FLAP: Flow-Adhering Planning with Constrained Decoding in LLMs

Shamik Roy   Sailik Sengupta   Daniele Bonadiman   Saab Mansour   Arshit Gupta
{royshami, sailiks, dbonadim, saabm, arshig}@amazon.com
aWS AI Labs

Abstract

Planning is a crucial task for agents in task oriented dialogs (TODs). Human agents typically resolve user issues by following predefined workflows, decomposing workflow steps into actionable items, and performing actions by executing APIs in order; all of which require reasoning and planning. With the recent advances in LLMs, there have been increasing attempts to use them for task planning and API usage. However, the faithfulness of the plans to predefined workflows and API dependencies, is not guaranteed with LLMs. Moreover, workflows in real life are often custom-defined and prone to changes; hence, adaptation is desirable. To study this, we propose the problem of faithful planning in TODs that needs to resolve user intents by following predefined flows and preserving API dependencies. To solve this problem, we propose **FLAP**, a **Fl**ow-**A**dhering **P**lanning algorithm based on constrained decoding with lookahead heuristic for LLMs. Our algorithm alleviates the need for finetuning LLMs using domain specific (plan/dependency) data, enables quick adaptation to predefined flows, and outperforms other decoding and prompting-based baselines. Further, our algorithm empowers smaller LLMs ($\approx$ 7B) to perform at par larger LLMs ($\approx$ 30B-40B).

1 Introduction

A successful task oriented dialog (TOD) often involves interaction with internal or external world through tools or APIs. For example, as shown in Fig. 1, booking a flight through a conversation may involve searching for flights, getting payment information, and so on. API usage is a difficult task for chatbot agents as it involves various reasoning and planning. Beyond predicting the correct API to use in a particular scenario, it requires task decomposition and resolution of inter-API dependencies (e.g., searching for flights requires extracting airport codes) (Zhang et al., 2021; Prasad et al., 2023; Wang et al., 2023; Lu et al., 2023b). Moreover,
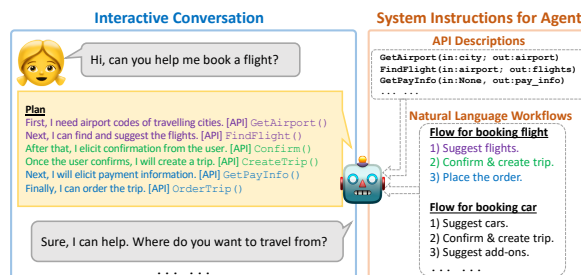

Figure 1: An agent-user interaction scenario. Agent plans how to resolve the user query by following given natural language workflows and API dependencies.

there may exist customized workflows to follow for API usage, such as "confirm before placing order" or "recommend add-ons before checking out", which can conveniently be defined using natural language (NL) instructions.

Recent advances in LLMs have enabled researchers to consider LLMs for task-planning (Huang et al., 2022; Valmeekam et al., 2023) and API usage (Qin et al., 2023b; Patil et al., 2023). In planning, studies have primarily focused on open-ended scenarios and emphasized mostly on goal attainment (Shridhar et al., 2020; Jansen, 2020) without considering any instructions on the process (Valmeekam et al., 2024), while works on API usage have mostly considered the appropriateness of API calls (Wang et al., 2023). However, considerations about API dependencies, and the faithfulness of generated API calls to predefined workflows or instructions are understudied.

In this paper, we propose a novel problem of zero-shot faithful planning with LLMs in TODs. As input, we provide (1) the domain definition in terms of APIs and their descriptions, (2) NL flows to fulfill various intents in the domain, and (3) a user query. The task is to generate a plan that fulfills the user's query by using the domain-specific APIs while adhering to the specified flows and API dependencies (§2).

Prior works attempted to teach LLMs to use

517

APIs by pretraining them with large amount of API related data (Schick et al., 2023; Yang et al., 2023; Qin et al., 2023b). However, in practice, API signatures and the NL flows to use them are customized to use cases and are prone to change (e.g., due to change in business policy). Hence, quickly adapting the LLMs to new flows and API dependencies is required. Unfortunately, pretraining LLMs cannot guarantee faithful planning with custom-defined APIs and flows which may significantly differ from the pretraining data (Blodgett et al., 2020; Bommasani et al., 2021); we experimentally verify this in §4.1. To alleviate the issue, we propose **FLAP**, a **F**low **A**dhering **P**lanning algorithm that uses constrained decoding of LLMs with lookahead heuristic (Och et al., 2001; Haghighi et al., 2007; Lu et al., 2022a; Huang et al., 2024) (§4.2). FLAP generates faithful plans to predefined flows and API dependencies, requires no pretraining, and is applicable on top of any autoregressive LLM regardless of its pretraining mechanism.

Given previous studies have shown that thoughts enhance the reasoning capabilities of LLMs (Wei et al., 2022) (even in the context of API usage (Yao et al., 2022)), FLAP prompts the LLMs to generate plans consisting of thoughts and corresponding APIs. FLAP aligns the generated thoughts to predefined NL flows, enabling thoughts to act as an intermediary (and address the vocabulary mismatch) between NL flows and APIs. In addition, FLAP captures the relations among NL flow steps and APIs using dynamic dependency graphs; these are used to score different dimensions related to API and flow dependency adherence in the lookahead heuristic function. Finally, FLAP also enforces structural constraints using lookahead heuristics.

To quantify the efficacy of current LLMs with promising prompt strategies, and our proposed approach for *faithful planning*, we construct a novel dataset comprising 4 domains, 13 intents and flows, 64 APIs, and 260 user queries, resembling real life use-cases (§3). We find that existing open-source LLMs and prompting techniques lack faithful planning capabilities and our proposed constrained decoding based algorithm, FLAP, improves the faithfulness of plans by reducing various errors, such as API and flow step dependency violations, redundant/lacking API in generated plan, etc. We also demonstrate that smaller LLMs (7B) with FLAP can achieve comparable performance against much larger LLMs (40B) (§5).

2 Task Formulation

In this section, we define the novel task of faithful planning in task oriented dialogues (TODs) to resolve queries by adhering to predefined natural language (NL) flows and API dependencies.

2.1 TOD Agent Scenario

In a typical TOD system, an agent is equipped with the following elements.

Domain: An agent usually has access to a specific domain, $\mathcal{D}$ (e.g., trip planning, banking, and so on).

Intents: In a domain $\mathcal{D}$, an agent has capabilities to handle a set of intents, $\mathcal{I}$. For example, an agent form the domain "trip planner" may only be able to book a hotel, search flights, and so on, however, unlikely to be able to create a bank account.

Flows: Agents are often provided with a sequence of natural language instructions, defined as "flows", $\mathcal{F}$ (e.g., "recommend add-ons before checking out"). Agents are expected to follow these flows *step-by-step* to resolve intents. Flows are typically defined by businesses and are prone to change.

APIs: Agents have access to a set of internal APIs, $\mathcal{A}$, to interact with the system (typically through GUI) and trigger actions based on the inputs from the user. For example, if a user is looking to book a car in Chicago, the agent may trigger the API, `FindRentalCar(location="Chicago")`.

2.2 Faithful Planning for Resolving Intent

In real life, when a user expresses an intent, an agent has to execute a sequence of APIs to fulfil it. Successful fulfillment of the user intent depends largely on the reasoning and planning capability of the agents because of the following challenges.

Identifying and following NL flows: Given an intent in $\mathcal{I}$, the execution steps need to be strictly consistent with a predefined flow in $\mathcal{F}$. Hence, the agents need to first identify the correct flow in $\mathcal{F}$ and then adhere to it.

Task decomposition and dependency resolution: APIs in $\mathcal{A}$ may have dependencies on other APIs in $\mathcal{A}$, and a flow step may require calling multiple APIs. For example, `GetAirports(city="Chicago")` is needed to be called before `FindFlight(airport="ORD")`, to execute the flow step "suggest flights". Hence, the agents have to decompose flow steps and preserve API dependencies.

Grounded in such challenges, we define the novel task of faithful planning for TOD agents (Fig.

1). Given, a domain $\mathcal{D}$, a set of flows $\mathcal{F}$ (for resolving intents $\mathcal{I}$), a set of APIs $\mathcal{A}$ (with parameters and descriptions), and a user query $\mathcal{Q}$; the task is to generate a plan $\mathcal{P}$ consisting of a sequence of APIs to fulfil the user query while obeying the flow steps in $\mathcal{F}$ and preserving API dependencies in $\mathcal{A}$.

In real world scenarios, agents may need to dynamically update the plan based on the situation (say, the user starts with query $\mathcal{Q}$ and then wants to do $\mathcal{Q}'$ as well). As a first step towards faithful planning in TODs, we study static planning (user does not change mind) in this paper, and leave dynamic (run-time) planning as a future work.

2.3 Zero-shot Planning with LLMs

We study the capabilities of LLMs in zero-shot planning by following predefined flows and preserving API dependencies in TODs. For that, we instruct a base LLM to act as a customer care agent and in context, we provide the domain definition consisting of APIs, their input-output parameters and descriptions, and NL flows for resolving intents in the domain. Given a user query in context, the LLM is instructed to plan how to resolve the query by using the given APIs and following the appropriate flow. The LLMs are expected to resolve API dependencies from their input-output parameters. The prompt structures are shown in §B. We study the problem in the following settings by varying the nature and complexity of the task.

All *vs*. relevant flow in-context: We experiment by providing either all flows or only the flow related to the user query in context. The former setting is more complex as the LLM has to first identify the relevant flow and then follow it to plan.

Reasoning and acting: Recent studies have shown that step-by-step thinking and acting improves the end goal achievement (Wei et al., 2022; Yao et al., 2022). Inspired by these works, we study the problem of plan generation in a setting where the LLMs are prompted to generate a thought and then an API in the plan to help it rationalize its selection of the API.

2.4 Evaluation Metrics

Number of edits to fix the plan: A generated plan is required to contain *only* the steps and APIs from the gold plan. We define number of edits as the sum of additions and deletions of steps/APIs in the generated plan to make it contain the same steps/APIs as the gold plan. The lower the number of edits, the better the generated plan.

Domains:	Trip Booking	Insurance	Banking	Restaurant & Ride Book
# of intents/flows	3	3	3	4
# of steps per intent/flow	5-6	4-5	3-5	4
# of APIs needed per step	1-2	1-2	1-3	1-4
# of APIs in domain	13	15	14	22
# of relationships among APIs	13	13	15	19
# of user queries	60	60	60	80

Intents in all domains			
Trip Booking book car, book hotel, book flight	**Insurance** buy insurance, cancel insurance, add member	**Banking** open account, report problem, cancel transaction	**Restaurant & Ride Book** book restaurant, book ride, cancel restaurant, cancel ride

(a) Created domains statistics.

Flow steps	Required API sequence (gold plan)
Suggest cars to the customer	`FindRentalCar()`
Confirm and create the trip	`Confirm(), CreateTrip()`
Extract and add promotional offers	`GetCarInsuranceDiscount(), UpdateTrip()`
Order the trip	`GetPaymentInformation(), OrderTrip()`

(b) Flow and required APIs for the intent "book car" in the domain "Trip Booking". The full list can be found in §A.1.

Table 1: Test data statistics (a) and example flow (b).

Inconsistent steps/APIs: A generated plan is required to obey flow steps and API dependencies. Hence, we measure the percentage of steps or APIs in the generated plan that do not follow the dependency structure (e.g., a step/API is generated before its parent steps/APIs).

Fine-grained metrics: We quantify other fine-grained metrics such as API hallucination, API repetition and parsability of the generated plans.

3 Data Collection

In this section, we describe the data collection procedure for studying faithful plan generation.

