L2P: A Python Toolkit for Automated PDDL Model Generation with Large Language Models

Marcus Tantakoun[1,2], **Christian Muise**[1,2], **Xiaodan Zhu**[1,3]
[1]Ingenuity Labs Research Institute, Queen's University
[2]School of Computing, Queen's University
[3]Department of Electrical and Computer Engineering, Queen's University
{20mt1, christian.muise, xiaodan.zhu}@queensu.ca

Abstract

The limitations of direct planning capabilities from Large Language Models (LLMs) have drawn interest in integrating neuro-symbolic approaches within the Automated Planning (AP) and Natural Language Processing (NLP) communities. With the proliferation of related techniques to convert NL to PDDL, we are seeing an ever-increasing set of related methods. To bring them together under a single computational umbrella, we created a unified framework that encompasses the vast majority of existing methods: *L2P*, [1] an open-source Python library developed to assist users in creating their own frameworks. We hope to see the L2P framework adopted by the community as a repository of existing advancements in LLM model acquisition and relevant papers, ensuring that users have access to the most current research and tools under a common framework for fair comparison.

1 Introduction

The advent of Large Language Models (LLMs) has marked a significant paradigm shift in AI, sparking claims regarding emergent reasoning capabilities within LLMs (Wei et al. 2022) and their potential integration into autonomous planning modules for agents (Pallagani et al. 2023). While LLMs, due to the prowess of distributed representation and learning, excel at System I tasks, planning—an essential aspect of System II cognition (Daniel 2017)—remains to be a significant bottleneck (Bengio 2020). Furthermore, LLMs face challenges with long-term planning and reasoning, often producing unreliable plans (Valmeekam, Stechly, and Kambhampati 2024, Pallagani et al. 2023, Momennejad et al. 2023), frequently failing to account for effects and requirements of actions as they scale (Stechly, Valmeekam, and Kambhampati 2024), and their performance degrades with self-iterative feedback (Stechly, Marquez, and Kambhampati 2023; Valmeekam, Marquez, and Kambhampati 2023; Huang et al. 2024).

Acknowledging the evident challenges that LLMs face in direct planning, Automated Planning (AP), or AI planning, presents a promising alternative with its ability to generate robust, optimal plans through logical and computational methods. LLMs, meanwhile, excel at extracting and refining classical planning models via Natural Language (NL), leaving the plan generation to classical planners and heuristic algorithms. Currently, researchers are exploring the nuances of varying pipelines to balance the effectiveness and limitations of LLMs in building such neuro-symbolic frameworks (Mahdavi et al. 2024; Guan et al. 2023; Gestrin, Kuhlmann, and Seipp 2024), specifically, encoded in the Planning Domain Definition Language (PDDL) McDermott et al. 1998.

To consolidate these under a cohesive computational framework—and to go beyond the conceptual connections found in previous frameworks—we developed an integrated system incorporating most current techniques: **Language-to-Plan (L2P)**. L2P plays a crucial role in advancing the NLP and AI planning fields by providing a structured, unified approach to translating natural language into formal planning language (PDDL). Its importance lies in how it simplifies and organizes the diverse techniques developed, offering a single framework that makes these approaches more accessible and comparable. L2P offers three major benefits:

(i) **Comprehensive Tool Suite**: users can easily plug in various LLMs for streamlined extraction experiments with our extensive collection of PDDL extraction and refining tools.

(ii) **Modular Design**: facilitates flexible PDDL generation, allowing users to explore prompting styles and create customized pipelines.

(iii) **Autonomous Capability**: supports a fully autonomous pipeline, reducing the need for manual authoring.

Our primary motivation stems from the lack of standardized frameworks in the field of PDDL modelling with LLMs. Existing works often employ disparate approaches, making meaningful comparisons difficult. Our goal is to develop a cohesive Python library that supports users in creating PDDL models using LLMs. This library will be flexible enough for users to design their own architectures while adhering to a standardized format that ensures consistency and enables effective comparative studies across different frameworks.

Why PDDL? The scope of this library is limited to PDDL for various reasons: its standardization and widespread use in the planning community strongly facilitate sharing and

[1]L2P toolkit is available at https://github.com/AI-Planning/l2p.

benchmarking. This, in turn, allows a wide selection of tools to support validation and refinement of models. Additionally, due to its flexibility, clear syntax, and declarative nature, it aligns well with LLMs' capabilities to translate descriptions into PDDL constructs, as all modern LLMs have likely encountered PDDL code in their training corpus.

2 Language-to-Plan (L2P) Framework

A standout feature of this library is its flexibility, enabling users to seamlessly switch between different LLM engines for experimentation within their own frameworks. Users can incorporate their preferred methods from L2P for extracting PDDL model components, tailoring the process to their specific needs. Additionally, the library offers extensive customization options for prompts and assumptions, allowing users to adapt LLM interactions to suit their unique requirements.

