CaStL: Constraints as Specifications through LLM Translation for Long-Horizon Task and Motion Planning

Weihang Guo, Zachary Kingston, and Lydia E. Kavraki

***Abstract*— Large Language Models (LLMs) have demonstrated remarkable ability in long-horizon Task and Motion Planning (TAMP) by translating clear and straightforward natural language problems into formal specifications such as the Planning Domain Definition Language (PDDL). However, real-world problems are often ambiguous and involve many complex constraints. In this paper, we introduce Constraints as Specifications through LLMs (CaStL), a framework that identifies constraints such as goal conditions, action ordering, and action blocking from natural language in multiple stages. CaStL translates these constraints into PDDL and Python scripts, which are then solved using an custom PDDL solver. Tested across three PDDL domains, CaStL significantly improves constraint handling and planning success rates from natural language specification in complex scenarios.**

I. Introduction

Task and Motion Planning (TAMP) methods for long-horizon robot decision-making are capable of solving complex manipulation tasks [1]. However, TAMP methods usually require explicit modeling of the problem domain in formal languages such as the Planning Domain Definition Language (PDDL) [2], thus burdening the system's end-user to specify desired behavior. Recent work in Large Language Models (LLMs) has provided a means of reasoning directly from natural language (NL). Some TAMP approaches use LLMs as task planners to directly generate task sequences [3–5]. However, while very powerful for reasoning in general domains, LLMs have inherent limitations (e.g., hallucinations [6–8]) that prove difficult to overcome in domains that require complex, non-monotone reasoning. Alternatively, LLMs can be used as translators [9, 10] to convert an NL task into a formal language specification, which can then be solved by a standard method. These methods achieve higher success in domains when the NL task is clear and straightforward—real-world problems are often ambiguous and involve intricate *constraints* on the solution. For example, a robot may need to avoid entering restricted areas in a warehouse, requiring navigation through many different rooms, or refrain from touching fragile objects while cleaning a table, thus requiring many additional, potentially non-intuitive steps to clear the table.

Recognizing the convenience of using NL to specify tasks and their constraints and taking advantage of the capabilities of modern LLMs, this work shows how to exploit LLMs - not only as translators but also as code generators - to get from NL task specifications to standardized formal specifications that can be used by classic TAMP solvers. More specifically, our approach, **CaStL** (Constraints as Specifications through LLMs), uses LLMs to identify four different types of constraints (i.e., attribute constraints, eventual constraints, implication constraints, and global constraints) in a task specification given in NL. These are then encoded into specifications for an SMT-based PDDL solver [11] through a combination of translation to PDDL (using the LLM as a translator) as well as a custom PDDL modeling Python API (using the LLM as a code generator). Our method also addresses motion constraints, essential in real-world robotics where task plans often fail due to unreachable objects or invalid grasps, by using a full TAMP stack, based on prior work in constraint-based TAMP [12]. PDDL was chosen as the end formal specification in this paper, but other formal languages are possible. Although our work is not the first to address the problem of transitioning NL to task specification (e.g., AutoTAMP [13], LLM+P [9]), our method provides a general extensible and robust framework that will, in the long run, facilitate interactions with human in the context of TAMP. We demonstrate the power of our approach in three different PDDL domains and in complex TAMP scenarios.

II. Preliminaries

Our work addresses translating NL tasks into specifications for Task and Motion Planning (TAMP) approaches. We assume that we are given a model of the robot, environment, a Planning Domain Definition Language (PDDL) domain and a partially specified problem that specifies what actions and predicates are available as well as what objects are in the scene. We denote this partially-specified PDDL problem with $(\mathcal{P}, \mathcal{A}, \mathcal{O})$, where $P(o_1, \ldots, o_n) \in \mathcal{P}, o_i \in \mathcal{O}, P \rightarrow \{0,1\}$ is the set of predicate functions and $A(o_1, \ldots, o_n) \in \mathcal{A}$ is the set of actions, both grounded by object arguments. We also include *object attributes* $D(o) \in \mathcal{D}, o \in \mathcal{O}, D \rightarrow \{0,1\}$, which encode additional properties about objects, e.g., if it is the color red, if it is heavy, and so on. A state of the world is a set of predicates $s = \{P_1, \ldots, P_n\}$ which are true, all other predicates are assumed to be false (a closed world). Actions have preconditions (a logical expression using predicates as atoms), effects (a set of predicates that become either $\{0,1\}$ at the next time step). In TAMP, actions are further grounded by a motion planner (e.g., finding a sequence of trajectories and a grasp pose for a manipulator to pick up a block). Even if an action's preconditions hold, it may not be the case that it is executable (e.g., a motion plan could not be found), thus meriting TAMP algorithms which consider the problem of

WG, ZK, and LEK are with the Department of Computer Science, Rice University, Houston TX, USA {wg25, zak, kavraki}@rice.edu. LEK is also affiliated with the Ken Kennedy Institute at Rice University. ZK is currently with the Department of Computer Science, Purdue University, West Lafayette IN, USA zkingston@purdue.edu. This work was supported in part by NSF RI 2336612 and Rice University funds.