Domain Creation. Previous datasets have developed domains, flows, and APIs for tackling intents in TODs. We find, none of the existing datasets is usable as it is in our problem. For example, the STAR (Mosig et al., 2020) and STARv2 (Zhao et al., 2023) datasets contain APIs (without flows) to resolve user queries, however, dependencies among the APIs are sparse, e.g., STAR contains only 10 dependencies among 25 APIs. In real life, API dependencies are more dense, hence, these datasets are not very useful for our study where the goal is to study the ability of LLMs to preserve dependencies. The ABCD dataset (Chen et al., 2021) contains flows for a few tasks, however, the related APIs needed to perform the steps in the flows are not given, hence, it is also not suitable for our study. Hence, inspired from these two datasets, we create in total, 4 domains, 13 flows (similar to ABCD) for resolving 13 distinct intents in these domains, and 64 APIs (similar to STAR) to execute different flow steps. We also annotate the gold plans to

	With Thought	All flows in-context					Only relevant flow in-context				
LLMs		Avg % of API Hallu.	Avg # of edit for		Avg % of inconsistent		Avg % of API Hallu.	Avg # of edit for		Avg % of inconsistent	
			Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)		Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
`santacoder-1.1b`	No	1.2%	30.3	25.9	15.4%	33.1%	1.8%	29.6	25.8	15.3%	34.1%
	Yes	13.0%	11.8	13.0	18.2%	34.1%	20.0%	9.5	11.5	19.5%	31.2%
`toolAlpaca-7b`	No	10.8%	4.6	5.1	20.4%	49.6%	7.0%	4.4	4.7	17.8%	49.2%
	Yes	19.0%	4.7	5.3	17.6%	44.8%	8.0%	3.0	3.5	10.6%	47.6%
`falcon-7b-inst.`	No	6.1%	47.5	40.3	16.6%	27.7%	6.6%	40.8	38.1	17.4%	23.9%
	Yes	19.0%	9.7	11.4	24.0%	38.8%	9.0%	9.7	9.2	21.5%	37.9%
`mpt-7b-inst.`	No	0.2%	16.2	13.4	14.7%	33.3%	0.2%	13.6	11.5	13.5%	35.6%
	Yes	4.0%	13.9	12.4	11.9%	29.6%	3.0%	9.9	8.9	9.1%	33.4%
`mistral-7b-inst.`	No	2.4%	2.9	3.0	10.6%	37.2%	1.4%	3.1	3.1	8.0%	30.6%
	Yes	0%	2.8	2.8	3.1%	40.3%	1.0%	2.7	2.6	2.8%	34.4%
`koala-13b`	No	6.3%	37.6	31.4	10.5%	24.0%	4.2%	29.4	23.8	11.3%	26.2%
	Yes	10.0%	8.8	9.1	9.0%	24.0%	9.0%	7.4	7.9	9.3%	22.6%
`vicuna-13b`	No	2.4%	4.1	4.3	15.1%	35.2%	2.4%	3.9	4.1	14.2%	31.7%
	Yes	4.0%	4.0	4.3	6.7%	35.5%	2.0%	3.1	3.4	5.4%	35.0%
`llama-13b`	No	2.4%	26.8	20.9	13.2%	31.7%	2.0%	24.7	19.1	11.3%	29.4%
	Yes	4.0%	9.2	8.7	4.7%	31.0%	3.0%	6.6	5.6	4.0%	38.4%
`mpt-30b-chat`	No	0.5%	3.3	3.4	9.1%	33.4%	0.2%	3.5	3.3	6.3%	33.3%
	Yes	1.0%	2.7	2.7	5.7%	32.5%	1.0%	2.1	2.1	4.6%	34.3%
`falcon-40b-inst.`	No	0.9%	8.1	7.2	18.0%	38.3%	0.9%	9.0	7.9	17.9%	35.5%
	Yes	5.0%	5.4	5.4	9.0%	36.4%	7.0%	4.0	4.1	8.7%	25.7%

Table 2: Zero-shot plan generation results using prompting-based approaches with Greedy decoding (average over all plans). Results with other metrics and standard deviation can be found in §D.

resolve the 13 intents. The statistics of the dataset and examples are presented in Tab. 1.

Test Query Generation. We aim to study plan generation in response to a user query in our created domains. To collect diverse user utterances related to the intents in Tab. 1a, we utilize the generative power of pretrained LLMs. We prompt an open-source LLM, `Falcon-40B-instruct` (Almazrouei et al., 2023)[1], to generate user queries related to the intents. We prompt the LLM to write paraphrases of a query with a specific intent. Prompt structure and generated examples are shown in §A.2. We manually go through the generated queries and discard any queries that are not related to the intent in the prompt. We find that 94% of the generated queries are correct. In this manner, we generate 20 user queries per intent resulting in 260 queries in total. We use this dataset for studying flow-constrained plan generation given a user query.

4 Constrained Decoding for Planning

In this section, first, we run an ablation on plan generation in various settings as explained in §2.3, to understand the complexity of the task. Then we explain our proposed constrained decoding based algorithm for faithful plan generation.

---

[1] We experimented with other open source LLMs and found `Falcon` to be the best in generating diverse utterances.

4.1 Ablation

To understand the complexity of zero-shot faithful planning in different settings, we run a zero-shot prompting-based ablation with a number of open-source LLMs pretrained with/without tools and of different parameter size. We use `Santacoder` (Allal et al., 2023), `Falcon` (Penedo et al., 2023), `Mpt` (Team, 2023), `Mistral-v0.1` (Jiang et al., 2023), `Koala` (Geng et al., 2023), `Vicuna` (Chiang et al., 2023), `Llama` (Touvron et al., 2023), `ToolAlpaca` (Tang et al., 2023), and `ToolLlaMA` (Qin et al., 2023b). The prompt structures for generating plans with and without thoughts are shown in §B. The results for plan generation in the two settings are summarized in Tab. 2. Our main observation is that all LLMs fail heavily in zero-shot planning regardless of the setting, especially LLMs fail to preserve the API and flow step dependencies in plans. For example, the average percentage of inconsistent APIs in the two settings, ranges from 22.6% to 49.6%. This finding is in line with previous studies on LLMs' capability in task planning and tool usage (Ruan et al., 2023; Valmeekam et al., 2022). We categorize our main observations below.

Effect of thoughts. Performance improves in flow faithfulness for almost all LLMs when prompted to generate thoughts and APIs compared to when prompted to generate only APIs in plan. This is consistent with the findings of Yao et al. (2022). However, with thoughts, API repetition and halluci-
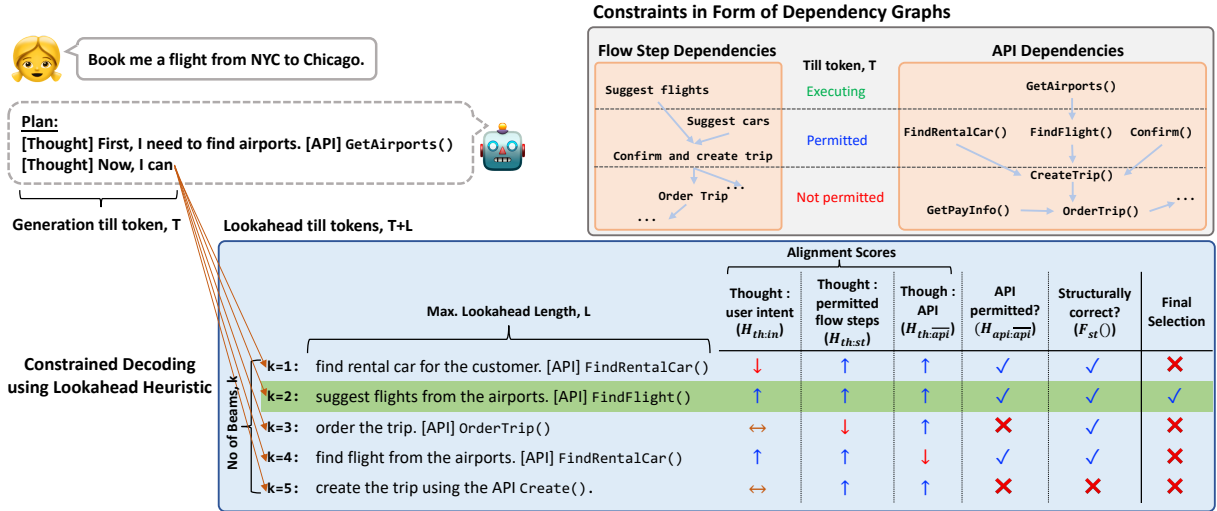
Figure 2: An example state of FLAP, our proposed lookahead heuristic-based constrained decoding algorithm for faithful planning, in the domain "Trip Planning" for a query related to "book flight". Here, $\uparrow$, $\downarrow$, and $\leftrightarrow$ indicate high, low, and mediocre alignment scores, respectively. The selected path based on the heuristic scores is highlighted.

nation increase and plan parsability (§D) decreases a bit. On some test examples, we notice that the LLMs enter into a repetitive loop with thoughts (examples in §E.1), in line with well-known phenomenon in greedy and beam-search decoding (Vijayakumar et al., 2016; Shao et al., 2017).

Effect of parameter size. Performance in most of the metrics improve in larger LLMs and smaller LLMs are prone to hallucination. We observe that different models of the same parameter size perform differently. For example, the 7B version of MPT is better than the 7B version of Falcon in most of the metrics and Mistral-7B is better than other 7B models and it performs as per 30B-40B models.

Effect of pretraining. We observe that LLMs pretrained on tools, e.g., ToolAlpaca fails heavily in API consistency and results in high API hallucination. With ToolLlama, we find that it fails to follow instructions and generates non-parsable responses (hence, not reported). These findings suggest that pretraining LLMs with tool data makes them heavily biased towards the pretraining data.

All flows vs. relevant flow in prompt. When only the relevant flow to the user query is given in-context, the performance improves with almost all LLMs indicating that LLMs are more confused when multiple flows are given in-context.

4.2 Proposed Algorithm

We now introduce **FLAP**, a **F**low **A**dhering **P**lanning algorithm based on constrained decoding with lookahead heuristic (Fig. 2). In FLAP, we first convert the flow steps and APIs to dependency

graphs. At decoding time, we keep track of these dependency graphs to enforce the dependencies as constraints during next token generation.

4.2.1 Constructing Dependency Graphs

For each domain, we construct dependency graphs for flow steps and APIs. The API dependency graph is constructed based on the input-output parameter dependencies among APIs. For example the API FindFlight(inputs:airport; outputs:flights) is dependent on the API GetAirport (inputs:city; outputs:airport) for the input parameter 'airport'. Hence, in the API dependency graph 'GetAirport' will be a parent of 'FindFlight'. Similarly for flow steps, we construct a separate flow-step dependency graph. While generating a plan, we maintain a list of the steps and APIs that are executed in these dependency graphs. At each planning step, only non-executed/under-execution APIs/steps in the dependency graphs with parents in the executed list are considered permitted.

4.2.2 Constrained Decoding of Plans with Lookahead Heuristic

The traditional text generation problem can be formulated as, given an input sequence of tokens $x = \{y_1, y_2, \cdots, y_{t-1}\}$, the next token generator predicts the next token y_t by solving the equation:

$$y_t = \underset{y_t' \in Y}{\arg\max} \, P(y_t'|x)$$

Here, Y is the set of all possible generations and $P(y_t'|x)$ is the likelihood of the token y_t' given the

521

input sequence x. In constrained decoding, the likelihood function takes the form $P'(y'_t|x) = P(y'_t|x) + H(y'_t|x)$, where $H(y'_t|x)$ is the score based on constraint satisfaction by generating y'_t. However, often it is difficult to determine the constraint satisfaction score by only looking at the next token. The lookahead mechanism comes handy here where the constraint satisfaction score for a candidate next token, y'_t is estimated by looking at the estimated future generations if y'_t is selected. Hence, instead of $H(y'_t|x)$, the heuristic function takes the form $H(y'_t, \cdots, y'_{t+L}|x)$, where L is lookahead length.

For constrained plan generation, we prompt LLMs to generate thoughts and corresponding APIs. During generation, for each candidate next token, we lookahead till the end of one thought+API and score the candidate based on their constraint satisfaction. Hence, our heuristic function takes the form $H_c = H([thought][api]|x)$. $[thought][api]$ may contain some already generated tokens; we also consider those when scoring. We score each $[thought][api]$ in the aspects below.

