Emergent Capabilities of LLMs: Meta-Evaluation of Planning and Reasoning in Large Language Models

Carlo Merola, Jaspinder Singh, Othmane Dardouri

{carlo.merola, jaspinder.singh, othmane.dardouri}@studio.unibo.it

July 2025

Abstract

Large Language Models (LLMs), such as LLaMA, have shown impressive general capabilities across natural language tasks, yet they struggle with structured, domain-specific planning and reasoning-intensive tasks. This project investigates whether LLaMA can solve deterministic planning problems using iterative, validator-guided prompting. A symbolic validator acts as a proxy for domain knowledge, evaluating plans for logical correctness and providing structured feedback to guide up to four rounds of plan refinement.

From this process, we construct a dataset of plan attempts, validation outcomes, and feedback messages, enabling the training of meta-networks for confidence estimation. We evaluate four models, including a FFN baseline using symbolic features and a RoBERTa-based classifier that incorporates both semantic and temporal information across iterations. The best-performing model achieves almost perfect predictive accuracy, highlighting the value of temporal dynamics in confidence estimation.

Our results show that structured feedback not only improves LLM planning quality but also supports risk-aware evaluation of model reliability. This approach represents a step toward scalable, trustworthy LLM-based reasoning systems in high-stakes domains.

1 Introduction

Large language models (LLMs) such as LLaMA 3.3 70B Instruct have recently shown strong emergent capabilities across a wide spectrum of natural language tasks. These include summarization, translation, arithmetic reasoning, and even code synthesis. However, one key area that remains underexplored is structured, domain-specific planning: the ability to generate sequences of actions that satisfy goal conditions under explicit constraints.

Planning tasks pose unique challenges for LLMs because they require consistency across steps, respect for domain rules, and goal-directedness—capabilities that do not naturally emerge from next-token prediction alone. This project investigates the extent to which an LLM can perform such planning tasks and how its performance can be improved through structured feedback.

Our contributions are threefold:

1. We construct a benchmark suite of planning tasks and use prompt engineering combined with a domain-aware validator to evaluate the Llama model's planning abilities.

2. We implement a feedback-guided refinement loop in which the LLM iteratively corrects its plans based on validator-generated messages.

3. We build and analyze a dataset from these interactions and train meta-networks to predict the likelihood that a given problem will be solved, enabling a form of confidence estimation.

2 Related Work

Recent work has highlighted the potential of large language models (LLMs) in reasoning-intensive tasks, including logical inference, code generation, and program repair. Prompt engineering has been widely used to guide models toward better performance in such structured tasks. Techniques such as chain-of-thought prompting and few-shot examples have demonstrated that proper prompting can substantially boost performance without the need for additional fine-tuning [8, 5].

Beyond prompting, several studies have explored feedback loops and self-refinement strategies, in which the model critiques, evaluates, or re-generates its outputs based on internal signals or auxiliary prompts [6, 7]. These iterative mechanisms have been shown to improve consistency and reduce hallucinations. Separately, research on model calibration and confidence estimation has investigated ways to quantify the reliability of LLM outputs—ranging from temperature scaling and entropy-based metrics to dedicated uncertainty-aware meta-models—particularly in high-stakes or safety-critical scenarios [1, 2].

Regarding planning capabilities, some recent work frames LLM reasoning as a form of planning over latent representations or simulated environments. For instance, [3] proposes using LLMs as both agents and world models within a Monte Carlo Tree Search (MCTS) planning loop, while [4] argues that LLMs are not inherently planners but can aid planning within structured symbolic-verifier frameworks.

However, the intersection of feedback-driven LLM planning and external confidence estimation remains underexplored. Most current approaches either focus on planning via prompt-based decomposition or on estimating confidence post-hoc, without integrating both into a unified architecture. This project builds upon these threads by proposing a structured, validator-driven planning

loop that combines generation, evaluation, and confidence estimation to improve robustness and controllability in reasoning-intensive tasks.