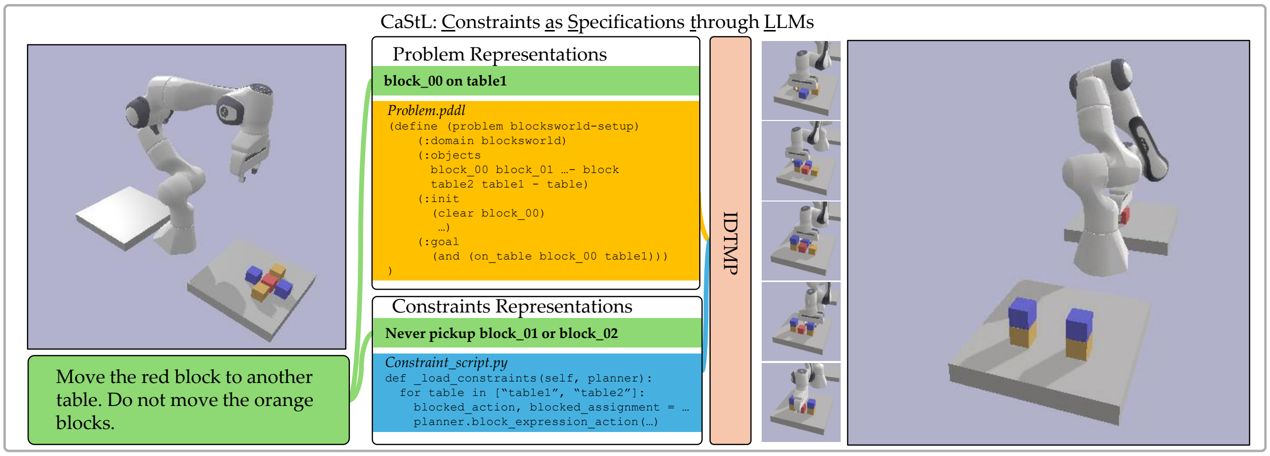


Fig. 1: Our proposed method, CaStL, allows specification of Task and Motion Planning (TAMP) problems with constraints in natural language using a multi-step process (detailed in Sec. IV). Here, a TAMP problem (*Move the red block to another table*) with an additional global constraint (*Do not move the orange blocks*) is specified. Our approach resolves ambiguities and breaks the problem down into a PDDL representation and a Python script, both of which provide constraints that are added to a SMT-based TAMP solver (IDTMP) [12] with a Python API. This solver is capable of resolving motion constraints (here, the red block cannot be grasped without moving one colored pair of blocks out of the way). The color of each step corresponds to the module with the same color in Fig. 2.

not only finding a feasible task plan (a sequence of actions $\pi = A_1, \ldots, A_n$), but a sequence of feasible motions that achieves the sequence of actions. For an extended discussion on the topic of grounding, see [12].

Core to any TAMP solver is the ability to account for motions failures by enumerating a number of *different task plans* or attempting many motion groundings. TAMP methods impose *constraints* on their task planning [12, 14], blocking actions from being taken in certain states (or under some logical expression, e.g., when objects are at specific locations). The presence of constraints reduces the possible task plans and several TAMP method (e.g., [12]) have mechanisms to produce task plans that are likely to be feasible. Our approach provides the means to incorporate a broad range of *task constraints* using NL, effectively limiting possible task plans and accelerating the solution of complex TAMP problems.

A. Task Constraints

We consider a number of different constraint classes, all of which are built upon logical expressions ϕ, where formulae are recursively defined with predicate atoms and operators such as *and*, *or*, *implies*, *not*, as well as universal $\forall$ and existential $\exists$ quantification over objects or object attributes. The constraints listed below are the ones considered in this work: many other constraints are possible and easy to encode in our framework, which is general to most logical expressions.

1) Attribute Constraints: As described, attributes of objects can be used in constraint expressions. While not directly supported in PDDL for quantification, our approach introduces a multi-stage approach to include attributes in constraint expressions, as described in Sec. IV-A.

2) Eventual Constraints: Eventual constraints specify that certain conditions must hold true at the end of the plan. These constraints ensure that specific tasks are completed at some point before the plan's completion, regardless of the order in which other actions are executed, and are expressed as a conjunction of predicates in the PDDL goal.

3) Global Constraints: In some problems it is desirable to *always* avoid certain conditions. For example, always avoiding a certain room, never picking up a specific block (Fig. 1). Much like preconditions, these are expressed as logical expressions that apply to the state of the world, but they must always hold over every state in the task plan.

4) Implication Constraints: We also want to enforce sequential constraints—i.e., preventing certain actions from being taken until another action has been taken, similar to concepts in temporal logic. These take form of a logical expression that must hold true before a specific grounding or quantification of an action can be taken, in addition to the actions normal preconditions. For example, *"blue blocks can only be picked up after the red blocks are stacked"* would be an implication constraint.