Generated thought to permitted flow step alignment ($H_{th:step}$). In our approach, thoughts are natural language sentences describing the next step to be taken. Ideally, they should correspond to the flow steps. Hence, we score the generated thoughts based on their semantic similarity to the permitted flow steps in the flow dependency graph. A flow step is permitted, if all of its parent flow steps are already executed or the flow step is under execution (e.g., flow step consisting of multiple API calls). First, we estimate the intended flow step, $\hat{s}$ by a thought, th by obtaining its semantically closest flow step using the formula below.

$$\hat{s} = \arg\max_{s \in S} sim(th, s)$$

Here, S is the set of all flow steps in the domain and $sim()$ is semantic similarity scorer. Then we calculate $H_{th:step}$ using the following equations.

$$H_{th:step} = \begin{cases} \alpha \times sim(th, \hat{s}), & \text{if } \hat{s} \text{ in } S_p \\ 0, & \text{if } \hat{s} \text{ in } S_n \end{cases}$$

$$\alpha = \begin{cases} \alpha_a, & \text{if } \hat{s} \text{ is in a flow} \\ \alpha_b, & \text{if } \hat{s} \text{ is deviating from flow} \\ \alpha_c, & \text{if } \hat{s} \text{ is same as the last step} \end{cases}$$

Here, S_p and S_n are permitted and not permitted flows steps at the current thought, respectively. We assign different weights to $H_{th:step}$ based on the

intended flow step $\hat{s}$. If the intended flow step $\hat{s}$ belongs to the flow that the model has been following so far, we assign the weight α_a. If the model deviates from the flow it has been following, we assign the weight α_b. In single intent use-cases, the model is required to follow only one flow, hence, setting $\alpha_a > \alpha_b$ prevents the model from deviating from the flow it has been following. If the model generates a flow step that is the same as the step it intended in its previous thought, we assign a weight α_c. One step may involve executing multiple APIs. Hence, setting $\alpha_c > \alpha_a$ encourages the model to complete the current step before moving to the next.

Generated API to permitted APIs alignment ($H_{api:\overline{api}}$). We score the generated API based on their alignment to the permitted APIs. Permitted APIs at a step are the non-executed APIs whose parents are already executed. We calculate $H_{api:\overline{api}}$ using the following equations.

$$\hat{a} = \arg\max_{a \in A} sim(a, \overline{a})$$

$$H_{api:\overline{api}} = \beta \times sim(\overline{a}, \hat{a})$$

$$\beta = \begin{cases} [0,1), & \text{if } \hat{a} \notin A_p \cup A_c \text{ or } \overline{a} \notin A \\ 1, & \text{if } \hat{a} \in A_p \\ 0, & \text{if } \hat{a} \in A_c \end{cases}$$

Here, $\overline{a}$ is the generated API, A is the set of all APIs in a domain, A_p and A_c are permitted APIs and already called APIs, respectively, and $A_p \cap A_c = \emptyset$. When APIs from none of the sets A_p and A_c are generated, the API is either hallucinated or generated from the non-permitted list (A_n). In both of the cases, we assign a lower weight than 1 in the alignment score to make it a soft constraint while making it zero will enforce a hard constraint against such cases. We keep this weight open to tuning for different LLMs because of their varying degree of hallucinations and errors.

Generated thought to user intent alignment ($H_{th:in}$). The plan should correspond to the user intent in the query. Hence, we score the generated thought, th based on its semantic similarity with the user query, in, using, $H_{th:in} = sim(th, in)$.

Generated thought to generated API alignment ($H_{th:\overline{api}}$). We generate thought to resonate the API usage. Hence, the generated API, $\overline{a}$ should be relevant to the generated thought, th. We align the generated thought to the generated API using the score, $H_{th:\overline{api}} = sim(th, \overline{a}_d)$, where $\overline{a}_d$ is the given

LLMs	Decoding Strategy	With Thought	Avg count per plan		% Plan Parsable (↑)	Avg % of APIs in plan		Avg # of edit for		Avg % of inconsistent	
			Thoughts	APIs		Repeat (↓)	Hallu. (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
mpt-7b-instruct — Baselines	Greedy Search	No	-	17.3	100%	11.6%	0.2%	16.2	13.4	14.7%	33.3%
	Greedy Search	Yes	16.2	16.1	97.7%	35.1%	4.0%	13.9	12.4	11.9%	29.6%
	Beam Search	Yes	9.9	9.9	70.8%	5.5%	3.0%	7.6	7.7	13.9%	41.0%
	Nucleus Sampling	Yes	12.3	12.4	76.9%	25.3%	4.0%	9.8	9.5	13.8%	31.7%
mpt-7b-instruct — Our model	FLAP.1 [soft api + align thought to (api, intent)]	Yes	10.3	10.4	100%	1.3%	0.0%	5.5	5.5	6.3%	1.3%
	FLAP.2 [soft api + align thought to (step, api)]	Yes	9.3	9.4	100%	3.0%	0.0%	5.7	5.6	4.3%	1.6%
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	9.2	9.4	100%	1.0%	0.0%	5.0	5.0	6.1%	1.3%
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	9.3	9.4	100%	3.6%	0.0%	5.5	5.4	4.8%	2.1%
mistral-7b-instruct — Baselines	Greedy Search	No	-	4.8	100%	0.3%	2.4%	2.9	3.0	10.6%	37.2%
	Greedy Search	Yes	4.7	4.8	99.6%	2.9%	0.0%	2.8	2.8	3.1%	40.3%
	Beam Search	Yes	3.6	3.8	98.1%	0.2%	0.0%	2.9	3.0	2.4%	38.9%
	Nucleus Sampling	Yes	5.7	5.9	93.9%	5.8%	2.0%	3.0	3.0	7.5%	34.3%
mistral-7b-instruct — Our model	FLAP.1 [soft api + align thought to (api, intent)]	Yes	6.6	6.7	100%	0.0%	0.0%	2.5	2.5	6.3%	4.6%
	FLAP.2 [soft api + align thought to (step, api)]	Yes	6.1	6.1	100%	0.1%	0.0%	2.3	2.3	5.1%	5.5%
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	6.3	6.4	100%	0.4%	0.0%	2.0	2.0	6.5%	3.6%
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	6.4	6.4	100%	0.5%	0.0%	2.5	2.5	5.6%	7.7%

(a) Setting: All flows in a domain are present in-context.

LLMs	Decoding Strategy	With Thought	Thoughts	APIs	% Plan Parsable (↑)	Repeat (↓)	Hallu. (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
mpt-7b-instruct — Baselines	Greedy Search	No	-	14.9	100%	8.5%	0.2%	13.6	11.5	13.5%	35.6%
	Greedy Search	Yes	12.4	12.3	98.9%	23.0%	3.0%	9.9	8.9	9.1%	33.4%
	Beam Search	Yes	8.0	8.0	81.5%	4.2%	2.0%	5.2	5.3	10.8%	41.4%
	Nucleus Sampling	Yes	10.0	10.3	75.0%	19.8%	5.0%	7.2	7.1	13.6%	33.0%
mpt-7b-instruct — Our model	FLAP.1 [soft api + align thought to (api, intent)]	Yes	9.6	9.6	100%	1.1%	0.0%	4.7	4.7	6.6%	1.2%
	FLAP.2 [soft api + align thought to (step, api)]	Yes	8.2	8.2	100%	0.9%	0.0%	3.5	3.4	5.1%	1.2%
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	8.8	9.0	100%	1.2%	0.0%	4.2	4.2	6.3%	2.4%
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	8.6	8.7	100%	3.0%	0.0%	3.7	3.4	5.6%	3.7%
mistral-7b-instruct — Baselines	Greedy Search	No	-	5.9	100%	0.6%	1.4%	3.1	3.1	8.0%	30.6%
	Greedy Search	Yes	5.6	5.7	100%	3.8%	1.0%	2.7	2.6	2.8%	34.4%
	Beam Search	Yes	4.2	4.3	100.0%	0.0%	0.0%	2.4	2.5	1.5%	36.7%
	Nucleus Sampling	Yes	5.6	5.9	98.1%	3.7%	1.0%	2.9	3.0	6.78%	30.2%
mistral-7b-instruct — Our model	FLAP.1 [soft api + align thought to (api, intent)]	Yes	7.0	7.0	100%	0.1%	1.0%	2.7	2.8	6.0%	4.1%
	FLAP.2 [soft api + align thought to (step, api)]	Yes	6.6	6.6	100%	0.4%	1.0%	2.3	2.3	5.9%	7.0%
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	6.7	6.8	100%	0.4%	0.0%	2.3	2.3	6.6%	4.3%
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	6.4	6.4	100%	0.4%	1.0%	2.2	2.2	6.1%	8.7%

(b) Setting: Only relevant flow to test query is present in-context.

Table 3: Comparison of our proposed approach, FLAP, with baselines in zero-shot faithful plan generation. Here, the numbers (FLAP.#) indicate different ablation versions of FLAP. Structural constraint is applied in all versions of FLAP. Results with standard deviations can be found in Tables 13 and 14 in §D. As reference for the "Avg count per plan" column, we note, there are 6.84 APIs per gold plan.

description of the generated API, $\overline{a}$. When $\overline{a}$ is a hallucination, we set $\overline{a}_d = \overline{a}$.

Structural constraint (F_{st}). For practical use-cases, it is convenient if the LLMs generate plans in a pre-defined format. It helps automatic parsability of the generated plans. Hence, we introduce a function, F_{st} that accounts for the structural consistency of the generated plan. It operates on the combined heuristic score, $H'_c = a \times H_{th:step} + b \times H_{api:\overline{api}} + c \times H_{th:in} + d \times H_{th:\overline{api}}$ (a, b, c, d are scaling factors) as follows.

$$F_{st}(H'_c) = \begin{cases} H'_c, & \text{if thought+API follows format} \\ 0, & \text{otherwise} \end{cases}$$

We use $H_c = F_{st}(H'_c)$ as the final heuristic score for a candidate token, y'_t and the final scoring function for next token selection is defined as follows.

$$S(y'_t|x) = (1 - \lambda) \times P(y'_t|x) + \lambda \times H_c^{y'_t}$$

The parameter λ controls how much weight is given in the LLM logits, $P(y'_t|x)$, and the heuristic score $H_c^{y'_t}$. We use top-k beams based on $P(y'_t)$ for the next token generation using constrained decoding. An example of our approach is shown in Fig. 2.

5 Experimental Evaluation

5.1 Experimental Setting

We evaluate FLAPs performance in faithful planning on the dataset described in §3, by comparing it with prompting-based baselines and different decoding methods. Note that, the approach with thoughts is similar to ReAct (Yao et al., 2022). Other approaches either depend on multiple modules, require finetuning, or tuned for a different task setting, hence, not directly applicable in our task. Based on our ablation (in Tab. 2), we apply FLAP on two LLMs in 7B parameter range, Mistral-7b-instruct and Mpt-7b-instruct. We leave application of FLAP on larger LLMs as future work because of their high resource requirement. We demonstrate the structure of plan, ([thought]...[API]...), through dummy start steps in the prompts (Fig. 6, §B) and enforce it through F_{st} during decoding. We use a pretrained sentence transformer (Reimers and Gurevych, 2019) to calculate semantic similarities ($sim()$). We report hyper-parameter tuning and other experimental details in §C.

5.2 Experimental Results

The experimental results are summarized in Table 3. First, we observe that the Beam Search and Nucleus Sampling (Holtzman et al., 2019) methods do not improve the overall performance compared to the Greedy Search method. Rather both Beam Search and Nucleus Sampling result in fewer number of parsable plans compared to Greedy Search. Note that Beam Search and Nucleus Sampling explore a wider search space compared to Greedy Search, and Nucleus Sampling increases diversity; we observe that without constraints, it results in more deviation from the instructions and often decreases performance in different metrics. Hence, in the rest of this section, we compare our models performance with "Greedy Search with thoughts". We also observe that different baselines and our approach show similar comparative trends in the two settings (all flows in context vs. relevant flow in context). Hence, we focus our rest of the discussion on the more difficult setting where all flows are given in the prompt (Tab. 3a).

In case of Mpt, different versions of our algorithm, FLAP, outperform the baselines by a great margin. In case of Mistral, the average percentage of inconsistent steps increases a bit (by 2-4%) with FLAP, however, performance improves in all other metrics, especially the inconsistent APIs decreases from 40.3% to 3.6%. A similar trend is observed in the relevant flow in-context setting (Tab. 3b).

