An Evaluation of LLMs on Multi-Agent Planning Problems

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have recently been used for a variety of planning-related tasks, including plan generation, translating planning problems to different representations, and assisting classical planners. However, most existing works have primarily focused on single-agent planning tasks. In contrast, evaluating the capabilities of LLMs for multi-agent planning tasks remains largely unexplored. In this work, we explore the ability of LLMs to solve cooperative multi-agent tasks specified in unfactored MA-PDDL format. We investigate three zero-shot approaches. The first, being the *LLM-Repair* approach, where we prompt the LLM to output a plan that solves the common goal and subsequently rectify the invalid actions in the generated plan with the help of a pretrained transformer together with PYPERPLAN, replacing inconsistent actions with the most similar valid ones. In the second approach, *LLM-Modulo (Kambhampati et al., 2024)*, we integrate the LLM with a plan verifier that checks the plan's correctness and provides feedback to the LLM to produce a valid plan. In the third approach, *LLM-Merge*, we decompose loosely coupled multi-agent planning tasks into several single-agent tasks, solve them using PYPERPLAN, and then prompt the LLM to merge the generated single-agent plans into a multi-agent plan. We evaluate several recent LLMs on the problems from the International Competition of Distributed and Multi-Agent Planners (CoDMAP) with these approaches. Our results provide empirical insights into the strengths and limitations of recent LLMs in multi-agent planning problems.

1 Introduction

Automatically devising a rational plan of action to achieve a given task is a critical part of artificial intelligence. Planning is a complex activity core to human intelligence requiring logical reasoning, rational thinking, knowledge of the underlying world and adaptation to changes. The successful deployment of AI powered automated planners in real-world systems demonstrates the maturity and reliability of planning technology. Notable examples include the Mars 2020 Simple Planner onboard on NASA's Perseverance rover (Gaines et al., 2022), the Spike mission planning system employed by the Hubble Space Telescope (Johnston, 1990), as well as path-planning systems widely used in autonomous vehicles and robotic platforms. These applications provide evidence of trusted autonomy across critical domains such as space exploration, automotive systems, and robotics.

Large Language Models (LLMs) are in the forefront of discussion in the AI research community, due to their seemingly human-like ability to generate texts, translate languages, converse meaningfully and even solve math problems. LLMs are deep learning models with billions of parameters, pre-trained on a large corpus of internet data (some researchers call it the collective consciousness) and often fine-tuned to specialize on solving certain tasks. Due to the versatility of LLMs, there has been studies to gauge the reasoning and planning abilities of LLMs, particularly planning tasks of the type studied in the automated planning community (Nau et al., 2004) where a formal description of the environment, the planning agents, the initial state and the desired goal state are provided as part of the problem description and the expectation is a finite sequence of actions that leads from the initial state to the goal state. Planning tasks can be broadly categorized as single agent or multi-agent. In multi-agent planning (MAP) tasks, multiple autonomous agents operate within a shared environment and cooperate to achieve a common goal. Each agent is endowed with its own set of actions and capabilities, and agents' may interact by jointly affecting the environment or constraining one another. The challenge in MAP is to compute coordinated plans that ensure goal achievement while accounting for inter-agent dependencies and resource constraints. As the number of agents increases, the complexity of the planning problem grows combinatorially, making scalability and effective coordination key concerns in this setting. A number of recent studies on evaluating the ability of LLMs to solve single-agent planning tasks have been reported (Huang et al., 2022; Song et al., 2023; Valmeekam et al., 2023; Kambhampati et al., 2024; Wei et al., 2025). There are reports of both optimism and pessimism over LLMs ability to solve single-agent planning tasks (Huang et al., 2022; Valmeekam et al., 2022; Kambhampati et al., 2024; Wei et al., 2025). Recently, the authors in (Kambhampati

et al., 2024) strongly advocate that LLMs cannot reason and therefore the plans they produce cannot be trusted/may not be sound. The authors argue at the same time that LLMs can help in solving planning tasks when used in what is called the *LLM-modulo framework*. The ability of LLM-modulo framework in finding sound plans have been demonstrated against single-agent planning tasks (Kambhampati et al., 2024). However, to the best of our knowledge, there it no work that studies the strengths and limitations of LLMs on solving multi-agent planning tasks, which is our objective of this paper. We believe this is important because multi-agent tasks are generally more complex than single agent tasks and requires higher degree of reasoning.

In this work, we present an evaluation of some of the recent LLMs on multi-agent planning problems from a set of benchmarks released in the international competition of distributed and multi-agent planners (CoDMAP) (Komenda et al., 2016). The evaluation broadly consists of zero-shot prompting in an LLM-Repair framework, zero-shot prompting in an LLM-Modulo framework as proposed in (Kambhampati et al., 2024) and zero-shot prompting in an LLM supported plan merging framework, which we elaborate in the paper. We present an open-source code-base which can be extended to evaluate newer LLMs and/or additional multi-agent planning benchmarks. [1]

2 Related Work

2.0.1 PDDL Planning

Planning domain definition language (PDDL) planning was introduced in (Aeronautiques et al., 1998) where a problem would be described using domain and problem files in PDDL format, which is a Lisp-like specification language. These problems would be solved by planners such as FastDownward (FD) (Helmert, 2006), FastForward (FF) (Hoffmann, 2001), LAMA (Richter and Westphal, 2010), and Scorpion (Seipp, 2018), which use combinatorial forward search with heuristics to find a sequence of actions that solves the problem. PDDL planning is mainly concerned with single-agent tasks.