3 Prompting Strategies and Experiments

Several prompting schemes were tested to find the optimal balance between expressivity and syntactic precision. Notably, we experimented with the following approaches:

- **Chain-of-Thought Augmentation:** Using a reasoning-based prefix (Let's reason step by step...) via the cot_prompt() function, we encouraged the model to explicitly deliberate over the planning sequence. However, this strategy underperformed, yielding only 2% valid plans, as measured by validator approval (model_outputs_3).
- **Interactive One-Step Planning Feedback:** With planner_validator_prompt(), we guided the model to output a single action at a time, reacting to "Valid"/"Wrong" feedback. While conceptually sound, this approach was not pursued in final evaluations due to insufficient control over full plan synthesis.
- **Structured PDDL Prompts:** We tested multiple system prompts designed to explicitly communicate PDDL syntax constraints. These included: A minimal domain/task overview with no system prompt (model_outputs_2, 10% valid plans). A detailed, strict syntax-aware system prompt for PDDL planning (model_outputs_0, 2% valid plans). Iteratively updated and simplified versions of the system prompt (model_outputs_1 and model_outputs_4, 6% and 10% respectively). A final refined system prompt that combined syntactic clarity with pragmatic instruction (model_outputs_5, 14% valid plans)—the best-performing setup. Ultimately, the final system prompt was selected based on empirical performance, measured by the percentage of validator-approved plans across all tested scenarios. Interestingly, minimal prompting (or none at all) outperformed some of the more verbose, syntactically constrained formulations, highlighting that overly strict instruction can inhibit model flexibility. In the Appendix it's reported the final prompt combination chosen.

Prompt engineering was not limited to system-level instructions. Initially, we employed dynamic prompt revision strategies using the backprompt() function, which injected error-specific validator feedback into the user prompt after each failed plan. However, this approach was later replaced with a more robust method: validator feedback was integrated directly into the LLM message flow, with the model's plan responses formatted as system messages and the validator's feedback as user messages. This more natural conversational structure allowed the LLM to better align with the iterative correction process and respond to validation failures more effectively.

4 Iterative Feedback-Guided Planning

4.1 Introduction and Motivation

In complex domains like the Blocks World, accurate planning requires strict adherence to action preconditions, state transitions, and goal satisfaction. Large Language Models (LLMs), such as LLaMA 3.3 70B Instruct, can generate syntactically plausible plans in a single pass. However, these plans often contain subtle errors, such as applying actions in invalid states or omitting necessary preconditions.

This section presents a feedback-driven planning loop that augments a base LLM with an external symbolic validator. By iteratively refining plans based on validator feedback, the system simulates a learning-like behavior—improving output quality without modifying the model's weights or parameters.

4.2 System Architecture

Our iterative pipeline comprises three main components:

- **Planner (LLM):** LlaMA 3.3 70B Instruct is used to generate plans based on a PDDL domain and problem description. We selected the Instruct variant due to its improved alignment with human instructions, making it more responsive to structured prompts and validator feedback in multi-turn interactions.
- **Validator:** A symbolic validator (VAL) checks whether a generated plan conforms to the domain and satisfies the goal.
- **Controller:** Manages the interaction loop, injecting validator feedback into the prompt and triggering new plan generations.

At each iteration, the LLM receives updated context from the validator feedback, enabling it to improve upon previous attempts.

Prompting Strategy

- **Initial Prompt:** Contains the full PDDL domain and problem, instructing the LLM to generate a valid plan.
- **Follow-Up Prompts:** Include the original problem plus feedback extracted from VAL in natural language, specifying exactly where and why a plan failed.

Each new plan is generated from scratch, guided by validator feedback, allowing the LLM to construct a revised plan informed by prior errors.

4.3 Iteration Process

The feedback loop is limited to a maximum of four iterations. Each round includes:

1. **Plan Generation:** The LLM proposes a complete plan.
2. **Validation:** VAL checks the plan against PDDL constraints.
3. **Feedback Extraction:** If the plan is invalid, structured error messages are parsed from the validator's output. Common patterns such as "No matching action", "Plan failed", "Bad plan", and "Plan invalid" are extracted and distilled into concise natural language feedback. An example of the validator's feedback is the following:

   > "Action `pickup block1` is not applicable in state `holding block2`."
   > "Precondition `clear block1` not satisfied at step 3."