We observe that Mistral is good at following flows even without FLAP in the difficult all flows in-context setting, however, it fails to decompose when one flow step requires multiple API calls. Hence, without FLAP, the average API count per plan is lower (4.8) than the average number of APIs in gold plans (6.84). As a result, the avg. % of inconsistent steps is low (3.1%), however, % of inconsistent API is very high (40.3%). With FLAP, the decomposition capability improves and the model generates plans with number of APIs (6.1-6.7) closer to the gold reference value (6.84) with fewer number of edits and much less inconsistent APIs (3.6-7.7%), however, with a trade off of slightly increased (by 2-4%) inconsistent steps. We present qualitative examples in Tab. 4.

We perform ablation by dropping different components of the heuristic scoring function in FLAP. We observe, different components of the heuristic score affect the corresponding aspects they are designed for, especially in the difficult setting where

Query: Can you book a flight from Boston to San Francisco?
Generated Plan by Mistral-7b-instruct Without FLAP
[thought] To suggest flights, I need to find flights from Boston to San Francisco. [API] FindFlight()
[thought] Once I have the flight details, I can confirm and create the trip. [API] Confirm()
[thought] After confirming the trip, I can order the trip. [API] OrderTrip()
Generated Plan by Mistral-7b-instruct With FLAP
[thought] First, I need to find the airport codes for Boston and San Francisco. [API] GetAirports()
[thought] Once I have the airport codes, I can suggest flights to the customer. [API] FindFlight()
[thought] After suggesting flights, I need to confirm the trip details with the user. [API] Confirm()
[thought] Once the trip details are confirmed, I can create the trip. [API] CreateTrip()
[thought] To order the trip, I need to get the payment information from the user. [API] GetPayInfo()
[thought] After getting the payment information, I can order the trip. [API] OrderTrip()

Table 4: Generated plans by Mistral. It often fails in step decomposition without FLAP.

all flows are present in-context. For example, in case of Mpt, dropping $H_{th:step}$ (thought to flow step alignment) results in higher step dependency violation (FLAP.1). Another component $H_{th:in}$ (thought to intent alignment) is designed to keep the model focused on the flow related to the user intent and we observe an overall lower number of edits required when this component is added (FLAP.1, FLAP.3, FLAP.4), implying this component helps the model generate a plan related to the user intent. However, the effect of these components is not clear in case of Mistral. We conjecture, as Mistral is already good at following flows, the effect is lower. The effect of $H_{th:\overline{api}}$ that makes generated thoughts and APIs coherent, is more qualitative and we find that dropping these component (FLAP.3) results in inconsistent thought and API pairs even if the generated API is correct in the plan. Examples of such errors are presented in Tab. 5. Finally, enforcing structural consistency (F_{st}) during decoding makes the plans parsable 100% of the time.

In Fig. 3, we present the comparative performance of LLMs of different size and smaller 7B LLMs with FLAP. We observe that FLAP on top of Mistral-7b outperforms and Mpt-7b performs at par larger LLMs.

6 Related Work

While API usage and planning have been studied as separate concepts in literature, we argue that these two are interdependent. Recently, different approaches have been proposed to teach LLMs to use the correct API, however, the faithfulness of API usage to their dependencies is understudied (Ruan et al., 2023). For correct API usage, one line of research considers pretraining of LLMs with large API data that are mostly augmented using self-instruction (Schick et al., 2023; Patil et al., 2023; Qin et al., 2023b; Yang et al., 2023; Gao et al., 2023; Tang et al., 2023; Li et al., 2023). In some cases, authors considered modular approaches and

Query: I need to book a flight from NYC to Chicago.	
Generated plan with FLAP with thought to API alignment	**Generated plan with FLAP without thought to API alignment**
[thought] I can get nearby airport codes to suggest flights to the customer. [API] GetAirports()	[thought] The customer wants to book a flight from NYC to Chicago. [API] GetAirports()
[thought] I can suggest flights to the customer. [API] FindFlight()	[thought] I can suggest flights to the customer. [API] FindFlight()
[thought] I can confirm flights to the customer and create a trip. [API] Confirm()	[thought] I have suggested flights to the customer. [API] GetPayInfo()
[thought] I confirm flights to the customer and create a trip. [API] CreateTrip()	[thought] I can confirm and create the trip. [API] Confirm()
[thought] Before ordering the trip, I can get payment information from the customer. [API] GetPayInfo()	[thought] Confirms and create the trip. [API] CreateTrip()
[thought] I confirm flights to the customer and order a trip. [API] OrderTrip()	[thought] The trip is created. [API] OrderTrip()

(a) Generated plan with Mpt-7b-instruct.

[thought] In order to suggest flights to the customer, I need to get the airport codes. [API] GetAirports()	[thought] In order to suggest flights to the customer, I need to get the airport codes. [API] GetAirports()
[thought] Once I have the airport codes, I can suggest flights to the customer. [API] FindFlight()	[thought] Once I have the airport codes, I can suggest flights to the customer. [API] FindFlight()
[thought] After suggesting the flights, I need to confirm and create the trip. [API] Confirm()	[thought] After suggesting the flights, I need to confirm and create the trip. [API] Confirm()
[thought] To order the trip, I need to create a trip with the suggested flight. [API] CreateTrip()	[thought], [API] CreateTrip()
[thought] Before ordering the trip, I need to get the payment information from the customer. [API] GetPayInfo()	[thought] Once the trip is created, I can order the trip. [API] GetPayInfo()
[thought] Once I have the payment information, I can order the trip. [API] OrderTrip()	[thought], order the trip. [API] OrderTrip()

(b) Generated plan with Mistral-7b-instruct.

Table 5: Qualitative error analysis when the generated thought to generated API alignment component ($H_{th:\overline{api}}$) in the heuristic scoring function of FLAP is dropped. The models results in incoherent thoughts and APIs at steps if $H_{th:\overline{api}}$ is dropped; such steps are highlighted in red.

relied on feedback from humans, other LLMs, or API execution outputs to tune LLMs (Song et al., 2023; Qin et al., 2023a; Qiao et al., 2023; Lu et al., 2023b; Prasad et al., 2023; Chen et al., 2023). Another line of work focused on improving the reasoning mechanism (Qian et al., 2023) for API usage via prompting to generate thoughts (Yao et al., 2022), consider API documentation (Hsieh et al., 2023), etc. Unfortunately, pretraining-based methods are difficult and prohibitively costly when adaptation to new APIs is required and prompting-based approaches cannot reliably improve reasoning and planning which are key to faithful API usage (§4.1).

In literature, planning is defined as decomposition of the end task and identifying intermediate executable actions or reasoning steps to achieve the goal where the decomposition is open-ended and does not require obeying instructions (Jansen, 2020; Huang et al., 2022; Ahn et al., 2022; Wu et al., 2022; Lu et al., 2022b; Yu et al.; Lu et al., 2023a; Brahman et al., 2023). However, in many applications, such as customer care service or physical robots, planning needs to strictly adhere to predefined instructions (Reijers, 2003; Sengupta et al., 2019). To achieve this, we draw inspiration from decoding approaches (Holtzman et al., 2019; Meister et al., 2020) that enable enforcement of lexical and parametric constraints during generation (Hokamp and Liu, 2017; Lu et al., 2021, 2022a; Krause et al., 2021; Yang and Klein, 2021; Huang et al., 2024). We extend the idea to enforce constraints related to predefined instructions.

7 Conclusion

In this paper, we study the novel problem of faithful planning for performing tasks in TODs, that
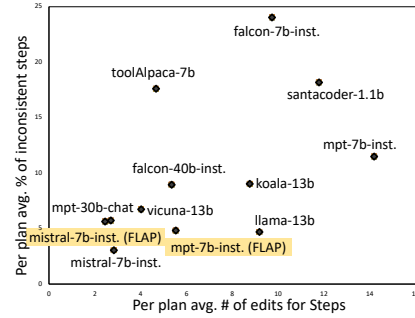
(a) Errors in generated flow steps.
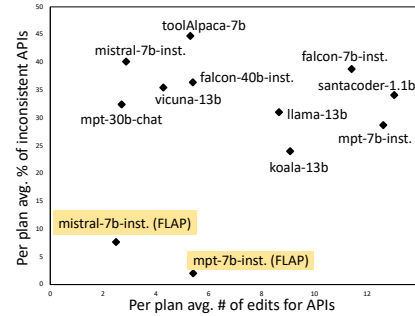
(b) Errors in generated APIs.

Figure 3: Comparison among different LLMs in flow step (top) and API (bottom) errors during plan generation in the setting where all flows are given in context.

adheres to predefined flows and preserves API dependencies. To solve the task, we propose **FLAP**, a constrained decoding algorithm based on lookahead heuristic. FLAP outperforms prompting-based baselines with different decoding methods and applying FLAP on two smaller LLMs we achieve comparable performance to larger ones. Our approach can be leveraged in accurate planning for various downstream applications such as next step prediction in conversations, user simulation, flow-following conversation data augmentation for training TOD models, and so on.

Limitations

We identify the following limitations of our study.

Simplifying Assumptions: In our study, we make the simplifying assumption that the user query can be resolved using the given APIs and workflows in context. However, that may not be the case in real life and users may have out-of-domain (OOD) queries. To tackle OOD intents, we can easily adapt our model by adding one additional workflow such as "cannot help", and instruct it to be selected if no other workflow is similar to the user query. Our constructed dataset also do not contain cases where multiple plans are valid based on the given workflows and APIs, however, our algorithm can be applicable in such cases without any modification.

Static Planning: In this paper, we study static planning. However, in real life, the agents may need to modify their plans in between a conversation because conversations are by nature dynamic. Some possible reasons for plan adaptation may be, based on API execution feedback, user changes their mind in between, and so on. Adapting such static plans based on the situation in dynamic conversations can be an interesting future work. We also focus only on API selection by following predefined flows and API dependencies. The correctness of parameter filling for the APIs is left as a future work as it is applicable mostly in the dynamic planning setting.

Runtime: We observe a higher runtime for plan generation using constrained decoding (§C.3). This is partly because of the inefficient deployment of the LLMs in the existing implementations (we use Huggingface implementations) and partly because of our resource constraint (discussed in details in §C.3). We intend to work on more efficient implementation of our algorithm by potentially parallelizing computations and implementing algorithmic optimizations (e.g., lookahead after every n tokens generation).

Usage of Open Source LLMs: We did not use any closed source LLMs for our research because of various reasons. Firstly, our algorithm requires the logits from the LLMs which may be difficult to obtain using closed source LLMs. Secondly, they might be changing inside the box and it will be difficult to replicate the results, hence, harming the benchmarking procedure. Thirdly, closed source LLMs are costly as they are accessible through paid APIs only. Due to resource constraint, we experimented with LLMs of at most 40B parameter size. However, as described in the paper, our approach can be applied on top of any autoregressive LLM.

Ethics Statement

In this paper, we present all implementation and dataset details to replicate the study (partially in the Appendix). All the relevant datasets, publicly available LLMs, and other models used in this paper are publicly available for scientific research and are cited adequately. All the results are reported with standard deviations and necessary error analyses are done (some parts in the Appendix), to provide the readers an idea about the potential error patterns and risks related to using our proposed models.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. 2023. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. Falcon-40B: an open large language model with state-of-the-art performance.

Su Lin Blodgett, Solon Barocas, Hal Daumé III, and Hanna Wallach. 2020. Language (technology) is

power: A critical survey of "bias" in NLP. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5454–5476, Online. Association for Computational Linguistics.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Faeze Brahman, Chandra Bhagavatula, Valentina Pyatkin, Jena D Hwang, Xiang Lorraine Li, Hirona J Arai, Soumya Sanyal, Keisuke Sakaguchi, Xiang Ren, and Yejin Choi. 2023. Plasma: Making small language models better procedural knowledge models for (counterfactual) planning. *arXiv preprint arXiv:2305.19472*.

Derek Chen, Howard Chen, Yi Yang, Alexander Lin, and Zhou Yu. 2021. Action-based conversations dataset: A corpus for building more in-depth task-oriented dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3002–3017.

Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Chatcot: Tool-augmented chain-of-thought reasoning on\\chat-based large language models. *arXiv preprint arXiv:2305.14323*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, and Jun Ma. 2023. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. *arXiv preprint arXiv:2308.14034*.

Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A dialogue model for academic research. Blog post.

Aria Haghighi, John DeNero, and Dan Klein. 2007. Approximate factoring for a* search. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 412–419.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. *arXiv preprint arXiv:1704.07138*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

James Y Huang, Sailik Sengupta, Daniele Bonadiman, Yi-an Lai, Arshit Gupta, Nikolaos Pappas, Saab Mansour, Katrin Kirchoff, and Dan Roth. 2024. Deal: Decoding-time alignment for large language models. *arXiv preprint arXiv:2402.06147*.

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

Peter Jansen. 2020. Visually-grounded planning without vision: Language models infer detailed plans from high-level instructions. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4412–4417.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. 2021. Gedi: Generative discriminator guided sequence generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4929–4952.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Apibank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023a. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, et al. 2022a. Neurologic a* esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 780–799.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Neurologic decoding:(un) supervised neural text generation with predicate logic constraints. In *Proceedings of*

the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4288–4299.

Yining Lu, Haoping Yu, and Daniel Khashabi. 2023b. Gear: Augmenting language models with generalizable and efficient tool resolution. *arXiv preprint arXiv:2307.08775*.

Yujie Lu, Weixi Feng, Wanrong Zhu, Wenda Xu, Xin Eric Wang, Miguel Eckstein, and William Yang Wang. 2022b. Neuro-symbolic procedural planning with commonsense prompting. In *The Eleventh International Conference on Learning Representations*.

Clara Meister, Tim Vieira, and Ryan Cotterell. 2020. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809.

Johannes E. M. Mosig, Shikib Mehri, and Thomas Kober. 2020. STAR: A Schema-Guided Dialog Dataset for Transfer Learning. *arXiv e-prints*.

Franz Josef Och, Nicola Ueffing, and Hermann Ney. 2001. An efficient a* search algorithm for statistical machine translation. In *Proceedings of the ACL 2001 Workshop on Data-Driven Methods in Machine Translation*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2023. Adapt: As-needed decomposition and planning with language models. *arXiv preprint arXiv:2311.05772*.

Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. Creator: Disentangling abstract and concrete reasonings of large language models through tool creation. *arXiv preprint arXiv:2305.14318*.

Shuofei Qiao, Honghao Gui, Huajun Chen, and Ningyu Zhang. 2023. Making language models better tool learners with execution feedback. *arXiv preprint arXiv:2305.13068*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Hajo A Reijers. 2003. *Design and control of workflow processes: business process management for the service industry*, volume 2617. Springer.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Sailik Sengupta, Zahra Zahedi, and Subbarao Kambhampati. 2019. To monitor or to trust: observing robot's behavior based on a game-theoretic model of trust. In *Proceedings of the Trust Workshop at AAMAS*.

Yuanlong Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope, and Ray Kurzweil. 2017. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2210–2219.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10737–10746. IEEE Computer Society.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *arXiv preprint arXiv:2306.06624*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

MosaicML NLP Team. 2023. Introducing mpt-7b: A new standard for open-source, commercially usable llms. Accessed: 2023-03-28.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Ashwin K Vijayakumar, Michael Cogswell, Ramprasaath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models.

Shufan Wang, Sébastien Jean, Sailik Sengupta, James Gung, Nikolaos Pappas, and Yi Zhang. 2023. Measuring and mitigating constraint violations of in-context learning for utterance-to-API semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7196–7207, Singapore. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Te-Lin Wu, Alex Spangher, Pegah Alipoormolabashi, Marjorie Freedman, Ralph Weischedel, and Nanyun Peng. 2022. Understanding multimodal procedural knowledge by sequencing multimodal instructional manuals. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4525–4542.

Kevin Yang and Dan Klein. 2021. Fudge: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3511–3535.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Xiao Yu, Maximillian Chen, and Zhou Yu. Prompt-based monte-carlo tree search for goal-oriented dialogue policy planning.

Yi Zhang, Sujay Kumar Jauhar, Julia Kiseleva, Ryen White, and Dan Roth. 2021. Learning to decompose and organize complex tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2726–2735.

Jeffrey Zhao, Yuan Cao, Raghav Gupta, Harrison Lee, Abhinav Rastogi, Mingqiu Wang, Hagen Soltau, Izhak Shafran, and Yonghui Wu. 2023. Anytod: A programmable task-oriented dialog system.

A Dataset Creation

A.1 Domain Creation

The flows and corresponding APIs for each domain are shown in Tab. 6. The API definitions in all domains, their input/output parameters, and descriptions are shown in Tab. 7.

A.2 Test Utterance Generation

The prompt for generating test queries related to the intent "book flight" in the domain "Trip Booking" is shown in Fig. 4. Similar prompts were used for the generation of queries for all other intents. Some generated user queries related to various intents are shown in Tab. 8.

B Prompt Structures for Generating Plans

The prompt structure for generating plans consisting of only APIs (no thoughts) is shown in Fig. 5. The prompt structure for planning with thoughts and APIs is shown in Fig. 6.

C Experimental Details

C.1 Hyper-parameter Tuning

In this section, we discuss the hyper-parameter selection process in FLAP, our proposed constrained decoding based algorithm in §4.2.

In our dataset, all user queries are single intent, hence, we set $\alpha_c > \alpha_a > \alpha_b$ ($\alpha_a = 0.5$, $\alpha_b = 0.1$, and $\alpha_c = 1$). We empirically determine $\beta = 0.1$ when the APIs are hallucinated or violates the dependency, i.e., a soft alignment of generated APIs to permitted APIs.

```
Our goal is to generate customer utterances in the customer care domain: Trip Planner

Here is an example of customer utterance and the intent(s) in the utterance:
Customer utterance: Please book a flight from {source} to {destination}.
Intents: [book_flight]

Now, write 20 paraphrases of this customer utterance where the intent(s) are [book_flight].
Make sure the paraphrases are very diverse and crisp.
Make sure you replace the parameter(s) [destination, source] with various US city names
such as 'Miami', 'New Orleans', 'Chicago', and so on.

Answer:
Paraphrase-1: Can you book a flight from NYC to Chicago for me?
Paraphrase-2:
```

Figure 4: Prompt for generating test utterances for the intent "book flight" in the domain "Trip Booking".

```
You are playing the role of a customer care agent who has access to the following APIs.
{
FindFlight(inputs: airport_code; Outputs: flight_id): retrieves flights to/from the airport.
FindRentalCar(inputs: ; Outputs: car_id): retrieves cars to/from the travelling city
...
...
...
}

# To book a hotel follow the below steps sequentially.
start processing the requests from the customer
suggest hotels to the customer
...
...

# To book a car follow the below steps sequentially.
start processing the requests from the customer
suggest cars to the customer
...
...

# To book a flight follow the below steps sequentially.
start processing the requests from the customer
suggest flights to the customer
...
...

### Customer request: "Can you please help me book a flight from Dallas to Austin?"
Plan step by step to fulfill the customer request using the given APIs and instructions.
Complete the plan below in one line.
Plan: InitSystem(), Start(),
```

Figure 5: Prompt format for generating a plan consisting of APIs only (without thoughts) in the domain "Trip Booking". The first two dummy APIs InitSystem(), Start() are provided in context to guide the model to follow the plan format.

To tune the values of the weight on the heuristic score (λ) and beam size (k), we run an ablation with a subset of the data (5 user utterances per intent) in Tab. 1a, in the setting where only relevant subflow is given in-context, using Mistral-7b-instruct. We experiment with $k = [5, 10, 15]$ and $\lambda = [0.3, 0.5, 0.7]$. The results are summarized in Tab. 10. We find that $k = 10$ and $\lambda = 0.7$ result in overall best performance, especially in terms of API dependency violation (refer to "per plan avg % of inconsistent APIs"). Hence, we report the results with these values.

We set the values of all the scaling factors (a, b, c, d) to 1 implying equal weight to all heurictic score components. We set the lookahead length, $L = 32$

in our experiments. During plan generation, if the API Finish() is generated, we consider that as end-of-plan (as per the flows in Tab. 6) and terminate the generation.

For the Beam Search baseline, we set number of beams to 3 and use no_repeat_ngram_size=10 to prevent the model from generating the same thought and API pair repeatedly. For the Nucleus Sampling baseline we use top_p=0.9.

C.2 Infrastructure and Libraries

We use eight NVIDIA A10G Tensor Core GPUs for all of our experiments. We use Pytorch[2] for

---

[2] https://pytorch.org

```
You are playing the role of a customer care agent who has access to the following APIs.
{
FindRentalCar(inputs: ; Outputs: car_id): retrieves cars to/from the travelling city
FindFlight(inputs: airport_codes; Outputs: flight_id): retrieves flights to/from the airports.
...
...
...
}

# To book a hotel follow the below steps sequentially.
start processing the requests from the customer
suggest hotels to the customer
...
...

# To book a car follow the below steps sequentially.
start processing the requests from the customer
suggest cars to the customer
...
...

# To book a flight follow the below steps sequentially.
start processing the requests from the customer
suggest flights to the customer
...
...

### Customer request: "Can you book a flight from NYC to Chicago for me?"

Identify the customer intent and plan step by step to fulfill the customer request using the given APIs
and instructions. State the thought and API call at each step.

[thought] To start processing requests from the customer, I need to initialize system. [API] InitSystem()
[thought] Now, I can start processing request from the customer. [API] Start()
[thought]
```

Figure 6: Prompt format for generating a plan (with thoughts) in the domain "Trip Booking". The first two dummy steps, e.g., thoughts related to `InitSystem()` and `Start()` are provided in context to guide the model to follow the plan format.

all implementations. For all the LLMs used in this paper, we use their Huggingface[3] implementations. We build our proposed constrained decoding algorithm on top of Huggingface generation module[4]. We used the Huggingface implementation of the `all-MiniLM-L6-v2`[5] sentence transformer for measuring semantic similarities (sim()).

C.3 Runtime Analysis

The runtimes of different models in plan generation are shown in Tab. 9. Note that our proposed constrained decoding algorithm is run on only one GPU of 24GB memory with a batch size of 1. This is one of the reasons of the high runtime. We explored parallelization techniques by deploying the LLMs in multiple GPUs to make the process faster. However, it was observed that deploying the LLMs (Huggingface implementation) in multiple GPUs made the inference time of the base LLMs even slower. Our intended future work includes more efficient deployment of the LLMs in potentially multi-GPU machines which is out-of-scope of this paper.

D Additional Results

The ablation results with standard deviations over all test data points with the prompting-based plan generation using numerous open-source models in the settings where all flows are given in-context and only relevant flows are given in-context, can be found in Tab. 11 and Tab. 12, respectively.

The results with standard deviations for plan generation using our proposed constrained decoding algorithm, FLAP, in the setting where all flows are given in-context and only relevant flows are given in-context, can be found in Tab. 13 and Tab. 14, respectively.

E Qualitative Error Analysis

E.1 Model entering into repetitive loop with thoughts

Example of error when model enters into a repetitive loop when prompted to generate plans with thoughts, can be observed in Tab. 15.

E.2 API hallucinations by pre-trained LLMs with tool data

API hallucination examples by `Alpaca-7B` are shown in Tab. 16.

