GenPlanX. Integrating LLMs and Classical AI for Generation of Plans and Execution

Daniel Borrajo* **Giuseppe Canonaco** **Tomás de la Rosa** **Alfredo Garrachón**

Sriram Gopalakrishnan **Simerjot Kaur** **Marianela Morales** **Sunandita Patra**

Alberto Pozanco **Keshav Ramani** **Charese Smiley** **Pietro Totis** **Manuela Veloso**

Abstract

AI Planning techniques generate sequences of actions for complex tasks. However, they lack the ability to understand planning tasks when provided using natural language. Now, Large Language Models (LLMs) provide novel capabilities in human-computer interaction. In the context of planning tasks, LLMs have shown to be particularly good in interpreting human intents among other uses. This paper introduces GENPLANX that integrates LLMs for natural language-based description of planning tasks, with a classical AI planning engine, alongside an execution and monitoring framework. We demonstrate the efficacy of GENPLANX in assisting users with office-related tasks, highlighting its potential to streamline workflows and enhance productivity through seamless human-AI collaboration.

1 Introduction

The rapid advancement of AI has led to the development of techniques capable of understanding and executing complex tasks. Among these, Large Language Models (LLMs) have emerged as a powerful tool for interpreting natural language, enabling machines to comprehend and respond to human requests with remarkable accuracy [1]. However, the challenge remains in translating these requests into valid (and ideally optimal) plans that can be executed in real-world environments.

In particular, we are interested in planning problems that involve the integration of standard office-related tasks, such as handling emails/calendars, managing presentations or databases, connecting to company APIs, or even running machine learning tasks. One of the pioneering efforts in this domain is the development of softbots, as introduced by Etizioni et al. [2]. These Softbots are software agents that perform tasks by interacting with software environments. The recent development of OpenAI Agent[2] shows that this is the future of office assistants.

The current preferred strategy for tackling these tasks involves using LLMs to create office co-pilots that generate plans in various formats [3]. While these tools are powerful, they require a carefully designed LLM pipeline due to the lack of guarantees regarding the accuracy or optimality of the solutions. On the other hand, classical AI planning offers reliable methods for creating action sequences to achieve goals from initial states within a specified domain model [4]. Although effective in real-world scenarios like logistics [5], satellite/rover control [6], elevator management [7], and tourist planning [8], they are unable to process natural language task descriptions, which is a requirement for current users. In this paper, we present the GENPLANX system, GENeration of PLANs and eXecution, which is designed to receive requests in natural language about office-related tasks, generate plans to achieve the users intents, execute the actions in the generated plans,

*All authors are affiliated to JPMorganChase.

[2] https://openai.com/index/introducing-chatgpt-agent/

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

continuously monitor for successful execution, and replan in case of failed execution. Given a specific application in planning, the domain model is fixed and defined by humans. So, the users specify a planning task by providing the problem description. The problem is composed of a set of objects, an initial state and a set of goals (also called intents). Furthermore, GENPLANX is designed to allow seamless integration of new tools, making it adaptable to new applications. This is done by adding new actions to the PDDL domain description as well as their python code counterpart as explained later. By integrating LLMs with classical planning techniques, we can create techniques that not only understand natural language requests, but also generate and execute plans that utilize various tools and databases to address these requests effectively. This paper will review the related work, describe GENPLANX components, and report our experimental evaluation where GENPLANX exhibit good results compared to pure LLM-based approaches.

2 Related Work

The integration of planning algorithms with LLMs has attracted considerable interest as researchers aim to improve AI systems' ability to understand and execute complex tasks. This section reviews key contributions in this area, focusing on advancements and challenges in merging these technologies. The first subsection discusses works that rely entirely on LLMs for planning. The second subsection explores hybrid approaches that combine LLMs with planning. The third subsection examines planning approaches within the software domain.

2.1 Planning using LLMs

Recent advancements in prompting strategies have enabled LLMs to orchestrate multi-step reasoning internally. Chain-of-Thought (CoT) prompting [9] instructs models to generate intermediate reasoning steps, decomposing complex tasks into simpler sub-problems. Building upon CoT, Tree-of-Thought (ToT) [10] explores multiple reasoning paths concurrently before selecting the most promising sequence of actions. More complex approaches include ReAct [11], which interleaves reasoning with concrete actions in an interactive feedback loop and ADaPT [12], which focuses on adaptive decomposition by dynamically breaking down tasks based on context. These methods represent a trend toward LLM-based planning using internal guiding mechanisms rather than external planners. While these techniques can handle open-ended planning tasks without explicit domain models, they lack guarantees regarding soundness or optimality that classical AI planning approaches provide. In contrast, GENPLANX does not use LLMs for plan generation but relies on a classical AI planner, ensuring these formal guarantees.

2.2 LLM and Automated Planning

Initial attempts to use LLMs for planning tasks through direct prompting highlighted significant limitations [13, 14], demonstrating that LLMs alone struggle to generate valid plans when evaluated against standard planning benchmarks.This has led to hybrid approaches combining LLMs with external planners or validators [15, 16], which aligns with our work in developing GENPLANX, where we leverage the strengths of both LLMs and classical planning techniques.

In these hybrid frameworks, LLMs typically parse input requests and generate structured representation of the intent, while dedicated planners handle the actual problem-solving. Several approaches use LLMs to generate PDDL (Planning Domain Description Language [17]) problems and domains [18, 19, 20, 21]. GENPLANX differs by having LLMs generate JSON outputs rather than PDDL, as this format is easier for LLMs to produce and can represent objects beyond symbols, such as file paths or email addresses.

Other work has focused on integrating structured domain knowledge with neural architectures [22, 23], showing that embedding explicit domain rules within transformer models improves mapping accuracy from natural language to formal planning constructs. Similarly, GENPLANX aligns user language and intents to formal planning representations, enhancing plan robustness and providing guarantees.

Several frameworks facilitate this integration of LLMs and planning algorithms. Huang et al. [24] presented a framework where LLMs generate high-level plans refined by classical planning algorithms. Liu et al. [19] introduced LLM+P, in which an LLM translates a natural-language task into PDDL specifications before invoking a classical planner. Finally, the plan is translated back to a readable

form. These approaches underscore the complementary strengths of LLMs in understanding language and planning algorithms in optimizing task execution. Mottaghi et al. [25] created a framework for integrating LLMs with planning in interactive environments. Singh et al. [26] proposed TwoStep for multi-agent planning, where LLMs decompose goals into independent sub-goals while symbolic planners handle sub-problems optimally. Benyamin *et al.* [27] use LLMs to orchestrate the different planning steps. These approaches and GENPLANX, leverage the complementary strengths of LLMs in language understanding and planning algorithms in optimizing task execution.

2.3 Softbots and Planning

Most real-world applications of AI planning technology are related to control of physical systems, such as robots [6] or satellites [28, 29]. Significantly less work has focused on software actions. Early work [2] introduced the concept of softbots (software bots), which utilized planning to interact with software environments. Among other works that have used AI planning for software tasks –where actions are functions/commands to be executed in a computer– we can mention web service composition [30, 31], business workflows generation [32], networks [33] or machine learning workflows[34].

