3}, null); # step 2: craft 12 planks from 3 log  
craft({'stick':4}, {'planks':2}, null); # step 3: craft 4 stick from 2 planks  
mine({'cobblestone':2}, null); # step 4: mine 2 cobblestone without tool  
craft({'stone_sword':1}, {'cobblestone':2, 'stick':1}, 'crafting_table'); # step 5: craft 1 stone_sword  
return 'stone_sword'
```

Descriptor: I succeed on step 1, 2, 3, but I fail on step 4 "mine({'cobblestone':2}, null);". My inventory: Explainer: Because mining cobblestone needs to use the tool wooden_pickaxe.

User: Please fix the above errors and replan the task "obtain 1 stone_sword".

RePlanner: The code for crafting stone_sword is as bellows: def obtain_1_stone_sword(inventory = {'plan

action: controller, execute the provided sub-goals sequentially through a goal-conditioned policy for minecraft

5 Action Matching

DRAFT: ONLY NOTES

TODO INTRO

Action execution: agents decisions/steps/tasks to concrete action executions acting in environment some actions could CHANGE the environment! diverse: internet or special page search (wiki), code interpreter, database,... external models, query workflow: how to interact with tools and get the results/observations/information also e.g RAG/RAG pipeline as a tool wording: TALM: tool augmented language models? LLM as action executor, interaction with environment by LLM executor generated actions, LLM can et list of possible actions/skills and/or demonstrations Can query memory before execution to get additional information

Different types of actions Wang et al. [2023e] perspective of action goal/objectives of actions: task completion, communication, environment exploration action space perspective: external tools, "internal" knowledge of LLM internal: planning step/decomposition, conversation

TODO some of the notes from Ghallab et al. [2016b] "Automated planning and acting" for intro

5.1 Action/Tool Selection

DRAFT: ONLY NOTES

select action (most often: tool) among diverse options tool definition prompt examples

TOOLS PREDEFINED PER EXPERIMENT Gao et al. [2024b] Efficient Tool Use with Chain-of-Abstraction Reasoning, see also offline planning grounding to real-world knowledge needed: can be achieved via tool calls need for holistic and efficient tool usage planning in multi-step reasoning Chain-of-Abstraction CoA/planning with abstract chains: decode reasoning chains with placeholders, then call domain tools: reify reasoning chain/fill in specific knowledge, eg use calculator on placeholder variables learn general reasoning strategy, parallel decoding and external tool calling (possibly less inference delay) LLM fine-tuned to generate abstract multi-step reasoning chain with placeholders on a user question. Call external tools to reify step with domain specific knowledge and fill in placeholders generate chain of reasoning once. Then add knowledge (other approaches rather interleaving) experiments show seems especially beneficial with long reasoning chains

argue: not practical, because as soon as there are several tools, tools must be chosen!

5.1.1 Reasoning on a List of Action/Tool Descriptions

Tool Availability via Descriptions

Shen et al. [2023] HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face AI tasks, different domains, modalities models for specific tasks how to handle complicated task? LLM as controller for different AI models ChatGPT LLM: task planning, select model (according to function description in HuggingFace) to execute subtask summarize response complex AI/ML tasks consisting of multiple subtasks, requiring different specialized models, expert models (e.g. finetuned to specific tasks) how to connect LLM and expert models? via language, language as interface needed: high-quality model descriptions model selection: Hugging Face model based ion their description task execution: invoke and execute model, results to LLM response generation: integrate all models answers select model best for task execute task, model via endpoint summarize all results JSON format for all formal specifications, templates prompt to select a model (tool)

#2 Model Selection Stage - Given the user request and the call command, the AI assistant helps the user

candidate model (tools) descriptions

```
{"model_id": model id #1, "metadata": meta-info #1, "description": description of model #1}
```

stable outputs: decoding temperature to 0 JSON format: set the logit_bias to 0.2 on the format constraints (e.g., "" and "") needs reliability of LLM, multiple interactions with LLM, limit of maxi-

maximum token length (eg to incorporate all model descriptions), instability (reliability of LLMS) Shen et al. [2023] HuggingGPT, Multi path but linearization (several prerequisite dependencies) task format: `[{"task": task, "id": task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": audio}}]` dep: dependency with id of previous task Prompt asks the LLM to generate task list in specific format and provides examples of task lists (demonstrations) to queries tasks can depend on several previous/prerequisite dependencies on resources produced by previous tasks

Huang et al. [2022b] Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents - only aspect of finding corresponding admissible action everyday household tasks plans which are not directly executable in environment - linguistic ambiguity or no fitting admissible action they: "enumerate all admissible actions and map the model's output phrases to the most semantically-similar admissible action (we use similarity measure between sentence embeddings produced by a RoBERTa model [27] in this work, but other choices are possible)" + generate actions by conditioning past actions made admissible by similarity measure... translate each step into admissible action via another pre-trained masked LLM then append generated action to tompz and then generate remaining steps when beneficial: need special format/structure of action, mapping of actions and objects in environment, lexical ambiguity "to embed the output action phrase and environment actions, we use a BERT-style LM [10, 27] pre-trained with Sentence-BERT [41] objective, to which we refer as Translation LM" after each step translation because better for guaranteeing achievability of each step interleave plan generation and action translation results in far more executable and correct plans (at least in robotic environment) interestingly, depending on LLM used for planning (CODEX 12B or GPT-3 175B) different translation LMs perform better: either Sentence RoBERTa (large) or Sentence Bert (base)

Ouyang and Li [2023] no in-context demonstrations (for actual rules in environment) instead list of possible actions, trial and error if can be applied action prompts for Hotpot QA:

```
Valid action formats are as follows:  
search[entity]  
lookup[keyword]  
finish[answer]
```

Formalize the following action strictly into the above valid action formats. If there are multiple actions

```
Action to formalize: {raw_action}  
Formalized action: {formalized_action}
```

Raman et al. [2023] actions/skills skill library skills described in natural language: need to ground skills in environment. Reason about: applicability, relevance and specific state changes in environment, eg query database: which concrete database with API, specific SQL query assumed: predefined skills with preconditions, goal description, observation and action space textual description if precondition violation (fail to execute action because precondition violation): CAPE (Corrective Actions from Precondition Errors) - LLM queried for corrective actions actions with preconditions: inspired by structured affordance models from symbolic?? planning, action preconditions: factorize states into preconditions. affordance: state components satisfied Options framework: O(s) over state space S: set of temporally extended actions, initiation set I(o): states in which option execution is afforded, termination condition/terminal state of skill(effect) $\beta_o(s)$ current state != option initiation state - precondition error

Yuan et al. [2024] EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction transform tool documentation to concise description diverse tool documentations: could be diverse, redundant, or incomplete framework to transform tool documentation into unified and concise tool instruction for easier tool usage, offer standardized tool descriptions and functionalities for LLM-based agents experiments: significantly reduces token consumption and can improve performance of LLM-based agents on tool utilization tool documentation + demonstrations: consume many tokens issues of tool documentations: inconsistency/diversity, formats, redundancy, incompleteness approach assumed: Task planning-stage with decomposition, then tool retrieval, tool selection and tool execution 2 stages: tool description generation prompt with instruction for LLM to convert tool documentation, summarize its general purpose include tool name, general purposes and build in functions prompt with demonstration tool functionality guidelines construction: parameters of tools for execution, extract parameters from tool documentation and organize into structured output, create examples of using tool with example scenario input parameters experiments on tool bench show greatly reduced token costs when using toolbench, the token cost was reduced by 70.43%, improves closed and open source models, also small like mistral-isntruct-7B or Vicuna-7B, huge improvement because before could solve tasks at all and then achieve high pass/win/success rate, impressive!

Du et al. [2024] AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls 16,000 APIs from Rapid API, solver: resolve user queries using set of API candidates + self-reflection: re-activation if solution impracticable planning as tool selection agent: gets tools per category and their description, identify most relevant API candidates for answering

I: Tool Description Generation
/* I: Task prompt */
Your task is to create a concise and effective tool usage description based on the tool documentation. You should ensure the description only contains the purposes of the tool without irrelevant information. Here is an example:
/* Examples */
{Tool Documentation}
Tool usage description:
{Tool_name} is a tool that can {General_Purposes}.
This tool has {Number} multiple built-in functions:
1. {Function_1} is to {Functionality_of_Function_1}
2. {Function_2} is to ...
/* Auto generation of tool description */
{Tool Documentation of 'Aviation Weather Center'}
Tool usage description:
<i>'Aviation Weather Center' is a tool which can provide official aviation weather data...</i>
II: Tool Function Guidelines Construction
/* Task prompt */
Your task is to create the scenario that will use the tool.
1. You are given a tool with its purpose and its parameters list. The scenario should adopt the parameters in the list.
2. If the parameters and parameters are both null, you should set: {"Scenario": XX, "Parameters":{}}.
Here is an example:
/* Examples */
{Tool_name} is a tool that can {General_Purposes}.
{Function_i} is to {Functionality_of_Function_i}
{Parameter List of Function_i}
One scenario for {Function_i} of {Tool_name} is:
{"Scenario": XX, "Parameters":{XX:XX}}
/* Auto-construction for Tool Function Guidelines */
'Ebay' can get products from Ebay in a specific country.
'Product Details' in 'Ebay' can get the product details for a given product id and a specific country.
{Parameter List of 'Product Details'}
One scenario for 'Product Details' of 'Ebay' is:
<i>{"Scenario": "if you want to know the details of the product with product ID 1954 in Germany from Ebay", "Parameters": {"product_id": 1954, "country": "Germany"}},</i>

Figure 23: EasyTool Prompts Yuan et al. [2024]

query two parts: API retriever identifies API candidates, solver addresses queries using generated API-candidate pool + self-reflection When without hierarchy in API retriever: Ablation, significant positive effects self-reflection also significant positive effects

Sun et al. [2023a] ALFWorld available actions, example of application, example of an action: ALFWorld:

```
# Go to a receptacle and update the agent's location.
# For example, 'On the countertop 1, you see a candle 1, a cloth 2, and a soapbar 1.' = goto('countertop 1')
...

```

MiniWoB++