III. Related Work

Task planning involves finding a sequence of actions to transition from a given start state to a desired goal condition (e.g., STRIPS [15]). Many logics and languages can encode task planning problems, such as Linear Temporal Logic (LTL) [16–18], Signal Temporal Logic (STL) [19], context-free grammars [20], and others. This work uses the Planning Domain Definition Language (PDDL) [2] due to its common usage, human-readable format, and factored representation. Moreover, in PDDL 3.0 [21], the ability to consider *constraints* on the solution was added, which enforces additional logical conditions that must hold over the found task—but this feature is poorly supported by solvers. Given the limited PDDL 3.0 training data available to train LLMs, directly translating constraints into PDDL 3.0-style specifications is unreliable and is avoided in our work as described in Sec. IV

Having the task planner consider motion constraints is essential in TAMP solving, as many actions a robot might take might be infeasible due to geometric conditions, e.g., an object is blocking the gripper from picking up a block, and so on. Works such as IDTMP [12] and COAST [14] use constraint-aware solvers in order to focus the search and more efficiently enumerate possible solutions. In particular, Satisfiability Modulo Theories (SMT) solvers [11] used by IDTMP are a flexible approach for incorporating these constraints into TAMP. We extend the SMT-based solving of IDTMP in this work to consider the four types of constraints discussed in Sec. II-A.

A. LLMs for Task and Motion Planning

There are two broad categories that describe how LLMs have been used to solve TAMP specified in NL: LLMs as task planners, and LLMs as translators.

First, LLMs-as-task-planners use the reasoning capabilities (e.g., Huang et al. [22]) of LLMs directly to generate a task plan, either step-by-step or as a whole. Initial works [3, 23] use zero-shot generation of action sequences from an NL description, but they suffer from poor execution success; later methods either generate a new plan upon failure [4, 24–26] or iteratively find the next action to execute [27–29]. To further ground the actions (i.e., finding feasible motions to execute), some works [30–33] use affordance functions or other heuristics to guide LLM inference, learn downstream networks to use LLM output [29], or integrate environment data into planning [5, 28, 34]. These methods perform very well on tasks that require "common-sense" reasoning and have many independent actions, but fail to scale to more complex problems, due to limitations such as context-faithfulness [6], hallucination [7], and principle reasoning [8]. In this work, we are concerned with tasks with many *constraints* on valid actions where solving them may require complex, non-monotone reasoning, to which LLMs-as-task-planners are ill-suited.

To overcome the natural limitations of LLM reasoning, many approaches instead use the LLM as a *translator*, to convert NL requests into a formal language, for example, LTL [35, 36], STL [13, 37], and MTL [38]. Relevant to this work, many approaches have translated NL into PDDL [9, 10, 39–41] and have even generated the PDDL domain [42, 43]. However, none of PDDL translation approaches have *directly* addressed translation of NL tasks that include constraints in addition to reaching the goal. One approach, DELTA [10], splits the NL task into subgoals, handling some constraints that require actions done in a certain order. However, this approach does not guarantee optimal makespan, and was not designed to handle complex constraints—performance remains low on many relevant problems (see Sec. V). Some approaches which translate NL to temporal logics (e.g., NL2TL [37] and AutoTAMP [13]) directly handle ordering constraints due to the nature of the formal language. Among previous works, AutoTAMP is the closest to this work. However, it demonstrate results only in 2D domains and consider only two actions: *enter* and *not-enter*. While not able to handle the full gamut of temporal constraints, our approach handles

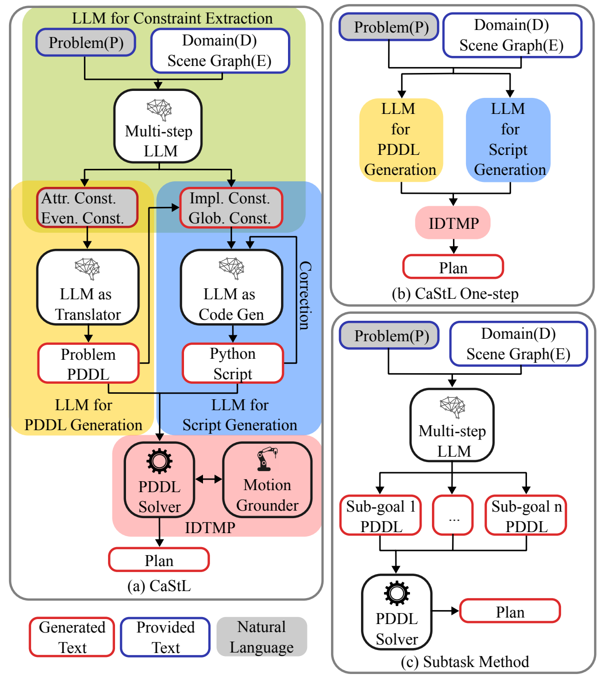


Fig. 2: Illustrations of approaches. (a) In our approach, CaStL, constraints are first extracted through multi-step LLM queries (Sec. IV-A). Then, the LLM translates natural language constraints into PDDL (Sec. IV-B) as well as Python scripts which use an API on our SMT-based PDDL solver within a TAMP algorithm, IDTMP [12] (Sec. IV-C). (b) CaStL One-step is an ablation of CaStL, without the multi-step LLM process for extracting constraints. (c) Baseline. The problem is decomposed into natural language subproblems, which are translated sequential into PDDL.