2.1 Builder Classes

The Builder classes contain functions for extracting specific components found in PDDL usage. Specifically, it allows the user to prompt the LLM to generate the domain types, predicates, and actions—including their respective parameters, preconditions, and effects. L2P also supports task specification, which includes objects, initial, and goal states that correspond to a given domain. Inspired by (Guan et al. 2023) and (Gestrin, Kuhlmann, and Seipp 2024), L2P not only includes PDDL builder classes but also features a customized Feedback Builder class that incorporates both LLM-generated feedback and human input, or, optionally, a combination of both. This system refines the LLM's responses in cases of unsatisfactory results. Additionally, L2P comes with a custom syntax validation tool that checks for common Python or PDDL syntax errors, leveraging this feedback to further enhance the accuracy of the generated responses. Appendix C—Figure 8 showcases an LLM giving feedback to refine a PDDL problem specification.

2.2 Template Customization

LLMs have shown that their outputs are significantly sensitive to prompting—raising questions about whether they are better off functioning as machine *translators* or *generators*. Liu et al. (2023) demonstrate that highly explicit descriptions improve translation accuracy, while (Gestrin, Kuhlmann, and Seipp 2024) and (Smirnov et al. 2024) leverage minimal descriptions, relying on LLMs' internal world knowledge to enrich outputs. Given a structured formatting template for LLM output, L2P enables users to generate domain-agnostic environments with their own customized prompts, ensuring outputs can be properly and easily extracted and mapped into Python types. Specifically, users can swap out different roles and LLM prompt techniques (i.e. CoT) to solve the problem. Figure 1 demonstrates an example of how one might create a simple Blocksworld domain predicate. Further information can be found on the L2P documentation page.

```
[ROLE]: Define the PDDL predicates of an AI
    agent's actions. End your final answers
    starting with "### New Predicates" with
    ''' ''' comment blocks as so:

### New Predicates
```
- (predicate_name_1 ?t1 - type_1 ?t2 -
    type_2): 'predicate_description'
- (predicate_name_2 ?t3 - type_3 ?t4 -
    type_4): 'predicate_description'
```
------------------------------------------------
[TECHNIQUE]: CoT, few-shot, etc.
------------------------------------------------
[TASK]:
## Domain
{domain_desc}

## Natural Language Actions
{nl_actions}

## Types
{types}
```

Figure 1: Example core template structure to prompt LLM PDDL predicate generation.

3 Demonstration Overview

This demonstration overview showcases L2P usage and further details how users can curate these tools to create their own frameworks. Appendix C—Figure 7 is a complete example of L2P creating a PDDL task specification. The LLM output is found below (Figure 2).

```
### LLM OUTPUT (GPT-4o)
(define
   (problem blocksworld_problem_01)
   (:domain blocksworld_problem)
   (:objects
      blue_block - object
      red_block - object
      yellow_block - object
      green_block - object
   )
   (:init
      (on_top blue_block red_block)
      (on_top red_block yellow_block)
      (on_table yellow_block)
      (on_table green_block)
      (clear blue_block)
      (clear green_block)
      (empty arm)
   )
   (:goal
      (and
         (on_top red_block green_block)
      )
   )
)
```

Figure 2: L2P usage - generating simple PDDL task specification on simple Blocksworld domain

```
### LLM OUTPUT
My concrete suggestions are the following:
- Add the predicate to indicate that the red block is not clear:
  - (clear red_block) should be removed from the initial state since the red block is covered by the blue block.

Final output should reflect this change in the initial state:
```
(clear blue_block): blue block is clear
(clear yellow_block): yellow block is clear
(clear green_block): green block is clear
```
Overall, the feedback is: Yes, the initial state needs to reflect that the red block is not clear.
```

Figure 3: Example of feedback given by LLM on a PDDL problem specification on the Blocksworld domain.

3.1 Paper Reconstructions

L2P is capable of recreating and encompassing previous frameworks for converting natural language to PDDL, serving as a comprehensive foundation that integrates past approaches. The L2P GitHub contains multiple paper reconstructions, such as NL2Plan (Gestrin, Kuhlmann, and Seipp 2024) and LLM+P (Liu et al. 2023), as examples of how existing methods can be implemented and compared within a unified system, demonstrating L2P's flexibility and effectiveness in standardizing diverse NL-to-PDDL techniques. An example of Guan et al. (2023) "action-by-action" algorithm can be found in Figure 4; action and predicate LLM output can be found in Appendix C—Figure 9. We hope the community embraces the L2P framework as a repository of existing advancements in LLM model acquisition and relevant papers, ensuring that users have access to the most current research and tools under a common framework for fair comparison.