```
# Action: press a key on the keyboard, the input can be one of the following:  
# enter, space, arrow_left, arrow_right, arrow_up, arrow_down, backspace  
# this function returns the string of the HTML code after taking the action # e.g., new_html_state = a
```

+SEACRH FOR TOOL CHAIN - FITS BOTH CATEGORIES? Qin et al. [2023] "by integrating LLMs with APIs, we can greatly expand their utility and empower them to serve as efficient intermediaries between users and the vast ecosystem of applications" ToolBench, an instruction-tuning dataset for tool use: constructed automatically using ChatGPT (gpt-3.5-turbo-16k) 16,464 real-world RESTful APIs prompt ChatGPT to generate diverse instructions with APIs (understand API functionality and generate possible instructions involving this API) and determine also relevant APIs for an instruction, – (instruction, relevant API) pairs, prompt needs: description of instruction generation task, API documentations, 3 in-context examples (human written instruction generation) and prompt ChatGPT to search for valid solution path (chain of API calls) - valid action sequence "To let ChatGPT finish an action sequence, we define two additional functions, i.e., "Finish with Final Answer" and "Finish by Giving Up". " depth-first search-based decision tree algorithm: evaluate multiple reasoning traces ToolEval: automatic evaluator, ChatGPT, evaluates LLMs ability to execute an instruction within limited budgets and compare quality and usefulness of solutions paths: high correlation with human evaluation, assessment ToolLLaNA: finetuned LLaMA on ToolBench, recommends appropriate APIs, evaluate using ToolEval ToolLama "outperforms Text-Davinci-003 and Claude-2, achieves comparable performance to the "teacher model" ChatGPT, and is only slightly inferior to GPT4" generalizability to new APIs: only needs documentation CoT and React limitations: error propagation (propagate errors further, then trapped in faulty loop, hallucinating APIs,, +limited exploration because only one possible direction therefore here construct decision tree) expand decision tree, proceed along promising path or end at a node by finish by fiving up, DFS: one valid path is sufficient, less API calls to OpenAI generate 126, 486 (instruction, solution path) pairs, which are used to train ToolLLaMA ToolLLaMA We fine-tune LLaMA-2 7B

5.1.2 Action/Tool Space Search?

Zhuang et al. [2023] TOOLCHAIN*: EFFICIENT ACTION SPACE NAVIGATION IN LARGE LANGUAGE MODELS WITH A* SEARCH –interesting aspect: how to assign cost to actions for using A* search series of API calls candidate APIS: action space tree search-based planning algorithm, action space = decision tree of API calls in a solution plan, explore acton space A* algorithm, task specific cost function, efficient search not requiring exhaustive exploration, less steps than DFS or MCTS task description into ordered sequence of API function calls A* requires less function calls needs components of cost function $f(n)=g(n)+h(n)$ cumulative cost g: task specific heuristic function, based on long-term memory with plans, highest longest common sub-sequence between current plan and all plans in memory and self-consistency frequency of different actions future cost h: task.-specific heuristic function: from long term memory, relative average position score of an action appearing in the plans and Imagination score by LLM: imagined plan and further steps to estimate remaining steps/less steps (a) Selection: Pick a frontier node with the lowest summation of cumulative cost and future cost (b) Expansion: Expand the nodes with potential next steps. (c) Update: Update the value functions of the newly added nodes in the search tree. GPT-3.5-turbo and GPT-4, LLaMA-2 7B and 13B, GPT-4 better than 3.5, results worse than GPT, bit good with finetuned LLama far more efficient than other tree search methods...but requires intricate...of the values for the cost function (because not based on simulation!) Prompts and general execution: Like ToolBench/ based on ToolBench

5.1.3 Action/Tool Selection Learning

Ahn et al. [2022] SayCan, Tasks/Skills - see also iterative planning! combine: need feasible and contextually appropriate actions subtask decomposition: only reasonable if: context knowledge about robot capabilities, current state and information about environment LLM interpret instruction + likelihood evaluate how a skill can help progress towards goal + affordance function) how likely it will succeed: LLM "aware" of capabilities + explainable reasoning/choosing of steps/actions Affordance functions: learned by RL grounding via affordances: leads to doubled performance on robotic tasks set of skills and affordance function for probability of completing skill succesfully from state s LLM should propose optimal skill multiply LLM probability of skill being useful for high-level instruction with affordance function of successful execution probability break down high-level instruction into sequence of available low-level skills set of skills, skill: policy (e.g. learned), value function (TD/RL), short language nondescript (e.g.g "pick up the can") condition policies on language: pre-trained large sentence encoder language model. generate embeddings by passing in Text descriptions of each skill - this is input to policy and value function

Schick et al. [2023] Toolformer: Language Models Can Teach Themselves to Use Tools LMs can teach themselves to use external tools via simple APIs Toolformer: model trained: decide on which API to call, when to call, which arguments to pass and how to incorporate results self-supervised: some demonstrations for each API then zero-shot learning: in-context learning, generate data set, let LLm annotate dataset with potential API calls and determine loss:

which API calls would have helped in answering the question pretrained GPT-J model (Wang and Komatsuzaki, 2021) with 6.7B parameters assumption: input/output of API calls are text (only then can they be inserted into text) Augment dataset of plain text, with API call sequences, sample number of API calls, execute, and check if responses helpful to predict future original tokens in the text generate API calls: This generated dataset is used to finetune a model Tools:

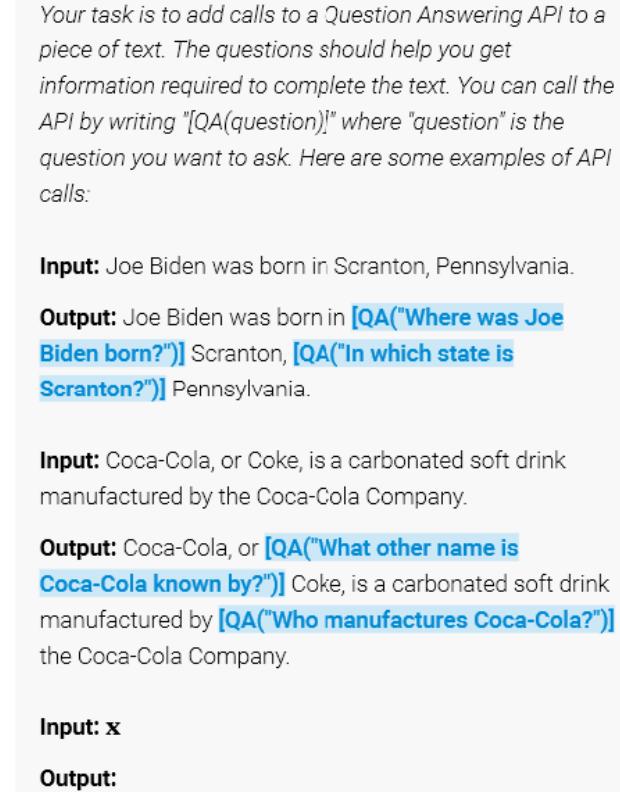


Figure 24: Toolformer Dataset Schick et al. [2023]

input and outputs must be text sequences + demonstrations of their use available no tool use in interactive way, very sensitive on prompt

5.2 Action via Code Generation

Tool Making

Wu et al. [2023b] AutoGen code execution or LLM-suggested function calls

Silver et al. [2023] Generalized Planning in PDDL Domains with Pretrained Large Language Models + see environment feedback given: domain, training tasks produce plans for other tasks in that domain (PDDL domains) GPT-4 synthesize python programs this work "only" for generalized planning, not planning! (for set of tasks, synthesize program that produces valid plans for all the tasks) CoT summarization and then synthesize program automated debugging: re-prompt LLM with four types of feedback "just two training tasks are often sufficient for strong generalization" GPT4 should write python program that solves tasks in a planning domain prompt: domain + training tasks , both in PDDL python program: take task description and generate a plan flow: 1) write summary of PDDL domain in natural language, then describe a solution strategy, then write it in python environment feedback: errors from code execution, give error to LLM and ask it to fix code itself analysis: important are using GPT-4 (compared to 3.5), PDDL names, automated debugging (env exception feedback) produced generalized plan in python code can be inspected (maybe instead of iteratively adding new tasks) + not many queries + this not limited to context window size only classical planning: deterministic, fully observable PDDL STRIPS subset, plan then finite sequence of actions 3 stages: write summary (need to truncate training tasks to fit into context window)

Domain: [PDDL Domain]

Example problems: [PDDL Training Tasks]

Write a short summary of this domain in words.

strategy proposal, asg for strategy

There is a simple strategy for solving all problems in this domain without using search. What is that?

strategy implementation:

```
Implement the strategy as a Python function. The code should be of the form def
get_plan(objects, init, goal):
# Your code here
return plan
where
- objects is a set of (object name, type name) tuples
...
```

Feedback: types: python exceptions, timeout, plan syntax, plan semantics exception:

```
Given this task: [PDDL Training Task]
The code raised the following exception:
File "<file-name-omitted>", line 86
lift_at = {atom[1]: atom[2] ...}
```

```
IndexError: tuple index out of range
Fix the code.
```

plan semantics

```
Given this task: [PDDL Training Task]
The code failed. It returned the following plan:
['(pick-up paper-1 loc-0)', ...].
NOTE: (pick-up paper-0 loc-0) has an unsatisfied precondition at time 3
(Set (at loc-0) to true)
Fix the code.
```

Wang et al. [2024b] Executable Code Actions Elicit Better LLM Agents, CodeAct further realization: open source LLMs better with text instruction than JSON, closed source rather better with JSON format (both worse than CodeAct)

Gur et al. [2023] "acts via programming on real websites by grounding sub-instruction and HTML snippet into executable Python codes" WebAgent: Flan-U-PaLM (Chowdhery et al., 2022; Chung et al., 2022) for grounded code generation for open-ended action spaces like in web automation elements to interact with a website they: act via programming, conditional code generation, Flan-U-PaLM: executable Python program with Selenium WebDriver, program synthesis

TODO Plan to Code execution, where to sort? Qiao et al. [2023] 01.12.2023 TaskWeaver: A Code-First Agent Framework main focus? domain-specific data analytics tasks with rich data structures? (nested lists, dictionaries, or data frames), how to handle these structures user requests to executable code, can use user-defined plugins dynamic plugin selection domain specific knowledge through examples requirements: use custom plugs, support rich data structures, stateful execution., Reasoning and acting, natural language response, code generation, domain knowledge usage, persisting artefacts Components: Planner, Code Generator (CG), and Code Executor (CE) Planner: break request into subtasks, execution process with self reflection, high-level plan CG: generate code, incorporate function calls (for plugins) plugins: function name, its description, the arguments it accepts, and what it returns optionally: code generation examples Code generator generates code for one step. Code executor executes the code. Result is sent back to planner and planner determines the next step (modify initial plan if outcome different to expected) Task order: sequential, interactive (need for human or LLM intervention between) or no dependency (eg can be executed in parallel) self reflection: if outcomes of previous steps diverge from anticipated results - modify plan code execution error: CI can re-generate code code feedback to planner: return code, logs, output, artefacts optional examples: for task planning, for code generation possible plugins- activated via function call: web API call, a software module, a customized algorithm, or a deep learning model

Wang et al. [2023c] VOYAGER: An Open-Ended Embodied Agent with Large Language Models lifelong learning agent in Minecraft GPT-4 develops skills: temporally extended, interpretable, compositional 2) an ever-growing skill library of executable code for storing and retrieving complex behaviours, and store action programs that helped solve a

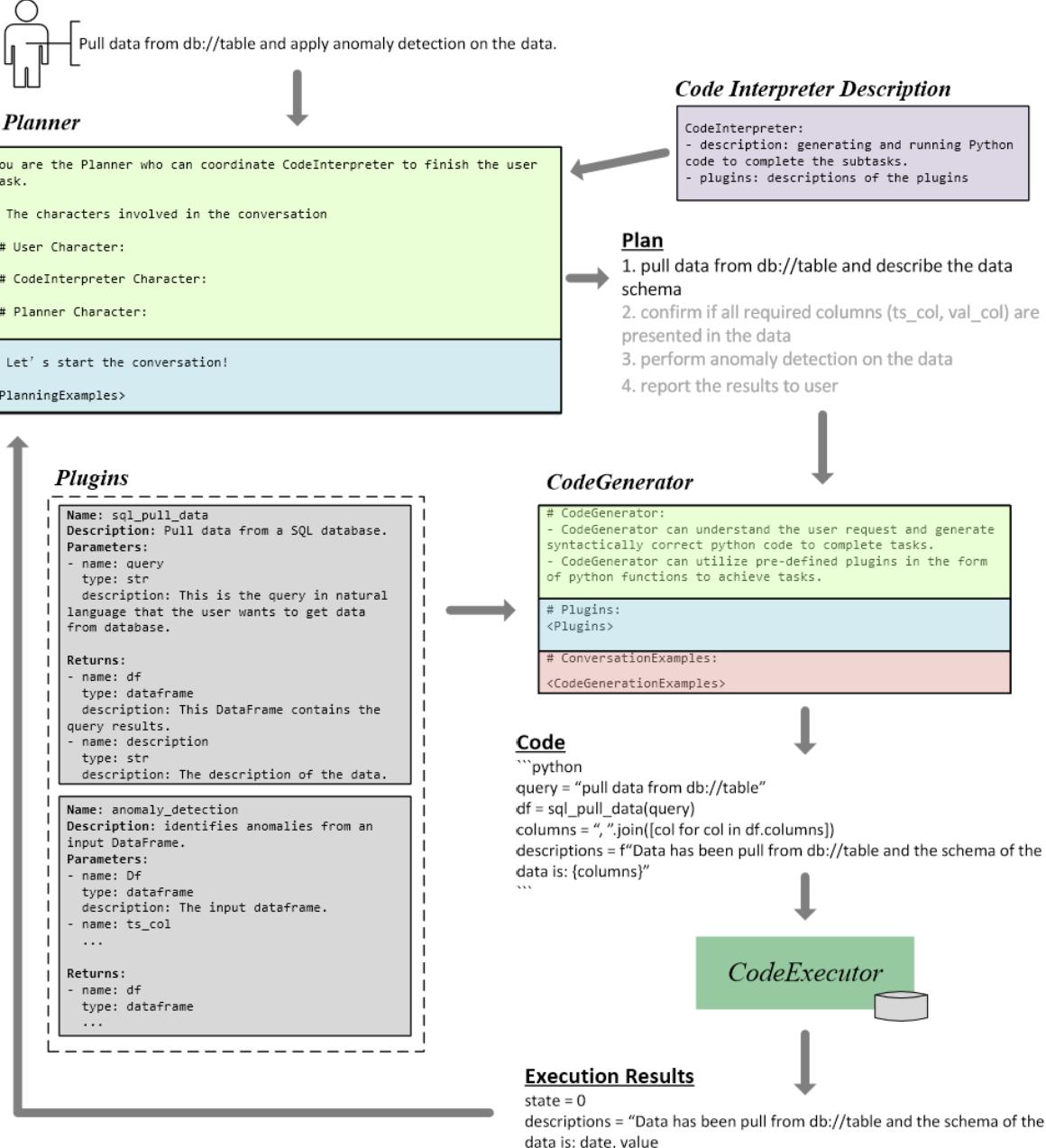


Figure 25: TaskWeaver Workflow Qiao et al. [2023]

task successfully description to program, generated by GPT 3.5 text-embedding-ada-002 [51] API for text embedding skill library: vector database, key= embedding vector of program description, value=program query library: GPT-3.5 uses pretrained knowledge to add context to a task, then embedding of task used to retrieve relevant skills from library Skill /code generation input to prompt: Guidelines for code generation, APIs and relevant skills from library, generated code from last round with feedback, agent current state in game, CoT prompting for reasoning before code generation Skill library and feedbacks significant in ablation studies, also very huge difference to using

Wang et al. [2024b] Executable Code Actions Elicit Better LLM Agents, CodeAct LLM used usually to generate JSON or text for actions/tools, predefined tools, no composition instead use executable python code, unified action space, execute code actions revise actions, new actions: observations/environment interaction finetuned from Llama2 and Mistral generate executable python code as actions, including available API calls within a python call/code execution,

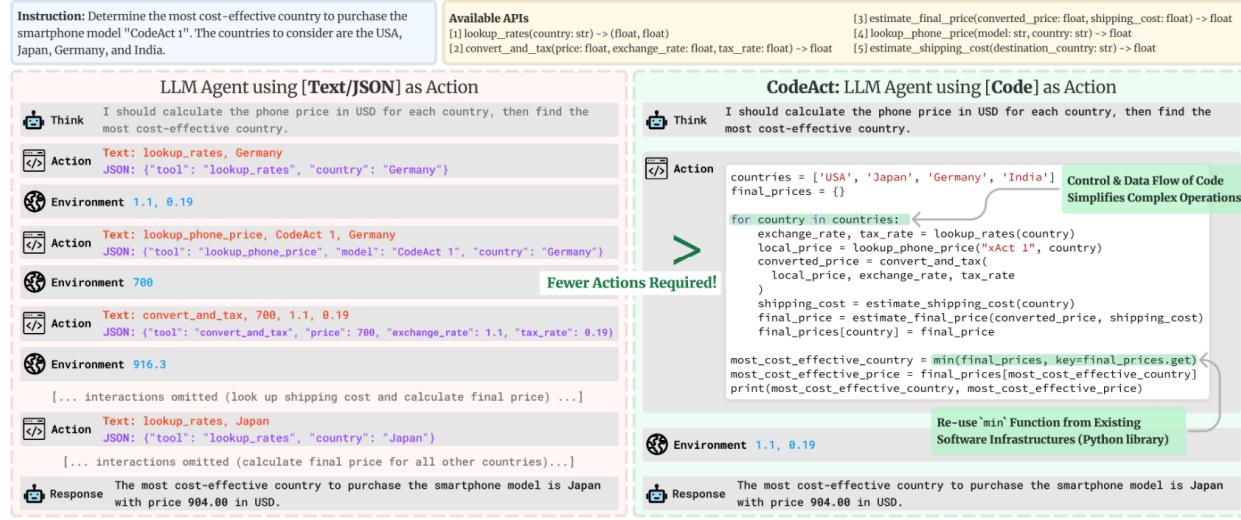


Figure 26: CodeAct: Action As Code Wang et al. [2024b]

inherent control and data flow within piece of code CodeActInstruct instruction-tuning dataset, on LLaMA-2 and Mistral-7B but also tested overall 17 LLMs? based on: programming knowledge aquired by LLMs during pretraining receive human instruction, environment feedback: code execution results as observations (results, errors), optional: CoT planning for action fewer interactions/turns zero-shot setting human instruction as usual task, environment feedback: code execution results as observations (results, errors) can use arbitrary python libraries within the generated code, e.g. pandas , Scikit-Learn etc example zero-shot system prompt, used in deploy instance of CodeAct:

```
<| im_start |> system A chat between a curious user and an artificial intelligence assistant . The assis
The assistant can interact with an interactive Python (Jupyter Notebook) environment and receive the co
The assistant should attempt fewer things at a time instead of putting too much code in one <execute>
The assistant should stop <execute> and provide an answer when they have already obtained the answer :
The assistant 's response should be concise , but do express their thoughts
<| im_end |>
```

tool definition

```
[5] view: Return the current view in string format of the rendered webpage. It has no arguments.
Returns the rendered content of the webpage.
You should call this when you want to see the rendered content of the current webpage.
Signature: view() -> str
```

Zhao et al. [2024] Empowering Large Language Model Agents through Action Learning, LearnAct Framework expand action space instead of fixed action space, learn actions from experience LearnAct Framework, actions: python functions iteration: LLM revises and updates actions based on errors on training tasks Training Stage and Test stage training stage: create actions, optimize based on execution feedback., expand actions space, eg add to existing basic actions more complex constructs like conditional statements or loops + generate policy(information about potential application of action, including description with anticipated outcome and necessary conditions and usage example) + executes actions on task and identifies failures and iteratively updates action set, action code or description Test stage: use the learned actions in sequential decision making ALFWorld and robotic tasks new actions generated based on atomic actions, similar to higher level actions and lower level learn from environment interaction Action space expandable: add new actions and instructions for using the actions (policy) GPT-4 and GPT-3.5 Turbo experiments show that learning in action space in Alfworld and robotic tasks better than eg ReAct+Reflection which writes policy prompts/learned verbal policies: can have better planning but no improved skillset closed-loop approach like here better than other approaches not interacting with environment like CodeAsPolicy iteration like here better than without like in Voyager ablation studies usage examples and descriptions contribute to performance, usage examples more (despite being often incorrect!?) example prompt, for action creation:

```
{basic task instruction}
Please propose several high-level steps for this task.
```

Each high-level step should be a Python function encompassing multiple (at least two) basic actions. All the provided actions are Python functions and can be executed directly, for example, ““python {basic actions}””. No additional interfaces besides the provided actions are available. All the code should be wrapped by `Here are examples: {created action example}`
Now please write your solution:

6 Feedback and Reflection

DRAFT: ONLY NOTES

NOTES OR INTRO: reflect refine long horizon errors new observations unpredictability non-executability assumed that more complex/longer/more steps feedback beneficial for: plan correction, plan/task improvement, refinement based upon: Observations, past actions,...

TODO add CRITIC Gou et al. [2024]

6.1 Internal Feedback with Self-Reflection

DRAFT: ONLY NOTES

self reflection: use LLM itself, verbal feedback mainly, examine intermediate steps also mention reflection based on memory, but only mention and refer to log.term memory - experience summary possible? error detection: repeated sub-steps - plan/iteration correction?

6.1.1 Corrective Prompting

TODO add SelfCheck: corrective Zero-hot prompt to check CoT reasoning Miao et al. [2023]

Self-refine Madaan et al. [2023] SELF-REFINE: Iterative Refinement with Self-Feedback "improving initial outputs from LLMs through iterative feedback and refinement" self-feedback on output few-shot, include feedback for output refinement generate initial output, then same LLM provides feedback to this output to refine itself iteratively uses a single LLM as the generator, refiner and the feedback provide GPT-3.5 and GPT-4 improves in task performance alternates between two generative steps—FEEDBACK and REFINE "Given an initial output generated by a model M, we pass it back to the same model M to get feedback. Then, the feedback is passed back to the same model to refine the previously-generated draft. This process is repeated either for a specified number of iterations or until M determines that no further refinement is necessary" needs examples,few-shot prompting, examples of feedback / of refinements "SELF-REFINE consistently improves over base models across all model sizes, and additionally outperforms the previous state-of-the-art across all tasks" Vicuna-13B problems with refinement process, was not able to consistently generate feedback, probably (large) instruction based model are best suited

Wu et al. [2023b] AutoGen, feedback via conversation (different agents but nonetheless internal)

Li et al. [2023b] Critic-In-The-loop: critic agent, select proposals, provide feedback to role-playing agents, similar to

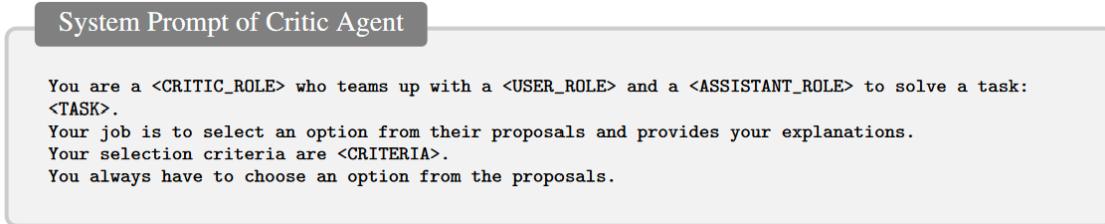


Figure 27: CAMEL CriticLi et al. [2023b]

tree-search, critic can be AI agent or human

Kim et al. [2023] , RCI, Language Models can Solve Computer Tasks general computer tasks, keyboard and mouse actions just prompting scheme: agent Recursively Criticizes and Improves its output (RCI) MiniWoB++ benchmark best: InstructGPT-3+RLHF LLM just several demonstrations per task combination with CoT effective LLM generates an output based on zero-shot prompting first. RCI prompt to identify possible problems and to generate updated output

apply in 3 stages for grounding: "actions are task-appropriate (task grounding), feasible in the agent's current state (state grounding), and admissible to be executed (agent grounding)" task grounding: plan for task solving, improve plan's success rate state grounding: actions in environment to be feasible in the current state agent grounding: admissible for agent general Schema:

```
<Initial output generation>
A: ...
<Critique>
Review your previous answer and find problems with your answer.
<Improve>
Based on the problems you found, improve your answer.
```

Task grounding: Find problems with this plan. ... Based on this, the improved plan for the agent to complete the task are as follows. Execution: According to the current plan, the next proper instruction should be State grounding: Considering the output on the webpage, the specific instruction for solving the task should, executable in current context Agent grounding: Therefore, the single instruction that matches one of the regular expressions is, executable by the agent (repeatedly run until action is executable) RCI prompts (highlighted) mainly InstructGPT-3 + RLHF models

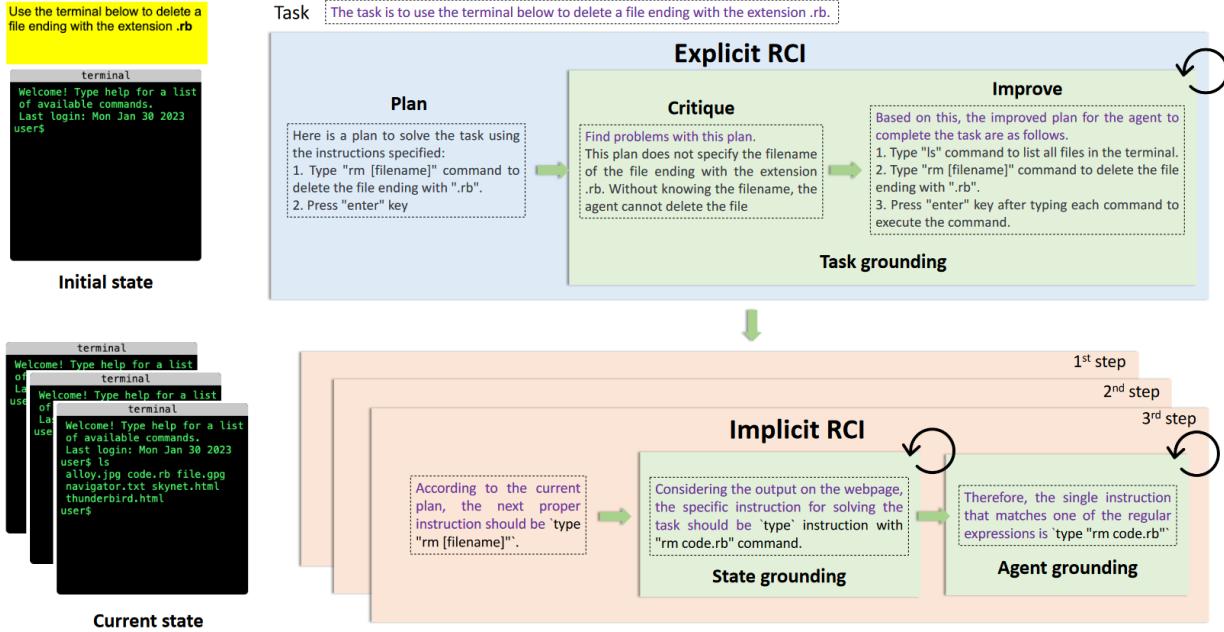


Figure 28: RCI Prompting Kim et al. [2023]

(gpt-3.5-turbo, gpt-4) evaluation/ablation: each grounding stage essential for performance

Kim et al. [2024] PROSPECTOR: IMPROVING LLM AGENTS WITH SELF-ASKING AND TRAJECTORY RANKING outperforms ReAct and Reflexion in ALFWorld, WebShop LLM agents based on few-shot in-context learning (based on ReAct and Reflexion) – "Prospector, a reflective LLM agent that features Self-Asking and Trajectory Ranking" two components Self-Asking and Trajectory Ranking Self-Asking steps, generate question and answer itself, demonstrations which questions to ask, eg to verify if observations match the planned steps, eg also "given" candidates "which is the most proper to select" Trajectory Ranking: create diverse trajectories and select most rewarding by using prediction models ALFWorld, WebShop "generates diverse trajectories with a number of trials and then selects the most rewarding trajectory as the final action" "1) Self-Asking, which can generate promising trajectory candidates, and 2) Trajectory Ranking, which can select the most rewarding trajectory from candidates" assumption: agent receives observation, generates action, receives reward, agent wants to maximize cumulative reward, like Reinforcement learning but without fine-tuning but: many environments no reward, therefore: LLM as reward estimator, prompt templates Few-shot LLM Critics,example for ALFWorld:

```
Evaluate if the instruction given in the input is accomplished by performing a sequence of actions (fa)
### Input:
```