---

[3]https://huggingface.co
[4]https://huggingface.co/docs/transformers/main_classes/text_generation
[5]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Flow steps	Required API sequence
Domain: Trip Booking, Intent: Book Car	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Suggest cars to the customer	`FindRentalCar()`
Confirm and create the trip	`Confirm()`, `CreateTrip()`
Extract and add promotional offers	`GetCarInsuranceDiscount()`, `UpdateTrip()`
Order the trip	`GetPaymentInformation()`, `OrderTrip()`
Finish processing request	`Finish()`
Domain: Trip Booking, Intent: Book Flight	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Suggest flights to the customer	`GetAirports()`, `FindFlight()`
Confirm and create the trip	`Confirm()`, `CreateTrip()`
Order the trip	`GetPaymentInformation()`, `OrderTrip()`
Finish processing request	`Finish()`
Domain: Trip Booking, Intent: Book Hotel	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Suggest hotels to the customer	`FindHotel()`
Confirm and create the trip	`Confirm()`, `CreateTrip()`
Order the trip	`GetPaymentInformation()`, `OrderTrip()`
Finish processing request	`Finish()`
Domain: Insurance, Intent: Buy Insurance	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Provide quote for the item to be insured	`GetItem()`, `GetQuote()`
Get customer details	`GetDemographicDetails()`
Order the insurance	`OrderInsurance()`
Finish processing request	`Finish()`
Domain: Insurance, Intent: Cancel Insurance	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Validate insurance	`GetInsuranceID()`, `ValidateInsurance()`
Record the refund method	`GetRefundMethod()`
Confirm and cancel the Insurance	`Confirm()`, `CancelInsurance()`
Finish processing request	`Finish()`
Domain: Insurance, Intent: Add Member	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Validate insurance	`GetInsuranceID()`, `ValidateInsurance()`
Update the insurance with new member information	`GetAdditionalMemberInformation()`, `UpdateInsurance()`
Finish processing request	`Finish()`
Domain: Banking, Intent: Open Account	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Check eligibility for opening an account	`GetCustomerIncome()`, `GetDateOfBirth()`, `CheckEligibility()`
Finalize the account type	`GetAccountTypes()`
Open the account	`Confirm()`, `OpenAccount()`
Finish processing request	`Finish()`
Domain: Banking, Intent: Report Problem	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Record the problem and notify the internal team	`RecordIssue()`, `NotifyComplaintDepartment()`
Finish processing request	`Finish()`
Domain: Banking, Intent: Cancel Transaction	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Validate transaction details	`GetTransactionInformation()`, `ValidateTransaction()`
Cancel the transaction	`Confirm()`, `CancelTransaction()`
Finish processing request	`Finish()`
Domain: Restaurant & Ride Book, Intent: Book Restaurant	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Suggest restaurants to the customer	`GetTime()`, `GetCity()`, `GetBudget()`, `FindRestaurants()`
Finalize the restaurant booking	`GetCustomerName()`, `GetSelectedRestaurant()`, `BookRestaurant()`
Finish processing request	`Finish()`
Domain: Restaurant & Ride Book, Intent: Book Ride	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Suggest ride options to the customer	`GetTime()`, `GetSource()`, `GetDestination()`, `FindRideShareOptions()`
Finalize the ride service booking	`GetPaymentMethod()`, `GetSelectedRideShareOption()`, `BookRide()`
Finish processing request	`Finish()`
Domain: Restaurant & Ride Book, Intent: Cancel Restaurant Booking	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Validate the restaurant booking	`GetRestaurantBookingID()`, `Authenticate()`
Cancel the restaurant booking	`CancelRestaurantBooking()`
Finish processing request	`Finish()`
Domain: Restaurant & Ride Book, Intent: Cancel Ride Booking	
Start processing the requests from the customer	`InitSystem()`, `Start()`
Validate the ride booking	`GetRideBookingID()`, `Authenticate()`
Cancel the ride booking	`GetRefundMethod()`, `CancelRide()`
Finish processing request	`Finish()`

Table 6: Flows and required APIs in all domains. We study faithful plan generation in a zero-shot setting. Hence, we additionally add the dummy first (`InitSystem()`, `Start()`) and last (`Finish()`) steps in each flow to guide the model by indicating the start and end of processing a request in the zero-shot prompt.

APIs	Input Parameters	Output Parameters	Description
Domain: Trip Booking			
Confirm()	None	confirmation_status	confirm trip details with the customer before creating a trip
FindRentalCar()	None	car_id	retrieves cars to/from the travelling city
FindHotel()	None	hotel_id	finds and retrieves hotels in a city
GetPayInfo()	None	pay_info	gets payment information from the customer
Start()	init_status	True/False	starts processing customer requests
InitSystem()	None	init_status	initializes the system to start processing customer requests
OrderTrip()	trip_id,pay_info, confirmation_status	order_status	places the bookings
UpdateTrip()	offer_id, trip_id	True/False	updates a trip with special offers
FindFlight()	airport_code	flight_id	retrieves flights to/from the airport
GetCarInsuranceDiscount()	car_id	offer_id	retrieves insurances related discounts related to the car options
CreateTrip()	flight_id/hotel_id/car_id, confirmation_status	trip_id	creates a new trip before ordering
Finish()	order_status	True/False	finishes processing customer requests
GetAirports()	None	airport_code	returns nearby airport codes based on the travelling city
Domain: Insurance			
CancelInsurance()	insurance_id, confirmation_status, refund_method	cancellation_status	cancels the insurance
GetAdditionalMemberInfo()	None	additional_member_info	asks additional member info from the customer and returns it
GetInsuranceID()	None	insurance_id	asks the insurance id from the customer and returns it
OrderInsurance()	pay_info	order_status	places order for purchasing the insurance
GetPaymentInformation()	None	pay_info	gets payment information from the customer
Finish()	order_status/ cancellation_status/ update_status	True/False	finishes processing customer requests
GetRefundMethod()	None	refund_method	asks the preferred refund method from the customer and returns it
Confirm()	insurance_id	confirmation_status	confirms insurance details with the customer
GetItem()	None	item_id	gets the item to be insured from the customer
InitSystem()	None	init_status	initializes the system to start processing customer requests
GetDemographicDetails()	None	demographics	asks demographic information of the customer and returns it
Start()	init_status	True/False	starts processing customer requests
GetQuote()	item_id	quote_id	provides quote to the customer for a given item
ValidateInsurance()	insurance_id	insurance_validated	validates if the insurance exists
UpdateInsurance()	additional_member_info, insurance_id	update_status	updates an existing insurance by adding a new member
Domain: Banking			
Start()	init_status	True/False	starts processing customer requests
GetTransactionInfo()	None	transaction_id	asks transaction info to the customer and returns the transaction id
CheckEligibility()	date_of_birth, annual_income	eligibility_status	checks if customer is eligible to open account given age, income
GetAccountTypes()	date_of_birth,annual_income, eligibility_status	account_type	get the account type based on customer eligibility, age and income
CancelTransaction()	transaction_id, confirmation_status	cancellation_status	cancels a transaction
ValidateTransaction()	transaction_id	None	validates if the transaction was actually executed
Finish()	open_status/ cancellation_status/ notification_status	True/False	finishes processing customer requests
GetCustomerIncome()	None	annual_income	ask customer annual income and returns it
RecordIssue()	None	issue_id	records the issue in internal database for review by team
OpenAccount()	confirmation_status, account_type	open_status	opens a new account
InitSystem()	None	init_status	initializes the system to start processing customer requests
Confirm()	None	confirmation_status	confirms with the customer before opening an account or cancelling a transaction
GetDateOfBirth()	None	date_of_birth	asks the date of birth to the customer and returns it
NotifyComplaintDepartment()	issue_id	notification_status	notifies the complaint department about an problem reported by the customer
Domain: Restaurant & Ride Booking			
GetBudget()	None	budget	asks customer the budget and return it
FindRideShareOptions()	source, destination, time	ride_options	retrieves ride-share options
GetPaymentMethod()	None	pay_method	asks customer the payment method and return it
GetDestination()	None	destination	asks destination to the customer and return it
GetCustomerName()	None	name	asks customer their name and return it
BookRide()	pay_method, rideshare_id	ride_booking_status	books the selected rideshare service
InitSystem()	None	init_status	initializes the system to start processing customer requests
GetSelectedRestaurant()	restaurant_list	restaurant_id	presents customer with the list of restaurants, ask their choice and return the selected restaurant id
Authenticate()	ride_id/restaurant_book_id	True/False	authenticates the customer and their booking ids
Finish()	None	ride_booking_status/ restaurant_booking_status/ restaurant_cancellation_status/ ride_cancellation_status	finishes processing customer requests
GetRestaurantBookingID()	None	restaurant_book_id	asks the customer the restaurant booking id and return it
GetCity()	None	city	asks customer the city and return it
GetTime()	None	time	asks time to the customer and return it
GetSelectedRideShareOption()	ride_options	rideshare_id	presents customer with the list of rides, asks their choice and return the selected rideshare id
GetSource()	None	source	asks source to the customer and return it
BookRestaurant()	restaurant_id, time, name	restaurant_booking_status	books the selected restaurant
CancelRide()	refund_method, ride_id	ride_cancellation_status	cancels the ride
FindRestaurants()	time, city, budget	restaurant_list	retrieves list of restaurants
CancelRestaurantBooking()	restaurant_book_id	restaurant_cancellation_status	cancels the restaurant booking
Start()	init_status	True/False	starts processing customer requests
GetRideBookingID()	None	ride_id	asks the customer the ride booking id and return it
GetRefundMethod()	None	refund_method	asks customer the refund method and return it

Table 7: API definition, their input/output parameters, and descriptions in four constructed domains. Some API names are clipped for better visibility (e.g., GetAdditionalMemberInformation() is clipped to GetAdditionalMemberInfo()).

Intents and Generated Queries in Domain Trip Booking

Book Car: Could you arrange for car rental for my Minneapolis trip from June 25th to July 3rd?
Book Hotel: I'm looking to book a hotel room in Los Angeles from August 18th to August 25th.
Book Flight: I need to fly from Miami to Toronto, can you please help me with that?

Intents and Generated Queries in Domain Insurance

Buy Insurance: I want to safeguard my family's health with an insurance policy.
Cancel Insurance: My current policy doesn't cover all the expenses, would you please cancel it?
Add Member: Would you please add my aunt to my mobile insurance policy?

Intents and Generated Queries in Domain Banking

Open Account: My wife and I are looking to open a joint bank account together. What steps do we need to take?
Report Problem: My card is declined at the ATM. Can you help?
Cancel Transaction: The balance of my savings account has been altered by an unauthorized transfer of $500. I would appreciate if you could get that money refunded.

Intents and Generated Queries in Domain Restaurant & Ride Book

Book Restaurant: Could you assist me in making a reservation at a restaurant in Manhattan?
Book Ride: I am in Washington D.C. looking for a ride to Fort Lauderdale.
Cancel Restaurant: I made a restaurant booking in Los Angeles this morning but I cannot make it due to an emergency. Can you cancel my booking?
Cancel Ride: May I cancel my ride from Brooklyn to Queens?

Table 8: Examples of generated user queries for each intent.

LLM	Models	With Thought	Avgerage Runtime Per Plan (seconds)	
			All flows in context	Relevant flow in context
mpt-7b-ins.	Greedy Search	No	7.94	5.4
	Greedy Search	Yes	14.07	11.3
	Beam Search	Yes	27.94	26.69
	Nucleus Sampling	Yes	17.74	17.55
	FLAP.1 [soft api + align thought to (api, intent)]	Yes	264.0	371.41
	FLAP.2 [soft api + align thought to (step, api)]	Yes	199.28	295.12
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	378.43	336.4
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	216.16	327.79
mistral-7b-ins.	Greedy Search	No	19.72	18.81
	Greedy Search	Yes	8.78	9.91
	Beam Search	Yes	10.24	9.76
	Nucleus Sampling	Yes	13.09	11.57
	FLAP.1 [soft api + align thought to (api, intent)]	Yes	250.67	252.45
	FLAP.2 [soft api + align thought to (step, api)]	Yes	221.63	234.47
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	228.61	117.58
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	238.82	113.22

Table 9: Average runtime of various models over all generated plans.