4 Collaborative Feature and Outlook

L2P automates the conversion of natural language into PDDL, offering researchers a unified framework to evaluate methods and foster rigorous experimentation and benchmarking. While the library currently supports only basic PDDL extraction tools for fully observable deterministic planning, it does not yet address conditional, temporal, or numeric planning. We aim to expand L2P's capabilities with the help of the Planning community, driving research into the challenges LLMs face in these areas. We invite the community to adopt this library, integrate it into their projects, and provide valuable feedback through issue reporting or feature suggestions. These contributions will refine and expand L2P, building a repertoire of standardized benchmarks and methodologies, enabling fair comparisons, and fostering the creation of high-impact research and applications.

```python
import os
from l2p import *

def run_aba_alg(model: LLM, action_model, domain_desc,
                hierarchy, prompt, max_iter: int=2
      ) -> tuple[list[Predicate], list[Action]]:

  actions = list(action_model.keys())
  pred_list = []

  for _ in range(max_iter):
    action_list = []
    # iterate each action spec. + new predicates
    for _, action in enumerate(actions):
      if len(pred_list) == 0:
        prompt = prompt.replace('{predicates}',
        '\nNo predicate has been defined yet')
      else:
        res = ""
        for i, p in enumerate(pred_list):
          res += f'\n{i + 1}. {p["raw"]}'
          prompt = prompt.replace('{predicates}', res)
      # extract pddl action and predicates (L2P)
      pddl_action, new_preds, response = (
        builder.extract_pddl_action(
          model=model,
          domain_desc=domain_desc,
          prompt_template=prompt,
          action_name=action,
          action_desc=action_model[action]['desc'],
          action_list=action_list,
          predicates=pred_list,
          types=hierarchy["hierarchy"]
        )
      )
      # format + add extracted actions and predicates
      new_preds = parse_new_predicates(response)
      pred_list.extend(new_preds)
      action_list.append(pddl_action)
    pred_list = prune_predicates(pred_list,action_list)

  return pred_list, action_list

if __name__ == "__main__":
  builder = DomainBuilder() # create domain build class
  # retrieve prompt information
  base_path='paper_reconstructions/llm+dm/prompts/'
  action_model=load_file(
      f'{base_path}action_model.json')
  domain_desc=load_file(
      f'{base_path}domain_desc.txt')
  hier=load_file(
      f'{base_path}hierarchy_requirements.json')
  prompt=load_file(f'{base_path}pddl_prompt.txt')

  # initialise LLM engine (OpenAI in this case)
  api_key = os.environ.get('OPENAI_API_KEY')
  llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

  # run "action-by-action" algorithm
  pred, action = run_aba_alg(model=llm, action_model,
      domain_desc, hier, prompt)
```

Figure 4: L2P code reconstruction of "action-by-action algorithm" (Guan et al. 2023). This iterates through each action description and generates its PDDL specifications while maintaining a dynamically generating predicate list.

References

Agarwal, S.; and Sreepathy, A. 2024. TIC: Translate-Infer-Compile for accurate 'text to plan' using LLMs and logical intermediate representations. *CoRR*, abs/2402.06608.

Bengio, Y. 2020. Deep Learning for System 2 Processing. *AAAI 2020*.

Birr, T.; Pohl, C.; Younes, A.; and Asfour, T. 2024. AutoGPT+P: Affordance-based Task Planning using Large Language Models. In *Robotics: Science and Systems XX*, RSS2024. Robotics: Science and Systems Foundation.

Chen, G.; Yang, L.; Jia, R.; Hu, Z.; Chen, Y.; Zhang, W.; Wang, W.; and Pan, J. 2024. Language-Augmented Symbolic Planner for Open-World Task Planning. *CoRR*, abs/2407.09792.

Collins, K. M.; Wong, C.; Feng, J.; Wei, M.; and Tenenbaum, J. B. 2022. Structured, flexible, and robust: benchmarking and improving large language models towards more human-like behavior in out-of-distribution reasoning tasks. *arXiv preprint arXiv:2205.05718*.

da Silva, L. M. V.; Köcher, A.; Gehlhoff, F.; and Fay, A. 2024. Toward a Method to Generate Capability Ontologies from Natural Language Descriptions. *CoRR*, abs/2406.07962.

Dagan, G.; Keller, F.; and Lascarides, A. 2023. Dynamic Planning with a LLM. *CoRR*, abs/2308.06391.

Daniel, K. 2017. *Thinking, fast and slow*. Macmillan.

Ding, Y.; Zhang, X.; Amiri, S.; Cao, N.; Yang, H.; Kaminski, A.; Esselink, C.; and Zhang, S. 2023. Integrating Action Knowledge and LLMs for Task Planning and Situation Handling in Open Worlds. *CoRR*, abs/2305.17590.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions. *CoRR*, abs/2405.04215.

Grover, S.; and Mohan, S. 2024. A Demonstration of Natural Language Understanding in Embodied Planning Agents. In *ICAPS 2024 System's Demonstration track*.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Han, M.; Zhu, Y.; Zhu, S.; Wu, Y. N.; and Zhu, Y. 2024. InterPreT: Interactive Predicate Learning from Language Feedback for Generalizable Task Planning. *CoRR*, abs/2405.19758.

Huang, J.; Chen, X.; Mishra, S.; Zheng, H. S.; Yu, A. W.; Song, X.; and Zhou, D. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Huang, S.; Lipovetzky, N.; and Cohn, T. 2024. Planning in the Dark: LLM-Symbolic Planning Pipeline without Experts. *arXiv preprint arXiv:2409.15915*.