```
{Example success trajectory}
### Response: success
### Input:
{Example fail trajectory}
### Response: fail
### Input:
{Input trajectory}
### Response:
```

alternative: Fine-tuned LLM Critics, LLMs fine-tuned on trajectory data, when hard /not effectively working with few-shot examples less efficient than eg ReAct, many prompts

6.1.2 Predefined Heuristics

if reward score prediction not by LM but predefined heuristics same issue as or discriminator

6.1.3 Memory-based Reflection

Wang et al. [2024c] RAT: Retrieval Augmented Thoughts Elicit Context-Aware Reasoning in Long-Horizon Generation chain of thoughts + information retrieval retrieval-augmented thoughts (RAT) initial zero-shot CoT, then RAG to revise CoT but: step by step: "LLMs produce the response step-by-step following the CoT (a series of subtasks), and only the current thought step will be revised based on the information retrieved with task prompt, the current and the past CoTs" each thought step revised with retrieved information relevant to the task query, current and past thought steps GPT-3.5, GPT-4, and CodeLLaMA-7b "substantially improves their performances on various long-horizon generation tasks" mitigate hallucinations Retrieval Augmented Generation (RAG) "utilizes retrieved information to facilitate more factually grounded reasoning" RAG+ long horizon reasoning RAT: strong advantages over vanilla CoT and RAG Code generation, MBPP, GSM8K, MineCraft task planning, creative writing "Two questions still persist: (1) what is relevant information to retrieve; (2) how to effectively correct reasoning steps with relevant factual information" rough algorithm: Generate zero-shot initial step-by-step thoughts T Draft answer T* initialized with the first thought step T1 repeat, revision rounds until all steps have revised thoughts: Generate query Qi based on current draft answer T* Retrieve information Ri from corpus or Internet Revise draft answer T* based on retrieved text Ri Append the next thought step Ti+1

Shinn et al. [2023] Reflexion: Language Agents with Verbal Reinforcement Learning to learn from trial and error; memory: see also memory linguistic feedback, verbally reflect on task feedback episodic memory buffer to save reflective texts - for usage in subsequent trials types of feedback: scalar values, free-form language feedback sources: externally, internally simulated can convert binary or scalar environment feedback to verbal feedback/textual summary, add as context for LLM agent in the next episode self-reflective feedback experience summaries stored in long-term memory "more explicit and interpretable form of episodic memory over prior experiences" use as explicit hints for actions in future episodes Actor model: use CoT or ReAct prompting on LLM, generate policy via model, use memory as context produce trajectory /carry out actions in environment, evaluate trajectory with evaluator model generate self-reflection with self-reflection model and add to memory, until evaluator model judges trajectory to be correct evaluator model: get generated trajectory, computes reward score for performance in task, eg via heuristic functions or LLM self-reflection model: LLM, crucial, generates verbal self-reflections, given: reward signal, current trajectory and persistent memory and store into memory Memory: long-term and short-term. Actor model uses short term and long term memory, short-term: trajectory history, long-term: outputs of self-reflection model Reflexion process: iterative optimization reflexion additionally to ReAct significantly outperforms in Alfworld long-term memory: can only use part: sliding window with maximum capacity Needs a whole trial and then do next trial to improve Webshop not much better than react: "We conclude that Reflexion is unable to solve tasks that require a significant amount of diversity and exploration" needs some form of evaluation

6.2 Environment Feedback

feedback from environment observations after actions or general "states"

TODO add from Voyager Wang et al. [2023c] (see: iterative task generation and Action via code generations)

LLM-Planner: check object mismatches and unattainable plans? Song et al. [2023]

Sun et al. [2023a] refine plan, especially for better performance problem complexity and long plan horizon, used for action and plan observations from text-based environment, each step, each action/interaction with environment leads to

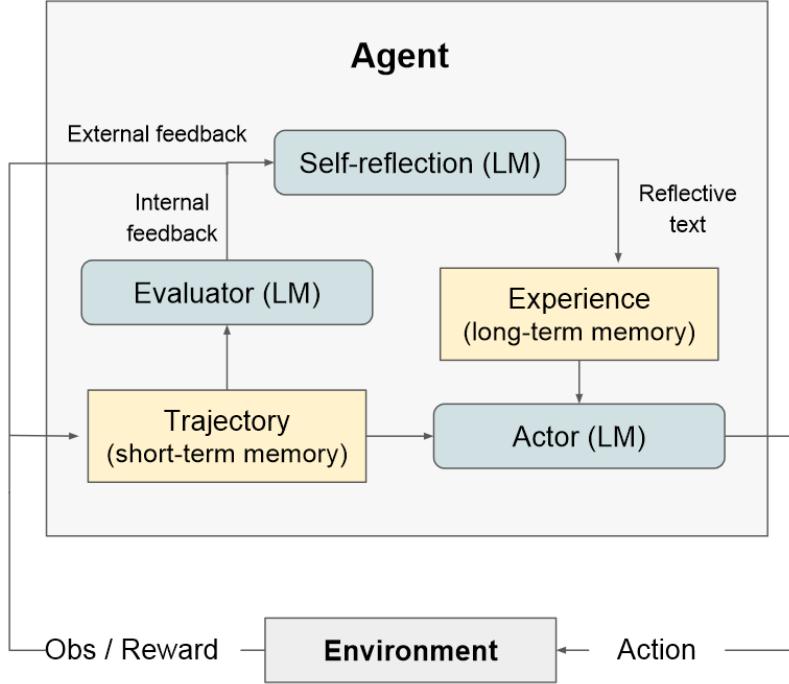


Figure 29: Reflexion Shinn et al. [2023]

an observation for each step, python like code is generated, is executed via environment interface (grounding actions, return environment observations) askLLM action: self-query, reasoning on information from observation

Ouyang and Li [2023] and SIR reflection: summarization, flaw identification, plan revision SIR: reward 0-1, depending if goal reached in environment according to observation SIR reflection: 1) Summarize the interaction history, 2) Identify the flawed steps of the plan, 3) Revise the flawed steps needs fine grained feedback to be effective/to use it efficiently, meaning eg missing index, validity of action, validity of precondition etc

Identify which step of plan you are at. Show your thought about the one next action. Your thought shou...

Task finished. The ground truth answer is "dancer Gregory Hines" and the correct entities to search ar...

Identify all flawed parts of the plan (not flawed action).

Suggest revision to the current flawed part of the plan. Only the flawed part.

? Ouyang and Li [2023] preprint: 26.10.2023 augments LLM prompt with a task-solving plan, prompt-based iterative experience collection and reflection no in-context demonstrations (for actual rules in environment) initial incorrect plan - interaction experiences, revise plan based on reflection natural language plan Different types of prompts: Thought-prompt: identify current step and think about next step summary-prompt: interaction history summary flaw-prompt: identify flawed parts of plan/action rev-prompt: revision suggestion of flawed part Upd-prompt: rewrite current plan, including summary and flawed action revision suggestions starts with empty plan, at each iteration random selection of number of task instances, for each task instance LLM agent creates sequence of thoughts and actions responding to environment observations

Raman et al. [2023] environment feedback +internal feedback (knowledge about action preconditions?) reasoning from action preconditions: corrective actions to resolve precondition errors huge correctness improvement compared to SayCan include precondition error feedback inspired by structured affordance models from symbolic?? planning, action preconditions: factorize states into preconditions. affordance: state components satisfied Options framework: $O(s)$ over state space S : set of temporally extended actions, initiation set $I(o)$: states in which option execution is afforded, termination condition/terminal state of skill(effect) $\beta_o(s)$ current state $s \neq$ option initiation state - precondition error control theory: closed-loop system: feedback from output, adaptive control precondition

HotpotQA: Plan Update

Task Description:
 Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be of three types:
 (1) search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to search.
 (2) lookup[keyword], which returns the next sentence containing keyword in the current passage.
 (3) finish[answer], which returns the answer and finishes the task. Call finish[] if the answer is not found.

Current Task Plan:

1. Identify the main keywords of entities.
2. Search for the main entity of keyword on Wikipedia using search[entity].
3. Look for the next sentence containing the keyword in the current Wikipedia page.
4. Repeat step 2 and 3 as necessary until the answer is found.
5. Finish the task with finish[answer].

Question 1: Maurice Hines and his brother were famous for what?
Summary 1: I searched for "Maurice Hines" on Wikipedia and found information about him and his brother Gregory Hines. I then looked up the keyword "tap dance" and found that both Maurice and Gregory Hines were famous for tap dancing. The answer provided was that Maurice Hines and his brother Gregory Hines were famous for tap dancing.
Flaws 1: Step 3 does not account for the possibility that the answer might already be present in the first paragraph returned by the search[] action. In such cases, the lookup[] action might not be necessary, and the assistant can directly proceed to step 5 to provide the answer.
Revision 1: Step 3. If the answer is not found in the first paragraph returned by search[entity], use lookup[keyword] to look for the next sentence containing the keyword in the current Wikipedia page opened by search[entity].

Based on the above experiences of the task, rewrite the current task plan. Pay more attention to summary of successful questions, and flawed actions and suggested revision of failed questions. The plan should not be specific to one question but generalizable to all questions. The actions in the plan should also be in the form as in task description.

> New Task Plan:

1. Identify the main keywords of entities.
2. Search for the main entity of keyword on Wikipedia using search[entity].
3. If the answer is not found in the first paragraph returned by search[entity], Look for the next sentence containing the keyword in the current Wikipedia page.
4. Repeat step 2 and 3 as necessary until the answer is found.
5. Finish the task with finish[answer].

Figure 30: Ouyang and Li [2023]

errors=corrective prompts current action history in working memory: for plan correction Re-prompting strategies: either only inform LLM that action failed Task Failed, implicit cause: or prompt template with name of failed action an environment objects I cannot <action> <object>, or explicit cause prompt template: precondition violation I cannot <action> <object> because <precondition violation> (only for ablation testing of corrective: Re-sampling: no need to reprompt. Evaluate top k admissible actions. re-evaluate if one action is not executable and pick another action) 100% executability because of re-prompting

DEPS: outcomes + reasoning about failures Wang et al. [2023d] self-explanation of feedback of failures action fails: descriptor: summary of current state end execution outcome of most recent goal. LLM: explainer/self-explanation to locate error, re-plan current task/generate revised plan (to original) prompt-template: python like code not only environment feedback that execution failed, but "describe, explain and plan" descriptor for feedback: human feedback/human description or pre-trained vision-language model, or returned information from (simulation) environment/available symbolic information explainer: few-shot CoT

Wang et al. [2024b] Executable Code Actions Elicit Better LLM Agents, CodeAct , environment feedback: code execution results as observations (results, errors)

Logeswaran et al. [2022] Few-shot Subgoal Planning with Language Models, multi-path! Lm predicts several subgoal sequences, re-rank to a ranked list of subgoal sequence, execute next subgoal from highest ranking plan observations: update state representation environment feedback: train supervised plan ranking model based on information from

environment/environment feedback/execution of plans and success/failure outcomes of the plans in the environment: construct labelled dataset of instructions with corresponding plans and agent state s, significant improvement

Silver et al. [2023] automated debugging (env exception feedback) very important for performance environment feedback: errors from code execution, give error to LLM and ask it to fix code itself Feedback: types: python exceptions, timeout, plan syntax, plan semantics exception:

```
Given this task: [PDDL Training Task]
The code raised the following exception:
File "<file-name-omitted>", line 86
lift_at = {atom[1]: atom[2] ...}
```

```
IndexError: tuple index out of range
Fix the code.
```

plan semantics

```
Given this task: [PDDL Training Task]
The code failed. It returned the following plan:
['(pick-up paper-1 loc-0)', ...].
NOTE: (pick-up paper-0 loc-0) has an unsatisfied precondition at time 3
(Set (at loc-0) to true)
Fix the code.
```

Gou et al. (2023a) introduced a framework called CRITIC critiquing by tool usage, own outputs check with appropriate external tools to evaluate output text

Huang et al. [2022a] Inner Monologue: Embodied Reasoning through Planning with Language Models ENVIRONMENT+ HUMAN Feedback (Inner Monologue: task completion, passive and active scene descriptions) embodied problems, understand available skills and how skills influence world natural language feedback from environment: success detection, scene description, and human interaction closed-loop language feedback inner monologue: thinking in language, thought process: actions, observations about outcomes, corrective actions embodied feedback with language actions: set of robot manipulation skills, textual descriptions perception models, robotic skills, and human feedback; chain in language prompt "retry under observed stochastic failure, replan under systematic infeasibility, or request human feedback for ambiguous queries, resulting in significantly improved performance in dynamical environments" possible observations: observation o may be success detection, object detection, scene description, visual-question answering, or even human feedback; can also be explicitly requested by planner can reason in long horizon setting without affordance-based grounding method incorporate textual feedback into prompts "closed-loop language feedback enables replanning even in complex unseen settings" reduce planning failures/failure rate, replan/recover from policy mistakes, long-horizon tasks, disturbances InstructGPT

Sun et al. [2023b] Interactive Planning Using Large Language Models for Partially Observable Robotics Tasks here of interest: partially observable aspects of environment. gain information through feedback Large Language Model for Partially Observable Task Planning(LLMPOP) framework to interactively plan with uncertainties robot collects environment observations several kinds of uncertainty: environmental uncertainty (non observable properties), skill execution uncertainty: outcome of execution uncertainty eg due to disturbances POMDPD partially observable Markov decision process (decision scenarios without complete state information) belief state: probabilistic estimation of full state, select action based on that and get observation of state transition and update belief state LLM needs to establish belief state and plan actions LLM input: task description from a user, the current observation from the robot, and the historical action and observation sequence from previous steps output: executable sequence of actions with text explanation 2 steps: LLM planner+ LLM evaluator prompt template for LLM Planner includes: Environment description, action options, output rules (set by user, constant) Task description (from users) Example outputs: in-context examples for planning Current observation and historical information (text format) prompt for LLM evaluator: background information, task description, and history observations; queries the planner LLM to do state abstraction (reason about missing information) and belief update, correct execution errors (from history) example prompt for planner

```
[Environment Context] A Franka Panda robot is placed in front of a table with cubes on top. The robot ...
[Task Description] <mass> Stack the lighter block on the heavier block.
[History] None.
[Action Options]
1) REACH <location> // end effector move to a desired pose
```