3 Architecture

The GENPLANX architecture is designed to integrate natural language processing with classical planning for automating complex tasks. It comprises several components that process requests and execute plans, as shown in Figure 1. A user request includes (1) Entities: relevant objects such as file names, slide titles, or calendar appointments); (2) Initial state: The task's starting configuration, like a database in a CSV file or text provided for translation; and (3) Goals: The desired outcomes, such as generating a PowerPoint or creating a decision tree from CSV data. Since users describe tasks in natural language without needing to understand AI planning elements, an LLM-based component can map English task descriptions into formal planning tasks, which then can be solved with an automated planner. The following sections will detail each component in the pipeline.

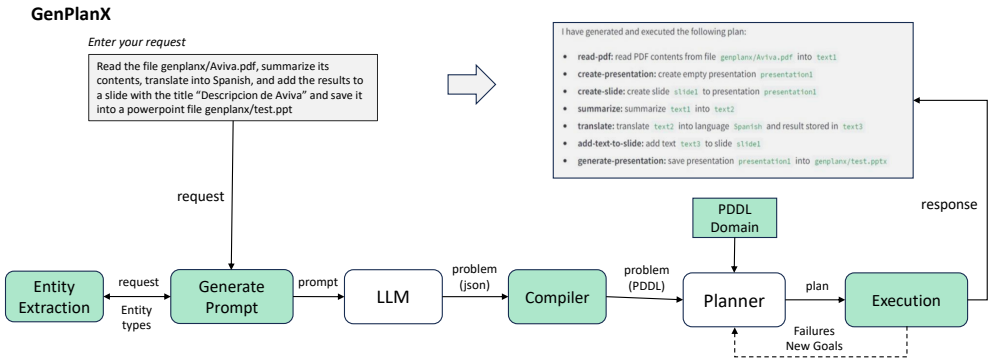

Figure 1: Architecture of GENPLANX. Green boxes are the unique implementations for our approach and contributions of this paper. White boxes are components integrated from existing AI tools.

4 Entity Extraction

The entity extraction module in GENPLANX builds on the capabilities described in [35], which focused on an email handling solution. We integrated this to improve prompts with domain-specific entities not covered in LLM training. This is especially important in the finance sector, where extracted entities include unique identifiers (for firms, clients), security IDs (CUSIP, SEDOL, ISIN), trade details (amount, currency, dates), portfolio IDs, and account numbers, among others. Utilizing an ensemble approach that combines deep learning, pattern-based techniques, and domain expertise, the module effectively extracts entities from text. For instance, in case the input request contains references to entities such as `F34GP5`, `US1234567892`, `16-07-24`, `A12345`, and `P6763` the module categorizes them correctly as `firm identifier`, `ISIN`, `trade date`, `account number`, and

`portfolio id`, respectively. The extractor returns a structured output detailing the types and lists of entities found.

5 Domain Model

A PDDL domain model defines the knowledge of a planning environment by specifying key elements such as types, predicates, and actions. Types categorize objects within the domain, while predicates describe properties and relationships among these objects. Actions are defined by their preconditions and effects, detailing the operations that can be performed and how they change the state of the world. In our work, we have named the domain as `assistant`, and it is designed to handle a variety of tasks within a professional office environment such as data manipulation, presentation, and communication.

The `assistant` domain includes a diverse set of types, such as `file`, `dataframe`, `email`, `contents` or `object`. The domain also defines predicates such as `(in ?c - contents ?c1 - contents)` and `(available ?o - object)`, that indicate which contents are associated with each other and to determine the availability of objects, respectively. Among the actions in this domain, let us take the `read-data` action (shown in Listing 1) as an example. This action reads data from a `data-file` (a parameter for the action), which can be an `excel-file` or `csv-file`, and makes it `available` as a dataframe `?d`. The property or fluent `available ?d` would be added to the state to indicate that after the action completes. The sole precondition is that the file contains the dataframe contents. The action is assigned a cost of 1.

Listing 1: `read-data` action.

```
(: action  read-data
   : parameters  (?a - ai-agent ?d - dataframe ?f - data- file )
   : precondition  (and (in ?d ?f))
   : effect  (and ( available  ?d)
                  ( increase  ( total -cost) 1)))
```

6 Prompting and LLMs

In this section, we explore the structure of the input prompt, detailing all the components necessary for GENPLANX to effectively fulfill user requests. This includes the required intents, as well as the structure of the output, supplemented with examples.

6.1 Prompt Description

GENPLANX must interpret a given request, and identify a set of intents within the request. The prompt used by GENPLANX is a carefully crafted set of instructions designed to assist in generating responses to office-related queries which can utilize structured data from multiple systems of records (SORs), understand user intents, and provide the necessary initial and goals for the planner. The prompt encompasses several key components, each serving a distinct purpose to ensure the response process is both efficient and accurate. We leverage the robust few-shot learning capabilities of LLMs to adapt GENPLANX to several user requests. Below is an overview of the prompt-components, along with the rationale for their inclusion. A complete description of the prompt can be found at F.

Generic Task Definition The prompt starts by defining the generic task: addressing office operation queries. This sets the context and scope, aiding the LLM in understanding its main goal. By clearly stating the task, the prompt offers a focused framework for generating relevant responses. While the current prompt configuration is static, the Model Context Protocol (MCP)[3] can be used to dynamically augment this and other sections of the prompt. This approach allows for automatic tool and action integration based on current information, enhancing the system's flexibility in handling diverse user queries.

[3] https://modelcontextprotocol.io/

Database Schemas The prompt provides schemas for relevant SORs in case the request refers to the available datasets. Databases relate to several office processes such as sales, operations, communications, etc. Each schema lists the relevant columns, such as `trade-id`, `client-id`, `date`, etc. The prompt instructs the LLM to pay attention to the case sensitivity of these fields when creating database queries. These schemas outline the available data fields, guiding the LLM in selecting the appropriate data sources for each query. This ensures that responses are based on accurate and relevant data.

Domain Types, Actions and Predicates The prompt includes a complete list of domain types, predicates and actions by including all these elements from the input domain model in PDDL. This ensures that the LLM can accurately model the data and actions required to fulfill the query.

Set of Intents A set of predefined intents is included to guide the identification of user requests; this is to facilitate few-shot learning to the LLM. By categorizing requests into specific intents, the prompt helps streamline the response generation process, ensuring that each query is addressed appropriately.

Response Format and Dictionary Structure The response to a query is expected to be a Python dictionary that encapsulates the identified intents. The dictionary must include keys for `init_state` and `goals`, both of which are mandatory. The rest of the keys represent the objects that should be considered when solving the planning task. The values of all keys are definition dictionaries that include specific elements such as `type`, `value`, and other context-specific keys (e.g., `to`, `body`, or `subject` for emails). These structural constraints in the response ensure consistency and standardization in the output, making them easier to interpret and process.

State Representation The prompt also includes definitions related to both `init_state` and `goals`. They should be strings formatted as a sequence of literals. Each literal is a tuple with elements separated by spaces, where the first element is a predicate from the predefined list of predicates. The prompt also specifies that literals from the goals should not be included in the initial state.

Other instructions If multiple intents are identified in a request, the prompt instructs to merge the dictionaries into a single comprehensive dictionary. This involves combining all entities found and merging the `init_state` and `goals` from all intents. The prompt also imposes constraints, such as not defining functions or using external tools, and ensuring that the output is formatted correctly in a single response.