4 types of constraints (Sec. II-A) in PDDL domains with 3D manipulation workspaces.

IV. CaStL

The goal of our method, CaStL (Constraints as Specifications through LLMs), is to parse natural language (NL) tasks in a given Planning Domain Definition Language (PDDL) domain into specifications understood by a TAMP solver to find a feasible TAMP plan to execute on a system. Our approach focuses on specifications as a set of *constraints* (detailed in Sec. II), that involve not only a desired end goal condition (*eventual constraints*), but also additional constraints that limit what actions can be taken at a given step in the plan, e.g., never taking certain actions (*global constraints*) or only taking an action once another condition has been fulfilled (*implication constraints*).

To achieve this, we first perform a multi-step query to an LLM (Sec. IV-A). We provide the PDDL domain, an environment scene graph, and the user's NL query as inputs to the LLM, which translates the NL query into a list of constraints. Next, the LLM translates the *eventual constraints* into a PDDL problem (Sec. IV-B). The *global* and *implication constraints* are converted by multi-step LLM prompt with semantic, syntactic, and error checking into a Python script which adds the constraints to the TAMP solver (Sec. IV-C). The fully specified problem is then solved by a TAMP planner, considering motion feasibility, and a final plan is generated if possible within the time limit.

```python
def _load_constraints(self, planner):
    backyard_unvisited = self.
    make_grounded_predicate("visited", ["robot1", "
    backyard"])

    # Create constraints for all the rooms except
    backyard
    rooms = ["kitchen", "bedroom1", "bedroom2", "
    restroom"]
    constraints = []
    for room in rooms:
        condition = self.make_grounded_predicate("
    visited", ["robot1", room])
        constraints.append(condition)

    and_condition = pd.make_and(constraints)

    # Block the 'move' action to the backyard if
    not all other rooms are visited
    blocked_action, blocked_assignment = self.
    make_action_assignment("move", ["robot1", "
    living-room", "backyard"])
    planner.block_expression_action(blocked_action,
     blocked_assignment, pd.make_not(and_condition)
    , pd.Assignment())
```

Listing 1: Python script for the constraint *visiting all rooms before the backyard*. This script is automatically generated by GPT-4o.

A. LLM for Multi-step Constraint Extraction

The LLM takes a user-specified problem in NL, a PDDL domain, and an environment scene graph as input. First, the LLM resolves ambiguities by matching pronouns and *attribute constraints* to the environment. For example, the original problem, *"You must visit the rooms that have a bed. But, you should visit the one with the largest bed first."* becomes *"The robot must visit room05, and room06. The robot should visit room05 first."* In this example *"rooms with a bed"* is an attribute constraint. Second, after resolving ambiguities, the LLM determines if the NL problem expresses implication and global constraints. If those constraints are present, our work identifies eventual, implication, and global constraints, guided by in-context examples. Otherwise, only eventual constraints are identified. In this case, *"The robot must visit room05, room06."* is an eventual constraint, and *"The robot should visit room05 first."* is an implication constraint.

B. LLM for PDDL Problem Generation

We employ the same approach as LLM+P [9] for PDDL generation. The input consists of all eventual constraints in NL, a PDDL domain, the environment, and in-context examples in the same domain. The output is a PDDL problem specification. The key difference between CaStL and LLM+P in this module is that we use the output of the prior multi-step query to disambiguate the problem as input. We include a *one-step* ablation, which does not have the above multi-step process in our experiments (Sec. V-B) to show that adding this step significantly increases success.

C. LLM for Constraint Script Generation and Correction

Although PDDL supports constraint specification since 3.0 [21], it is rarely used and thus not present in LLM training data or well supported. Thus, the success rate for LLMs to directly translate constraints into PDDL (i.e., the `:constraints` field) is low. Therefore, we instead prompt the LLM to extract and translate constraints into a Python script that uses a custom API to build logical expressions and add constraints to the task planner.

The translation process is achieved in multiple steps: first, each additional constraint in natural language (NL) is paraphrased into a constraint-specific format. Implication constraints are paraphrased into `Never/Always <expression>` and global constraints are paraphrased into `Do not <action> until <expression> is/is not true`. Next, the paraphrased expressions are translated into a Python script by the LLM (an example is shown in Lst. 1). Syntax errors are handled with corrective re-prompting [44], where the error message is given to the LLM to regenerate the script. We also use the LLM to evaluate whether the generated script was semantically consistent with the original instructions.