2.0.2 Multi-Agent Planning

For multi-agent planning, several new languages have been proposed, such as the multi-agent planning language (MAPL) (Brenner, 2003) and multi-agent PDDL (MA-PDDL) (Kovacs et al., 2012), which offer additional language constructs for specifying privacy and concurrency. The CoDMAP competition evaluates a range of solvers for multi-agent planning problems described in the MA-PDDL format. The solvers can be broadly categorized as centralized and distributed. In the centralized approach, the multi-agent task is solved by a single planning entity with a global view of the environment, thereby generating joint plans for all agents, rather than having agents plan independently with local knowledge. The advantage of this process is that the external communication between agents is not needed, and existing single-agent planning techniques can be easily reused with suitable adaptations. Some examples of planners using the centralized approach include the Agent Decomposition-based Planner (ADP) and the Multi-agent Planner for Required Cooperation (MARC) (Sreedharan et al., 2015). The distributed approach, on the other hand, involves a multi-agent planning task with multiple planning agents, each possessing a local view of the task, while collectively working towards a common goal. In this approach, the agents make decisions autonomously and coordinate their activities through explicit communications. Given a task with n agents, the distributed planners can be executed across multiple machines, one for each agent, which makes communication among agents essential. However, the defining characteristics of this approach are the distribution of computation across agents rather than the hardware architecture itself. The advantage of this approach is that it can effectively preserve each agent's privacy. Some examples of distributed solvers are Forward Multi-Agent Planning (FMAP) (Torreno et al., 2014) and Planning State Machines (PSM) (Torreno et al., 2015).

2.0.3 Plan Validation and Repair

Plan validation servers as a very important step in planning. Given a plan, tools like VAL (Howey et al., 2004) helps in plan validation by checking the type mismatches between objects and action parameters, performing step-wise action verification where at each step it is checked if the actions preconditions are being met and finally whether the goal is achieved by the plan. It can also produce repair advice for the domain and problem files if errors are found in the problem formalization itself (e.g., missing constraints, unsolvable goals).

2.0.4 LLMs for Planning

Recently, there has been a significant effort in using LLMs for generating plans for single-agent tasks. Some widely used paradigms for LLM-assisted plan generation include using LLMs for task decomposition, taking LLM assistance while running planning algorithms, and fine-tuning LLMs for planning-related tasks, such as natural language-to-PDDL translation. Task Decomposition involves breaking abstract goals into manageable subgoals, making it amenable to reasoning. It can be done sequentially (Singh et al., 2023; Sun et al., 2023), in parallel (Yang et al., 2025) or asynchronously (Lin et al., 2024). For each of these approaches, the task decomposition can be performed recursively (Prasad et al., 2024). LLMs are also being used with classical planners to convert natural-language problems into formal specifications, such as in PDDL, for solving with classical planners. LLMs have been shown to be able to support search algorithms for expansions, suggesting actions based on the environments, as substitute world models (Hao et al., 2023)-simulating an environment

[1] The source code will be made publicly available upon acceptance.

by keeping track of the states (configurations) and actions (transitions)-in order to augment reasoning, and LLMs have also been shown to help state progress evaluation towards the goal via LLM/LVM ratings (Hao et al., 2023; Yao et al., 2023). Another paradigm for LLM-based planning is fine-tuning the LLMs for planning-related tasks to improve planning performance (Chalvatzaki et al., 2023). LLMs are also fine-tuned for general agentic tasks that involve planning components, thereby improving both planning performance and related abilities. The performance of LLM in planning can be evaluated based on completeness (Guan et al., 2023; Aghzal et al., 2023; Valmeekam et al., 2024; Katz et al., 2024), executability (Song et al., 2023; Lin et al., 2024; Brohan et al., 2023; Singh et al., 2022; Prasad et al., 2024), optimality (Ju et al., 2024; Lehnert et al., 2024), generalization (Yang et al., 2022; Silver et al., 2024; Wang et al., 2023), and efficiency (Sun et al., 2023; Gonzalez-Pumariega et al., 2024; Hu et al., 2025).

2.1 Plan Merging

Plan merging has been studied in various settings to obtain a global plan from multiple individual plans. In a multi-agent setting, plan merging gives rise to a multi-agent plan coordination problem (MPCP). Early works focused on identifying and resolving conflicts arising from agent interactions in a shared environment. Several methods can resolve these conflicts, such as introducing synchronization constraints (Georgeff, 1988) and exploiting positive interactions, in which actions performed by one agent support the goals of another (Martial, 1990). Algorithms for merging plans by removing or rearranging actions in the merged plan have been proposed (Mali, 1999). (Cox and Durfee, 2009a) proposed efficient distributed methods for solving the MPCP problem. More recent approaches, such as PMR (Luis et al., 2020), combine distributed planning with centralized repair mechanisms that correct invalid merged actions across agents. Other related works (Britanik and Marefat, 1995) have explored plan merging in Hierarchical Task Network (HTN) planning, where merging between partial-order plans occurs at multiple abstraction levels, as well as methods for merging partial-order plans containing durative actions for robotic tasks (Mudrova et al., 2016). Additionally, several works (Brafman and Domshlak, 2006) have explored decomposition mechanisms of planning tasks into various subtasks to maximize the efficiency of merging.

3 Background

In this section, we provide the necessary background for MAP problems and introduce a formalization that can be generalized to the various types discussed later. The formalization can be compiled into languages such as MA-PDDL. Our formalization of the MAP task will use the MA-STRIPS (Brafman and Domshlak, 2008) model for simplicity. As a minimalist multi-agent version of the well-known STRIPS (Fikes and Nilsson, 1971) model, MA-STRIPS has become the standard formalism for defining MAP problems. We define a MAP problem:

Definition 3.1. *(Brafman and Domshlak, 2008; Kovacs et al., 2012) A* MAP problem Π *is a* 4*-tuple* (Dom, Prob, $\mathcal{I}$, $\mathcal{G}$) *where*