6.2 Intents

The intents refer to the set of goals that GENPLANX should achieve to fulfill the user request. Each intent is a tuple composed of the name of the intent, its description and the expected json output from the LLM. These intents enable GENPLANX to efficiently handle file and data management tasks, such as reading and saving files in various formats, including PDFs and Word documents, and performing database operations like adding, deleting, or modifying records. GENPLANX also supports data visualization and presentations creation, enabling users to generate charts and comprehensive PowerPoint presentations with ease. Additionally, it facilitates effective communication by allowing users to send emails and notifications, ensuring that important information is shared promptly.

Beyond basic data handling, GENPLANX offers advanced information processing and organizational tools. Users can explain, translate, or summarize text, find information within files, and conduct deep research on specific topics. GENPLANX also aids in scheduling by identifying free slots in calendars and provides web search capabilities for additional information. For more complex interactions, it can query a large language model for intelligent responses and match files based on specific criteria, showcasing its versatility and adaptability in addressing a wide range of tasks. The following is a list of current intents:

- **File manipulation:** Read file, Save file, Read PDF file, Read Word file, Save PDF file.
- **Database primitive operations:** Add, delete or modify records. Count or add value.
- **Office-related operations:** Send email, Notify by email, Generate Chart, Create PowerPoint, Create chart slide, Create text slide, Create table slide.

- **LLM-related operations:** Explain, Translate, Summarize, Deep research, Ask LLM.
- **Other:** API access, Data access, Find information in file, Find free slots, Search web, Match files.

6.3 Output of the LLM

The output of the LLM is a Python dictionary/json that should contain the required information to fulfill the user request. This dictionary is composed of the joint information from the different set of intents selected by the LLM. It contains information on objects to be considered in the planning episode, the initial state representing the state at the beginning of the planning task, and the goals representing the desired partial state. This dictionary must adhere to the following structure:

- The keys of the dictionary are the elements of the task (objects), `init_state`, and `goals`. All keys must be in lowercase.
- Each object is defined as a dictionary with at least the two keys `type` and `value`. `type` represents the PDDL type of the object. `value` is the value that the object will take at execution time. Initially, it can have any arbitrary value for most cases. When actions operate with an object, the value of the object during execution will be saved there. Additionally, it can have other specific keys such as `to`, `body`, or `subject` for emails.
- The types of the different objects must adhere to a pre-defined set of types, the ones provided as input in the prompt that appear in the domain file.
- `init_state` and `goals` are mandatory keys and are composed of a sequence of literals.

A complete dictionary output example is available in Appendix A.

7 Planning and Execution

Given the dictionary returned by the LLM, the compiler module translates its information into a PDDL problem definition (see Appendix B). The domain and problem files serve as input to the architecture described in Figure 1. The Planner then returns a Plan, i.e., a sequence of actions described in PDDL that achieve the goal stated in the problem. GENPLANX uses Fast-Downward [36] through the Unified Planning library [37], but any other planning engine can be used, since the domain and problem definitions are specified in the PDDL standard language. Thus, GENPLANX is planner-independent. In Listing 2, we show an example of a plan in PDDL, which is the solution to the problem stated in B. Every action is represented as a tuple composed of the name and the parameters of the action.

Listing 2: Plan Example.

```
(read-data ai dataframe1 data-file1)
(query-data ai query1 dataframe1 filtered-dataframe)
(create-response ai chat-response)
(add-to-response ai filtered-dataframe chat-response)
(send-response ai chat-response)
```

8 Execution and Monitoring

The resulting plan is sent to the Monitoring and Execution module, which translates it into the actual code that will be executed into the real-world. This translation is accomplished through a mapping linking PDDL actions to the corresponding python executable functions. Each PDDL action has a corresponding python function with the same name that implements it. These python functions share the signature `(parameters, state)`, which allows the execution module to update the execution state at each step. See the implementation of *read-data* in Appendix C

This module will also periodically sense the real-world to check whether the plan is going as expected or the real-world has deviated from the planned states. As opposed to other planning-execution

architectures [38], currently GENPLANX only monitors the low-level state without translating it back to the high-level PDDL state. More specifically, monitoring is done through boolean functions that check whether the execution of the action into the real-world yielded the expected effects. See an example in Appendix C. For an end-to-end example, showing from intents to executed actions see Appendix D.

9 Experimental Evaluation

In this section, we present the results of a set of preliminary experiments designed to compare the planning abilities of GENPLANX, which generates a PDDL problem using an LLM, and solves it using a classical planner, with two LLM-based planning approaches:

1. LLM-DIRECT-PLAN, which generates a plan given the user request and a natural language description of the assistant domain. LLM-DIRECT-PLAN runs an LLM client with a prompt designed to generate a plan from the request by considering the same actions in the domain, and carefully guiding the LLM towards a valid, and syntactically correct output (i.e., a PDDL plan compliant with the given domain).

2. LLM-TRANSLATE-AND-PLAN, which first translates the user request into a PDDL problem, and generates a plan for it; both steps use an LLM. The translation part works in the same way as GENPLANX produces the planning task. In the second step the prompt consists of a list of actions with the corresponding parameters and a detailed instruction to produce a syntactically correct PDDL plan.

We evaluated all three approaches with two LLM clients, GPT-4o and o3-mini. We manually designed 50 user requests[4] that covered several intents each and handled prototypical examples within the assistant domain. We then ran each planning configuration five times (due to the stochasticity of the call to the LLM) and report metrics (mean and standard deviation) for success ratio, and time taken (for the full process, LLM steps, and planning time). We used GPT-4o with temperature 0, and max output of 4096 tokens, and o3-mini with temperature 0, and max tokens 32768. Success is measured by checking whether the generated plan is valid using a formal plan validation tool and if the plan is equivalent to the *expected plan* set as ground truth. The expected plan is the sequence of actions manually created for each of the 50 user request in the test set. See Appendix E.2 for details on plan validation. Table 1 summarizes the main results.

Approach	Success ratio	Total time	Translation time	Planning time
GENPLANX (with GPT-4o)	**0.50 (0.50)**	**9.0 (2.1)**	**5.5 (1.8)**	**3.5 (1.4)**
LLM-DP (with o3-mini)	0.27 (0.44)	16.6 (5.7)	NA	16.6 (5.7)
LLM-TP (with GPT-4o)	0.29 (0.45)	9.3 (2.7)	5.6 (2.8)	3.7 (0.9)

Table 1: Results of comparing GENPLANX against LLM-based approaches. We report the mean (and standard deviation) for the best performing LLM Client (in parentheses) within each approach. Results for all configurations are shown in Appendix E. LLM-DP = LLM-DIRECT-PLAN; LLM-TP = LLM-TRANSLATE-AND-PLAN.

We observe that GENPLANX demonstrates a significantly higher success ratio of 0.50 (0.50), indicating its robustness in generating valid plans compared to LLM-DIRECT-PLAN and LLM-TRANSLATE-AND-PLAN, which have success ratios of 0.27 (0.44), 0.29 (0.45) respectively. The high success ratio of GENPLANX suggests that it is more reliable in performing planning-related tasks, and in particular within the office assistant domain, even compared with a "reasoning" LLM as o3-mini.

In terms of time metrics, GENPLANX demonstrates better efficiency with a total time of 9.0 (2.1) seconds, which is lower than the best-performing LLM approaches: LLM-DIRECT-PLAN's 16.6 (5.7) seconds and LLM-TRANSLATE-AND-PLAN's 9.3 (0.45) seconds. This indicates that GENPLANX is also very efficient in the planning phase.

[4]We repeated the experiments with randomly generated instances by GPT4o and results were equivalent.