LLM	Varying Parameter	Parameter Values	With Thoughts	Avg count per plan		% Plan Parsable (↑)	Avg. % of APIs in plan that are		Avg # of edit for		Per plan avg % of inconsistent	
				Thought	APIs		Repeated (↓)	Hallucinated (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
mistral-7B-Ins.	λ	top k=10, λ=0.7	Yes	6.85 ± 2.96	6.98 ± 3.04	100.0	0.81 ± 4.65	0.0 ± 2.15	2.2 ± 2.94	2.19 ± 2.8	6.92 ± 9.12	2.99 ± 8.39
	λ	top k=10, λ=0.5	Yes	6.02 ± 2.73	6.03 ± 2.72	100.0	0.0 ± 0.0	0.0 ± 2.46	2.51 ± 2.58	2.55 ± 2.63	7.06 ± 11.59	16.39 ± 15.81
	λ	top k=10, λ=0.3	Yes	5.11 ± 1.58	5.11 ± 1.58	100.0	0.22 ± 1.76	0.0 ± 0.0	2.09 ± 2.01	2.17 ± 1.98	4.22 ± 8.77	20.62 ± 18.75
	λ	λ=0 (Greedy)	Yes	5.66 ± 3.48	5.77 ± 3.6	100.0	3.07 ± 10.67	1.0 ± 4.3	2.78 ± 3.53	2.86 ± 3.45	3.15 ± 7.39	31.25 ± 22.82
	top k	top k=5, λ=0.7	Yes	6.52 ± 2.25	6.52 ± 2.25	100.0	0.0 ± 0.0	0.0 ± 2.07	2.36 ± 2.58	2.42 ± 2.63	7.42 ± 9.53	4.36 ± 9.33
	top k	top k=10, λ=0.7	Yes	6.85 ± 2.96	6.98 ± 3.04	100.0	0.81 ± 4.65	0.0 ± 2.15	2.2 ± 2.94	2.19 ± 2.8	6.92 ± 9.12	2.99 ± 8.39
	top k	top k=15, λ=0.7	Yes	6.02 ± 2.11	6.19 ± 2.19	100.0	0.31 ± 2.25	0.0 ± 2.25	2.43 ± 2.46	2.46 ± 2.48	10.42 ± 11.14	4.14 ± 11.22

Table 10: Ablation for tuning constrained decoding parameters of FLAP: beam size for lookahead (top k) and lookahead heuristic score weight (λ). The ablation was done using a subset of the data (5 user utterances per intent). There are 6.84 average APIs per gold plan (reference for the "Avg. count per plan" column).

LLMs	With Thought	Avg count per plan		% Plan Parsable (↑)	Avg % of APIs in plan that are		Avg # of edit for		Per plan avg % of inconsistent	
		Thoughts	APIs		Repeated (↓)	Hallucinated (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
santacoder-1.1b	No	-	33.73 ± 49.05	100.0	19.84 ± 37.01	1.21 ± 9.43	30.34 ± 47.63	25.91 ± 39.8	15.42 ± 14.02	33.06 ± 24.21
	Yes	15.09 ± 8.66	14.98 ± 8.59	99.62	27.61 ± 30.64	13.0 ± 21.75	11.78 ± 7.66	13.02 ± 8.27	18.17 ± 16.39	34.13 ± 27.51
toolAlpaca-7b	No	-	4.13 ± 2.16	100.0	0.0 ± 0.0	10.76 ± 23.62	4.64 ± 2.05	5.12 ± 2.42	20.41 ± 19.98	49.6 ± 26.21
	Yes	4.88 ± 4.26	4.88 ± 4.26	90.38	3.32 ± 12.31	19.0 ± 21.98	4.67 ± 4.07	5.3 ± 3.23	17.6 ± 17.27	44.78 ± 25.18
falcon-7b-instruct	No	-	55.97 ± 56.1	100.0	42.65 ± 44.96	6.06 ± 19.0	47.48 ± 51.98	40.34 ± 45.91	16.61 ± 20.56	27.67 ± 26.48
	Yes	11.62 ± 6.44	11.69 ± 6.31	100.0	39.02 ± 30.8	19.0 ± 27.34	9.73 ± 6.45	11.41 ± 7.34	24.03 ± 25.24	38.83 ± 26.52
mpt-7b-instruct	No	-	17.32 ± 32.42	100.0	11.58 ± 28.7	0.17 ± 1.39	16.17 ± 31.7	13.42 ± 24.88	14.65 ± 14.12	33.33 ± 21.74
	Yes	16.2 ± 7.59	16.08 ± 7.49	97.69	35.11 ± 30.54	4.0 ± 10.64	13.92 ± 8.03	12.38 ± 7.69	11.86 ± 15.74	29.61 ± 23.69
mistral-7b-instruct	No	-	4.78 ± 1.7	100.0	0.32 ± 3.04	2.36 ± 7.04	2.87 ± 1.75	3.02 ± 1.79	10.6 ± 15.84	37.23 ± 25.28
	Yes	4.72 ± 2.38	4.76 ± 2.41	99.62	2.92 ± 11.73	0.0 ± 3.59	2.78 ± 2.56	2.8 ± 2.53	3.09 ± 8.21	40.31 ± 23.89
koala-13b	No	-	45.79 ± 44.81	100.0	39.64 ± 41.95	6.29 ± 17.54	37.6 ± 40.87	31.4 ± 32.87	10.51 ± 13.37	23.98 ± 18.54
	Yes	12.06 ± 4.15	12.2 ± 4.21	97.31	24.37 ± 30.11	10.0 ± 20.04	8.76 ± 5.23	9.08 ± 5.52	9.02 ± 11.68	24.04 ± 19.39
vicuna-13b	No	-	5.94 ± 2.68	100.0	0.57 ± 2.75	2.38 ± 6.99	4.12 ± 2.51	4.29 ± 2.54	15.08 ± 16.23	35.22 ± 20.51
	Yes	6.94 ± 3.53	6.99 ± 3.6	92.31	4.46 ± 9.76	4.0 ± 9.55	4.02 ± 3.03	4.28 ± 3.11	6.73 ± 11.49	35.49 ± 22.61
llama-13b	No	-	29.41 ± 37.77	100.0	25.84 ± 38.81	2.36 ± 9.19	26.77 ± 37.17	20.87 ± 29.83	13.16 ± 16.08	31.72 ± 25.31
	Yes	10.21 ± 6.9	10.13 ± 6.79	98.85	21.87 ± 33.24	4.0 ± 12.7	9.18 ± 7.12	8.66 ± 7.05	4.68 ± 10.61	31.04 ± 27.63
mpt-30b-chat	No	-	5.7 ± 1.93	100.0	1.73 ± 5.81	0.53 ± 3.94	3.26 ± 2.13	3.35 ± 2.08	9.08 ± 14.36	33.41 ± 19.78
	Yes	5.57 ± 2.07	5.87 ± 2.25	96.92	2.91 ± 7.67	1.0 ± 5.15	2.7 ± 2.58	2.7 ± 2.6	5.72 ± 10.18	32.45 ± 22.35
falcon-40b-instruct	No	-	10.21 ± 21.88	100.0	3.1 ± 15.86	0.89 ± 4.24	8.07 ± 19.65	7.15 ± 14.25	18.0 ± 17.61	38.27 ± 20.27
	Yes	7.6 ± 4.83	7.75 ± 4.96	100.0	10.25 ± 21.52	5.0 ± 12.61	5.35 ± 4.55	5.4 ± 4.29	8.95 ± 12.34	36.42 ± 24.54

Table 11: Zero-shot plan generation results using greedy decoding when all flows are given in the prompt. There are 6.84 average APIs per gold plan (reference for the "Avg count per plan" column).

LLMs	With Thought	Avg count per plan		% Plan Parsable (↑)	Avg % of APIs in plan that are		Avg # of edit for		Per plan avg % of inconsistent	
		Thoughts	APIs		Repeated (↓)	Hallucinated (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
santacoder-1.1b	No	-	34.08 ± 49.7	100.0	19.81 ± 37.22	1.79 ± 11.73	29.58 ± 46.88	25.75 ± 39.29	15.34 ± 15.5	34.14 ± 23.77
	Yes	13.14 ± 9.84	13.07 ± 9.76	100.0	22.27 ± 29.81	20.0 ± 28.74	9.48 ± 8.03	11.53 ± 8.73	19.48 ± 19.06	31.21 ± 25.47
toolAlpaca-7b	No	-	4.13 ± 1.88	100.0	0.0 ± 0.0	6.99 ± 18.96	4.38 ± 2.25	4.73 ± 2.52	17.77 ± 19.97	49.17 ± 26.49
	Yes	3.69 ± 1.01	3.7 ± 1.02	96.15	2.19 ± 7.56	8.0 ± 16.92	3.01 ± 1.82	3.52 ± 2.27	10.56 ± 15.83	47.59 ± 22.9
falcon-7b-instruct	No	-	50.45 ± 52.86	100.0	39.54 ± 45.13	6.63 ± 20.89	40.76 ± 48.73	38.09 ± 44.17	17.43 ± 20.33	23.85 ± 21.46
	Yes	11.52 ± 8.03	11.47 ± 7.95	97.69	37.26 ± 33.21	9.0 ± 19.72	9.74 ± 8.56	9.15 ± 7.74	21.49 ± 24.2	37.91 ± 28.25
mpt-7b-instruct	No	-	14.89 ± 29.87	100.0	8.5 ± 25.06	0.19 ± 1.83	13.55 ± 29.35	11.52 ± 24.08	13.52 ± 14.29	35.64 ± 20.75
	Yes	12.42 ± 8.3	12.34 ± 8.19	98.85	23.02 ± 30.15	3.0 ± 9.44	9.87 ± 8.32	8.93 ± 7.8	9.09 ± 15.46	33.44 ± 23.48
mistral-7b-instruct	No	-	5.85 ± 2.61	100.0	0.57 ± 6.06	1.38 ± 5.06	3.1 ± 2.93	3.13 ± 2.14	8.02 ± 12.87	30.61 ± 20.76
	Yes	5.57 ± 3.1	5.7 ± 3.5	100.0	3.76 ± 12.18	1.0 ± 3.97	2.65 ± 3.48	2.6 ± 2.85	2.75 ± 6.78	34.36 ± 22.77
koala-13b	No	-	36.08 ± 41.04	100.0	28.75 ± 39.8	4.18 ± 13.41	29.37 ± 37.85	23.78 ± 29.36	11.25 ± 13.33	26.15 ± 17.19
	Yes	10.71 ± 4.44	10.99 ± 4.78	100.0	20.25 ± 28.69	9.0 ± 20.52	7.42 ± 5.61	7.93 ± 6.06	9.28 ± 10.64	22.64 ± 17.01
vicuna-13b	No	-	5.98 ± 2.53	100.0	0.32 ± 1.95	2.4 ± 7.3	3.91 ± 2.51	4.12 ± 2.5	14.21 ± 15.44	31.66 ± 17.25
	Yes	6.12 ± 3.08	6.19 ± 3.14	92.31	3.68 ± 8.51	2.0 ± 7.09	3.08 ± 2.68	3.37 ± 2.79	5.4 ± 9.27	35.02 ± 22.05
llama-13b	No	-	27.92 ± 38.12	100.0	23.04 ± 38.37	2.0 ± 9.53	24.68 ± 37.43	19.1 ± 30.62	11.34 ± 14.81	29.43 ± 22.09
	Yes	8.73 ± 6.97	8.67 ± 6.86	100.0	18.87 ± 33.3	3.0 ± 10.05	6.58 ± 7.58	5.6 ± 6.28	4.03 ± 12.27	38.39 ± 30.38
mpt-30b-chat	No	-	6.68 ± 5.29	100.0	3.88 ± 11.23	0.19 ± 1.78	3.53 ± 5.38	3.33 ± 3.05	6.32 ± 10.9	33.33 ± 20.31
	Yes	5.43 ± 2.22	5.73 ± 2.32	97.31	3.12 ± 8.4	1.0 ± 3.97	2.13 ± 2.05	2.09 ± 1.86	4.56 ± 9.25	34.33 ± 23.63
falcon-40b-instruct	No	-	11.22 ± 23.02	100.0	4.01 ± 18.09	0.89 ± 4.63	8.97 ± 21.72	7.87 ± 16.12	17.88 ± 18.08	35.5 ± 20.26
	Yes	7.14 ± 4.31	7.26 ± 4.37	97.69	10.55 ± 21.04	7.00 ± 14.17	3.95 ± 4.3	4.11 ± 4.1	8.66 ± 15.45	25.68 ± 20.45

Table 12: Zero-shot plan generation results using greedy decoding when only the flow related to the user query is given in the prompt. There are 6.84 average APIs per gold plan (reference for the "Avg count per plan" column).