- Dom *is a three-tuple* $\langle \mathcal{AG}\ \mathcal{F}, \{A^i\}_{i=1}^n \rangle$ *defining the planning domain with the components:*
  - *i.* $\mathcal{AG}$ *is a finite set of* n *planning agents.*
  - *ii.* $\mathcal{F}$ *is a finite set of atoms or propositions.*
  - *iii.* $\mathcal{A}^i$ *is the finite set of planning actions of the agent* $i \in \mathcal{AG}$*. The set of actions in* Π *is defined as* $\mathcal{A} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{A}^i$.
- Prob *defines a planning problem instance, which is a collection of problems* $\mathcal{P}^i$*, one each for an agent* $i \in \mathcal{AG}$*.* $\mathcal{P}^i$ *is two-tuple* $\langle \mathcal{I}^i, \mathcal{G}^i \rangle$ *where*
  - *i.* $\mathcal{I}^i \subseteq \mathcal{F}$ *defines the* initial state *of agent* i*. The initial state* $\mathcal{I}$ *of* Π *is given by* $\mathcal{I} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{I}^i$.
  - *ii.* $\mathcal{G}^i \subseteq \mathcal{F}$ *defines the* goal *for the agent* i*. The goal* $\mathcal{G}$ *of* Π *is the conjunction of the agent goals given by* $\mathcal{G} = \bigcap_{\forall i \in \mathcal{AG}} \mathcal{G}^i$.

In MA-STRIPS, a task is represented by a sequence of states. Each state is described by a set of propositions (or atoms), and the states evolve as planning actions are executed. We define an action as follows:

Definition 3.2. *An* action *(Torreño et al., 2018) in MAP is a tuple* $\alpha = \langle pre(\alpha), add(\alpha), del(\alpha) \rangle$*, where* $pre(\alpha)$*,* $add(\alpha)$ *and* $del(\alpha)$ *are sets of propositions that denote the* preconditions*,* add effects*, and* delete effects *of the action, respectively.*

An action α can be performed in a state S only if all of its preconditions are met within that state, that is, $\forall f \in pre(\alpha)$, $f \in S$. When the action is executed, it changes the state of the world to a new state, S', such that $S' = S \setminus del(\alpha) \cup add(\alpha)$.

Now, we define a plan for a MAP problem. The responsibility for a *planner* in a *centralized* approach for a MAP task is to schedule a sequence of actions for each agent i that meets $\mathcal{G}^i$, and converge to meet the goal $\mathcal{G}$ of Π. For a *distributed* approach, each agent i plans their sequence of actions independently to meet end objectives while avoiding conflict (Cox and Durfee, 2009b). We first define a plan for an agent, and later use those plans to define a plan for a MAP problem Π.

Definition 3.3. *A plan* ϕ^i *for an agent* $i \in \mathcal{AG}$ *is a sequence of actions,* $\phi^i = a_1^i a_2^i \ldots a_n^i$*, where* $a_j^i \in A^i$*.* ϕ^i *satisfies* $\mathcal{G}^i$*, i.e., if the sequence of actions in* ϕ^i *are executed, it moves the agent* i *from its initial state* $\mathcal{I}^i$ *to the goal* $\mathcal{G}^i$*.*

Definition 3.4. *A plan* Φ *for a MAP problem* Π *is a collection of action sequences* ϕ^i*, that is,* $\Phi = [\phi^1, \phi^2, \ldots, \phi^n]$ *for each agent* $i \in \mathcal{AG}$*, where actions* a_k^i *and* a_l^j *for agent* i *and* j *may be executed concurrently.*

For example, in the given plan Φ (see Equation 1) for a MAP problem, the actions a_1^2 and a_1^n should be executed

concurrently. Similarly, the case is for the actions a_2^1 and a_2^n.

$$\Phi = \left[\begin{array}{cccc} \multicolumn{4}{c}{Agents\ Plan} \\ \phi^1 & \phi^2 & \cdots & \phi^n \\ \hline a_1^1 & - & \cdots & - \\ - & a_1^2 & \cdots & a_1^n \\ a_2^1 & - & \cdots & a_2^n \\ \vdots & \vdots & \cdots & \vdots \end{array}\right] \quad (1)$$

In the following section, we describe our approaches for solving the multi-agent planning problems.

4 Solving MAP Problems with LLMs

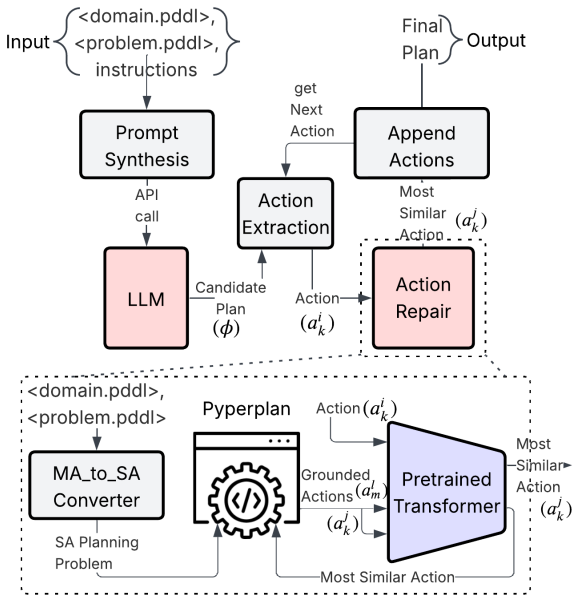

Figure 1: LLM Repair process, where the invalid actions in the LLM-generated candidate plan are repaired and replaced in the Action Repair block. The Action Repair block contains a pretrained transformer and Pyperplan, which work together to replace an invalid action with the most similar valid action in the current state. The most similar valid action is found using the pretrained transformer.