We use the SMT-based [11] TAMP algorithm from Dantam et al. [12], where the solver maintains a constraint stack to generate alternate task plans with an increasing horizon. In addition to internally generated motion constraints, we provide a Python API for scripting additional constraints onto the constraint stack, examples of which are visible in Lst. 1.

An ablation of this module is evaluated in Sec. V-D, where instead of having the LLM generate Python, we ask it to translate the constraints into a JSON schema (Lst. 2), which is then parsed into the solver. The results show that JSON translation performs worse than scripting at accurately modeling constraints.

V. Validation

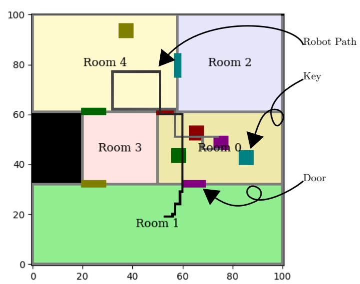
We evaluate on three domains: HouseChip (HC), Kitchen (KT), and BlocksWorld (BW), described further in Sec. V-A. For each domain, we consider two classes of randomly generated environments, one simple (ending with 1) and one complex (ending with 2), with complexity based on the number of objects. We also consider increasing complexity of tasks: only eventual constraints (No), eventual and implication constraints (Impl), eventual and global constraints (Glob), eventual, implication, and global constraints (Impl Glob), finally all types of constraints (Impl Glob Attr). 11 trials of each task in each environment are evaluated for each method, discussed in Sec. V-B. We use GPT-4o for all our experiments due to its balance between cost and performance.

A. Domain Descriptions

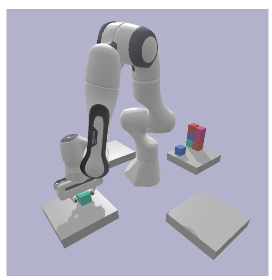
The **HouseChip (HC)** domain (Fig. 3a) is inspired by the HouseWorld and Chip's Challenge domains from AutoTAMP [13]. Despite their similarity, adapting AutoTAMP to this domain for a fair comparison is non-trivial. AutoTAMP simplifies doors, keys, walls, and rooms into a single *room* entity, focusing on two actions: *enter* or *not enter*. Additionally, we found that some parameters in their algorithm are sensitive to the specific setup. Thus, we excluded AutoTAMP from the comparison. In their experiment, eventual constraints are to

Cases	Methods	Problems and Constraints					Average	Time
		No Logic / Motion	Impl Logic / Motion	Glob Logic / Motion	Impl Glob Logic / Motion	Impl Glob Attr Logic / Motion	Input Tokens	in LLM (s)
HC 1	Subtask	91%/91%	91%/91%	73%/73%	27%/27%	18%/18%	9140	36.13
	CaStL (one step)	**100%/100%**	91%/91%	91%/91%	27%/27%	-/-	10415	11.35
	CaStL	**100%/100%**	**100%/100%**	**100%/100%**	**73%/73%**	**73%/73%**	15112	10.79
HC 2	Subtask	82%/82%	91%/91%	18%/18%	82%/82%	**91%/91%**	7767	17.60
	CaStL (one step)	91%/91%	**100%/100%**	91%/91%	**91%/91%**	18%/18%	7321	8.24
	CaStL	**100%/100%**	**100%/100%**	**100%/100%**	**91%/91%**	**91%/91%**	9195	9.42
KT 1	Subtask	55%/-	9%/-	-/-	-/-	-/-	9140	36.13
	CaStL (one step)	36%/-	**100%/-**	55%/-	91%/-	**100%/-**	10415	11.35
	CaStL	**91%/-**	**100%/-**	**82%/-**	**100%/-**	91%/-	15040	11.44
KT 2	Subtask	9%/-	45%/-	-/-	-/-	-/-	11448	45.71
	CaStL (one step)	55%/-	73%/-	45%/-	**64%/-**	27%/-	10711	10.86
	CaStL	**100%/-**	**100%/-**	**82%/-**	**64%/-**	**64%/-**	15938	11.76
BW 1	Subtask	-/-	9%/-	-/-	-/-	-/-	-	-
	CaStL (one step)	82%/45%	55%/55%	64%/36%	36%/55%	36%/36%	4520	8.11
	CaStL	**100%/64%**	**64%/64%**	**100%/73%**	**73%/73%**	**64%/73%**	7424	10.52
BW 2	Subtask	55%/-	27%/-	27%/-	9%/-	27%/-	4124	12.32
	CaStL (one step)	64%/55%	82%/55%	18%/18%	**64%/55%**	**64%**/36%	7681	15.80
	CaStL	**82%/82%**	**100%**/64%	**55%/45%**	**64%**/45%	**64%/45%**	8021	16.41

TABLE I: Success rates for pure task planning (*Logic*) and after grounding with motion planning (*Motion*) are presented for the BW, HC, and KT domains. The KT domain and the Subtask method do not ground motions, denoted by '-'. A success rate of 0% is also marked as '-'. *Average input tokens* is the average total number of tokens spent on each trial with task planning success, average time on LLM queries given by *Time in LLM*. *No, Impl, Glob*, and *Attr* refer respectively to problems with no, implication, global, and attribute constraints. Constraints are added to the same initial *No* problem for each case. All experiments use GPT-4o.