4. **Prompt Revision:** The feedback is reformulated into a user-facing prompt—e.g., "The plan you provided is invalid. Plan failed due to precondition violation at step 3. Please provide a corrected plan."—and prepended to the next input to guide regeneration.

If the validator confirms the plan is valid and achieves the goal, the loop is terminated early.

4.4 Feedback and Error Types

The validator provides correctness-based feedback focusing on:

- **Precondition Violations:** An action is invoked when its preconditions are not satisfied.
- **Infeasible Sequences:** Actions are sequenced in a way that makes intermediate states unreachable.
- **Redundant or Missing Steps:** These may cause failures to achieve the goal or waste actions.
- **Syntax or Action Mismatches:** The plan references undefined or malformed actions not supported in the domain.

4.5 Qualitative Examples

Example 1: Successful Repair

Initial Plan:

```
1. (pickup block1)
2. (stack block1 block2)
```

Feedback: "Precondition `clear block2` not satisfied at step 2."

Revised Plan:

```
1. (unstack block3 block2)
2. (putdown block3)
3. (pickup block1)
4. (stack block1 block2)
```

Example 2: Unresolved Case

In some problems, even after four iterations, the model fails to produce a valid plan—often due to misunderstandings of object constraints or overly complex goal conditions. These failures highlight limitations in the LLM's grounding or in the quality of feedback incorporation.

5 Dataset Construction

To evaluate and understand the iterative planning behavior of the LLM, we systematically collected data from each planning attempt across all problems and iterations. The resulting dataset captures the evolution of plan quality, validation outcomes, and response to feedback over time.

5.1 Data Collection Process

For each planning problem, we allowed the system up to four iterations. At each iteration, the following elements were generated and logged:

- The full plan generated by the LLM.
- The corresponding validation output from the VAL validator.
- The feedback extracted and embedded into the next prompt.
- The final success/failure outcome for that plan.

These artifacts were parsed to extract structured metrics that reflect both the correctness and quality of the generated plans.

5.2 Feature Set

Each row in the dataset corresponds to a single plan attempt (i.e., one LLM generation in one iteration for a given problem). For each entry, we recorded the following features:

- **Percentage of Valid Actions:** The fraction of actions in the plan that are valid according to the VAL validator. This captures partial correctness even in invalid plans.

- **Number of Consecutive Valid Steps:** The number of valid steps from the beginning of the plan until the first error occurs. This metric serves as a proxy for planning stability.
- **Plan Length:** Total number of actions generated in the plan. This helps analyze verbosity and potential over-planning.
- **Number of Logical Violations:** The number of precondition failures or domain rule violations reported by the validator. This reflects the degree of incorrectness.
- **Problem Name:** The name or identifier of the planning problem instance. Useful for grouping results by difficulty or domain.
- **Iteration Index:** The index of the current iteration (ranging from 1 to 4). This enables tracking how plan quality changes over successive iterations.
- **Plan Success (Binary):** A boolean value indicating whether the plan successfully reaches the goal and passes validation.

5.3 Dataset Size and Scope

The dataset contains entries from all planning problems tested during the evaluation. For each problem, we collected up to four plan attempts, resulting in a maximum of four data points per problem. Problems that were solved early (e.g., in the first or second iteration) have fewer associated rows, as no further iterations were executed.

5.4 Limitations

While the dataset provides a detailed view of planning behavior, it is constrained by the expressiveness of the VAL validator and the structure of PDDL domains. Plans that are semantically correct but syntactically invalid (or vice versa) are treated uniformly as failures. Additionally, because all plans are generated from scratch per iteration, there is no explicit tracking of local plan edits or deltas.

Despite these limitations, the dataset offers a rich foundation for understanding how LLMs perform under iterative symbolic supervision and can inform future work on planning robustness and repairability.