We examine the ability of the state-of-the-art LLMs across some multi-agent planning tasks. In particular, we explore the ability of LLMs to generate valid plans for the benchmarks proposed in the CoDMAP competition. We consider three strategies: LLM-Repair, inspired by the open-loop approach proposed in (Silver et al., 2022) in which plan-generation follows a predefined set of instructions without incorporating adjustments based on feedback from the output, LLM-Modulo, adapted from (Kambhampati et al., 2024) an iterative approach to plan generation where a plan is repeatedly refined based on the feedback from critics such as binary critics, and LLM-assisted plan-merging, where the MAP problem is decomposed into single agent problems, each

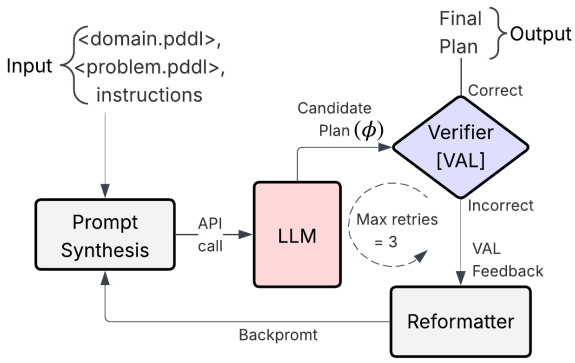

Figure 2: The LLM-Modulo process in which VAL verifies an LLM-generated plan and returns feedback when the plan is incorrect, allowing the LLM to refine it.

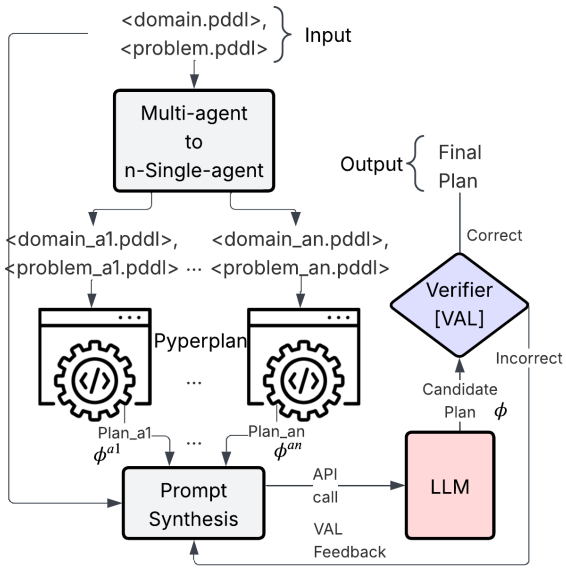

Figure 3: Plan merging process in which a multi-agent problem is decomposed into individual single-agent problems, which are solved by Pyperplan one at a time, and then the generated plans are merged by the LLM. In case of an invalid merged plan, it is refined using the LLM modulo process.

solved using a classical planner and the resulting plans are merged by the LLM into a multi-agent plan. In each of these strategies, we verify the correctness of the final plan using VAL. A schematic diagram of our evaluation pipeline is shown in Figure 4.

4.1 LLM Repair Framework

Figure 1 illustrates the LLM repair process. In this strategy, we prompt an LLM with the unfactored MA-PDDL description of the benchmark problem, which consists of a single domain and a problem file in PDDL, and some instructions. We eliminate some redundancy in the domain and problem files to reduce input token usage and provide instructions on the plan format. All interactions are done through API calls to the LLM. The

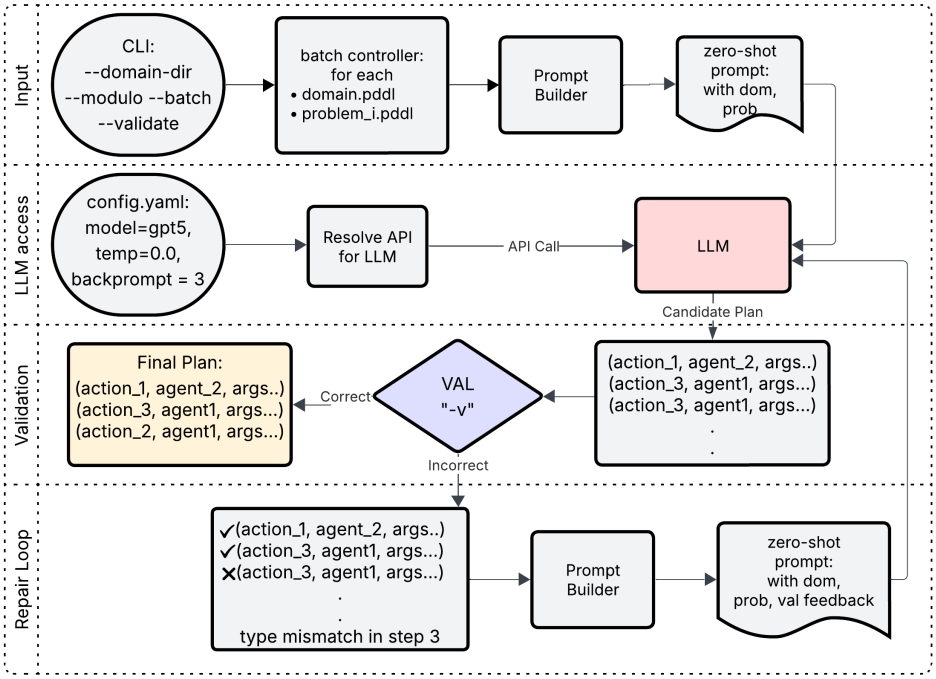

Figure 4: Pipeline of the LLM-based planning framework with the modulo approach. The user provides the domain directory containing the PDDL domain and problem files, along with the configuration specifying the LLM and runtime parameters. The batch iterator then generates a zero-shot prompt for each problem instance and feeds it to the LLM. The LLM generates a candidate plan, which is validated by VAL in the verbose mode. VAL declares the errors and their cause; this information, along with the plan, is incorporated into the prompt, and the LLM is queried again. The yellow box indicates the final valid plan.

LLM then returns a plan as a sequence of actions.