9.1 Role of LLM Clients

We evaluated GENPLANX, LLM-DIRECT-PLAN and LLM-TRANSLATE-AND-PLAN with GPT-4o and o3-mini LLM clients. For both GENPLANX and LLM-TRANSLATE-AND-PLAN, which use the LLM for translating the user request in natural language to PDDL, GPT-4o is outperforming o3-mini as observed in Figure 2 (left). This indicates that gpt-4o is better at translating natural language to PDDL than o3-mini. In terms of planning, LLM-DIRECT-PLAN and LLM-TRANSLATE-AND-PLAN use the LLM Client, whereas, GENPLANX uses a classical planner. The classical planner is outperforming both GPT-4o and o3-mini. In all cases, o3-mini takes longer to respond to the queries as seen in Figure 2 (right). LLM-TRANSLATE-AND-PLAN is the most time consuming planner because it uses the LLM Client twice. For detailed breakdown of translating vs planning time, please see Appendix E, Figure 7.

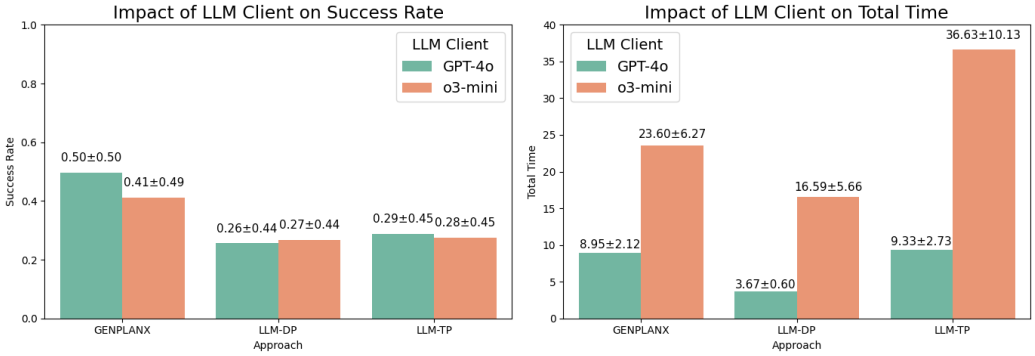

Figure 2: We experimented with two LLM clients, GPT-4o and o3-mini for each of our planning approaches, GENPLANX, LLM-DIRECT-PLAN (LLM-DP) and LLM-TRANSLATE-AND-PLAN (LLM-TP). Left: The impact of LLM Client on success ratio. Right: The total time taken (in seconds per run) by the LLM Clients for the respective planning approaches shown in X-axis.

9.2 Failure Analysis

In this section, we examine the cases where the planners fail, and discuss the potential reasons for failure. The cases where GENPLANX fail (as well as the LLM-TRANSLATE-AND-PLAN, since they share the same initial step) are primarily due to the LLM's inability to generate a valid PDDL problem description (dictionary). This issue often arises when an action requires a parameter of a specific type, but the problem description supplies an object of a subtype. Refining the prompt engineering might boost GENPLANX's performance even further.

In Figure 3 (left), we examine the incorrect plans and categorize them. LLM-DIRECT-PLAN and LLM-TRANSLATE-AND-PLAN return invalid plans for a singificant number of cases. This is due to incorrect positioning of actions within a plan, incorrect action parameters and not including relevant actions. GENPLANX and LLM-TRANSLATE-AND-PLAN fail in the PDDL problem formulation step, and end up solving a different task compared to the user request. We categorize these failures as 'Different Task'. Finally, if the problem is formed incorrectly such that no solution exists, GENPLANX doesn't return a plan ('No plan' category).

In Figure 3 (right), we calculate the overlap of the incorrect plans with the ground truth. If all the actions in the result are part of the ground truth, overlap is 1. If no actions in the generated plan are a part of the ground truth, the overlap is 0. We observe that even though LLM-TRANSLATE-AND-PLAN is the worst performing planner in our experiments, its incorrect plans have a highest overlap with ground truth.

9.3 Role of Action Semantics

For LLM-DIRECT-PLAN and LLM-TRANSLATE-AND-PLAN, we conducted tests without and with providing the action semantics of the planning domain in natural language as a part of the prompt. This includes the list of objects and a description of how the action changes the environment. We

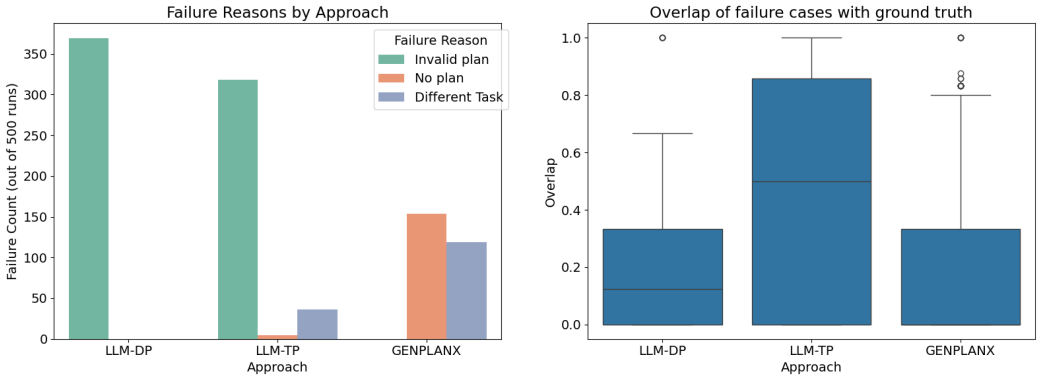

Figure 3: Analysis of failures for our three planning approaches, GENPLANX, LLM-DIRECT-PLAN (LLM-DP), and LLM-TRANSLATE-AND-PLAN (LLM-TP). Left: We observe the different categories of failures. Right: We study the overlap of the incorrect plans with the ground truth.

expected that providing the description of action semantics will improve the reasoning of the LLMs and guide them to come up with better plans. However, it led to a degradation in performance for both approaches, resulting in lower success ratios compared to the base prompt without the action semantics. For the LLM-DIRECT-PLAN model, the success ratio dropped from 25.6% to 21.2% with gpt-4o and from 26.8% to 19.2% with the o3-mini (see Table 2 for all metrics). Similarly, the LLM-TRANSLATE-AND-PLAN model experienced a decrease in success ratio from 28.8% to 17.2% with the gpt-4o and from 27.6% to 10.8% with o3-mini. Given the information value of action semantics in natural language for the tasks, we expected the performance to improve. However, this drop in performance is not entirely unexpected as prior work [39, 40, 41] has shown that reasoning performance can drop with the length of the context, and generating plans requires complex reasoning over dependencies and costs.

10 Conclusions and Future work

In this paper we have introduced GENPLANX. It can understand user requests in natural language, generate plans to address those requests, execute the plans in a real office environment, and monitor the execution. Our contributions include a new domain that implements common office-related actions, the implementation of those actions, as well as an architecture that integrates LLMs and AI classical planning. Additionally, we reported an experimental evaluation in which GENPLANX demonstrated a success ratio that was 20 percentage points higher compared to LLM-based alternatives. This also includes the benefit of reduced response time due to the simpler nature of translating user requests into a formal task representation.