(a) HOUSECHIP(HC)



(b) BLOCKSWORLD(BW)

Fig. 3: (a) The HC domain features a robot starting in Room 0, tasked with visiting a list of rooms, each of which locked by a corresponding key. (b) The BW domain, which consists of pick, place, stack, and unstack actions with a number of blocks and tables.

visit rooms, with all doors initially locked and can be unlocked by obtaining the corresponding keys. The robot must navigate within 5 units of the room's center to mark it as visited; A^* [45] is used to ground motions. Implication constraints specify room visitation order. Global constraints restrict room access or key collection. This domain is custom-designed, so LLMs lack prior training data.

The **KITCHEN (KT)** domain is adapted from the 2014 International Planning Competition [46]. Eventual constraints task the robot with making and delivering sandwiches to some or all children. Some children are allergic to gluten, requiring both normal and gluten-free sandwiches. Implication constraints force delivery order or force preparation of all sandwiches before serving. Global constraints block certain ingredients or trays.

In **BLOCKSWORLD (BW)**, a manipulator arranges blocks in a specified order using the Franka Emika Panda robot, with RRT-Connect [47] to ground motion. Implication constraints prevent picking up certain blocks until other criteria are met. Global constraints prohibit moving certain blocks or placing specific blocks on specific tables. LLMs likely have ample training data for this domain. Block positions are randomized in each trial, and thus the motion planner might succeed in one instance but fail in another.

B. Algorithm Ablations and Baseline

We present an ablation of CaStL (shown in Fig. 2b), where LLMs directly translate into PDDL and Python script, bypassing multi-step constraint extraction (CaStL One-step). We also implemented a baseline approach following the DELTA [10] architecture (Fig. 2c) which decomposes the problem into sub-problems and sequentially solves generated PDDL problems (Subtask). Since DELTA does not provide their full prompt, we created our own to achieve problem decomposition. We use the same prompt and in-context examples as CaStL for translation. While DELTA includes PDDL domain generation via LLMs, we excluded this as it is not the focus of this paper.

C. Results

Results are shown in Tbl. I. Two success rates are reported: one for pure task planning, one for task and motion planning. We also record the average number of input tokens and the time taken by the LLM per run. The TAMP solver is given a timeout of 60 seconds. Success is evaluated with a script to compare against ground truth constraints.

The full CaStL method overall outperforms the one-step approach. The one-step method often reports false positives on problems without constraints, demonstrating the utility of the multi-step prompting strategy. The total time spent on LLM queries is comparable to the one-step method, even though CaStL uses more queries due to its multi-step process. This is likely because each step is simpler, allowing the LLM to process them efficiently. While the baseline approach can handle some implication constraints, it struggles with global constraints, and has longer LLM processing times due to individual translation of sub-problems.

Cases	Problems	Methods	Success Rate
BW 1	Glob	Script	**100%**
		Script w/o correction	91%
		JSON	**100%**
	Impl	Script	**64%**
		Script w/o correction	**64%**
		JSON	**64%**
	Impl Glob	Script	**73%**
		Script w/o correction	64%
		JSON	**73%**
	Impl Glob Attr	Script	**64%**
		Script w/o correction	45%
		JSON	**64%**
BW 3	Glob	Script	**36%**
		Script w/o correction	18%
		JSON	-
	Impl	Script	**91%**
		Script w/o correction	**91%**
		JSON	73%
	Impl Glob	Script	**45%**
		Script w/o correction	36%
		JSON	-
	Impl Glob Attr	Script	**45%**
		Script w/o correction	**45%**
		JSON	-

TABLE II: Comparison of the Python *Script* and JSON approaches for representing constraints on task planning success. The rest of the CaStL method is kept the same. *Impl*, *Glob*, and *Attr* refer to problems expressing implication, global, and attribute constraints. The *Script* method handles natural language ambiguity and many-to-one mappings with loops, while the JSON approach requires explicit specification of all constraints.

D. Variants for Constraint Script Generation

We compare an ablation of our Python script generation for implication and global constraint specification (e.g., Lst. 1) to an approach where constraints are translated and parsed to and from a JSON schema (Lst. 2). We evaluate on the BW domain, both with and without correction. The results, shown in Tbl. II, compare the BW 3 setup, which shares the same environment as BW 2 but includes additional constraints. These constraints included conditions like *"the robot cannot move blocks 1, 2, and 3 when block 4 is on block 5"* or *"all blocks can only be placed on their original table."*

We found that NL action names often lack one-to-one mappings (e.g., "touch" maps to both *pick-up* and *unstack* in BW). NL also tends to omit indirect objects and use quantification, causing a single sentence to map to many constraints. The Python script approach efficiently handles this with loops, while JSON requires all constraints to be explicitly specified. For small numbers of constraints, JSON performs well, sometimes surpassing the Python script approach without correction. However, as the number of constraints increase, LLMs struggle to capture all of them, leading to a significant drop in success rate.