However, sometimes these generated actions are not in the correct format; for instance, the arity of a given action may be incorrect, or the ordering of the action's arguments may be incorrect. However, to check the validity of the plan using the VAL tool, the plan needs to be in the correct format. To address this issue, we employ a two-component strategy consisting of the Pyperplan (Alkhazraji et al., 2020) engine and a pre-trained transformer, showing in the action repair block of Figure 1. Pyperplan is a single-agent planner that accepts planning problems in PDDL and produces a valid plan when one exists. More importantly, Pyperplan provides a Python interface that allows access to grounded actions, operators, and other internal planning artefacts. Since Pyperplan is a PDDL planner that is mainly used for single-agent planning problems, we first convert the MA-PDDL domain and problem files into a PDDL planning problem using a converter script from the CoDMAP competition. Whenever the LLM returns a plan, each of its actions is used to update the current state maintained by pyperplan, which then returns the set of all grounded actions possible in that state.

These grounded actions are used to repair any invalid actions. Specifically, at each step, the grounded actions are embedded into vector representations using a pre-trained transformer encoder. The corresponding action produced by the LLM is also embedded as a vector, and cosine similarity is computed between the LLM action embedding and the embeddings of all grounded actions. If there is a direct match, the action is retained; otherwise, it is replaced with the most similar action among the grounded actions. This way, VAL can execute the entire plan.

4.2 LLM Modulo Framework

Figure 2 describes our implementation of LLM modulo approach. Similar to LLM repair approach, we prompt the LLM via API calls with the unfactored domain and problem files along with the instructions on the expected plan format. We also compress the input prompt just as in LLM repair approach. The LLM then attempts to generate a candidate plan in a format compatible with VAL.

The generated plan is then verified by VAL, and if it is valid, the process terminates successfully. Otherwise, if it is invalid — either due to syntactic errors or violations of preconditions and constraints — we extract the verbose diagnostic output from VAL, which shows type checking and updates through add/delete effects, among other things. Then we condense this output and keep only the specific causes of failure, such as unsatisfied precondition or incorrect use of actions.

We provide this condensed feedback to the LLM together with the original domain and problem description and the candidate plan which triggered the error. This

becomes the new prompt and we ask the LLM to rectify the plan with this information. This generate-verify-rectify process continues iteratively until we find a valid plan or a retry limit is reached.

4.3 LLM Merge Framework

Figure 3 describes the LLM-based plan merging process. In this process, given a multi-agent planning problem, particularly in loosely coupled systems, we first decompose it deterministically into several per-agent single-agent problems. Each problem consists of its own domain and problem files. These problems retain the same goal as the original multi-agent problem but include only the actions available to a particular agent. Then, for each generated domain-problem pair, we run the PYPERPLAN planner to obtain individual plans, if one exists. These individual-agent plans can achieve the goals independently, but generally require significantly more actions/steps than coordinated multi-agent plans. That is why, next, we prompt the LLM with the original multi-agent domain and problem, along with the individual agent plans, and prompt it to generate a merged candidate plan in a normalized action format that resolves potential conflicts between agents. Then, from this plan, we remove malformed or non-action text and validate using the VAL plan validator. If it is valid, we accept the merged candidate plan. Otherwise, we follow an LLM-Modulo-style scheme, where we extract the relevant feedback from VAL, send a repair prompt to the LLM, and re-validate up to N attempts. Through this process, we get a multi-agent plan. This approach is similar to distributed multi-agent planning using classical planners, while LLM-Repair and LLM-Modulo are more aligned to centralized multi-agent planning using classical planners.

Tightly-coupled systems Plan merging in tightly coupled systems is less suitable, as the resolution scheme during merging generally assumes that each agent can solve a subset of the goal/task. In tightly coupled systems, however, such a decomposition is not possible because the agents have to closely coordinate to achieve the goals. Enabling such a decomposition would require a robust, possibly time-dependent, goal/task allocation scheme. But in tightly coupled systems, this process is difficult and often leads to several highly dependent subplans with very few actions, further complicating the merging process later. Therefore, for these domains it is generally better to directly attempt multi-agent planning rather than merging.

5 Experiments and Results

5.1 Benchmarks

We evaluate the strategies on the benchmark domains from the literature on multi-agent planning, some of which are extensions of the classical IPC domains to a multi-agent context. Specifically, we consider the following domains: Blocksworld, Depot, Driverlog, Elevators08, Logistics00, Rovers, Satellites, Sokoban, Woodworking, and Zenotravel. These domains can be categorised as either Loosely Coupled or Tightly Coupled Domains (Torreno et al., 2014). The latter category being more complex than the former, in terms of finding a feasible plan. In the following, we describe these categories and the domains that fall under each category. These benchmark domains each contain 20 problem instances, resulting in a total of 200 instances.

Loosely Coupled Domains These domains consist of homogeneous agents, each with the same planning abilities (i.e., action operators) and the potential to solve tasks individually. As a result, cooperation between agents is not strictly necessary. The five loosely coupled domains we have considered in our evaluation are: Blocksworld, Driverlog, Rovers, Satellite, and Zenotravel.

Tightly Coupled Domains These domains consist of specialized agents with different planning abilities (i.e., action operators), and solving the planning task requires coordination between the agents. A single agent cannot accomplish the goal on its own. The tightly coupled domains considered in the evaluation are, namely, Depot, Logistics, Taxi, Elevators and Woodworking.

5.2 Experimental Setup

Baseline As a baseline, the LLMs are queried to generate a plan with a zero-shot prompt. The generated plans are evaluated by VAL. The results are reported in Table 1, where each entry represents the average of three runs to account for the variability in outputs that occur due to the non-deterministic nature of LLM-based generation. All queries to the respective LLMs are issued via API calls, with a temperature of 0.0. The maximum output length is set to 8000 tokens, and the plan length bound is set to 100 while prompting LLMs. The input query is then compressed by stripping whitespace and comments from the PDDL files to reduce the number of tokens.