```

...
[Output Instruction]
The output has two components: explanation and action sequence. Explanation start with EXPLAIN, then g
[Output Examples(Not exactly for this task)]
EXPLAIN To move cubeA to the left of cubeB, pick up cubeA, and place it to the left of cubeB.
EXECUTE
ACTION PICK cubeA
...
History]
== Round 0 ==
[Response History] Plan explanation: First, I will pick up cubeA and lift it slightly to check its weight.
Action 0: PICK cubeA
Observation after step 0: CubeA: [0.11 0.13 1.20 0.01 0.00 0.39 0.92] ...

```

example prompt for evaluator

[Environment Context] A Franka Panda robot is placed in front of a table with cubes on top. The robot ...
[Task Description] <mass> Stack the lighter block on the heavier block.
[History] None.

Based on the [Task description], [History], please first evaluate if the robot has successfully completed the task. If the task is not finished, please explain:

1. Is there any missing information to solve the task? If [History] is included, also explain:
2. From the [History], what is the new task-related information we get? Show which action and observation:
3. Are there any failed actions (observation after the action is not as expected)? Recognize the action and its effect:

Example:

SUCCESS The robot has successfully completed the task.

FAIL Task is not finished. ...

Zhou et al. [2023a] LANGUAGE AGENT TREE SEARCH UNIFIES REASONING ACTING AND PLANNING IN LANGUAGE MODELS LATS (Language Agent Tree Search), a general framework inspired by monte carlo tree search LLM: agent, value function, optimizer external feedback from environment crucial? self-reflection external memory expand ReAct, outperforms ReAct introduce an LM-based Monte Carlo tree search variant to deliberately construct the best trajectory from sampled actions, using heuristics from LM external feedback and self-reflection: learn from experiences mitigate: autoregressive sampling like in CoT neglecting alternatives, some approaches like CoT relying only on internal LLM representation resulting in eg hallucinations, eg ToT suing simple search like BFS/DFS+no env feedback? other approaches also constrained to tasks where LM can act as the world model Usually MCTS requires environment model ReAct basis: receive observation , take action a following policy base on instruction+ggf few-shot examples ReAct actions: permissible actions and reasoning traces (thoughts) policy: $\pi(a_t|x, o_1 \dots o_{t-1}, a_1 \dots a_{t-1})$ sample n actions, assumption: for complex decision making tasks probably several correct trajectories/reasoning paths + mitigate stochastic text generation + greater exploration MCTS: tree with nodes = states, of input and action and observations sequence state: $s = [x, a_1 \dots a_t, o_1 \dots o_t]$ model-free operations: selection, expansion, evaluation, simulation, backpropagation, and reflection, selection: UCT algorithm expansion: sample n actions, give to environment and get feedback as observation, add child nodes evaluation: LLM reason about state, attribute scalar value for correctness of trajectory simulate: until terminal state is reached, backpropagate value, self-reflection, eg unsuccessful terminal node about errors in reasoning or acting Observe performance gain of using MCTS over variants like DFS higher computational cost for tasks where performance is more important than efficiency (eg programming?) OWN ANALYSIS: valuable if actions take short time and are fully reversible! because several actions are actually executed!, sometimes no roll-back or high costs!

Singh et al. [2023] PROGPROMPT: program generation for situated robot task planning using large language models environment feedback: assertions in code, get current semantic state of environment and check if assertion holds, otherwise corrective action Singh et al. [2023] PROGPROMPT: program generation for situated robot task planning using large language models problem: situated environment, executable actions "prompt the LLM with program-like specifications of the available actions and objects in an environment, as well as with example programs that can be executed" VirtualHome household tasks, physical robot arm tabletop tasks tasks: knowledge about: object affordances, logical sequences of actions, task relevance of objects and actions - needs state feedback "situated-awareness in LLM-based robot task planning" prompting scheme utilizing programming language structures (LLM pretrained among others programming/code) pythonic prompt structure import statement for available actions and parameters, list of environment objects and function definitions with sequences of actions operating on objects asserting preconditions in

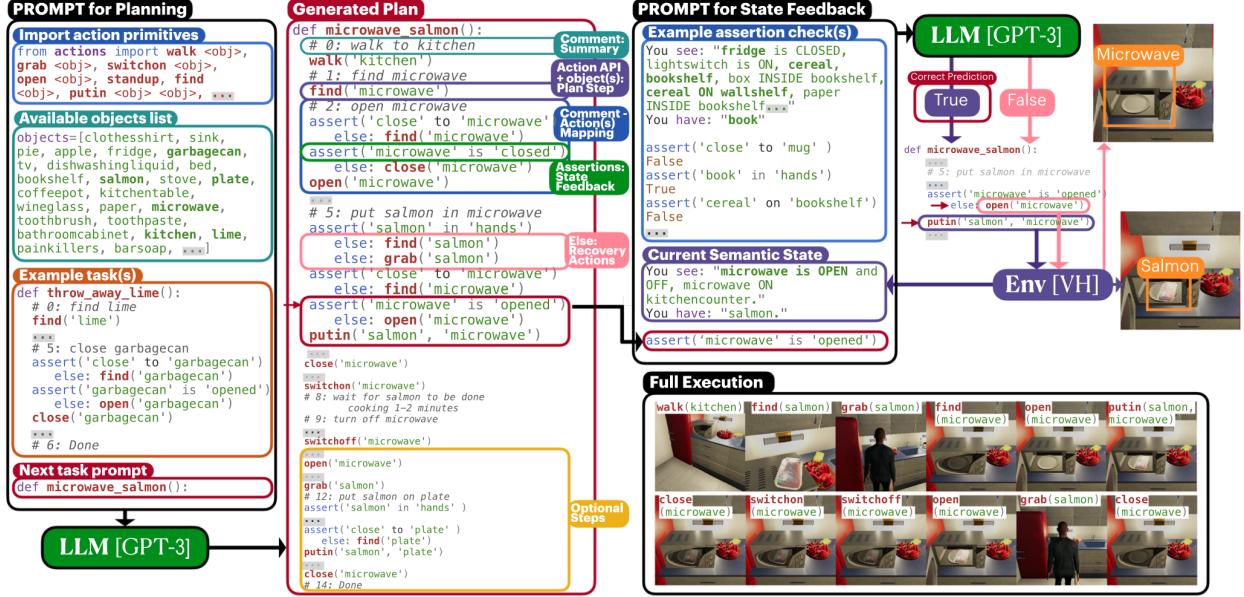


Figure 31: ProgPROMPT Singh et al. [2023]

environment, state feedback, recovery actions natural language comments to explain goal of action using comments for tasks and feedback (assertions and recovery actions) both better performance a test using not programming language structure but natural language performed significantly worse

6.3 Human Feedback

subjective aliment: values, preferences

what can actions do? information retrieval...

Inner Monologue: ask for feedback

Chain of Hindsight: past outputs annotated with human feedback, reated, model finetuning

Zhou et al. [2023b] Open-source Framework for Autonomous Language Agents , supports human-agent interaction, humans can communicate and interact with language agents

Wu et al. [2023b] AutoGen, participate in agent conversation

7 Memory

DRAFT: ONLY NOTES

INTRO: what is meant by memory, state of planning, current...short-term memory, and long-term: past behaviour reading/retrieval see 121: recent, relevant, important, often only relevance! writing: duplicates? storage limit? normally: previous result(s)

7.1 Working Memory

INTRO: Why working memory is necessary! need to keep track of the state of planning and plan execution and intermediary results flow, observations, progress,... context information recent perceptions current trajectory

Zhou et al. [2023b] Open-source Framework for Autonomous Language Agents update via scratchpad, prompt (natural language)

Sun et al. [2023b] Interactive Planning Using Large Language Models for Partially Observable Robotics Tasks everything appended to history, which is natural language text appended to prompt