In future work we would like to improve GENPLANX in two main fronts. First, we would like to expand the set of actions considered in order to cover a wider ranger of office tasks. This process could be either manual as we are currently doing; or automated, by learning action models from observations [42, 43, 44]. Second, we would like to provide GENPLANX with goal reasoning [45] capabilities. In particular, we would like to let GENPLANX automatically generate new goals upon replanning [46], when monitoring detects opportunities upon changes in the environment [47], by analyzing the structure of the goals [48], or by predicting the appearance of new goals [49, 50].

11 Limitations

Our current evaluation, while demonstrating a 20 percentage point improvement over LLM-based alternatives, is limited in scope and lacks extensive testing across diverse office environments, task complexities, and user expertise levels. The performance of GENPLANX is partially dependent on the specific LLM used, yet we have not systematically evaluated how different language models affect overall system performance, robustness, and generalization capabilities. Additionally, the current implementation relies on a fixed manually designed domain model with predefined actions, creating

scalability challenges as the number of supported office tasks increases, potentially affecting planning efficiency and maintenance as the domain model grows in complexity.

References

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] 0. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, , and M. Williamson. An approach to planning with incomplete information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1992.

[3] Anand Narayanaswamy. Using copilot in microsoft 365. In *Microsoft Copilot for Windows 11: Understanding the AI-Powered Features in Windows 11*, pages 205–233. Springer, 2024.

[4] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Access Online via Elsevier, 2004.

[5] Javier García, José E. Florez, Álvaro Torralba, Daniel Borrajo, Carlos Linares-López, Ángel García-Olaya, and Juan Sáenz. Combining linear programming and automated planning to solve intermodal transportation problems. *European Journal of Operations Research*, 227(1):216–226, 2013.

[6] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J.C.-J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B.G. Chafin, W.C. Dias, and P.F. Maldague. MAPGEN: Mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems*, 19(1):8–12, feb 2004.

[7] Jana Koehler and Kilian Schuster. Elevator control as a planning problem. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 331–338, 2000.

[8] Isabel Cenamor, Sergio Núñez, Tomás de la Rosa, and Daniel Borrajo. Planning for tourism routes using social networks. *Expert Systems with Applications*, 69:1–9, 2017. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2016.10.030. URL `http://dx.doi.org/10.1016/j.eswa.2016.10.030`.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL `https://arxiv.org/abs/2201.11903`.

[10] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL `https://arxiv.org/abs/2305.10601`.

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

[12] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models, 2024. URL `https://arxiv.org/abs/2311.05772`.

[13] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

[14] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

[15] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Kaya Stechly, Mudit Verma, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

[16] Tomás de la Rosa, Sriram Gopalakrishnan, Alberto Pozanco, Zhen Zeng, and Daniel Borrajo. TRIP-PAL: Travel planning with guarantees by combining large language models and automated planners. *arXiv e-prints*, abs/2406.10196, 2024. URL `https://arxiv.org/abs/2406.10196`.

[17] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

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

[19] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[20] James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *34th International Conference on Automated Planning and Scheduling*, 2024.

[21] Vishal Pallagani, Kaushik Roy, Bharath Muppasani, Francesco Fabiano, Andrea Loreggia, Keerthiram Murugesan, Biplav Srivastava, Francesca Rossi, Lior Horesh, and Amit Sheth. On the prospects of incorporating large language models (llms) in automated planning and scheduling (aps). *arXiv preprint arXiv:2401.02500*, 2024.

[22] Rajesh Gupta, Suresh Patel, and Ming Lee. Transformers for natural language to structured planning: Integrating domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4567–4574, 2021.

[23] Li Zhang, Mei Wang, and Hui Chen. Structured representations for natural language to planning problem translation. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5):2003–2015, 2022.

[24] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Roozbeh Mottaghi. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[25] Roozbeh Mottaghi, Chuang Gan Wu, and Ali Farhadi. Nocturne: A scalable driving benchmark for bringing multi-agent learning one step closer to the real world. In *Conference on Robot Learning*, pages 647–656. PMLR, 2020.

[26] Ishika Singh, David Traum, and Jesse Thomason. Twostep: Multi-agent task planning using classical planners and large language models, 2024. URL `https://arxiv.org/abs/2403.17246`.

[27] Yarin Benyamin, Argaman Mordoch, Shahaf S Shperberg, and Roni Stern. Toward pddl planning copilot. *arXiv preprint arXiv:2509.12987*, 2025.

[28] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998. URL `citeseer.nj.nec.com/muscettola98remote.html`.

[29] María Dolores Rodríguez-Moreno, Daniel Borrajo, and Daniel Meziat. An AI planning-based tool for scheduling satellite nominal operations. *AI Magazine*, 25(4):9–27, Winter 2004. URL `http://hdl.handle.net/10016/6792`. http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1782.

[30] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*, pages 380–394. Springer, 2004.

[31] David Camacho, Ricardo Aler, Daniel Borrajo, and José Manuel Molina. A multi-agent architecture for intelligent gathering systems. *AI Communications*, 18(1):15–32, 2005. URL `http://hdl.handle.net/10016/5898`.

[32] María Dolores Rodríguez-Moreno, Daniel Borrajo, Amedeo Cesta, and Angelo Oddi. Integrating planning and scheduling in workflow domains. *Expert Systems with Applications*, 33(2):389–406, October 2007.

[33] Sunandita Patra, Alex Velazquez, Myong Kang, and Dana Nau. Using online planning and acting to recover from cyberattacks on software-defined networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 15377–15384, 2021.

[34] Fernando Fernández, Daniel Borrajo, Susana Fernández, and David Manzano. Assisting data mining through automated planning. In P. Perner, editor, *Proceedings of the International Conference on Machine Learning and Data Mining (MLDM 2009)*, volume 5632 of *Lecture Notes in Artificial Intelligence*, pages 760–774, Leipzig (Germany), 2009. Springer Verlag. URL `mldm09.pdf`.

[35] Simerjot Kaur, Charese Smiley, Keshav Ramani, Elena Kochkina, Mathieu Sibue, Samuel Mensah, Pietro Totis, Cecilia Tilli, Toyin Aguda, Daniel Borrajo, and Manuela Veloso. Advanced messaging platform (amp): Pipeline for automated enterprise email processing. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Industry Track)*, Vienna, Austria, July 2025. Association for Computational Linguistics.

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

[37] Andrea Micheli, Arthur Bit-Monnot, Gabriele Röger, Enrico Scala, Alessandro Valentini, Luca Framba, Alberto Rovetta, Alessandro Trapasso, Luigi Bonassi, Alfonso Emilio Gerevini, Luca Iocchi, Felix Ingrand, Uwe Köckemann, Fabio Patrizi, Alessandro Saetti, Ivan Serina, and Sebastian Stock. Unified planning: Modeling, manipulating and solving ai planning problems in python. *SoftwareX*, 29:102012, 2025. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2024.102012. URL `https://www.sciencedirect.com/science/article/pii/S2352711024003820`.

[38] César Guzmán, Vidal Alcázar, David Prior, Eva Onaindía, Daniel Borrajo, and Juan Fdez-Olivares. Building a domain-independent architecture for planning, learning and execution. PELEA. In *Proceedings of the ICAPS 2011 System Demonstrations*, Freiburg (Germany), 2011. AAAI Press. URL `icaps11-demo-pelea.pdf`.

[39] Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models. *arXiv preprint arXiv:2402.14848*, 2024.

[40] Yury Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.