LLM-Repair The results of the LLM-Repair approach, averaged over three evaluations, are reported in Tables 2 and 3. Similar to the baseline, the LLM calls are made via the corresponding APIs with temperature set to 0.00, maximum output tokens of 8000, and compressed input queries. Our evaluation framework is implemented in Python, with Pyperplan as the classical planner to extract the grounded operators. It was also used in (Silver et al., 2022) for plan generation along with LLMs. We also use the MA-PDDL to single-agent PDDL conversion scripts from the CoDMAP competition. For action comparison and replacement, we use the pre-trained sentence transformer *Paraphrase-MiniLM-L6-v2*, an encoder-based model trained on a collection of sentence-pair datasets for sentence-level semantic similarity.

LLM-Modulo For the LLM-Modulo approach, the results are averaged over two LLM evaluations in Table 2

and 3. All LLM calls, including both the generation and correction prompts, are made through an API call with a temperature of 0.0. For each problem instance, up to three back-prompts (feedback iterations) are permitted, with feedback from VAL used to guide the LLM in generating valid plans. The remainder of the experimental setup, including the evaluation framework and supporting tools, is identical to the LLM-Repair approach.

LLM-Merge The results of this approach are averaged over three LLM evaluations. For each problem instance, the maximum output tokens during merging is set to 8000, and the temperature is set to 0.0. Plans for each agent are generated using Pyperplan with greedy best-first search and the hFF heuristic. These plans are then merged using an LLM with an underlying modulo approach, and the resulting merged plans are refined with up to three backprompt cycles. VAL runs with the verbose option after each merging attempt, and the tail end of its output is taken as feedback for the next iteration. The rest of the experimental setup, including the supporting tools and conversion scripts, is identical to the LLM-Repair approach. Unlike the repair approach, there is no explicit plan length bound imposed during the merging step.
All experiments were conducted on a desktop computer with an Intel® Core™ i9-14900KS CPU and 32 GB of RAM, running Linux 22.04 LTS.

5.3 Results

In table 1 we describe the results of the baseline approach. Tables 2 and 3 show the performance of the LLMs as the percentage of planning problems of each type correctly solved under the LLM-Repair and LLM-Modulo. In table 4 we describe the results of the LLM-Merge approach.

Baseline In this approach, smaller models such as GPT-OSS-20B show poor performance, often generating invalid plans. One reason for this is because some plans are declared invalid because VAL cannot execute them due to syntactic errors or malformed actions. Larger models show modest improvements, especially in the loosely coupled domains, which require less coordination and reasoning about dependent actions. In contrast, tightly coupled domains require more coordination, leading to significant performance drops even with larger models. Overall, the results indicate the scope of better coverage with more sophisticated techniques

LLM-Repair Similar to the baseline approach, we observe significant variance across models and domains. Smaller models, such as GPT-OSS-20B, are significantly worse than larger ones at generating valid plans. Interestingly, we observe that the improvement over the baseline remains limited with the repair approach, suggesting that a single-pass correction of LLM-generated plans is insufficient to produce valid plans. Domain difficulty also plays an important role but appears less sensitive to domain difficulty compared to the modulo approach, as we see in average columns of tables 2 and 3, where the average values for loosely and tightly coupled systems remain relatively close. Among loosely coupled domains, the Rovers domain shows lower success rates than the others, likely because it has a large action space and more object types. Similarly, among tightly coupled systems, elevators and the woodworking domain show lower correctness because these domains impose additional optimization constraints where a utility function must be minimized, and each action incurs a fixed cost. Overall, DeepSeek-V3.2 shows the best performance with the repair approach, both in loosely and tightly coupled systems.

LLM-Modulo In this approach, we see a consistent increase in performance across all the problems and models consistently. Even with a limit of three backprompt cycles, the feedback mechanism is able to correct the invalid plans to valid solutions. Notably, we observe significant improvements in the GPT-OSS-20B model, especially in loosely coupled domains, suggesting that external feedback can compensate for limited reasoning capabilities. Also, we see that the performance gap between models becomes less pronounced under the Modulo approach. Similar to the repair approach, we see performance is slightly worse in the tightly coupled domain, but the difference in performance between loosely and tightly coupled domains is more pronounced than in the repair approach. Overall, Gemini-3-Pro shows the best performance with the modulo approach, both in loosely and tightly coupled domains.

LLM-Merge In this approach, we observe further improvements in coverage than the vanilla modulo approach in several domains, suggesting merging can sometimes outperform direct generation. We see that the performance is exceptionally good for the blocksworld, driverlog, and the zenotravel domains, while the rovers domain shows a lower success rate. This could be because, in some rover instances, individual agents cannot fully achieve the goal independently; only a few agents possess the abilities required to achieve it. In addition, the rover domain contains a larger number of objects and object types that must be catered to, which increases the difficulty of the merging step. Interestingly, we see a significant increase in average percentage coverage of the domains for GPT-OSS-20B, which increases from 0.58 in the modulo approach to 0.74 in the merge approach. This suggests that structured intermediate plans effectively reduce the reasoning burden on the LLM. We also observe that mid-sized LLMs perform very well, with GPT-5 outperforming Gemini-3-pro across several domains. This could indicate that plan-merging relies more on structured reasoning than sheer model size. Overall, these results indicate that LLM-based plan merging can be an effective strategy for loosely coupled multi-agent planning problems.

Model	Loosely Coupled						Tightly Coupled					
	BW	DL	RV	SAT	ZT	Avg	DP	LOG	TX	ELV	WW	Avg
GPT-OSS-20B	0.00	0.21	0.00	0.15	0.23	0.11	0.01	0.11	0.21	0.03	0.05	0.08
GPT-OSS-120B	0.60	0.63	0.06	0.71	0.68	0.54	0.18	0.41	0.60	0.31	0.11	0.32
GPT-5	0.71	0.65	0.21	0.73	0.70	0.60	0.13	0.45	0.51	0.15	0.15	0.28
DeepSeek-V3.2	0.50	0.65	0.00	0.28	0.50	0.39	0.10	0.35	0.33	0.05	0.13	0.19

Table 1: Baseline coverage across planning domains grouped by system coupling type. The average column records the mean coverage across all domains. BW: Blocksworld, DL: Driverlog, RV: Rovers, SAT: Satellite, ZT: Zenotravel, DP: Depot, LOG: Logistics, TX: Taxi, ELV: Elevators, WW: Woodworking.