Izquierdo-Badiola, S.; Canal, G.; Rizzo, C.; and Alenyà, G. 2024. PlanCollabNL: Leveraging Large Language Models for Adaptive Plan Generation in Human-Robot Collaboration. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, 17344–17350. IEEE.

Kelly, J.; Calderwood, A.; Wardrip-Fruin, N.; and Mateas, M. 2023. There and Back Again: Extracting Formal Domains for Controllable Neurosymbolic Story Authoring. In Eger, M.; and Cardona-Rivera, R. E., eds., *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, October 08-12, 2023, Salt Lake City, UT, USA*, 64–74. AAAI Press.

Lin, K.; Agia, C.; Migimatsu, T.; Pavone, M.; and Bohg, J. 2023. Text2Motion: from natural language instructions to feasible plans. *Auton. Robots*, 47(8): 1345–1365.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *CoRR*, abs/2304.11477.

Liu, W.; Nie, N.; Mao, J.; Zhang, R.; and Wu, J. 2024a. Learning Compositional Behaviors from Demonstration and Language. In *8th Annual Conference on Robot Learning*.

Liu, Y.; Palmieri, L.; Koch, S.; Georgievski, I.; and Aiello, M. 2024b. DELTA: Decomposed Efficient Long-Term Robot Task Planning using Large Language Models. *CoRR*, abs/2404.03275.

Liu, Y.; Palmieri, L.; Koch, S.; Georgievski, I.; and Aiello, M. 2024c. Towards Human Awareness in Robot Task Planning with Large Language Models. *CoRR*, abs/2404.11267.

Lyu, Q.; Havaldar, S.; Stein, A.; Zhang, L.; Rao, D.; Wong, E.; Apidianaki, M.; and Callison-Burch, C. 2023. Faithful Chain-of-Thought Reasoning. *CoRR*, abs/2301.13379.

Mahdavi, S.; Aoki, R.; Tang, K.; and Cao, Y. 2024. Leveraging Environment Interaction for Automated PDDL Generation and Planning with Large Language Models. *CoRR*, abs/2407.12979.

McDermott, D.; M. Ghallab, A. H.; Knoblock, C.; A. Ram, M. V.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

Momennejad, I.; Hasanbeig, H.; Frujeri, F. V.; Sharma, H.; Jojic, N.; Palangi, H.; Ness, R. O.; and Larson, J. 2023. Evaluating Cognitive Maps and Planning in Large Language Models with CogEval. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Oates, T.; Alford, R.; Johnson, S.; and Hall, C. 2024. Using Large Language Models to Extract Planning Knowledge from Common Vulnerabilities and Exposures.

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Srivastava, B.; Horesh, L.; Fabiano, F.; and Loreggia, A. 2023. Understanding the Capabilities of Large Language Models for Automated Planning. *CoRR*, abs/2305.16151.

Sakib, M. S.; and Sun, Y. 2024. Consolidating Trees of Robotic Plans Generated Using Large Language Models to Improve Reliability. *CoRR*, abs/2401.07868.

Singh, I.; Traum, D.; and Thomason, J. 2024. TwoStep: Multi-agent Task Planning using Classical Planners and Large Language Models. *CoRR*, abs/2403.17246.

Singh, S.; Swaminathan, K.; Arora, R.; Singh, R.; Datta, A.; Das, D.; Banerjee, S.; Sridharan, M.; and Krishna, K. M. 2024. Anticipate & Collab: Data-driven Task Anticipation and Knowledge-driven Planning for Human-robot Collaboration. *CoRR*, abs/2404.03587.

Smirnov, P.; Joublin, F.; Ceravola, A.; and Gienger, M. 2024. Generating consistent PDDL domains with Large Language Models. *CoRR*, abs/2404.07751.

Stechly, K.; Marquez, M.; and Kambhampati, S. 2023. GPT-4 Doesn't Know It's Wrong: An Analysis of Iterative Prompting for Reasoning Problems. *CoRR*, abs/2310.12397.

Stechly, K.; Valmeekam, K.; and Kambhampati, S. 2024. Chain of Thoughtlessness: An Analysis of CoT in Planning. *CoRR*, abs/2405.04776.

Valmeekam, K.; Marquez, M.; and Kambhampati, S. 2023. Can Large Language Models Really Improve by Self-critiquing Their Own Plans? *CoRR*, abs/2310.08118.

Valmeekam, K.; Stechly, K.; and Kambhampati, S. 2024. LLMs Still Can't Plan; Can LRMs? A Preliminary Evaluation of OpenAI's o1 on PlanBench. *arXiv preprint arXiv:2409.13373*.

Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; Chi, E. H.; Hashimoto, T.; Vinyals, O.; Liang, P.; Dean, J.; and Fedus, W. 2022. Emergent Abilities of Large Language Models. *CoRR*, abs/2206.07682.

Wong, L.; Mao, J.; Sharma, P.; Siegel, Z. S.; Feng, J.; Korneev, N.; Tenenbaum, J. B.; and Andreas, J. 2023. Learning adaptive planning representations with natural language guidance. *CoRR*, abs/2312.08566.