```
ACTION PICK cubeA
...
[History]
== Round 0 ==
[Response History] Plan explanation: First, I will pick up cubeA and lift it slightly to check its weight.
Action 0: PICK cubeA
Observation after step 0: CubeA: [0.11 0.13 1.20 0.01 0.00 0.39 0.92] ...
```

Wang et al. [2024d] RecMind: Large Language Model Powered Agent For Recommendation provides zero-shot personalized recommendations external knowledge, tools, planning self-inspiring algorithm, at each step self-inspire by considering all previous states when planning for the next step - by this use historical information, integrate multiple reasoning paths key components: planning: break recommendation tasks into steps, each step: thought, action, observation Memory: Personalized memory(user information eg reviews) and world knowledge (domain specific metadata+ knowledge accesses eg via web) tools: eg for information retrieval, database , search, text summarization , expert models (HuggingFace), SQL Tool, Search Tool based on ToT, explore all previous paths Recmind often outperforms fully-trained models for rating prediction tasks direct recommendation tasks, fully trained models for recommendation usually better because of long context needed for RecMind, sequential recommendation: comparable results with fully trained models advantage: easily transferable to unseen domains compared to specifically trained models Llama2 70b, GPT-3.5, text-davinci-003, and GPT-4, not overly dependent on foundation LLM, but problems when input context length limited Self-inspiring prompt: "You are given multi-step problem-solving steps towards finishing the task task. The previous steps are previous_steps. You already have the thought, action, and observation in the current step current_step. Your mission is to decide if there is an alternative thought in the current step that can help finish this task following the previous steps. If there is, directly output the thought. If not, please respond empty_response." CoT prompt: "Solve a recommendation task with interleaving Thought, Action, and Observation steps." thought sampling prompt is "Given the previous previous_steps, list five possible thoughts for the next step towards finishing the task task." The decision-making prompt is "Given an instruction and several choices, decide which choice is most promising. Your instruction is task_specific_instruction. Your available options are option_list. Analyze each choice, then conclude in the last line, 'The best choice is s', where s is the integer id of the choice."

7.2 Long-Term Memory

? TODO also add to memory? Paul [2023] Sequential Planning in Large Partially Observable Environments guided by LLMs, neoplanner, assumption: partial observability sequential planning in large state space and action space LLMs: if state space, action space and observations can be represented in natural language hybrid agent: state space search + queries to LLM for action plan, use reward signal balance of exploration and exploitation ". Learnings from each trial are stored as entity relationships in text format", used in future queries test environment: scienceworld: "interactive text-based environment that demands intricate interactive reasoning processes for resolving a multitude of science-theory-based tasks", no specific state information available, only observations combination of LLM and reinforcement learning (RL) "propose to build a state space model of the environment by trying out different actions, recording observations and rewards" LLM predicts best sequence of actions build memory of learnings about environments state space graph "As states are not directly accessible from the environment, the observations can be encoded in certain way to derive the latent state" needed: some form of reward assigned to a state for known states can select actions according to future rewards for exploitation for exploration, let a LLM generate action plan from current state LLM for action plan generation LLM as learner to generate learnings about environment: trace of episode, previous learnings, feedback, generate learnings as list of texts, with subjects, objects and relations between them reward transformed into natural language feedback GPT4-turbo possible learnings, relations in environment: "thermometer in kitchen can be moved to inventory" query LLM once to find a sequence of actions, carry out, less LLM calls roles: Action plan generator prompt System: You are an AI action planner for an autonomous agent. You are situated in a task environment, as provided by the user,... Learner prompt System: You are an expert assistant.

7.2.1 Experiences: Storing Trajectories

Kagaya et al. [2024] RAP: Retrieval-Augmented Planning with Contextual Memory for Multimodal LLM Agents "dynamically leverage past experiences corresponding to the current situation and context, thereby enhancing agents' planning capabilities" "storing past experiences in memory, retrieving them appropriately based on the similarity with present context including multimodal information, and generating subsequent actions via in-context learning" framework, which consists of four core components: Memory, Reasoner, Retriever, and Executor memory: databases with logs of prior successful task executions what is saved: Task information, overall plan and trajectory (plans, actions, observation sequences) similarity for retrieval: similarity score, weighted average of task similarity, overall plan alignment and retrieval key congruence (with log trajectory)

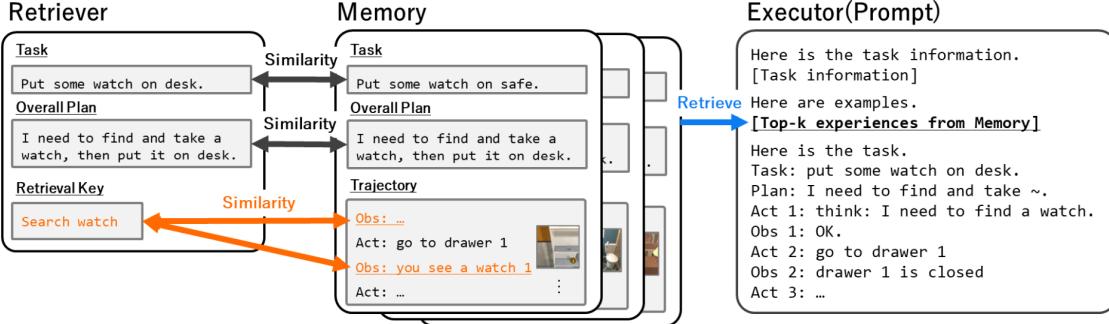


Figure 32: RAP Memory Retrieval Kagaya et al. [2024]

Du et al. [2024] AnyTool: record of historical context, identified reason of unresolved user query, APIs that resolved to be irrelevant: can then be removed from candidate pool

Sun et al. [2023a] past failures and successes for planning, "Skill discovery process", accumulating successful experiences: enhance long term planning, + sample efficiency successfully plans and their interactions with environment: if task resembles skills, skills are used as few-shot example in agent prompt improves sample efficiency and reliability Skill Acquisition: unseen tasks trial and error, successful solution: candidate skill skill filtering: comparison planning with and without candidate skill as example in prompt, if higher success rate wit in prompt keep, else discard ablation study: skill discovery significantly enhanced performance

7.2.1.1 Format and Retrieval TODO: MAYBE NO SUBPARAGRAPHS AS NOT MANY EXPLANATIONS AVAILABLE IN LITERATURE ANYWAY!

Kagaya et al. [2024] RAP: Retrieval-Augmented Planning with Contextual Memory for Multimodal LLM Agents "dynamically leverage past experiences corresponding to the current situation and context, thereby enhancing agents' planning capabilities" "storing past experiences in memory, retrieving them appropriately based on the similarity with present context including multimodal information, and generating subsequent actions via in-context learning" framework, which consists of four core components: Memory, Reasoner, Retriever, and Executor memory: databases with logs of prior successful task executions what is saved: Task information, overall plan and trajectory (plans, actions, observation sequences) similarity for retrieval: similarity score, weighted average of task similarity, overall plan alignment and retrieval key congruence (with log trajectory)

7.2.1.1.1 Natural Language Log general log?, past plans/thoughts/actions/observations/feedback, interactions

7.2.1.1.2 Database ChatDB: database as symbolic memory module DB-GPT

7.2.1.1.3 Vectorstore vector store? fast retrieval needed embeddings

Zhou et al. [2023b] Open-source Framework for Autonomous Language Agents store and retrieve, VectorDB, semantic search, action histories, embedded by sentence transformers

GITM? key-value store of embeddings and natural language can improve agent in long run when "learning" from experiences (not finetuning, but changing prompts and incorporating into prompts/context) GITM: exploration store actions with which successfully completed task, can reuse

7.2.2 Experience Summaries: Generating and Storing Reflections

Shinn et al. [2023] Reflexion: Language Agents with Verbal Reinforcement Learning linguistic feedback, verbally reflect on task feedback episodic memory buffer to save reflective texts - for usage in subsequent trials types of feedback: scalar values, free-form language can convert binary or scalar environment feedback to verbal feedback/textual summary, add as context for LLM agent in the next episode self-reflective feedback experience summaries stored in long-term memory "more explicit and interpretable form of episodic memory over prior experiences" use as explicit hints for actions in future episodes Actor model: use CoT or ReAct prompting on LLM, generate policy via model, use memory as context generate self-reflection with self-reflection model and add to memory self-reflection model: LLM, crucial, generates verbal self-reflections, given: reward signal, current trajectory and persistent memory and store into memory Memory: long-term and short-term. Actor model uses short term and long term memory, short-term: trajectory history,

long-term: outputs of self-reflection model long-term memory: can only use part: sliding window with maximum capacity needs some form of evaluation

Park et al. [2023] Generative Agents: Interactive Simulacra of Human Behavior, one aspect: long-term memory, including the observations, reflections and plans agent architecture components: observation, planning, reflection every day tasks/believable individual and emergent behaviour of several autonomous agents 3 main components: 1 memory stream (long-term memory in natural language of agent experiences) + memory retrieval model: relevance+recency+importance 2 reflection: synthesize memories into higher level inferences, draw conclusions 3 planning translate conclusions from reflection + current environment into high-level action plans, recursively detailed role: a paragraph in natural language of identity description , including relations to others interaction: natural language statement output describing action (must be translated into actual action in the world) Agents communicate with each other in natural language memory

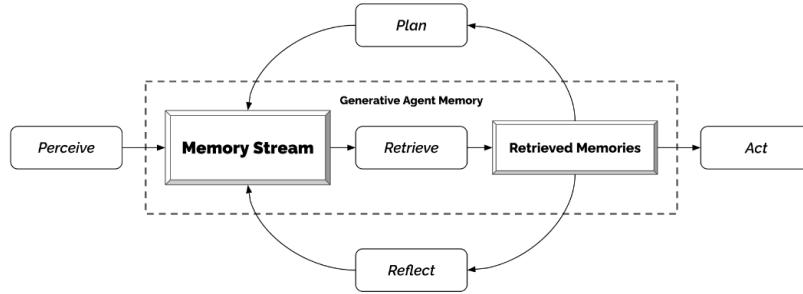


Figure 33: SimulacraEmergent-Architecture Park et al. [2023]

stream: database, does not fit into context window record: "It is a list of memory objects, where each object contains a natural language description, a creation timestamp, and a most recent access timestamp" basic element: observation retrieval: current situation + return subset of memory stream, retrieval function: recency (exponential decay function), importance (model assigns integer score from 1 to 10 from mundane to poignant at memory creation), relevance: related to current situation conditioned on query memory(language model creates embedding vector of text description of each memory + relevance as cosine similarity between memory and query embedding vectors). Normalize the three, add, return maximum of all memories synthesise records into higher level reflections reflection: second memory for reflections, generated periodically give language model recent experiences + prompt: "Given only the information above, what are 3 most salient highlevel questions we can answer about the subjects in the statements?"

Statements about Klaus Mueller

1. Klaus Mueller is writing a research paper
2. Klaus Mueller enjoys reading a book on gentrification
3. Klaus Mueller is conversing with Ayesha Khan about exercising [...]

What 5 high-level insights can you infer from the above statements? (example format: insight (because

This process generates statements such as Klaus Mueller is dedicated to his research on gentrification (because of 1, 2, 8, 15). store as reflection in memory stream + pointers every thing recorded - reason obver in natural language ChatGPT planning: for consistent behaviour, generate plan and store in memory, top down and more detailed recursively prompt: description of agent summary + "Here is Eddy's plan today in broad strokes: 1)" observations from environment prompt: <observation description + memories> "Should John react to the observation, and if so, what would be an appropriate reaction?"

8 Multiagent Planning

DRAFT: ONLY NOTES

Zhou et al. [2023b] Open-source Framework for Autonomous Language Agents, additionally to "usual" single agent approach also multiagent communication Zhou et al. [2023b] possible either hard-coded ordering or dynamic scheduling using one controller agent as moderator, next agent dependent on previous actions, environment and target of current states

specific, finite process vs dialogue/cycles/cyclic

multi-agent debate Du et al., 2023). Liang et al. (2023) degeneration-of-thought (DoT) problem , LLM no novel thoughts although initial solution incorrect instead multiple agents with opinions and a judge for managing debate and decide for solution, multiple LLM instances

8.1 Collective Planning in Fixed Processes

Hong et al. [2023] METAGPT: META PROGRAMMING FOR A MULTI-AGENT COLLABORATIVE FRAMEWORK meta-programming framework GPT-based Meta-Programming framework "MetaGPT models a group of agents as a simulated software company" human workflows in LLM-based multi-agent collaboration Standardized Operating Procedures (SOPs) to prompt sequences, streamlined workflows, domain-expertise agents, assign roles to agents create intermediate structured output documents, similar to usual software engineering workflow - increases success rate of target code generation message protocol, structures messages, publish-subscribe mechanism, message pool,



Figure 34: MetaGPT Hong et al. [2023]

communicate through documents and diagrams rather than dialogue, store messages in global message pool, everyone

can publish there and access messages from other agents specialized roles to handle specific tasks profile definition: name, profile, goal, constraints, + provide specific context and skills iterative code production: Engineer role can execute unit test cases and receives test results MetaGPT outperforms all preceding approaches on HumanEval and MBPP benchmarks, best with GPT-4? ablation studies effectiveness/significance of roles

8.2 Collective Planning via Conversation

communication: direct, shared memory

Li et al. [2023b] autonomous cooperation among communicative agents communicative agent framework: role-playing tackling problem: defining a suitable prompt for the problem role-playing with inception prompting, guide towards task completion multi-agent collaboration for complex-task solving cooperative: agents together with agents or humans, to achieve common goal, collaboration, coordination auto-prompting method: inception prompting human user input: idea human: role specification of the agents/select roles task specifier: make task more specific to well-defined task, concrete, imagination module role assignment: system message declares role of agent, assistant/user agents based upon

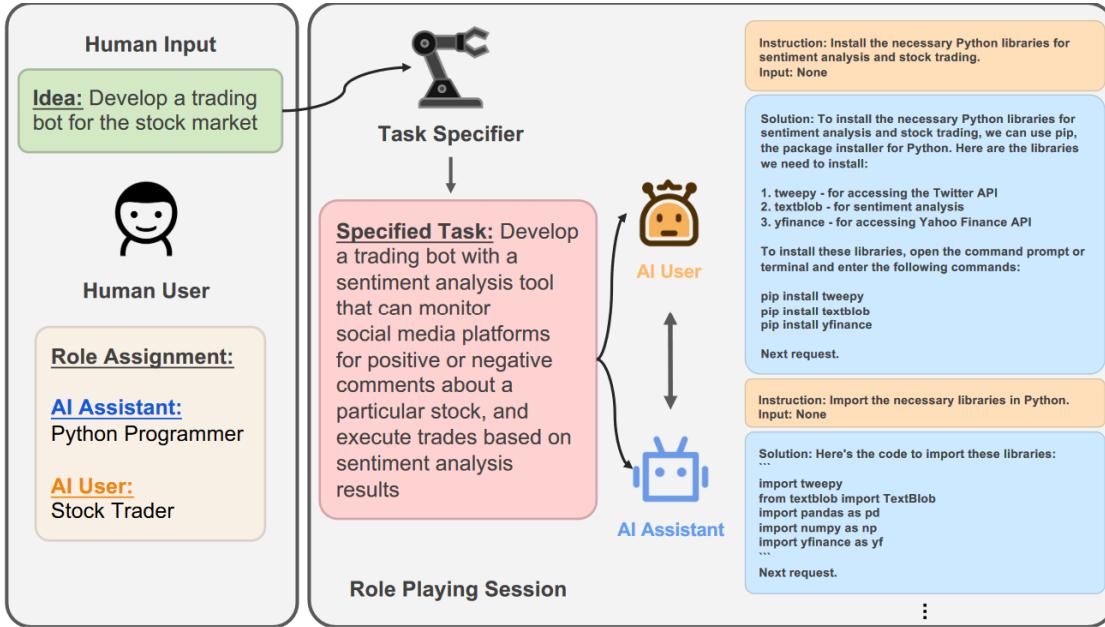


Figure 35: CAMEL Li et al. [2023b]

large-scale autoregressive language models 2 roles: AI user provides instructions, AI assistant responds with a solution that fulfils the instructions, set of conversational messages: $M_t = \{(I_0, S_0), \dots, (I_t, S_t)\} = \{(I_i, S_i)\}_{i=0}^t$ messages: automatic prompting in conversation, not predefined: inception prompting infinite loop of messages can occur (set max number of messages) other problems: repeats instruction instead of acting, flake replies (reply that will do a task but don't do it) Possible to use task agents/construct task agents, which specify the task and make a list of planned subtasks. Add task planner to original prompt to break down task into smaller subtasks embodied agents, physical entities, solve tasks in real world like browsing the internet Critic: rather see Feedback! solution produced using our proposed framework outperforms zero-shot-CoT by a large margin (with GPT-4 Evaluation) prompt ablation studies

Wu et al. [2023b] AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation agents can be based on: LLMs, tools, humans, or combination computation: actions of agent to compute response control flow: sequence of computations unified interface: send/receive, generate_reply agent auto reply mechanism, conversation driven control: receive, invoke generate_reply, send back or other, custom behaviour patterns natural language conversation and also Python code can be used for specifications example: retrieval-augmented chat: two agents, RAG user proxy agent with Vector DB with sentence transformers (Reimers & Gurevych, 2019) as context retriever, retrieves relevant chunks as context RAG assistant agent generates code or text to answer question, is either satisfied or asks for context update if initial most similar context does not include necessary information, it is updated...increases chance of finding relevant knowledge Wu et al. [2023b] AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation individual agent: role, built-in capability selection or extension, conversable (message receive, react, respond) LLM application