Model	Blocksworld		Driverlog		Rovers		Satellite		Zenotravel		Average	
	Repair	Modulo	Repair	Modulo	Repair	Modulo	Repair	Modulo	Repair	Modulo	Repair	Modulo
GPT-OSS-20B	0.00	0.45	0.25	0.67	0.01	0.05	0.28	0.92	0.13	0.80	0.13	0.58
GPT-OSS-120B	0.71	1.00	0.35	0.85	0.10	0.27	0.35	0.95	0.41	1.00	0.38	0.81
GPT-5	0.95	1.00	0.70	0.75	0.22	0.22	0.75	1.00	0.67	0.77	0.66	0.75
DeepSeek-V3.2	0.91	1.00	0.75	0.85	0.20	0.55	0.76	1.00	0.70	0.92	0.66	0.86
Gemini-3-Pro	0.88	1.00	0.66	0.87	0.16	0.70	0.58	1.00	0.61	1.00	0.58	0.91

Table 2: Coverage (proportion of valid plans) in **loosely coupled systems** using LLM-Repair and LLM-Modulo strategies. The average column records the mean coverage across all domains for the repair and the modulo strategies, respectively. The models are arranged in increasing order of their total parameters, with GPT-5 having approximately 600B parameters, DeepSeek-V3.2 with 671B, and Gemini-3-Pro having nearly 1T parameters.

Model	Depot		Logistics		Taxi		Elevators		Woodworking		Average	
	Repair	Modulo	Repair	Modulo	Repair	Modulo	Repair	Modulo	Repair	Modulo	Repair	Modulo
GPT-OSS-20B	0.05	0.05	0.00	0.42	0.36	0.72	0.10	0.27	0.07	0.15	0.12	0.32
GPT-OSS-120B	0.35	0.50	0.35	0.70	0.72	0.85	0.41	0.80	0.15	0.27	0.40	0.62
GPT-5	0.35	0.62	0.70	1.00	0.75	0.80	0.17	0.72	0.20	0.30	0.43	0.69
DeepSeek-V3.2	0.60	0.90	0.90	0.87	0.76	0.87	0.66	0.65	0.20	0.45	0.62	0.75
Gemini-3-Pro	0.36	0.95	0.81	1.00	0.33	1.00	0.08	0.65	0.13	0.45	0.34	0.81

Table 3: Coverage (proportion of valid plans) in **tightly coupled systems** using LLM-Repair and LLM-Modulo strategies. The last column records the mean coverage across all domains for both strategies.

Model	BW	DL	RV	SAT	ZT	Avg
GPT-OSS-20B	0.96	0.80	0.46	0.63	0.88	0.74
GPT-OSS-120B	1.00	0.91	0.50	0.76	0.96	0.82
GPT-5	1.00	1.00	0.60	0.76	1.00	0.87
DeepSeek-V3.2	0.76	0.81	0.41	0.63	0.95	0.71
Gemini-3-Pro	0.48	0.95	0.53	0.70	0.95	0.72

Table 4: Coverage across loosely coupled planning domains for the plan merging approach. BW: Blocksworld, DL: Driverlog, RV: Rovers, SAT: Satellite, ZT: Zenotravel. Avg denotes mean coverage across domains.

6 Conclusion

In this work, we have investigated the capabilities of Large Language Models (LLMs) in solving structured multi-agent planning described in MA-PDDL. We evaluated some of the current state-of-the-art LLMs across the benchmarks introduced in the CoDMAP competition using three different approaches, namely, LLM-Repair, LLM-Modulo, and LLM-Merge. All of these approaches use zero-shot prompting. From the baseline experimental results, we found that current state-of-the-art LLMs are capable of generating plans while maintaining a structured format. However, problems requiring greater coordination between agents remain a challenge. While the LLM-Repair approach shows only modest improvement over the baseline, the LLM-Modulo approach shows significant gains by incorporating iterative feedback from a plan validator. This shows the importance of external validators in LLM-based plan generation. Additionally, our LLM-Merge approach shows further improvements in Loosely Coupled Domains by decomposing multi-agent problems into single-agent tasks and merging their plans. This indicates that LLMs can be more effective when used as components of a structured planning pipeline rather than standalone planners. Finally, we make our source code publicly available to support further research.

Acknowledgment

We used Generative AI tools for minor paraphrasing and manuscript revision. All final content was reviewed by the authors.

Limitations

While our approach explores some LLM-based approaches and experimentally evaluates them, a few limitations remain. First, our approach relies solely on zero-shot prompting, which may not fully utilize the LLM's capabilities. Few-shot prompting could potentially be better. We have also not explored task-specific fine-tuning. Second, all our experiments were conducted on benchmarks from the CoDMAP competition. Although these domains are widely used in multi-agent planning research, they represent a limited set of problems. Evaluating the approaches on more robust benchmark data would help assess their generality. Finally, the LLM-merging approach depends on the degree of coupling between the agents and is primarily effective for loosely coupled systems. Effective strategies for task decomposition and plan merging in tightly coupled systems remain an open challenge, which we did not explore in this work.

References

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, and 1 others. 1998. Pddl—the planning domain definition language. *Technical Report, Tech. Rep.*

Mohamed Aghzal, Erion Plaku, and Ziyu Yao. 2023. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning. *arXiv preprint arXiv:2310.03249*.

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. 2020. Pyperplan. https://doi.org/10.5281/zenodo.3700819.