6 Meta-Networks for Confidence Estimation

To estimate the probability that the LLM will successfully solve a given planning task, we deployed four distinct meta-networks. Each network was designed with a different input representation, varying in granularity, feature usage, and modeling approach. All meta-networks were evaluated on an external test set composed of a wide range of planning problems across multiple domains, including varied goal specifications and domain rules.

6.1 Meta-Network Architectures

Meta-Network 1: Feature-Based on Final Generation The first meta-network is a small FFN model trained on symbolic features extracted solely from the last generation of the LLM (i.e., after the final backprompting iteration). The feature vector includes:

- **Valid Action Percent:** Fraction of actions deemed valid by the validator.
- **Consecutive Valid Steps:** Number of valid steps before the first violation.
- **Logical Violations:** Total number of precondition or domain rule violations.
- **Plan Length:** Number of actions in the generated plan.

This model serves as a baseline, capturing only the static final planning outcome.

Meta-Network 2: RoBERTa + MLP (Prompt + Plan Embedding) The second meta-network uses a RoBERTa encoder to embed the tokenized concatenation of the problem description and the generated plan from the last iteration. This representation is passed to a multilayer perceptron (MLP) to predict the binary success label. This setup captures the full contextual semantics of the planning task and the LLM's response.

Meta-Network 3: RoBERTa + MLP + Explicit Features This variant extends Meta-Network 2 by appending the explicit features from Meta-Network 1 (valid percent, violations, etc.) to the RoBERTa embedding. The resulting feature vector is fed to the MLP, enabling the model to combine symbolic structure with semantic representation for better prediction.

Meta-Network 4: Full Iteration Feature-Based The fourth meta-network uses a small FFN classifier that operates on the full set of handcrafted features extracted from *every iteration* of the LLM's planning process. That is, for each planning task, all intermediate plans generated across up to four iterations are included, each with:

- Valid Action Percent
- Consecutive Valid Steps
- Logical Violations
- Plan Length

Rather than aggregating statistics, this model uses the raw features from all iterations as input, allowing it to learn from temporal dynamics, stagnation, or correction patterns during the feedback loop.

6.2 Evaluation on External Test Set

All meta-networks were evaluated on a held-out external test set, which includes planning problems from diverse domains and varying complexity levels. Performance was assessed using the F1-score, with additional metrics such as precision and recall computed during analysis. The goal is to estimate the model's confidence in plan validity and flag uncertain or failure-prone instances. In next section we will see and discuss results and key findings.

7 Results and Analysis

Our experiments demonstrate that the iterative prompting approach, combined with validator feedback, substantially improves plan validity by correcting errors such as violating preconditions or repeating actions. The iterative refinement is most effective when early feedback is properly integrated by the LLM, preventing failure propagation from early mistakes.

We evaluated four meta-network models designed to estimate the confidence that an LLM-generated plan will successfully solve a given planning task. These models varied in input representation, ranging from handcrafted symbolic features extracted from final or multiple iterations to semantic embeddings generated by a RoBERTa encoder.

7.1 Key Findings

- The logistic regression model based on symbolic features extracted *only from the final plan generation* (Meta-Network 1) provided a reasonable baseline performance.
- The RoBERTa-based meta-networks (Meta-Network 2 and 3), which utilized token embeddings of the concatenated problem and final plan, did **not** achieve superior results. This underperformance is likely due to the limited size and diversity of our dataset, which was insufficient to effectively train a complex transformer model for confidence estimation.
- The best performance was consistently achieved by the FFN model that leveraged *features extracted from all iterations* of the planning process (Meta-Network 4). By incorporating temporal dynamics such as stagnation or correction patterns, this model was better able to capture failure modes that evolve over multiple iterations.

The handcrafted features used across models included:

- **Valid Action Percent:** The fraction of actions in the plan deemed valid by the validator.
- **Consecutive Valid Steps:** The count of valid steps before the first detected violation.

- **Logical Violations:** The total number of precondition or domain rule violations identified.
- **Plan Length:** The number of actions in the generated plan.