Xie, Y.; Yu, C.; Zhu, T.; Bai, J.; Gong, Z.; and Soh, H. 2023. Translating Natural Language to Planning Goals with Large-Language Models. *CoRR*, abs/2302.05128.

Ye, R.; Hu, Y.; Bian, Y. A.; Kulm, L.; and Bhattacharjee, T. 2024. MORPHeus: a Multimodal One-armed Robot-assisted Peeling System with Human Users In-the-loop. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, 9540–9547. IEEE.

Ying, L.; Collins, K. M.; Wei, M.; Zhang, C. E.; Zhi-Xuan, T.; Weller, A.; Tenenbaum, J. B.; and Wong, L. 2023. The Neuro-Symbolic Inverse Planning Engine (NIPE): Modeling Probabilistic Social Inferences from Linguistic Inputs. *CoRR*, abs/2306.14325.

Zhang, L.; Jansen, P.; Zhang, T.; Clark, P.; Callison-Burch, C.; and Tandon, N. 2024a. PDDLEGO: Iterative Planning in Textual Environments. *CoRR*, abs/2405.19793.

Zhang, T.; Zhang, L.; Hou, Z.; Wang, Z.; Gu, Y.; Clark, P.; Callison-Burch, C.; and Tandon, N. 2024b. PROC2PDDL: Open-Domain Planning Representations from Texts. *CoRR*, abs/2403.00092.

Zhang, X.; Qin, H.; Wang, F.; Dong, Y.; and Li, J. 2024c. LaMMA-P: Generalizable Multi-Agent Long-Horizon Task Allocation and Planning with LM-Driven PDDL Planner. *arXiv preprint arXiv:2409.20560*.

Zhou, Z.; Song, J.; Yao, K.; Shu, Z.; and Ma, L. 2023. ISR-LLM: Iterative Self-Refined Large Language Model for Long-Horizon Sequential Task Planning. *CoRR*, abs/2308.13724.

A Paper Overview

This section contains an overview of the core framework papers found within **Leveraging Large Language Models for Automated Planning and Model Construction: A Survey**. The table provides a structured distinction between the papers based on their methodological approaches and specific areas of focus, such as what aspect are they modelling, prompting style/weight, and feedback mechanism. Additionally, it highlights papers already covered by the L2P library, ensuring users have quick access to resources that have been implemented or analyzed in detail. These covered papers will be included in our GitHub repository for easy reference. The distinction in the table also aims to emphasize gaps in the literature, such as under explored techniques or novel applications of NL-PDDL through LLMs, paving the way for future research. To maintain relevance, we plan to keep an updated installment of all NL-PDDL works leveraging LLMs, ensuring this survey evolves alongside the field.

Framework	Task	Domain	NHI	Prompt	Feedback
(Collins et al. 2022)	✓	×	✓	Medium (Few-shot)	None
⋆(Xie et al. 2023)	✓	×	✓	Medium (Few-shot)	None
(Lyu et al. 2023)	✓	×	✓	Light (CoT)	None
(Grover and Mohan 2024)	✓	×	✓	Medium (Few-shot)	Env.
(Lin et al. 2023)	✓	×	✓	Medium (Few-shot)	LLM+Env.
⋆(Liu et al. 2023)	✓	×	✓	Heavy (Few-shot)	None
(Birr et al. 2024)	✓	×	✓	Light (CoT)	External tool
(Agarwal and Sreepathy 2024)	✓	×	×	Medium	None
⋆(Dagan, Keller, and Lascarides 2023)	✓	×	✓	Medium	LLM + Env.
(Zhang et al. 2024a)	✓	×	✓	Medium (Few-shot)	LLM
(Liu et al. 2024c)	✓	×	×	Extended Scene Graph	Env.
(Singh et al. 2024)	✓	×	×	Light	LLM+Deviation
(Izquierdo-Badiola et al. 2024)	✓	×	✓	Light	LLM
(Singh, Traum, and Thomason 2024)	✓	×	✓	Medium (One-shot+)	LLM Helper Agent
(Zhang et al. 2024c)	✓	×	✓	Light	Fast Downward + External Tool
(Oates et al. 2024)	×	✓	×	Medium (Few-shot, in-context)	Human
⋆(Zhang et al. 2024b)	×	✓	×	Medium (Few-shot)	None
⋆(Guan et al. 2023)	×	✓	×	Light	Human
(Huang, Lipovetzky, and Cohn 2024)	×	✓	✓	Light	Sentence Encoder Filter
(Wong et al. 2023)	×	✓	✓	Medium (Few-shot)	None
(Liu et al. 2024a)	×	✓	×	Heavy	Human
(Ding et al. 2023)	×	✓	✓	Light	LLM
(Chen et al. 2024)	×	✓	✓	Medium	Env.
(Kelly et al. 2023)	✓	✓	✓	Medium (One-shot+)	External tool
⋆(Smirnov et al. 2024)	✓	✓	✓	Light	LLM
(Liu et al. 2024b)	✓	✓	✓	Medium (One-shot+)	None
(Zhou et al. 2023)	✓	✓	✓	Heavy (Few-shot)	None
(Ye et al. 2024)	✓	✓	×	Heavy	Human
(Han et al. 2024)	✓	✓	×	Light	Human
⋆(Gestrin, Kuhlmann, and Seipp 2024)	✓	✓	✓	Medium (Few-shot)	LLM + Human
(da Silva et al. 2024)	✓	✓	✓	Light	LLM
⋆(Mahdavi et al. 2024)	✓	✓	✓	Light	LLM + Env.
(Sakib and Sun 2024)	✓	✓	✓	Light	None
(Ying et al. 2023)	✓	✓	✓	Medium (Few-shot)	Syntax rejection filter