[41] Zhan Ling, Kang Liu, Kai Yan, Yifan Yang, Weijian Lin, Ting-Han Fan, Lingfeng Shen, Zhengyin Du, and Jiecao Chen. Longreason: A synthetic long-context reasoning benchmark via context expansion. *arXiv preprint arXiv:2501.15089*, 2025.

[42] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, 2019. doi: 10.1016/j.artint.2019.05.003. URL `https://doi.org/10.1016/j.artint.2019.05.003`.

[43] Marianela Morales, Alberto Pozanco, Giuseppe Canonaco, Sriram Gopalakrishnan, Daniel Borrajo, and Manuela Veloso. On learning action costs from input plans. *arXiv preprint arXiv:2408.10889*, 2024.

[44] Alba Gragera, Raquel Fuentetaja, Ángel García-Olaya, and Fernando Fernández. A planning approach to repair domains with incomplete action effects. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 153–161, 2023.

[45] Héctor Muñoz-Avila, Ulit Jaidee, David W. Aha, and Elizabeth Carter. Goal-driven autonomy with case-based reasoning. In *Proceedings of Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning (ICCBR) 2010*, volume 6176 of *Lecture Notes in Computer Science*, pages 228–241. Springer, 2010.

[46] Alberto Pozanco, Daniel Borrajo, and Manuela Veloso. Generating replanning goals through multi-objective optimization in response to execution observation. In *Proceedings of ECAI*, Kraków (Poland), 2023.

[47] Daniel Borrajo and Manuela M. Veloso. Computing opportunities to augment plans for novel replanning during execution. In *Proceedings of ICAPS*, Guangzhou (China), 2021.

[48] Alberto Pozanco, Álvaro Torralba, and Daniel Borrajo. Computing planning centroids and minimum covering states using symbolic bidirectional search. In *Proceedings of ICAPS*, Banff (Canada), 2024.

[49] Alberto Pozanco, Susana Fernández, and Daniel Borrajo. Learning-driven goal generation. *AI Communications*, 31(2):137–150, 2018. ISSN ISSN on-line: 1875-8452. URL `aicomm18-learning.pdf`. https://content.iospress.com/articles/ai-communications/aic754, DOI: 10.3233/AIC-180754.

[50] Raquel Fuentetaja, Daniel Borrajo, and Tomás de la Rosa. Anticipation of goals in automated planning. *AI Communications*, 31(2):117–135, March 2018. ISSN ISSN on-line: 1875-8452. URL `aicomm18-anticipatory.pdf`.

A Output Example of Task description

As an example, given the request *What is the status of the trade TR123?*, the LLM could return the dictionary in Figure 4. This dictionary represents the output of the LLM tasked with extracting the status of a particular trade when provided with the trade ID, which is present in a downstream file. The file is identified by the type of the entity returned by the Entity extraction module. Going into more detail, what the LLM is describing at the output is:

```
{"data-file1": {"type": "data-file", "value": "./genplanx/file_1.csv"}
 "dataframe1": {"type": "dataframe", "value": []}
 "filtered-dataframe": {"type": "dataframe", "value": []}
 "chat-response": {"type": "response", "value": []}
 "query1": {"type": "query", "value": "df[(df["trade-id"] == "TR123")]"}
 "init_state": {"type": "state",
                "value": "(in dataframe1 data-file1) (available query1)
                          (query-result dataframe1 query1 filtered-dataframe)"}
 "goals": {"type": "state",
           "value": "(and (done-query query1)
                          (in filtered-dataframe chat-response)
                          (sent chat-response))"}}
```

Figure 4: Example of a structured output dictionary for getting the status of a trade.

- **Objects**: The dictionary includes the elements (as key-value pairs) that are required during execution, like the file where the information is stored, the dataframe that will be extracted from the file, the filtered-dataframe after filtering it regarding the user query and the chat response that would be generated and returned to the user
- **Initial State**: Represents the initial state of the task, where the dataframe is inside the file, the system has the query defined and there is a query result after applying the query to the dataframe inside the file.
- **Goals**: Represents the desired state where the query has been done, the filtered dataframe is present in the response to the user and the response is sent to the user with the information requested.

B PDDL Problem

The task dictionary is translated into a PDDL problem as shown in Listing 3.

Listing 3: Problem Example.

```
(define (problem test-llm)
  (:domain assistant)
  (:objects ai - agent
            data-file1 - data-file
            dataframe1 filtered-dataframe - dataframe
            chat-response - response
            query1 - query)
  (:init (in dataframe1 data-file1)
         (available query1)
         (query-result dataframe1 query1 filtered-dataframe))
  (:goal (and (done-query query1)
              (in filtered-dataframe chat-response)
              (sent chat-response))))
```

C Execution and Monitoring Functions

Each action in the plan has a corresponding python implementation that executes the task in the real-world. Action shown in Listing 4 shows an example where the action reads the content of a data file and loads it into a dataframe. Listing 5 shows the python function that monitors the execution of the `read-data` action. This function checks whether the data read from the file is a dataframe instance or not. In case the execution of any action fails, GENPLANX replans. Another reason for replanning can be an action execution adding a new goal to the execution state. As an example, a read-email action can read the contents of an email, and generate a new set of goals related to the intents expressed in the email.

Listing 4: read-data Python implementation.

```
def read_data (parameters, state):
    # obtains the object names from the parameters of the action
    data_var, file_var = parameters[1], parameters[2]
    # gets the file path from the file entry in the execution state
    path = state[file_var]["value"]
    df = pd.read_csv(path)
    columns = df.columns.tolist()
    state[data_var] = {"type": "dataframe",
                       "value": df,
                       "columns": columns}
    # adds columns of dataframe as new objects to the state
    for col in columns:
        if col not in state:
            name = col.replace("␣", "-") + "_column"
            name = name.lower()
            state[name] = {"type": "column", "value": col}
    return state
```

Listing 5: Function that monitors the success of the read-data action execution.

```
def read_data_success (action, state):
    data_var = action[2]
    success = (isinstance(state[data_var]["value"], pd.DataFrame))
    return success
```

D Example showing the utility of GENPLANX

In this section, we present an example that shows the utility of GENPLANX. For an office task related to annual report presentation, we present Example 1 and demonstrate how GENPLANX successfully optimally solves the task.

User Prompt for Example 1: *Read data and generate a barchart from 'balance' against reference column 'year. The available databases to read from are db1 with cost of reading 1 and db2 with cost of reading 3. db1 supports basic query. db2 supports optimized query. Create a slide with bar chart with title 'Balance over years', and add it to a presentation. Save the presentation on file genplanx/graph.pptx.*

The initial state and the goals generated by the LLM and compiled into PDDL are in Figure 5.

Initial State:

```
(in df1 db1) (in df2 db2) (reference reference1 reference-df)
(available query1) (query-result df1 query1 filtered-df)
(available query2) (query-result df1 query2 reference-df)
(database-query-optimized db2) (database-query-basic db1)
(= (database-cost database1) 1) (= (database-cost database2) 2)(= (total-cost) 0)
```

Goals:

```
(done-query query1) (done-query query2) (in-graph filtered-df reference-df bar-chart1)
(in bar-chart1 slide1) (in title1 slide1) (in slide1 presentation1) (in ppt1 ppt-file1)
```

Figure 5: Initial state, goals for Example1.