7.2 Performance Summary

Model ID	Meta-Network Type	Input Features / Representation	F1 Score
MN-1	FFN (Final Only)	Valid%, Consecutive Valid Steps, Violations, Plan Length	0.53
MN-2	RoBERTa + MLP	Token Embedding of Problem + Final Plan	0.71
MN-3	RoBERTa + MLP + Features	Token Embedding + Explicit Final Plan Features	0.71
MN-4	FNN (All Iterations)	Full feature set from each iteration (no aggregation)	**1.0**

Table 1: F1-score results of the four meta-networks evaluated on the external test set. The model using features from all iterations (MN-4) achieved the highest performance.

7.3 Results Analysis

The superior performance of MN-4 confirms that temporal patterns and correction dynamics across multiple iterations are critical signals for estimating plan success. In contrast, relying solely on the final plan's features or semantic embeddings without sufficient training data limits the model's ability to generalize and capture subtle failure modes.

The RoBERTa-based meta-networks underperformed, likely because the dataset size was insufficient to leverage the full representational power of transformer embeddings. This suggests that future work should focus on larger datasets or augmentations to enable effective semantic modeling or alternatively prioritize carefully engineered features reflecting domain knowledge.

In practical terms, our results advocate for confidence estimation frameworks that integrate iterative feedback and leverage interpretable, symbolic features extracted throughout the plan refinement process rather than solely end-stage representations. This approach offers a pragmatic and effective path toward robust LLM-based planning systems with reliable failure prediction capabilities.

To further evaluate the behavior of MN-4, we measured its predicted confidence (after applying the sigmoid to the output logit) on previously unseen problems. These problems were grouped by type (as derived from the `folder` field), and Table 7.3 reports the average confidence for each group. Notably, MN-4 expresses high certainty on deterministic, structurally constrained domains (e.g., `gripper`, `shoe-sock`), while remaining appropriately uncertain on ambiguous, complex, or deceptive domains (e.g., `blocksworld 9 blocks`, `labyrinth`, `hanoi`).

Problem Type	Avg. Confidence
blocksworld 4 blocks	0.4111
blocksworld 5 blocks	0.2849
blocksworld 6 blocks	0.1840
blocksworld 7 blocks	0.0990
blocksworld 8 blocks	0.1019
blocksworld 9 blocks	0.0509
blocksworld 10 blocks	0.0772
basic_move	0.7115
gripper	0.9989
hanoi	0.3038
labyrinth	0.0601
logistics	0.1137
monkey	0.2821
mshoe-sock	0.9988
travel	0.9955

Table 2: Average MN-4 confidence scores by problem type (aggregated over test instances)

8 Conclusions and Future Work

8.1 Summary

This study demonstrates the effectiveness of validator-guided feedback in enhancing the reasoning capabilities of large language models (LLMs), particularly within structured symbolic planning tasks. By iteratively refining its plans based on feedback, LLaMA 3.3 70B Instruct improves its output quality without requiring fine-tuning. Crucially, the **validator serves as a proxy for external domain knowledge**, providing structured, symbolic supervision that guides the LLM toward more logically consistent solutions.

A central contribution of this work is the construction of a comprehensive dataset capturing plan outputs, validation results, and iteration-level feedback. This enabled the training of meta-networks to estimate plan success likelihood, providing external **confidence** estimates. The highest-performing meta-model, leveraging temporal features across multiple iterations, achieved perfect predictive performance (F1-score = 1.0), highlighting the importance of capturing the model's correction dynamics over time.

8.2 Findings from Prompt Engineering

Our experiments also revealed that prompt design is critical to effective LLM planning. Notably, we found that verbose or overly rigid system prompts—despite being syntactically correct—often degraded model performance. In contrast, more concise and pragmatically oriented prompts performed better. For in-

stance, minimal prompting achieved a higher plan success rate (14%) compared to more complex syntax-aware prompts (as low as 2%).

These results suggest that **overly complex system prompts may obscure the problem context and hinder the model's ability to focus on the planning task**. Effective prompt engineering should prioritize clarity, task relevance, and model-aligned language over strict formalism.