Ronen I Brafman and Carmel Domshlak. 2006. Factored planning: How, when, and when not. In *AAAI*, volume 6, pages 809–814.

Ronen I. Brafman and Carmel Domshlak. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pages 28–35. AAAI.

Michael Brenner. 2003. A multiagent planning language. In *Proc. of ICAPS'03 Workshop on PDDL*, volume 3.

J Britanik and M Marefat. 1995. Hierarchical plan merging with application to process planning. In *IJCAI*, pages 1677–1685.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, and 1 others. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR.

Georgia Chalvatzaki, Ali Younes, Daljeet Nandha, An Thai Le, Leonardo FR Ribeiro, and Iryna Gurevych. 2023. Learning to reason over scene graphs: a case study of finetuning gpt-2 into a robot language model for grounded task planning. *Frontiers in Robotics and AI*, 10:1221739.

Jeffrey Cox and Edmund Durfee. 2009a. Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent and Grid Systems*, 5(4):373–408.

Jeffrey S. Cox and Edmund H. Durfee. 2009b. Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent Grid Syst.*, 5(4):373–408.

Richard Fikes and Nils J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208.

Dan Gaines, Steve Chien, Gregg Rabideau, Stephen Kuhn, Vincent Wong, Amruta Yelamanchili, Shannon Towey, Jagriti Agrawal, Wayne Chi, Andrea Connell, Evan Davis, and Colette Lohr. 2022. Onboard planning for the mars 2020 perseverance rover. In *16th Symposium on Advanced Space Technologies in Robotics and Automation*.

Michael Georgeff. 1988. Communication and interaction in multi-agent planning. In *Readings in distributed artificial intelligence*, pages 200–204. Elsevier.

Gonzalo Gonzalez-Pumariega, Wayne Chen, Kushal Kedia, and Sanjiban Choudhury. 2024. Query-efficient planning with language models. *arXiv preprint arXiv:2412.06162*.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Malte Helmert. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Jörg Hoffmann. 2001. Ff: The fast-forward planning system. *AI magazine*, 22(3):57–57.

R. Howey, D. Long, and M. Fox. 2004. Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32779–32798.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

M. D. Johnston. 1990. Spike: Ai scheduling for nasa's hubble space telescope. In *Proceedings of the Sixth Conference on Artificial Intelligence Applications*, page 184–190. IEEE Press.

Da Ju, Song Jiang, Andrew Cohen, Aaron Foss, Sasha Mitts, Arman Zharmagambetov, Brandon Amos, Xian Li, Justine T Kao, Maryam Fazel-Zarandi, and 1 others. 2024. To the globe (ttg): Towards language-driven guaranteed travel planning. *arXiv preprint arXiv:2410.16456*.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. 2024. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*.

Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi Araghi. 2024. Thought of search: Planning with language models through the lens of efficiency. *Advances in Neural Information Processing Systems*, 37:138491–138568.

Antonín Komenda, Michal Stolba, and Daniel L Kovacs. 2016. The international competition of distributed and multiagent planners (codmap). *AI Magazine*, 37(3):109–115.

Daniel L Kovacs and 1 others. 2012. A multi-agent extension of pddl3. 1. In *ICAPS 2012 Proceedings of the 3rd Workshop on the International Planning Competition (WS-IPC 2012)*, pages 19–37.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. 2024. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*.

Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony Cohn, and Janet B Pierrehumbert. 2024. Graph-enhanced large language models in asynchronous plan reasoning. *arXiv preprint arXiv:2402.02805*.

Nerea Luis, Susana Fernández, and Daniel Borrajo. 2020. Plan merging by reuse for multi-agent planning. *Applied Intelligence*, 50(2):365–396.

Amol Dattatraya Mali. 1999. Plan merging & plan reuse as satisfiability. In *European Conference on Planning*, pages 84–96. Springer.

F von Martial. 1990. Interactions among autonomous planning agents. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World 1989*.

Lenka Mudrova, Bruno Lacerda, and Nick Hawes. 2016. Partial order temporal plan merging for mobile robot tasks. *Frontiers in Artificial Intelligence and Applications*, 285(ECAI 2016).

Dana Nau, Malik Ghallab, and Paolo Traverso. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252.

Silvia Richter and Matthias Westphal. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.

Jendrik Seipp. 2018. Fast downward scorpion. *Ninth International Planning Competition (IPC-9): Planner Abstracts*, pages 77–79.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 20256–20264.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Pddl planning with pretrained large language models. In *NeurIPS 2022 foundation models for decision making workshop*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: program generation for situated robot task planning using large language models. *Auton. Robots*, 47(8):999–1012.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009.

Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. 2015. A first multi-agent planner for required cooperation (marc). *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP'15)(2015)*, pages 17–20.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245.

Alejandro Torreño, Eva Onaindia, Antonín Komenda, and Michal Stolba. 2018. Cooperative multi-agent planning: A survey. *ACM Comput. Surv.*, 50(6):84:1–84:32.

Alejandro Torreno, Eva Onaindia, and Oscar Sapena. 2014. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626.

Alejandro Torreno, Oscar Sapena, and Eva Onaindia. 2015. Global heuristics for distributed cooperative multi-agent planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 225–233.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. In *NeurIPS 2024 Workshop on Open-World Agents*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. PlanGenLLMs: A modern survey of LLM planning capabilities. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19497–19521, Vienna, Austria. Association for Computational Linguistics.

Ruihan Yang, Jiangjie Chen, Yikai Zhang, Siyu Yuan, Aili Chen, Kyle Richardson, Yanghua Xiao, and Deqing Yang. 2025. Selfgoal: Your language agents already know how to achieve high-level goals. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 799–819.

Ryan Yang, Tom Silver, Aidan Curtis, Tomas Lozano-Perez, and Leslie Pack Kaelbling. 2022. Pg3: policy-guided planning for generalized policy generation. *arXiv preprint arXiv:2204.10420*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.