| LLMs | Decoding Strategy | With Thought | Avg count per plan | | % Plan Parsable (↑) | Avg % of APIs in plan that are | | Avg # of edit for | | Per plan avg % of inconsistent | |
			Thoughts	APIs		Repeated (↓)	Hallucinated (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
mpt-7b-ins.	Greedy Search	No	-	17.32 ± 32.42	100.0	11.58 ± 28.7	0.17 ± 1.39	16.17 ± 31.7	13.42 ± 24.88	14.65 ± 14.12	33.33 ± 21.74
	Greedy Search	Yes	16.2 ± 7.59	16.08 ± 7.49	97.69	35.11 ± 30.54	4.0 ± 10.64	13.92 ± 8.03	12.38 ± 7.69	11.86 ± 15.74	29.61 ± 23.69
	Beam Search	Yes	9.94 ± 5.82	9.9 ± 5.8	70.77	5.48 ± 11.7	3.0 ± 7.31	7.56 ± 4.6	7.73 ± 4.75	13.93 ± 14.06	40.99 ± 19.49
	Nucleus Sampling	Yes	12.3 ± 6.4	12.41 ± 6.4	76.92	25.28 ± 22.63	4.0 ± 12.28	9.84 ± 5.73	9.46 ± 6.06	13.82 ± 15.15	31.72 ± 18.74
	FLAP.1 [soft api + align thought to (api, intent)]	Yes	10.25 ± 3.63	10.37 ± 3.68	100.0	1.26 ± 6.03	0.0 ± 0.36	5.51 ± 4.14	5.47 ± 4.02	6.3 ± 7.7	1.26 ± 4.28
	FLAP.2 [soft api + align thought to (step, api)]	Yes	9.28 ± 4.54	9.37 ± 4.57	100.0	2.99 ± 11.26	0.0 ± 0.62	5.68 ± 5.01	5.58 ± 4.85	4.34 ± 7.22	1.62 ± 5.15
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	9.16 ± 4.02	9.39 ± 4.07	100.0	1.04 ± 6.04	0.0 ± 0.0	5.02 ± 3.92	4.98 ± 3.87	6.1 ± 8.06	1.32 ± 4.34
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	9.29 ± 4.5	9.38 ± 4.55	100.0	3.63 ± 11.94	0.0 ± 1.55	5.53 ± 4.93	5.41 ± 4.74	4.81 ± 7.71	2.05 ± 6.53
mistral-7b-ins.	Greedy Search	No	-	4.78 ± 1.7	100.0	0.32 ± 3.04	2.36 ± 7.04	2.87 ± 1.75	3.02 ± 1.79	10.6 ± 15.84	37.23 ± 25.28
	Greedy Search	Yes	4.72 ± 2.38	4.76 ± 2.41	99.62	2.92 ± 11.73	0.0 ± 3.59	2.78 ± 2.56	2.8 ± 2.53	3.09 ± 8.21	40.31 ± 23.89
	Beam Search	Yes	3.61 ± 1.44	3.75 ± 1.62	98.08	0.2 ± 2.05	0.0 ± 1.55	2.88 ± 2.19	2.95 ± 2.15	2.37 ± 8.37	38.94 ± 26.87
	Nucleus Sampling	Yes	5.67 ± 2.83	5.91 ± 2.87	93.85	5.79 ± 12.25	2.0 ± 5.77	2.97 ± 2.51	3.02 ± 2.56	7.47 ± 13.13	34.28 ± 22.38
	FLAP.1 [soft api + align thought to (api, intent)]	Yes	6.62 ± 2.72	6.65 ± 2.73	100.0	0.0 ± 0.0	0.0 ± 2.85	2.45 ± 3.18	2.51 ± 3.23	6.26 ± 9.47	4.59 ± 10.35
	FLAP.2 [soft api + align thought to (step, api)]	Yes	6.09 ± 2.02	6.12 ± 2.09	100.0	0.12 ± 1.14	0.0 ± 3.03	2.25 ± 3.08	2.32 ± 3.16	5.06 ± 8.57	5.47 ± 11.81
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	6.27 ± 2.21	6.36 ± 2.21	100.0	0.39 ± 3.59	0.0 ± 0.0	1.97 ± 2.89	2.01 ± 2.84	6.46 ± 9.45	3.63 ± 9.06
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	6.37 ± 2.45	6.39 ± 2.44	100.0	0.54 ± 5.12	0.0 ± 2.93	2.45 ± 3.05	2.49 ± 3.04	5.63 ± 9.88	7.68 ± 13.15

Table 13: Zero-shot plan generation results with FLAP and other baselines, when all flows are given in the prompt. Here, the numbers (FLAP.#) indicate different ablation versions of FLAP. Structural constraint is applied in all versions of FLAP. There are 6.84 average APIs per gold plan (reference for the "Avg count per plan" column).

| LLMs | Decoding Strategy | With Thoughts | Avg count per plan | | % Plan Parsable (↑) | Avg % of APIs in plan that are | | Avg # of edit for | | Per plan avg % of inconsistent | |
			Thoughts	APIs		Repeated (↓)	Hallucinated (↓)	Steps (↓)	APIs (↓)	Steps (↓)	APIs (↓)
mpt-7b-ins.	Greedy Search	No	-	14.89 ± 29.87	100.0	8.5 ± 25.06	0.19 ± 1.83	13.55 ± 29.35	11.52 ± 24.08	13.52 ± 14.29	35.64 ± 20.75
	Greedy Search	Yes	12.42 ± 8.3	12.34 ± 8.19	98.85	23.02 ± 30.15	3.0 ± 9.44	9.87 ± 8.32	8.93 ± 7.8	9.09 ± 15.46	33.44 ± 23.48
	Beam Search	Yes	7.96 ± 5.7	7.96 ± 5.66	81.54	4.22 ± 10.69	2.0 ± 6.43	5.18 ± 4.41	5.3 ± 4.43	10.81 ± 13.0	41.44 ± 19.33
	Nucleus Sampling	Yes	9.98 ± 5.29	10.27 ± 5.58	75.0	19.82 ± 20.03	5.0 ± 10.53	7.19 ± 5.2	7.07 ± 4.94	13.55 ± 15.4	33.01 ± 21.27
	FLAP.1 [soft api + align thought to (api, intent)]	Yes	9.55 ± 3.83	9.63 ± 3.89	100.0	1.1 ± 6.69	0.0 ± 1.59	4.67 ± 4.05	4.65 ± 3.89	6.58 ± 8.19	1.2 ± 4.13
	FLAP.2 [soft api + align thought to (step, api)]	Yes	8.15 ± 3.68	8.2 ± 3.69	100.0	0.86 ± 5.29	0.0 ± 0.0	3.46 ± 3.99	3.42 ± 3.91	5.1 ± 8.36	1.24 ± 5.05
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	8.83 ± 4.02	9.0 ± 4.06	100.0	1.22 ± 7.4	0.0 ± 0.0	4.22 ± 4.07	4.15 ± 3.92	6.27 ± 7.77	2.41 ± 7.78
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	8.6 ± 4.48	8.65 ± 4.49	100.0	2.99 ± 11.21	0.0 ± 0.0	3.69 ± 4.44	3.37 ± 3.9	5.61 ± 9.66	3.66 ± 9.98
mistral-7b-ins.	Greedy Search	No	-	5.85 ± 2.61	100.0	0.57 ± 6.06	1.38 ± 5.06	3.1 ± 2.93	3.13 ± 2.14	8.02 ± 12.87	30.61 ± 20.76
	Greedy Search	Yes	5.57 ± 3.1	5.7 ± 3.5	100.0	3.76 ± 12.18	1.0 ± 3.97	2.65 ± 3.48	2.6 ± 2.85	2.75 ± 6.78	34.36 ± 22.77
	Beam Search	Yes	4.2 ± 1.2	4.28 ± 1.26	100.0	0.0 ± 0.0	0.0 ± 0.0	2.43 ± 1.96	2.51 ± 1.9	1.54 ± 5.59	36.72 ± 23.34
	Nucleus Sampling	Yes	5.63 ± 2.19	5.94 ± 2.42	98.08	3.73 ± 9.88	1.0 ± 4.71	2.94 ± 2.44	2.95 ± 2.48	6.77 ± 12.65	30.2 ± 20.14
	FLAP.1 [soft api + align thought to (api, intent)]	Yes	6.99 ± 2.94	7.01 ± 2.95	100.0	0.09 ± 1.02	1.0 ± 3.47	2.68 ± 3.35	2.75 ± 3.39	6.01 ± 9.7	4.06 ± 9.32
	FLAP.2 [soft api + align thought to (step, api)]	Yes	6.61 ± 2.33	6.63 ± 2.35	100.0	0.44 ± 4.76	1.0 ± 3.35	2.29 ± 2.86	2.32 ± 2.86	5.94 ± 10.51	7.01 ± 13.06
	FLAP.3 [soft api + align thought to (step, intent)]	Yes	6.73 ± 2.56	6.82 ± 2.61	100.0	0.36 ± 2.74	0.0 ± 1.05	2.27 ± 2.63	2.3 ± 2.65	6.58 ± 8.78	4.29 ± 9.98
	FLAP.4 [soft api + align thought to (step, api, intent)]	Yes	6.36 ± 2.32	6.37 ± 2.31	100.0	0.37 ± 3.06	1.0 ± 4.4	2.2 ± 2.76	2.23 ± 2.73	6.11 ± 10.14	8.66 ± 14.1

Table 14: Zero-shot plan generation results with FLAP and other baselines, when only the flow related to the user query is given in the prompt. Here, the numbers (FLAP.#) indicate different ablation versions of FLAP. Structural constraint is applied in all versions of FLAP. There are 6.84 average APIs per gold plan (reference for the "Avg count per plan" column).

Query: I want to go from San Diego to Dallas, can you please help me with that?

Generated plan by Mpt-7b-instruct using prompting with greedy decoding

[thought] I can suggest flights to the customer. [API] FindFlight()
[thought] I can suggest hotels to the customer. [API] FindHotel()
[thought] I can suggest cars to the customer. [API] FindRentalCar()
[thought] I can suggest insurances to the customer. [API] GetCarInsuranceDiscount()
[thought] I can suggest flights to the customer. [API] FindFlight()
[thought] I can suggest hotels to the customer. [API] FindHotel()
[thought] I can suggest cars to the customer. [API] FindRentalCar()
[thought] I can suggest insurances to the customer. [API] GetCarInsuranceDiscount()
[thought] I can suggest flights to the customer. [API] FindFlight()
[thought] I can suggest hotels to the customer. [API] FindHotel()
[thought] I can suggest cars to the customer. [API] FindRentalCar()
[thought] I can suggest insurances to the customer. [API] GetCarInsuranceDiscount()
[thought] I can suggest flights to the customer. [API] FindFlight()
[thought] I can suggest hotels to the customer. [API] FindHotel()
[thought] I can suggest cars to the customer. [API] FindRentalCar()

Table 15: Example of error when the LLMs enter into a repetitive loop of thought+API in Greedy decoding when prompted to generate thoughts and APIs.

Generated Plan by `ToolAlpaca-7B`	Correct APIs
Query: I would like to book a rental car for my visit to Las Vegas on June 28th.	
[thought] I need to suggest cars to the customer. [API] SuggestCars()	FindRentalCar()
[thought] I need to confirm and create the trip. [API] ConfirmTrip()	Confirm(), CreateTrip()
[thought] I need to extract and add promotional offers. [API] ExtractPromotionalOffers()	GetCarInsuranceDiscount(), UpdateTrip()
[thought] I need to order the trip. [API] OrderTrip()	GetPaymentInformation(), OrderTrip()
Query: Can you help me open an investment account?	
[thought] I need to check the eligibility of the customer to open an account. [API] CheckEligibility()	GetCustomerIncome(), GetDateOfBirth(), CheckEligibility()
[thought] Based on the eligibility, I can finalize the account type. [API] FinalizeAccountType()	GetAccountTypes()
[thought] Now, I can open the account. [API] OpenAccount()	Confirm(), OpenAccount()

Table 16: API hallucination examples by `ToolAlpaca`. Hallucinated APIs are highlighted.