8.3 Model Choice and Limitations

While LLaMA 3.3 70B Instruct was used in this study, its performance is bounded by its architectural limitations and inference latency. Model choice significantly affects task performance, and newer models may offer improved reasoning capacity and response times.

8.4 Implications

This work shows that validator-guided feedback, acting as a form of external domain knowledge, enables LLMs to generate more robust and logically consistent plans. Additionally, external meta-models can predict output validity without retraining the LLM itself. This modular architecture—decoupling plan generation from validation and reliability assessment—offers a scalable approach to **trustworthy** LLM-based reasoning.

8.5 Future Work

We outline several directions to extend this research:

- **Fine-tuning LLMs** using validator-generated feedback to encode domain-specific rules directly into the model.
- **Expanding the validator framework** to support a wider range of tasks.
- **Training meta-networks** that assess partial correctness and identify uncertainty in individual plan components.
- **Scaling the dataset** with more diverse planning problems to improve generalization in meta-models.
- **Exploring hybrid prompt strategies** that combine minimal instructions with dynamic feedback integration.

8.6 Conclusion

Validator-guided planning improves both plan accuracy and **transparency** in LLM reasoning, with the validator serving as a critical stand-in for **external domain** expertise. Our results also emphasize the importance of empirically

tuned prompts—avoiding unnecessary complexity to maximize model effectiveness. As LLMs continue to evolve, architectures that integrate symbolic supervision and modular confidence estimation will be essential for deploying these models in **risk-aware** conditions, particularly within high-stakes, logic-bound environments.

Acknowledgment

We would like to acknowledge the following resources and contributions that supported the development of this work:

- GPT web search for assisting in finding research papers and technical documentation online.
- Thesis work of *Antonio Pagnotta* titled *"Llama 2 as A Next-Generation partial order programming tool: Performance, benchmarks and implications for autonomous systems"*.
- Slides and teaching materials from Prof.ssa *Roberta Calegari*, Prof. *Giovanni Sartor*, and Prof.ssa *Francesca Lagioia*.
- Grammarly, for assistance in syntax and grammar correction throughout the writing process.

References

[1] Shrey Desai and Greg Durrett. Calibration of pre-trained transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 295–302. Association for Computational Linguistics, 2020.

[2] Jiahui Geng, Fengyu Cai, Yuxia Wang, Heinz Koeppl, Preslav Nakov, and Iryna Gurevych. A survey of confidence estimation and calibration in large language models. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 6577–6595. Association for Computational Linguistics, 2024.

[3] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 8154–8173. Association for Computational Linguistics, 2023.

[4] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[5] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[6] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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*, 2023.

[7] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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*, 2023.

[8] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

System Prompt PDDL

System:

```
You are an expert PDDL planner.
Always output only valid PDDL domain and problem
↪  definitions.
Follow strict PDDL syntax for `:domain`, `:requirements`,
↪  `:types`, `:predicates`, `:actions`, `:objects`,
↪  `:init`, `:goal` and other PDDL predicates.
A PDDL model has two parts: a DOMAIN describing the schema
↪  and a PROBLEM describing a specific instance.
Maintain a current state as you select actions.
Only perform actions whose preconditions are satisfied.
Your output should be a coherent and valid sequence of
↪  actions that leads from the initial state `:init` to
↪  the goal state `:goal`.
Do not include any narrative or explanation|output only
↪  plan.
```

Figure 1: Prompt template to instruct the LLM to generate valid PDDL plans.

User Prompt PDDL

User:

```
Problem description: you are an agent capable of
↪  solving the planning problems  described using the
↪  PDDL syntax.
In the following problem the agent can perform some
↪  actions (described later) to achieve a specific
↪  goal.

=== DOMAIN DEFINITION ===
{domanin}

=== PROBLEM DEFINITION ===
{problem}

Can you BUILD A PLAN?
Do not include any narrative or explanation|output only
↪  the chosen actions for the PLAN.
The actions should have no ':action' prefix since it is
↪  in the domain definition and should not be
↪  outputted in the plan text.
```

Figure 2: Prompt template to build the problem.