Figure 5: Summary of frameworks found in **Leveraging Large Language Models for Automated Planning and Model Construction: A Survey**. *NHI* = No Human Intervention. ⋆Papers reconstructed by L2P

B Automated Planning Background – Additional Information

Automated Planning is a specialized field within AI and NLP that can be challenging for those unfamiliar with its principles. To make it more accessible, we provide background information to bridge the knowledge gap. A key tool in classical planning is the Planning Domain Definition Language (PDDL), which models planning problems and domains. We illustrate its concepts with the Blocksworld problem, where blocks must be stacked in a specific order using actions like picking up, unstacking, and placing blocks, all while respecting constraints like moving only one block at a time or not disturbing stacked blocks.

The given problem can be encoded in PDDL. Demonstrated below is the PDDL domain file:

```
(define (domain blocksworld)
    (:requirements :strips)
    (:predicates (clear ?x) (on-table ?x) (arm-empty) (holding ?x) (on ?x ?y))
    (:action putdown
        :parameters  (?ob)
        :precondition (holding ?ob)
        :effect (and (clear ?ob) (arm-empty) (on-table ?ob) (not (holding ?ob)))
    )
    (:action pickup
        :parameters (?ob)
        :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
        :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob)) (not
            (arm-empty)))
    )
    (:action stack
        :parameters  (?ob ?underob)
        :precondition (and (clear ?underob) (holding ?ob))
        :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob) (not (clear ?underob))
            (not (holding ?ob)))
    )
    (:action unstack
        :parameters  (?ob ?underob)
        :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
        :effect (and (holding ?ob) (clear ?underob) (not (on ?ob ?underob)) (not (clear
            ?ob)) (not (arm-empty)))))
```

Predicates define relationships or properties that can be true or false, such as `(on ?x ?y)` for block `?x` on `?y`, `(ontable ?x)` for `?x` on the table, `(clear ?x)` for `?x` having nothing on top, and `(holding ?x)` for the robot holding `?x`. **Actions** describe state changes. For instance, `(pick-up)` contains the *parameter(s)* block `?x`, *preconditions* requiring `(clear ?x)` and `(ontable ?x)`, and *effects* updating the state to reflect the robot holding `?x`, which is no longer on the table and clear.

The following is the PDDL problem/task file:

```
(define (problem blocksworld-problem)
   (:domain blocksworld)
   (:objects A B C) ; Blocks
   (:init (ontable A) (ontable B) (on C A) (clear B) (clear C)) ; Initial state
   (:goal (and (on A B) (on B C)))) ; Goal state
```

Objects represent the entities involved, such as blocks A, B, and C. The **initial state** defines the starting arrangement, where blocks A and B are on the table, block C is on A, and both B and C are clear. The **goal state** specifies the desired configuration, where block A is stacked on B, and block B is stacked on C.

Given the above PDDL domain and problem definitions, a classical planner might generate the following (optimal) plan:

```
Unstack C from A
Put C on table
Pick up A
Stack A on B
Pick up B
Stack B on C
```

C L2P Usage Example

Below are example usages using our L2P library. Full documentation can be found on our website.

```python
import os
from l2p import *

domain_builder = DomainBuilder() # initialize Domain Builder class

# REPLACE WITH OWN API KEY
api_key = os.environ.get('OPENAI_API_KEY')
llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

# retrieve prompt information
base_path='tests/usage/prompts/domain/'
domain_desc = load_file(f'{base_path}blocksworld_domain.txt')
extract_predicates_prompt = load_file(f'{base_path}extract_predicates.txt')
types = load_file(f'{base_path}types.json')
action = load_file(f'{base_path}action.json')

# extract predicates via LLM
predicates, llm_output = domain_builder.extract_predicates(
    model=llm,
    domain_desc=domain_desc,
    prompt_template=extract_predicates_prompt,
    types=types,
    nl_actions={action['action_name']: action['action_desc']}
    )

# format key info into PDDL strings
predicate_str = "\n".join([pred["clean"].replace(":", " ; ") for pred in predicates])

print(f"PDDL domain predicates:\n{predicate_str}")

------------------------------------------------------------------------------------

### OUTPUT
(holding ?a - arm ?b - block) ;  true if the arm ?a is holding the block ?b
(on_top ?b1 - block ?b2 - block) ;  true if the block ?b1 is on top of the block ?b2
(clear ?b - block) ;  true if the block ?b is clear (no block on top of it)
(on_table ?b - block) ;  true if the block ?b is on the table
(empty ?a - arm) ;  true if the arm ?a is empty (not holding any block)
```