GENPLANX first needs to read the data from a database. Consider the situation where the data can be accessed from two databases, `db1` and `db2`, which may differ in access protocols and design. The read step has two alternative choices for action parameter, `read-data(db1)` with cost 1 and `read-data(db2)` with cost 2. The choice made at the read step has an effect on the query step. `db1` only support basic query (action `query-data-basic`) with cost 5, and `db2` supports optimized query (action `query-data-optimized`) with cost 2. Reading data from `db1` using a basic query results in a cost of 6 (1+5). Alternatively, when data is read from `db2` using an optimized query is both valid and cost-effective, with a total cost of 4 (2+2). Therefore, the optimal plan should include `read-data(db2)` and `query-data-optimized`. With the help of a classical planner, GENPLANX is able to generate an optimal plan (as shown in Figure 6). When we asked the LLM, GPT-4O, to generate a plan for this task, it was unable to generate the optimal plan. We also show that even the o3-mini reasoning model does not generate correct or optimal plan consistently without additional hints.

- **create-graph:** create graph `bar-chart1` of type `bar-chart`
- **read-data:** read data from file `./genplanx/annual-report.csv` into `dataframe1`
- **create-presentation:** create empty presentation `presentation1`
- **create-slide:** create slide `slide1` to presentation `presentation1`
- **query-data-optimized:** query database using `df['balance']` from `dataframe1` into `filtered-dataframe`
- **query-data-optimized:** query database using `df['year']` from `dataframe1` into `reference-dataframe`
- **add-to-graph:** add `filtered-dataframe` to graph `bar-chart1`
- **add-to-slide-basic:** add `bar-chart1` to slide `slide1`
- **generate-presentation:** save presentation `presentation1` into `./genplanx/graph.pptx`

Figure 6: Step by step plan generated and executed by GENPLANX for Example 1.

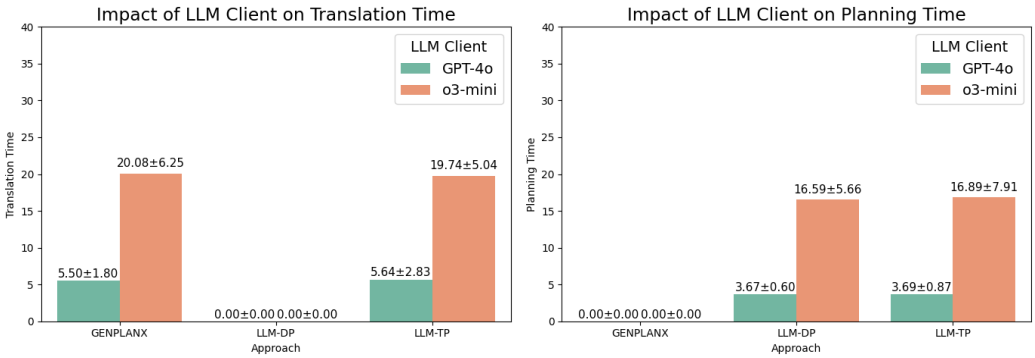

Figure 7: For each of our planning approaches, GENPLANX, LLM-DIRECT-PLAN, and LLM-TRANSLATE-AND-PLAN, we measure the time spent by the respective LLM Client on translating the user request from natural language to PDDL (left), and the time spent (in seconds per run) on generating the plan (right). Since LLM-DIRECT-PLAN generates the plan without translation, its translation time is 0. GENPLANX uses a classical planner for planning, and not the LLM Client, hence its planning time is 0 for LLM Client. To see GENPLANX's planning times, please refer to Table 2.

E Additional Experimental Results

This section presents detailed results for the experimental evaluation, including the impact of different LLM Clients and inclusion or exclusion (*base* case on Table 2) of action semantics.

Table 2: Results of comparing GENPLANX against LLM-based approaches. We report the mean (and standard deviation) for all configurations of LLM Client and prompts provided for all 50 user requests in our test set. Each configuration is run five times per user request. LLM-DP = LLM-DIRECT-PLAN; LLM-TP = LLM-TRANSLATE-AND-PLAN.

Approach	Prompt	Client	Success Ratio	Total Time	Translation Time	Planning Time
GENPLANX	base	GPT-4o	**0.50** (0.50)	8.95 (2.12)	**5.50** (1.80)	**3.45** (1.41)
GENPLANX	base	o3-mini	0.41 (0.49)	23.60 (6.27)	20.08 (6.25)	3.53 (1.07)
LLM-DP	base	GPT-4o	0.26 (0.44)	**3.67** (0.60)	NA	3.67 (0.60)
LLM-DP	base	o3-mini	0.27 (0.44)	16.59 (5.66)	NA	16.59 (5.66)
LLM-DP	actions semantics	GPT-4o	0.21 (0.41)	3.86 (0.60)	NA	3.86 (0.60)
LLM-DP	actions semantics	o3-mini	0.19 (0.39)	15.12 (5.03)	NA	15.12 (5.03)
LLM-TP	base	GPT-4o	0.29 (0.45)	9.33 (2.73)	5.64 (2.83)	3.69 (0.87)
LLM-TP	base	o3-mini	0.28 (0.45)	36.63 (10.13)	19.74 (5.04)	16.89 (7.91)
LLM-TP	actions semantics	GPT-4o	0.17 (0.38)	9.43 (2.90)	5.67 (3.08)	3.76 (0.86)
LLM-TP	actions semantics	o3-mini	0.11 (0.31)	40.96 (20.08)	20.21 (5.63)	20.75 (18.15)

E.1 Translation and Planning Time Breakdown for LLM Clients

Following up from the discussion in Section 9.1, Figure 7 presents the time spent by the LLM Clients on translating the user request from natural language to PDDL (left), and the time taken to generate a plan (right). No GPUs were needed, and the LLMs were accessed via APIs. All experiments were run on Intel Xeon Gold 6240R CPUs with 8 processors and 64GB RAM.

E.2 Plan Validation

Given a planning task in PDDL, a plan is valid if the sequence of actions can transform the initial state into a succession of states where the final state contains the goals. We perform this check as provided by the Unified

Planning library[5]. In GENPLANX, we require an additional validation because the LLM is also generating the planning task, which means a valid plan might not fulfill the original intent of the user's request. Therefore, for each request in the test set, we include an expected plan that resolves the goal in the user's request. This second validation against this plan is done for equivalence, as actions may appear in a different order due to plan symmetries, and the values of parameters (e.g., the identifier of a dataframe where a file is loaded) only need to represent equivalent objects. Note that plans can achieve the same goal even if these objects are named differently. This mechanism allowed us to identify failures categorized as '*Different task*', which are valid plans that did not correspond to the original request. We verified that this validation process is sound based on a sample of 160 executions drawn from different tasks and configurations.