E. Cross-domain Generalizability

CaStL generalizes effectively across different domains when provided with in-context examples. We further evaluate performance by in-context examples from a different domains, shown in Tbl. III, where CaStL solves the same problems in KITCHEN using in-context examples from the BLOCKSWORLD domain. We observed that the algorithm handles implication constraints more robustly than global

```
[
  {
    "type": "implication",
    "action": ["pick-up", "block0", "table0"],
    "condition": [["on", "block4", "block5"]]
  },
  {
    "type": "global",
    "condition": [["not", "on_table", "block0", "
    table1"]]
  },
  ...
]
```

Listing 2: Return implication and global constraints in JSON

Cases	Problems	Methods	Success Rate
KT 1	Glob	Same context	**91%**
		BW context	18%
	Impl	Same context	**100%**
		BW context	73%
	Impl Glob	Same context	**100%**
		BW context	27%
	Impl Glob Attr	Same context	**100%**
		BW context	18%
KT 2	Glob	Same context	**82%**
		BW context	27%
	Impl	Same context	73%
		BW context	**82%**
	Impl Glob	Same context	**73%**
		BW context	9%
	Impl Glob Attr	Same context	**65%**
		BW context	-

TABLE III: Comparison of same-domain in-context examples and cross-domain in-context examples for task planning. *Impl*, *Glob*, and *Attr* refer to natural language problems expressing implication, global, and attribute constraints.

constraints. Notably, for implication constraints in KT 2, the success rate is higher. Failure cases fall into two categories. First, LLMs occasionally miss action parameters or reference non-existent objects; second, they misinterpret whether a condition or its negation should be applied.

VI. CONCLUSION

We present CaStL, a method which efficiently uses multi-step queries to LLMs with few-shot in-context examples to translate attribute, eventual, implication, and global constraints from natural language into a formal specification for Task and Motion Planning (TAMP). By considering ablations of CaStL without multi-step querying and a baseline from the literature [10], we demonstrate the improved capability of our constraint-based approach in specifying complex tasks for TAMP solvers.

We also identified a few limitations. First, the prompt and in-context examples are crucial; while they can be reused within the same domain, designing them requires expertise. We aim to explore methods that do not rely on in-context examples. Second, we plan to support additional types of constraints, such as temporal and geometric constraints, and connect our approach to other task planning systems which may consider uncertainty and open worlds.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. "Integrated task and motion planning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.1 (2021), pp. 265–293.

[2] M. Fox and D. Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.

[3] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser. "Tidybot: Personalized robot assistance with large language models". In: *Autonomous Robots* 47.8 (2023), pp. 1087–1102.

[4] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. "Llm-planner: Few-shot grounded planning for embodied agents with large language models". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 2998–3009.

[5] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. "SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning". In: *7th Annual Conference on Robot Learning*. 2023.

[6] W. Zhou, S. Zhang, H. Poon, and M. Chen. "Context-faithful Prompting for Large Language Models". In: *The 2023 Conference on Empirical Methods in Natural Language Processing*. 2023.

[7] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung. "Towards mitigating LLM hallucination via self reflection". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. 2023, pp. 1827–1843.

[8] S. Kambhampati. "Can large language models reason and plan?" In: *Annals of the New York Academy of Sciences* 1534.1 (2024), pp. 15–18.

[9] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. "Llm+ p: Empowering large language models with optimal planning proficiency". In: *arXiv preprint arXiv:2304.11477* (2023).

[10] Y. Liu, L. Palmieri, S. Koch, I. Georgievski, and M. Aiello. "DELTA: Decomposed Efficient Long-Term Robot Task Planning using Large Language Models". In: *CoRR* abs/2404.03275 (2024).

[11] L. De Moura and N. Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[12] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. "An incremental constraint-based framework for task and motion planning". In: *The International Journal of Robotics Research* 37.10 (2018), pp. 1134–1151.

[13] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan. "AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 6695–6702.

[14] B. Vu, T. Migimatsu, and J. Bohg. "Coast: Constraints and streams for task and motion planning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 14875–14881.

[15] R. E. Fikes and N. J. Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4 (1971), pp. 189–208.

[16] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi. "Towards manipulation planning with temporal logic specifications". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 346–352.

[17] E. M. Clarke. "Model checking". In: *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*. Springer. 1997, pp. 54–56.

[18] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. "Temporal-logic-based reactive mission and motion planning". In: *IEEE transactions on robotics* 25.6 (2009), pp. 1370–1381.

[19] O. Maler and D. Nickovic. "Monitoring temporal properties of continuous signals". In: *International symposium on formal techniques in real-time and fault-tolerant systems*. Springer. 2004, pp. 152–166.