Figure 6: L2P usage - generating simple PDDL predicates

```python
import os
from l2p import *

task_builder = TaskBuilder() # initialize Task Builder class

# REPLACE WITH OWN API KEY
api_key = os.environ.get('OPENAI_API_KEY')
llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

# load in assumptions
problem_desc = load_file(r'tests/usage/prompts/problem/blocksworld_problem.txt')
extract_task_prompt = load_file(r'tests/usage/prompts/problem/extract_task.txt')
types = load_file(r'tests/usage/prompts/domain/types.json')
predicates_json = load_file(r'tests/usage/prompts/domain/predicates.json')
predicates: List[Predicate] = [Predicate(**item) for item in predicates_json]

# extract PDDL task specifications via LLM
objects, initial_states, goal_states, llm_response = task_builder.extract_task(
    model=llm,
    problem_desc=problem_desc,
    prompt_template=extract_task_prompt,
    types=types,
    predicates=predicates
    )

# format key info into PDDL strings
objects_str = task_builder.format_objects(objects)
initial_str = task_builder.format_initial(initial_states)
goal_str = task_builder.format_goal(goal_states)

# generate task file
pddl_problem = task_builder.generate_task(
    domain="blocksworld",
    problem="blocksworld_problem",
    objects=objects_str,
    initial=initial_str,
    goal=goal_str)

print(f"### LLM OUTPUT:\n {pddl_problem}")
-----------------------------------------------------------------------------------------
### LLM OUTPUT
(define
   (problem blocksworld_problem)
   (:domain blocksworld)
   (:objects
      blue_block - object
      red_block - object
      yellow_block - object
      green_block - object
   )
   (:init
      (on_top blue_block red_block)
      (on_top red_block yellow_block)
      (on_table yellow_block)
      (on_table green_block)
      (clear blue_block)
      (clear yellow_block)
      (clear green_block)
   )
   (:goal
      (and
         (on_top red_block green_block)
      )
   )
)
```

Figure 7: L2P usage - generating simple PDDL task specification

```python
import os
from l2p import *

feedback_builder = FeedbackBuilder()

api_key = os.environ.get('OPENAI_API_KEY')
llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

problem_desc = load_file(r'tests/usage/prompts/problem/blocksworld_problem.txt')
types = load_file(r'tests/usage/prompts/domain/types.json')
feedback_template = load_file(r'tests/usage/prompts/problem/feedback.txt')
predicates_json = load_file(r'tests/usage/prompts/domain/predicates.json')
predicates: List[Predicate] = [Predicate(**item) for item in predicates_json]
llm_response = load_file(r'tests/usage/prompts/domain/llm_output_task.txt')

objects, initial, goal, feedback_response = feedback_builder.task_feedback(
    model=llm,
    problem_desc=problem_desc,
    feedback_template=feedback_template,
    feedback_type="llm",
    predicates=predicates,
    types=types,
    llm_response=llm_response)

print("FEEDBACK:\n", feedback_response)
--------------------------------------------------------------------------------
### LLM OUTPUT
1. Are any necessary objects missing?
   All necessary objects are included based on the problem description. So: No.

2. Are any unnecessary objects included?
   All objects included are relevant to the problem. Hence: No.

3. Are any objects defined with the wrong type?
   All objects are correctly defined as "object". Therefore: No.

4. Are any unnecessary or incorrect predicates declared?
   All predicates used in the initial state are relevant and correctly applied. Thus: No.

5. Are any needed or expected predicates missing from the initial state?
   The initial state is missing the predicate for the red block being clear. Since the red block is covered by
   the blue block, it should not be clear. Therefore: Yes.

6. Is anything missing from the goal state?
   The goal state accurately reflects the desired outcome of having the red block on top of the green block. So: No.

7. Is anything unnecessary included in the goal description?
   The goal description is concise and only includes what is necessary. Therefore: No.

8. Should any predicate be used in a symmetrical manner?
   The predicates used do not require symmetry as they are directional in nature. So: No.

My concrete suggestions are the following:
- Add the predicate to indicate that the red block is not clear:
  - (clear red_block) should be removed from the initial state since the red block is covered by the blue block.

Final output should reflect this change in the initial state:
```
(clear blue_block): blue block is clear
(clear yellow_block): yellow block is clear
(clear green_block): green block is clear
```
Overall, the feedback is: Yes, the initial state needs to reflect that the red block is not clear.
```