F GENPLANX task generation prompt

Below is an example of the full prompt GENPLANX uses to call the LLM

```
    You are working in an office environment.
    You have to carry out office-related actions that include providing responses to
         queries from employees or clients related to sales. Information is present
        on several systems of records (SOR). The schemas (columns) of those systems
        and their values are the following:
Schema for SOR 1: [list of columns], Schema for SOR 2: [list of columns]
Pay attention to the upper or lower case of the fields in the provided schema when
    creating queries to the databases. Given a request, decide whether to use or
    not an SOR and if so choose the appropriate SOR, and identify a set of intents
    on that request. Then, return a Python dictionary that contains information on
    all intents. You should not define a function or provide python code, but
    return the dictionary as your output. Do not use external tools. The keys of
    the dictionary are the elements of the task (entities), the 'init_state' and
    the 'goals'. All keys have to be in lower case. 'init_state' and 'goals' are
    mandatory. The values of the dictionary keys are definition dictionaries. A
    definition dictionary is a python dictionary, where the keys are 'type', 'value
    ', and some other element specific keys, such as 'to', 'body', or 'subject' for
     emails. The types of the task are: pie-chart, bar-chart, histogram, column,
    value-counts, count, input-email, output-email, human-agent, ai-agent, excel-
    file, csv-file, regulation, dataframe, text, graph, title, api, data-file, pdf-
    file, word-file, text-file, powerpoint-file, section, news, row, file, email,
    data, model, data-contents, response, query, chat-history, ml-algorithm,
    presentation, appointments, appointments-item, slide, contents, data-type,
    agent, language, session, object. The entities' values should be extracted from
     the intents on the request. You cannot use as keys of dictionary the names of
    types and you cannot use repeated keys. The value of 'init_state' is a string
    whose contents is a state, where a state is a sequence of literals separated by
     spaces. The init_state represents what is known to be true at the beginning.
    Each literal is a tuple whose elements are separated by spaces. The first
    element of the literals is a predicate and the following elements are its
    arguments. Each argument ** must be a symbol (e.g. query1 or dataframe2) **.
    The following is a list of valid predicates, where each element is of the form
    (<predicate-name> <parameters>):
       (in ?c - contents ?c1 - contents)
       (in-data ?dt - data-type ?d - contents)
       (in-graph ?c - contents ?c1 - contents ?c3 - graph)
       (available ?o - object)
       (used ?c - contents)
       (data-type-contents ?dt - data-type ?dc - data-contents)
       (web-search-result ?q - text ?r - text)
       (deep-research-result ?q - text ?r - text)
       (merged-answer ?q - text ?r - text)
       (last-result ?q - text ?l - text)
       (query-result ?d - dataframe ?q - query ?d1 - data)
       (answer-llm ?q - text ?r - text)
       (extract-result ?a - api ?c - column ?v - text ?d - data)
       (done-merge)
```

[5] https://unified-planning.readthedocs.io/en/latest/operation_modes.html#planvalidator

```
        (reference ?c - column ?d - dataframe)
        (done-query ?q - query)
        (done-question ?q - text)
        (modified ?d - dataframe ?c - contents ?co - column ?t - text ?d1 -
            dataframe)
        (added-value ?d - dataframe ?c - contents ?co - column ?t - text ?d1 -
            dataframe)
        (merged ?d - dataframe ?d1 - dataframe ?d2 - dataframe)
        (deleted ?d - dataframe ?d1 - dataframe ?d2 - dataframe)
        (sent-contents ?rc - contents ?r - response)
        (sent ?r - response)
        (sent-email ?e - output-email ?c - contents)
        (read-email-contents ?c - contents ?f - file ?e - input-email)
        (replied-email ?e - input-email)
        (email-read ?e - input-email)
        (email-parsed ?e - input-email)
        (notified ?e - output-email ?c - contents)
        (explained ?c - contents ?c1 - contents)
        (translated ?c - contents ?l - language ?c1 - contents)
        (summarized ?c - contents ?c1 - contents)
        (search-result ?t - text ?q - text ?t1 - text)
        (news-result ?q - text ?r - dataframe)
        (fact-checked ?r - text)
        (checked-regulation ?q - text ?r - regulation)
        (before ?t - text ?t2 - text)
        (merged-text ?t1 - text ?t2 - text ?t3 - text)
        (relevant-to ?m - text ?t - text)
        (generated-from ?m - text ?r - text)
        (info-on ?d - data-contents ?g - graph ?p - presentation)
        (appointments-read ?s - appointments)
        (appointments-contents ?s - appointments ?d - dataframe)
        (free-slots ?d - dataframe ?d1 - dataframe)
        (learned-model ?l - ml-algorithm ?m - model ?d - dataframe ?c - column)
        (matched ?f1 - file ?f2 - file ?f3 - file ?m - dataframe)
The parameters are defined as: <variable> - <type>. The value of 'goals' is the list
     of intents, also represented as a state (list of literals). Take the types
    into account when defining the literals in the states (init_state and goal).
    For example, if you want to express that a file F contains a dataframe D, add
    to the init_state the literal '(in D F)'. ** You cannot use elements of other
    types as arguments of the corresponding predicate. Make sure all parameters of
    all literals that appear in the 'init_state' and 'goals' have an entry in the
    output dictionary. Everything that is true at start should be in the '
    init_state'. Everything that you would like to be true at the end should be
    specified in the 'goals'. If you find more than one intent in the request,
    merge the dictionaries into a single dictionary. In order to merge the
    dictionaries, add all entities found. Also, the merged 'init_state' will be the
     list of all literals in all the intents' 'init_state'. Likewise, the merged '
    goals' will be the list of all literals on all intents' 'goals'. Do not include
     literals from the goals in the initial state. Use one chat response. Make sure
     you format the output properly and take into account all the previous
    constraints. When creating queries to databases please take into account
    semantics. As an example, if the request asks about not matched transactions,
    check for all semantically equivalent values, as unmatched. Do not return
    Output: in the output. **
I will give you now several examples with their corresponding output as '
Intent: <intent>
Output:
<dictionary>'.
Examples:
Intent: Summarize
Output:
{'text1': {"type": "text", "value": "matched"}, 'text2': {'type': 'text', 'value': '
    text2'}, 'init_state': {'type': 'state', 'value': ''}, 'goals': {'type': 'state
    ', 'value': '(and (summarized text1 text2))'}}
```

```
Intent: Explain
Output:
{'text1': {"type": "text", "value": "matched"}, 'text2': {'type': 'text', 'value': '
    text2'}, 'init_state': {'type': 'state', 'value': '(available text1)'}, 'goals':
     {'type': 'state', 'value': '(and (explained text1 text2))'}}

Intent: Unknown Intent/Anything else/Something unrelated to the above intents
Output:
{"chat-response": {"type": "response", "value": "Apologies, I'm not able to help
    with that. Try another question!"}, "init_state": {"type": "state", "value": "(
    available chat-response)"}, "goals": {"type": "state", "value": "(and (sent
    chat-response))"}}

Intent:
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction are regarding the efficacy of GENPLANX and how its results compare with LLM approaches. This reflects the scope and contributions of the paper as outlined in Section 9.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 11 addresses the current limitations of our work.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA] .

   Justification: There are not theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.

- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer:[Yes]

   Justification: The prompts, models and their parameters have been disclosed in Sections 6 – 9

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: While our work does not disclose the actual code and data, all efforts have been taken to ensure maximum transparency in our architecture, modeling and approach.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so No is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All details regarding the LLMs used, planners and other modules have been disclosed in the paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All experiments across the paper contain the average results across five runs per configuration for each of the 50 user requests and also show the standard deviation.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Information on the compute resrouces have been provided as a part of the Appendix Section E.1. Justification: For all experiments, the time of execution along with the mean and standard deviation has been reported.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Our work currently is a method to incorporate planning and LLMs into automating office tasks. This does not have direct societal impact in its current state.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our current work does not involve the release of data or models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The creators of the models have been properly cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: Our work does not release any assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our work does not involve crowd sourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Our work does not involve crowd sourcing or research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are important to GENPLANX and this has been sufficiently described.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.