AI Society Inception Prompt

Task Specifier Prompt:

Here is a task that <ASSISTANT_ROLE> will help <USER_ROLE> to complete: <TASK>. Please make it more specific. Be creative and imaginative. Please reply with the specified task in <WORD_LIMIT> words or less. Do not add anything else.

<p>Assistant System Prompt:</p> <p>Never forget you are a <ASSISTANT_ROLE> and I am a <USER_ROLE>. Never flip roles! Never instruct me!</p> <p>We share a common interest in collaborating to successfully complete a task.</p> <p>You must help me to complete the task.</p> <p>Here is the task: <TASK>. Never forget our task!</p> <p>I must instruct you based on your expertise and my needs to complete the task.</p> <p>I must give you one instruction at a time.</p> <p>You must write a specific solution that appropriately completes the requested instruction.</p> <p>You must decline my instruction honestly if you cannot perform the instruction due to physical, moral, legal reasons or your capability and explain the reasons.</p> <p>Unless I say the task is completed, you should always start with:</p> <p>Solution: <YOUR_SOLUTION></p> <p><YOUR_SOLUTION> should be specific, and provide preferable implementations and examples for task-solving.</p> <p>Always end <YOUR_SOLUTION> with: Next request.</p>	<p>User System Prompt:</p> <p>Never forget you are a <USER_ROLE> and I am a <ASSISTANT_ROLE>. Never flip roles! You will always instruct me.</p> <p>We share a common interest in collaborating to successfully complete a task.</p> <p>I must help you to complete the task.</p> <p>Here is the task: <TASK>. Never forget our task!</p> <p>You must instruct me based on my expertise and your needs to complete the task ONLY in the following two ways:</p> <ol style="list-style-type: none"> 1. Instruct with a necessary input: Instruction: <YOUR_INSTRUCTION> Input: <YOUR_INPUT> 2. Instruct without any input: Instruction: <YOUR_INSTRUCTION> Input: None <p>The "Instruction" describes a task or question. The paired "Input" provides further context or information for the requested "Instruction".</p> <p>You must give me one instruction at a time.</p> <p>I must write a response that appropriately completes the requested instruction.</p> <p>I must decline your instruction honestly if I cannot perform the instruction due to physical, moral, legal reasons or my capability and explain the reasons.</p> <p>You should instruct me not ask me questions.</p> <p>Now you must start to instruct me using the two ways described above.</p> <p>Do not add anything else other than your instruction and the optional corresponding input!</p> <p>Keep giving me instructions and necessary inputs until you think the task is completed.</p> <p>When the task is completed, you must only reply with a single word <CAMEL_TASK_DONE>.</p> <p>Never say <CAMEL_TASK_DONE> unless my responses have solved your task.</p>
---	--

Figure 36: CAMEL Inception Prompt Li et al. [2023b]

workflows as multi-agent conversations interaction: conversation-centric computation and control Wu et al. [2023b] AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation agents converse to solve tasks, chat among themselves and also with human in the loop flexible agent conversation patterns single- or multi-turn dialogs different human involvement modes and static vs. dynamic conversation

8.3 Separate Planning and Agent Debate

Li et al. [2024] "We find that, simply via a sampling-and-voting method, the performance of large language models (LLMs) scales with the number of agents instantiated" "LLM performance may likely be improved by a brute-force scaling up the number of agents instantiated" sampling-and-voting method with 2 phases: query into LLMs, generate multiple outputs, majority votin to determine final result, majority voting: by cumulative similarity for each sample relative to the others (need to quantify similarity), choose sample with highest cumulative similarity no additional hand crafted prompt design tasks applied: arithmetic reasoning, general reasoning, code generation, different scaled Llama 2 and GPT models accuracy gain by just using approach, already significant is using 10 instead of 1, ensemble size

9 LLM-Based Agents Evaluations and Benchmarks?

DRAFT: ONLY NOTES

TODO Remove because all Benchmarks/evaluations already mentioned in LLM comparison?!

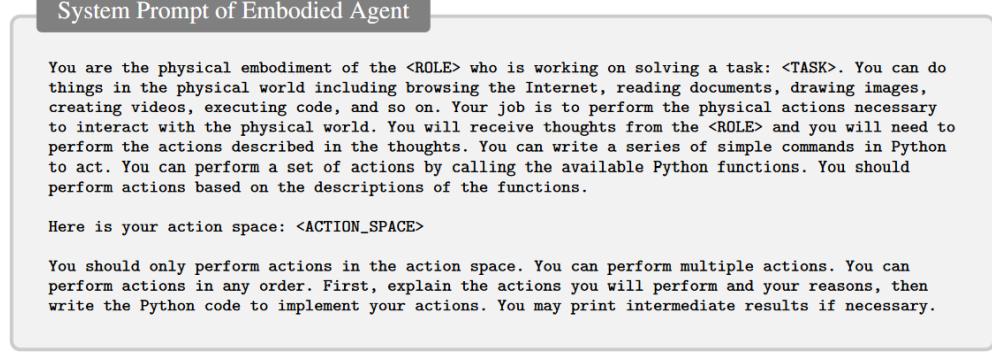


Figure 37: CAMEL Embodied ActionLi et al. [2023b]

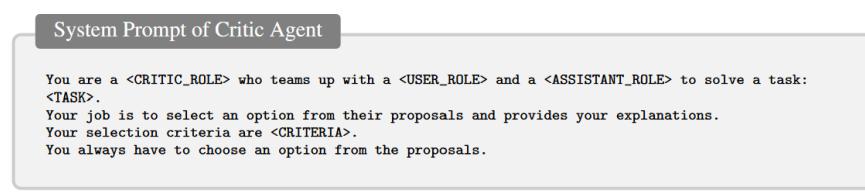


Figure 38: CAMEL Li et al. [2023b]

evaluation: for tasks similar to use case?! not game/simulation like ALFWorld can approaches be compared/how are whole approaches evaluated and how are the single components evaluated?

Ablation studies: understanding the impact of every component in the system How to measure performance of single/isolated component? [https://en.wikipedia.org/wiki/Ablation_\(artificial_intelligence\)](https://en.wikipedia.org/wiki/Ablation_(artificial_intelligence))

human evaluation: criteria: helpfulness,...score/rank results or similar to Turing test metrics/benchmarks

has that been done?: Comparative study: only LLM, LLM+RAG, LLM-based agent Evaluation: Same test protocol for every configuration Maybe human evaluation of answers! Borda count? ranked voting

AgentSims? ToolBench WebShop? Gentbench

Gur et al. [2023] percentage of acquired attributes due to the request (eg matching the real-estate request)

Sun et al. [2023a] ALFWorld text based virtual household environment, MiniWoB++ simulation environment of computer tasks, success rate (number of successful episodes over total number of episodes) ablation study, on gpt version gpt-3.5-turbo underperforms, not using code style significant performance drop.... skill discovery ALFWorld (Shridhar et al., 2021) tztext based virtual household environment with household tasks WebShop. WebShop (Yao et al., 2022) online shopping website environment, website navigation and decision making Minecraft: Prasad et al. [2023] TextCraft, text-only environment for minecraft, crafting minecraft items However , in these environments like ALFWorld etc, success (for success rate) is soundly? defined. But this is not the case in setting assumed here!

Ouyang and Li [2023] human evaluation on model predictions on HotpotQA check accuracy, check consistency with supporting facts

Li et al. [2023b] Evaluation: HumanEval [18] and HumanEval+ [69], and GPT4 for ChatBot Evaluation (GPT4 agent decides of two solutions which is better)

Raman et al. [2023] evaluation Human evaluation: via crowdsourcing platform prolific, annotator evaluate grounded okab in English if accomplishes task objective, semantic correctness and relevance, consider Fleiss Kappa model for inter-annotator agreement metrics: number of steps, number of corrections, number of prompts

Wang et al. [2024b] Executable Code Actions Elicit Better LLM Agents, CodeAct benchmarks: API-Bank, correctness metric curate new benchmark: M^3 ToolEval , human curated tasks requiring multiple calls to multiple tools in multi-turn interactions

Shen et al. [2023] HuggingGPT, eval Evaluation: GPT-4 evaluated the generated plans (via prompt), bewrten selected feasible task, relationships between tasks, gts positive and negative demonstrations

Ma et al. [2024] AGENTBOARD: AN ANALYTICAL EVALUATION BOARD OF MULTI-TURN LLM AGENTS benchmarking agent across diverse scenarios especially: partially-observable environments, multi-round interactions progress rate metric

Li et al. [2023a] API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs how effective LLMs in using tools? benchmark for tool-augmented LLMs 73 API tools, tool use dialogues

Mialon et al. [2023b] GAIA: A Benchmark for General AI Assistants

Nan et al. [2023] On Evaluating the Integration of Reasoning and Action in LLM Agents with Database Question Answering question answering dataset designed to evaluate how Large Language Models (LLMs) interact with a SQL interpreter, generate multiple SQL queries "Sequential: The LLM agent systematically tackles the sub-tasks in a linear, step-by-step fashion, with predetermined sequence: interaction planning, tool employment, and information synthesis" "Iterative: The LLM agent cyclically alternates between interaction planning and tool employment"

Valmeekam et al. [2023] "benchmark suite based on the kinds of domains employed in the International Planning Competition. On this benchmark, we evaluate LLMs in three modes: autonomous, heuristic and human-in-the-loop"

Guo et al. [2023] multturn, multi-modal instructions PowerPoint Task Completion (PPTC) benchmark to assess LLMs' ability to create and edit PPT files based on user instructions

Wu et al. [2023a] SMARTPLAY : A BENCHMARCH FOR LLMS AS INTELLIGENT AGENTS SmartPlay consists of 6 different games, including Rock-Paper-Scissors, Tower of Hanoi, Minecraft

Pallagani et al. [2023] Understanding the Capabilities of Large Language Models for Automated Planning "consider six classical planning domains represented in PDDL, released as a part of the IPC, to assess planning capabilities in LLMs" Comparison with FastDownward Planner (as ground truth solution)

Chen et al. [2024] When is Tree Search Useful for LLM Planning? It Depends on the Discriminator multi-step LLM planning we systematically analyze different planning methods in a unified generator/discriminator framework "ur framework consists of a generator that proposes (partial) action sequences, a discriminator that evaluates the outcomes of these actions, and a planning method that ranks the actions according to their outcomes and manages the interaction between the two models" text-to SQL parsing, mathematical reasoning

10 Issues, Open Challenges, Discussion

DRAFT: ONLY NOTES

Planning Problem Representation and Assumptions Long-term planning vs finite context window of LLMs alignment robustness, reliability: sensitive to prompt changes, developer trial and error strategy hallucinations efficiency/amount of requests/energy and costs long-term, path, adapt plan natural language: ambiguity, not exact...

privacy, sensitive data - privacy protocol?! errors: biases, hallucinations unintended consequences: tool usage, side-effects

user education, transparency, ethical check/guidelines, human oversight

critical reflection social, ethical, legal aspects

no discussion about: problem assumptions like environment characteristics, planning problem assumptions, nothing about complexity (is it really "better" to make a plan of tasks with preconditions and effects by reasoning over LLM instead of just using classical forward-search planning?? so it is an interesting (from engineering perspective) but scientifically horrible topic ;-)

SEL aspects in the analysed literature itself?

LLM-Powered Planning AI Agents: Open Issues and Research Current Further Developments Further prompting techniques Evaluations and benchmarks

Open Issues Finite context window vs. long-term planning Systematic prompting assumptions State space and world model Planning problem assumptions (observability, determinism, ...) Explicit concept of planning approaches (HTN, conditional, probabilistic, ...) Other combinations of LLMs and symbolic knowledge Human-in-the-loop reliability, energy consumption/sustainability, economical, explainability, autonomy

How will the work of engineers be influenced?

computational power of training phase potential copyright infringements in training phase power consumption in inference agent – agency??

referring to Davis and Marcus [2015] web mining based approach: LLM does statistical reasoning over language. No other kinds of reasoning are involved! If other kinds seem to be appropriate from human perspective, maybe need to be added separately

Privacy and Security Issues in Deep Learning: A Survey A critical overview of privacy in machine learning limitations, inconsistencies or shortcomings of previous studies alignment of language agents? explainable AI: why did the LLM decide/reason on this task/sequence? prompt injection attack: customer (or there internal attacker) prompt injection, attacks LLM to return classified information-> because the LLM has ACCESS to the information! maybe also misuse the LLM to do other tasks, writing internal e-mails etc, steal personal information WHICH contents can be reported to the customer? -> Which documents are customers allowed to see? How does the LLM understand the probability of the answer/output? How to establish trustworthiness of the result? -> Answer customer an estimation of trustworthiness. Possibility: scores from last output layer -> if low, maybe need more information or re-computation tasks in internet eg web automation can be critical: security + unintended side effects (eg booking is done, illegal content accessed)

Manning [2022] general risks of using large models on specialized tasks, because they can be easily adapted: maybe only small number of models available= expensive and time consumption of training, model owners: power and influence biases in the models not knowing if models are safe, cannot easily be checked because of size of models and training data

PDDL with incomplete knowledge??

11 Conclusion

References

- Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, June 1995. ISSN 0269-8889, 1469-8005. doi:10.1017/S0269888900008122. URL https://www.cambridge.org/core/product/identifier/S0269888900008122/type/journal_article.
- Michael Wooldridge. Intelligent Agents: The Key Concepts. In G. Goos, J. Hartmanis, J. Van Leeuwen, Vladimír Mařík, Olga Štěpánková, Hana Krautwurmová, and Michael Luck, editors, *Multi-Agent Systems and Applications II*, volume 2322, pages 3–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-43377-4 978-3-540-45982-8. doi:10.1007/3-540-45982-0-1. URL <http://link.springer.com/10.1007/3-540-45982-0-1>. Series Title: Lecture Notes in Computer Science.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The Rise and Potential of Large Language Model Based Agents: A Survey, September 2023. URL <http://arxiv.org/abs/2309.07864>. arXiv:2309.07864 [cs].
- Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark), February 2023. URL <http://arxiv.org/abs/2302.06706>. arXiv:2302.06706 [cs].
- Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of Artificial General Intelligence: Early experiments with GPT-4, April 2023. URL <http://arxiv.org/abs/2303.12712>. arXiv:2303.12712 [cs].
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face, May 2023. URL <http://arxiv.org/abs/2303.17580>. arXiv:2303.17580 [cs].
- Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized Planning in PDDL Domains with Pretrained Large Language Models, December 2023. URL <http://arxiv.org/abs/2305.11014>. arXiv:2305.11014 [cs].
- Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency, September 2023a. URL <http://arxiv.org/abs/2304.11477>. arXiv:2304.11477 [cs].

- Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic Planning with a LLM, August 2023. URL <http://arxiv.org/abs/2308.06391> [cs].
- Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Aslı Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented Language Models: a Survey, February 2023a. URL <http://arxiv.org/abs/2302.07842>. arXiv:2302.07842 [cs].
- Lilian Weng. LLM Powered Autonomous Agents, June 2023. URL <https://lilianweng.github.io/posts/2023-06-23-agent/>.
- Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. BOLAA: Benchmarking and Orchestrating LLM-augmented Autonomous Agents, August 2023b. URL <http://arxiv.org/abs/2308.05960>. arXiv:2308.05960 [cs].
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, December 2024a. ISSN 2095-2228, 2095-2236. doi:10.1007/s11704-024-40231-1. URL <https://link.springer.com/10.1007/s11704-024-40231-1>.
- Pengyu Zhao, Zijian Jin, and Ning Cheng. An In-depth Survey of Large Language Model-based Artificial Intelligence Agents, September 2023. URL <http://arxiv.org/abs/2309.14365>. arXiv:2309.14365 [cs].
- Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive Architectures for Language Agents, September 2023. URL <http://arxiv.org/abs/2309.02427>. arXiv:2309.02427 [cs].
- Thorsten Händler. A Taxonomy for Autonomous LLM-Powered Multi-Agent Architectures:. In *Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 85–98, Rome, Italy, 10 2023. SCITEPRESS - Science and Technology Publications. ISBN 978-989-758-671-2. doi:10.5220/0012239100003598. URL <https://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0012239100003598>.