Figure 8: L2P usage - generating LLM-feedback on task specification

```
## PREDICATES
{'name': 'truck-at',
    'desc': 'true if the truck ?t is currently at location ?l',
    'raw': '(truck-at ?t - truck ?l - location): true if the truck ?t is currently at location ?l',
    'params': OrderedDict([('?t', 'truck'), ('?l', 'location')]),
    'clean': '(truck-at ?t - truck ?l - location): true if the truck ?t is currently at location ?l'}
{'name': 'package-at',
    'desc': 'true if the package ?p is currently at location ?l',
    'raw': '(package-at ?p - package ?l - location): true if the package ?p is currently at location ?l',
    'params': OrderedDict([('?p', 'package'), ('?l', 'location')]),
    'clean': '(package-at ?p - package ?l - location): true if the package ?p is currently at location ?l'}
{'name': 'truck-holding',
    'desc': 'true if the truck ?t is currently holding the package ?p',
    'raw': '(truck-holding ?t - truck ?p - package): true if the truck ?t is currently holding the package ?p',
    'params': OrderedDict([('?t', 'truck'), ('?p', 'package')]),
    'clean': '(truck-holding ?t - truck ?p - package): true if the truck ?t is currently holding the package ?p'}
{'name': 'truck-has-space',
    'desc': 'true if the truck ?t has space to load more packages',
    'raw': '(truck-has-space ?t - truck): true if the truck ?t has space to load more packages',
    'params': OrderedDict([('?t', 'truck')]),
    'clean': '(truck-has-space ?t - truck): true if the truck ?t has space to load more packages'}
{'name': 'plane-at',
    'desc': 'true if the airplane ?a is located at location ?l',
    'raw': '(plane-at ?a - plane ?l - location): true if the airplane ?a is located at location ?l',
    'params': OrderedDict([('?a', 'plane'), ('?l', 'location')]),
    'clean': '(plane-at ?a - plane ?l - location): true if the airplane ?a is located at location ?l'}
{'name': 'plane-holding',
    'desc': 'true if the airplane ?a is currently holding the package ?p',
    'raw': '(plane-holding ?a - plane ?p - package): true if the airplane ?a is currently holding package ?p',
    'params': OrderedDict([('?a', 'plane'), ('?p', 'package')]),
    'clean': '(plane-holding ?a - plane ?p - package): true if the airplane ?a is currently holding package ?p'}
{'name': 'connected-locations',
    'desc': 'true if location ?l1 is directly connected to location ?l2 in city ?c',
    'raw': '(connected-locations ?l1 - location ?l2 - location ?c - city): ?l1 is connected to ?l2 in city ?c',
    'params': OrderedDict([('?l1', 'location'), ('?l2', 'location'), ('?c', 'city')]),
    'clean': '(connected-locations ?l1 - location ?l2 - location ?c - city): ?l1 is connected to ?l2 in city ?c'}

## ACTIONS
{'name': 'load_truck', 'parameters': OrderedDict([('?p', 'package'), ('?t', 'truck'), ('?l', 'location')]),
    'preconditions': '(and\n    (truck-at ?t ?l)\n    (package-at ?p ?l)\n    (truck-has-space ?t)\n)',
    'effects': '(and\n    (not (package-at ?p ?l))\n    (truck-holding ?t ?p)\n)'}
{'name': 'unload_truck', 'parameters': OrderedDict([('?p', 'package'), ('?t', 'truck'), ('?l', 'location')]),
    'preconditions': '(and\n    (truck-at ?t ?l)\n    (truck-holding ?t ?p)\n)',
    'effects': '(and\n    (not (truck-holding ?t ?p))\n    (package-at ?p ?l)\n)'}
{'name': 'load_airplane', 'parameters': OrderedDict([('?p', 'package'), ('?a', 'plane')]),
    'preconditions': '(and\n    (package-at ?p ?l)\n    (plane-at ?a ?l)\n)',
    'effects': '(and\n    (not (package-at ?p ?l))\n    (plane-holding ?a ?p)\n)'}
{'name': 'unload_airplane', 'parameters': OrderedDict([('?p', 'package'), ('?a', 'plane'), ('?l', 'location')]),
    'preconditions': '(and\n    (plane-at ?a ?l)\n    (plane-holding ?a ?p)\n)',
    'effects': '(and\n    (not (plane-holding ?a ?p))\n    (package-at ?p ?l)\n)'}
{'name': 'drive_truck',
    'parameters': OrderedDict([('?t', 'truck'), ('?l1', 'location'), ('?l2', 'location'), ('?c', 'city')]),
    'preconditions': '(and\n    (truck-at ?t ?l1)\n    (connected-locations ?l1 ?l2 ?c)\n)',
    'effects': '(and\n    (not (truck-at ?t ?l1))\n    (truck-at ?t ?l2)\n)'}
{'name': 'fly_airplane',
    'parameters': OrderedDict([('?a', 'plane'), ('?l1', 'location'), ('?l2', 'location'), ('?c', 'city')]),
    'preconditions': '(and\n    (plane-at ?a ?l1)\n    (connected-locations ?l1 ?l2 ?c)\n)',
    'effects': '(and\n    (not (plane-at ?a ?l1))\n    (plane-at ?a ?l2)\n)'}
```

Figure 9: L2P formatted predicate and actions output from LLM (action-by-action algorithm) of Logistics domain.