[20] N. Dantam and M. Stilman. "The motion grammar: Analysis of a linguistic method for robot control". In: *IEEE Transactions on Robotics* 29.3 (2013), pp. 704–718.

[21] A. Gerevini and D. Long. "Plan Constraints and Preferences in PDDL3 The Language of the Fifth International Planning Competition". In: *ICAPS 2006* (Jan. 2005).

[22] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 9118–9147.

[23] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor. "Chatgpt for robotics: Design principles and model abilities". In: *IEEE Access* (2024).

[24] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang. "Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[25] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. "Reflexion: Language agents with verbal reinforcement learning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[26] S. S. Raman, V. Cohen, I. Idrees, E. Rosen, R. Mooney, S. Tellex, and D. Paulius. "CAPE: Corrective Actions from Precondition Errors using Large Language Models". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 14070–14077.

[27] T. Yoneda, J. Fang, P. Li, H. Zhang, T. Jiang, S. Lin, B. Picker, D. Yunis, H. Mei, and M. R. Walter. "Statler: State-Maintaining Language Models for Embodied Reasoning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 15083–15091.

[28] R. Fu, J. Liu, X. Chen, Y. Nie, and W. Xiong. "Scene-llm: Extending language model for 3d visual understanding and reasoning". In: *arXiv preprint arXiv:2403.11401* (2024).

[29] J. Zhang, L. Tang, Y. Song, Q. Meng, H. Qian, J. Shao, W. Song, S. Zhu, and J. Gu. "FLTRNN: Faithful Long-Horizon Task Planning for Robotics with Large Language Models". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 6680–6686.

[30] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *Conference on Robot Learning*. PMLR. 2023, pp. 287–318.

[31] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. "Text2Motion: from natural language instructions to feasible plans". In: *Autonomous Robots* 47.8 (Nov. 2023), pp. 1345–1365.

[32] D. Shah, M. R. Equi, B. Osiński, F. Xia, B. Ichter, and S. Levine. "Navigation with large language models: Semantic guesswork as a heuristic for planning". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2683–2699.

[33] D. Paulius, A. Agostini, and D. Lee. "Long-Horizon Planning and Execution With Functional Object-Oriented Networks". In: *IEEE Robotics and Automation Letters* 8.8 (2023), pp. 4513–4520.

[34] J. Xiang, T. Tao, Y. Gu, T. Shu, Z. Wang, Z. Yang, and Z. Hu. "Language models meet world models: Embodied experiences enhance language models". In: *Advances in Neural Information Processing Systems* 36 (2024).

[35] F. Fuggitti and T. Chakraborti. "NL2LTL–a python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 13. 2023, pp. 16428–16430.

[36] J. Pan, G. Chou, and D. Berenson. "Data-efficient learning of natural language to linear temporal logic translators for robot task specification". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11554–11561.

[37] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan. "Nl2tl: Transforming natural languages to temporal logics using large language models". In: *arXiv preprint arXiv:2305.07766* (2023).

[38] K. Manas, S. Zwicklbauer, and A. Paschke. "TR2MTL: LLM based framework for Metric Temporal Logic Formalization of Traffic Rules". In: *2024 IEEE Intelligent Vehicles Symposium (IV)*. 2024, pp. 1206–1213.

[39] R. Arora, S. Singh, K. Swaminathan, A. Datta, S. Banerjee, B. Bhowmick, K. M. Jatavallabhula, M. Sridharan, and M. Krishna. "Anticipate & Act: Integrating LLMs and Classical Planning for Efficient Task Execution in Household Environments". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 14038–14045.

[40] T. Birr, C. Pohl, A. Younes, and T. Asfour. "AutoGPT+ P: Affordance-based Task Planning with Large Language Models". In: *Robotics: Science and Systems*. 2024.

[41] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu. "LLM3: Large Language Model-based Task and Motion Planning with Motion Failure Reasoning". In: *Multi-modal Foundation Model meets Embodied AI Workshop @ ICML2024*. 2024.

[42] K. Shirai, C. C. Beltran-Hernandez, M. Hamaya, A. Hashimoto, S. Tanaka, K. Kawaharazuka, K. Tanaka, Y. Ushiku, and S. Mori. “Vision-Language Interpreter for Robot Task Planning”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 2051–2058.

[43] M. Han, Y. Zhu, S.-C. Zhu, Y. N. Wu, and Y. Zhu. *InterPreT: Interactive Predicate Learning from Language Feedback for Generalizable Task Planning*. 2024.

[44] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex. “Planning with large language models via corrective re-prompting”. In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022.

[45] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[46] M. Vallati, L. Chrpa, M. Grześ, T. L. McCluskey, M. Roberts, S. Sanner, et al. “The 2014 international planning competition: Progress and trends”. In: *AI Magazine* 36.3 (2015), pp. 90–98.

[47] J. Kuffner and S. LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. 2000, 995–1001 vol.2.