- Yule Wang. A Complete Guide to LLMs-based Autonomous Agents (Part I), October 2023. URL <https://medium.com/@yulemoon/a-complete-guide-to-langs-based-autonomous-agents-part-i-69515c016792>.
- Zhuosheng Zhang, Yao Yao, Aston Zhang, Xiangru Tang, Xinbei Ma, Zhiwei He, Yiming Wang, Mark Gerstein, Rui Wang, Gongshen Liu, and Hai Zhao. Igniting Language Intelligence: The Hitchhiker’s Guide From Chain-of-Thought Reasoning to Language Agents, November 2023. URL <http://arxiv.org/abs/2311.11797>. arXiv:2311.11797 [cs].
- Vinod Muthusamy, Yara Rizk, Kiran Kate, Praveen Venkateswaran, Vatche Isahagian, Ashu Gulati, and Parijat Dube. Towards large language model-based personal agents in the enterprise: Current trends and open problems. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6909–6921, Singapore, 12 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.findings-emnlp.461. URL <https://aclanthology.org/2023.findings-emnlp.461>.
- Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, and Xiuqiang He. Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects, January 2024. URL <http://arxiv.org/abs/2401.03428>. arXiv:2401.03428 [cs].
- Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of LLM agents: A survey, February 2024. URL <http://arxiv.org/abs/2402.02716>. arXiv:2402.02716 [cs].
- Stuart J. Russell, Peter Norvig, and Ernest Davis. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, 3rd ed edition, 2010. ISBN 978-0-13-604259-4.
- Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016a. doi:10.1017/CBO9781139583923.
- Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, New York, NY, 2016b. ISBN 978-1-107-03727-4. URL <https://projects.laas.fr/planning/>.
- A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003. ISSN 0004-3702. doi:[https://doi.org/10.1016/S0004-3702\(02\)00374-0](https://doi.org/10.1016/S0004-3702(02)00374-0). URL <https://www.sciencedirect.com/science/article/pii/S0004370202003740>. Planning with Uncertainty and Incomplete Information.

- Ugur Kuter, Dana Nau, Marco Pistore, and Paolo Traverso. A hierarchical task-network planner based on symbolic model checking. *ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pages 300–309, 01 2005.
- Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning: theory and practice*. Elsevier/Morgan Kaufmann, Amsterdam ; Boston, 2004. ISBN 978-1-55860-856-6. URL <https://projects.laas.fr/planning/aptop/index.html>.
- Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing, July 2021. URL <http://arxiv.org/abs/2107.13586>. arXiv:2107.13586 [cs].
- Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. The Impact of Reasoning Step Length on Large Language Models, January 2024. URL <http://arxiv.org/abs/2401.04925>. arXiv:2401.04925 [cs].
- Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive Planning from Feedback with Language Models. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 58202–58245. Curran Associates, Inc., 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/b5c8c1c117618267944b2617add0a766-Paper-Conference.pdf.
- Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: program generation for situated robot task planning using large language models. *Autonomous Robots*, August 2023. ISSN 0929-5593, 1573-7527. doi:10.1007/s10514-023-10135-3. URL <https://link.springer.com/10.1007/s10514-023-10135-3>.
- Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents, January 2024. URL <http://arxiv.org/abs/2401.13178>. arXiv:2401.13178 [cs].
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore, 2023a. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-main.187. URL <https://aclanthology.org/2023.emnlp-main.187>.
- Linyong Nan, Ellen Zhang, Weijin Zou, Yilun Zhao, Wenfei Zhou, and Arman Cohan. On Evaluating the Integration of Reasoning and Action in LLM Agents with Database Question Answering, November 2023. URL <http://arxiv.org/abs/2311.09721>. arXiv:2311.09721 [cs].
- Vishal Pallagani, Bharath Muppasan, Keerthiram Murugesan, Francesca Rossi, Biplav Srivastava, Lior Horesh, Francesco Fabiano, and Andrea Loreggia. Understanding the Capabilities of Large Language Models for Automated Planning, May 2023. URL <http://arxiv.org/abs/2305.16151>. arXiv:2305.16151 [cs].
- Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Nan Duan. PPTC Benchmark: Evaluating Large Language Models for PowerPoint Task Completion, November 2023. URL <http://arxiv.org/abs/2311.01767>. arXiv:2311.01767 [cs].
- Yue Wu, Xuan Tang, Tom M. Mitchell, and Yuanzhi Li. SmartPlay: A Benchmark for LLMs as Intelligent Agents, December 2023a. URL <http://arxiv.org/abs/2310.01557>. arXiv:2310.01557 [cs].
- Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. TPTU: Large Language Model-based AI Agents for Task Planning and Tool Usage, November 2023. URL <http://arxiv.org/abs/2308.03427>. arXiv:2308.03427 [cs].
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners, January 2023. URL <http://arxiv.org/abs/2205.11916>. arXiv:2205.11916 [cs].
- Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models, May 2023a. URL <http://arxiv.org/abs/2305.04091>. arXiv:2305.04091 [cs].
- Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate Before Use: Improving Few-Shot Performance of Language Models, June 2021. URL <http://arxiv.org/abs/2102.09690>. arXiv:2102.09690 [cs].
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages

- 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.
- Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models, March 2023. URL <http://arxiv.org/abs/2212.04088>. arXiv:2212.04088 [cs].
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and Narrowing the Compositionality Gap in Language Models, October 2023. URL <http://arxiv.org/abs/2210.03350>. arXiv:2210.03350 [cs].
- Yew Ken Chia, Guizhen Chen, Luu Anh Tuan, Soujanya Poria, and Lidong Bing. Contrastive Chain-of-Thought Prompting, November 2023. URL <http://arxiv.org/abs/2311.09277>. arXiv:2311.09277 [cs].
- Yanfang Zhang, Yiliu Sun, Yibing Zhan, Dapeng Tao, Dacheng Tao, and Chen Gong. Large Language Models as an Indirect Reasoner: Contrapositive and Contradiction for Automated Reasoning, February 2024. URL <http://arxiv.org/abs/2402.03667>. arXiv:2402.03667 [cs].
- Chang Gao, Haiyun Jiang, Deng Cai, Shuming Shi, and Wai Lam. StrategyLLM: Large Language Models as Strategy Generators, Executors, Optimizers, and Evaluators for Problem Solving, February 2024a. URL <http://arxiv.org/abs/2311.08803>. arXiv:2311.08803 [cs].
- Shreyas Sundara Raman, Vanya Cohen, David Paulius, Ifrah Idrees, Eric Rosen, Ray Mooney, and Stefanie Tellex. CAPE: Corrective Actions from Precondition Errors using Large Language Models, October 2023. URL <http://arxiv.org/abs/2211.09935>. arXiv:2211.09935 [cs].
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models, July 2022a. URL <http://arxiv.org/abs/2207.05608>. arXiv:2207.05608 [cs].
- Lajanugen Logeswaran, Yao Fu, Moontae Lee, and Honglak Lee. Few-shot Subgoal Planning with Language Models, May 2022. URL <http://arxiv.org/abs/2205.14288>. arXiv:2205.14288 [cs].
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models, March 2023b. URL <http://arxiv.org/abs/2203.11171>. arXiv:2203.11171 [cs].
- Jieyi Long. Large Language Model Guided Tree-of-Thought, May 2023. URL <http://arxiv.org/abs/2305.08291>. arXiv:2305.08291 [cs].
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with Language Model is Planning with World Model, October 2023. URL <http://arxiv.org/abs/2305.14992>. arXiv:2305.14992 [cs].
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models, December 2023a. URL <http://arxiv.org/abs/2305.10601>. arXiv:2305.10601 [cs].
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models, December 2023a. URL <http://arxiv.org/abs/2310.04406>. arXiv:2310.04406 [cs].
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawska, Hubert Niewiadomski, Piotr Nyczek, and Torsten Hoefer. Graph of Thoughts: Solving Elaborate Problems with Large Language Models, November 2023. URL <http://arxiv.org/abs/2308.09687>. arXiv:2308.09687 [cs].
- Bilgehan Sel, Ahmad Al-Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models, September 2023. URL <http://arxiv.org/abs/2308.10379>. arXiv:2308.10379 [cs].
- Ziru Chen, Michael White, Raymond Mooney, Ali Payani, Yu Su, and Huan Sun. When is Tree Search Useful for LLM Planning? It Depends on the Discriminator, February 2024. URL <http://arxiv.org/abs/2402.10890>. arXiv:2402.10890 [cs].
- Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating Natural Language to Planning Goals with Large-Language Models, February 2023. URL <http://arxiv.org/abs/2302.05128>. arXiv:2302.05128 [cs].

- Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning, November 2023. URL <http://arxiv.org/abs/2305.14909>. arXiv:2305.14909 [cs].
- Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis, October 2023. URL <http://arxiv.org/abs/2307.12856>. arXiv:2307.12856 [cs].
- Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. SwiftSage: A Generative Agent with Fast and Slow Thinking for Complex Interactive Tasks, May 2023. URL <http://arxiv.org/abs/2305.17390>. arXiv:2305.17390 [cs].
- Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models, May 2023. URL <http://arxiv.org/abs/2305.18323>. arXiv:2305.18323 [cs].
- Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models, October 2023. URL <http://arxiv.org/abs/2304.09842>. arXiv:2304.09842 [cs].
- Silin Gao, Jane Dwivedi-Yu, Ping Yu, Xiaoqing Ellen Tan, Ramakanth Pasunuru, Olga Golovneva, Koustuv Sinha, Asli Celikyilmaz, Antoine Bosselut, and Tianlu Wang. Efficient Tool Use with Chain-of-Abstraction Reasoning, February 2024b. URL <http://arxiv.org/abs/2401.17464>. arXiv:2401.17464 [cs].
- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, August 2022. URL <http://arxiv.org/abs/2204.01691>. arXiv:2204.01691 [cs].
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models, March 2023b. URL <http://arxiv.org/abs/2210.03629>. arXiv:2210.03629 [cs].
- BabyAGI. URL <https://github.com/yoheinakajima/babyagi>.
- Yohei Nakajima. BabyBeeAGI: Task Management and Functionality Expansion on top of BabyAGI, April 2023. URL <https://yoheinakajima.com/babybeeagi-task-management-and-functionality-expansion-on-top-of-babyagi/>.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, October 2023c. URL <http://arxiv.org/abs/2305.16291>. arXiv:2305.16291 [cs].
- Yu Du, Fangyun Wei, and Hongyang Zhang. AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls, February 2024. URL <http://arxiv.org/abs/2402.04253>. arXiv:2402.04253 [cs].
- Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Xin Zhao, and Ji-Rong Wen. ChatCoT: Tool-Augmented Chain-of-Thought Reasoning on Chat-based Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14777–14790, Singapore, 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.findings-emnlp.985. URL <https://aclanthology.org/2023.findings-emnlp.985>.
- Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-Needed Decomposition and Planning with Language Models, November 2023. URL <http://arxiv.org/abs/2311.05772>. arXiv:2311.05772 [cs].
- Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents, October 2023d. URL <http://arxiv.org/abs/2302.01560>. arXiv:2302.01560 [cs].
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A Survey on Large Language Model based Autonomous Agents, September 2023e. URL <http://arxiv.org/abs/2308.11432>. arXiv:2308.11432 [cs].
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents, March 2022b. URL <http://arxiv.org/abs/2201.07207>. arXiv:2201.07207 [cs].

- Siqi Ouyang and Lei Li. AutoPlan: Automatic Planning of Interactive Decision-Making Tasks With Large Language Models, October 2023. URL <http://arxiv.org/abs/2305.15064>. arXiv:2305.15064 [cs].
- Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction, February 2024. URL <http://arxiv.org/abs/2401.06201>. arXiv:2401.06201 [cs].
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs, October 2023. URL <http://arxiv.org/abs/2307.16789>. arXiv:2307.16789 [cs].
- Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. ToolChain*: Efficient Action Space Navigation in Large Language Models with A* Search, October 2023. URL <http://arxiv.org/abs/2310.13227>. arXiv:2310.13227 [cs].
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools, February 2023. URL <http://arxiv.org/abs/2302.04761>. arXiv:2302.04761 [cs].
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation, October 2023b. URL <http://arxiv.org/abs/2308.08155>. arXiv:2308.08155 [cs].
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable Code Actions Elicit Better LLM Agents, February 2024b. URL <http://arxiv.org/abs/2402.01030>. arXiv:2402.01030 [cs].
- Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, Minghua Ma, Pu Zhao, Si Qin, Xiaoting Qin, Chao Du, Yong Xu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. TaskWeaver: A Code-First Agent Framework, December 2023. URL <http://arxiv.org/abs/2311.17541>. arXiv:2311.17541 [cs].
- Haiteng Zhao, Chang Ma, Guoyin Wang, Jing Su, Lingpeng Kong, Jingjing Xu, Zhi-Hong Deng, and Hongxia Yang. Empowering Large Language Model Agents through Action Learning, February 2024. URL <http://arxiv.org/abs/2402.15809>. arXiv:2402.15809 [cs].
- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing, February 2024. URL <http://arxiv.org/abs/2305.11738>. arXiv:2305.11738 [cs].
- Ning Miao, Yee Whye Teh, and Tom Rainforth. SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning, October 2023. URL <http://arxiv.org/abs/2308.00436>. arXiv:2308.00436 [cs].
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback, May 2023. URL <http://arxiv.org/abs/2303.17651>. arXiv:2303.17651 [cs].
- Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society, November 2023b. URL <http://arxiv.org/abs/2303.17760>. arXiv:2303.17760 [cs].
- Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language Models can Solve Computer Tasks, November 2023. URL <http://arxiv.org/abs/2303.17491>. arXiv:2303.17491 [cs].
- Byoungjin Kim, Youngsoo Jang, Lajanugen Logeswaran, Geon-Hyeong Kim, Yu Jin Kim, Honglak Lee, and Moontae Lee. Prospector: Improving LLM Agents with Self-Asking and Trajectory Ranking, 2024. URL <https://openreview.net/forum?id=YKK1jXEWja>.
- Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. RAT: Retrieval Augmented Thoughts Elicit Context-Aware Reasoning in Long-Horizon Generation, March 2024c. URL <http://arxiv.org/abs/2403.05313>. arXiv:2403.05313 [cs].
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning, October 2023. URL <http://arxiv.org/abs/2303.11366>. arXiv:2303.11366 [cs].
- Lingfeng Sun, Devesh K. Jha, Chiori Hori, Siddarth Jain, Radu Corcodel, Xinghao Zhu, Masayoshi Tomizuka, and Diego Romeres. Interactive Planning Using Large Language Models for Partially Observable Robotics Tasks, December 2023b. URL <http://arxiv.org/abs/2312.06876>. arXiv:2312.06876 [cs].

- Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Xiangru Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. Agents: An Open-source Framework for Autonomous Language Agents, December 2023b. URL <http://arxiv.org/abs/2309.07870>. arXiv:2309.07870 [cs].
- Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Xiaojiang Huang, Yanbin Lu, and Yingzhen Yang. RecMind: Large Language Model Powered Agent For Recommendation, February 2024d. URL <http://arxiv.org/abs/2308.14296>. arXiv:2308.14296 [cs].
- Swarna Kamal Paul. Sequential Planning in Large Partially Observable Environments guided by LLMs, December 2023. URL <http://arxiv.org/abs/2312.07368>. arXiv:2312.07368 [cs].
- Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. RAP: Retrieval-Augmented Planning with Contextual Memory for Multimodal LLM Agents, February 2024. URL <http://arxiv.org/abs/2402.03610>. arXiv:2402.03610 [cs].
- Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior, August 2023. URL <http://arxiv.org/abs/2304.03442>. arXiv:2304.03442 [cs].
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework, November 2023. URL <http://arxiv.org/abs/2308.00352>. arXiv:2308.00352 [cs].
- Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More Agents Is All You Need, February 2024. URL <http://arxiv.org/abs/2402.05120>. arXiv:2402.05120 [cs].
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for General AI Assistants, November 2023b. URL <http://arxiv.org/abs/2311.12983>. arXiv:2311.12983 [cs].
- Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM*, 58(9):92–103, August 2015. ISSN 0001-0782, 1557-7317. doi:10.1145/2701413. URL <https://dl.acm.org/doi/10.1145/2701413>.
- Christopher D. Manning. Human Language Understanding & Reasoning. *Daedalus*, 151(2):127–138, May 2022. ISSN 0011-5266, 1548-6192. doi:10.1162/daed_a_01905. URL <https://direct.mit.edu/daed/article/151/2/127/110621/Human-Language-Understanding-and-Reasoning>.