

Article

LLMPC: Large Language Model Predictive Control

Gabriel Maher 

Independent Researcher, San Francisco, CA 94115, USA; gabriel.d.maher@gmail.com

Abstract: Recent advancements in planning prompting techniques for Large Language Models have improved their reasoning, planning, and action abilities. This paper develops a planning framework for Large Language Models using model predictive control that enables them to iteratively solve complex problems with long horizons. We show that in the model predictive control formulation, LLM planners act as approximate cost function optimizers and solve complex problems by breaking them down into smaller iterative steps. With our proposed planning framework, we demonstrate improved performance over few-shot prompting and improved efficiency over Monte Carlo Tree Search on several planning benchmarks.

Keywords: large language models; model predictive control; artificial intelligence

1. Introduction

Recent work has demonstrated remarkable success in enhancing Large Language Model (LLM) capabilities through integration with planning and control algorithms. From early prompting techniques like chain-of-thought [1] and self-consistency [2], to more sophisticated approaches combining LLMs with tree search [3,4] and reinforcement learning [5], researchers have found that guiding LLM outputs with structured planning significantly improves performance on complex reasoning tasks.

In robotics and embodied AI, frameworks like SayCan [6] and Voyager [7] successfully use LLMs to generate action plans for physical systems. LLM-Assist demonstrated improved autonomous vehicle planning by combining LLMs with traditional planners [8]. Knowledge-guided approaches like KnowAgent [9] further enhance LLM planning by incorporating domain expertise and constraints. Enabling LLMs to solve planning problems involving safety constraints was explored with the SageAgent benchmark [10].

Several approaches have explored using LLMs for hierarchical and multi-agent planning. Two-step goal decomposition [11] showed that LLMs could break down complex multi-agent tasks more effectively than traditional PDDL planners. ISR-LLM [12] and hierarchical constraint planning [13] demonstrated improved performance on problems requiring coordination between multiple constraints and objectives.

Search-based methods have proven particularly effective when combined with LLMs. Tree search algorithms [3,4,14,15] use LLMs to guide exploration while value functions evaluate candidate solutions. Scattered forest search [16] showed improved solution diversity in code generation tasks. Natural language planning [17] demonstrated that describing search spaces in natural language improved code generation performance. In the context of web agents, some work has explored the use of LLMs as world models [18,19].

Recent work has also explored enhancing LLM planning through learning. Supervised fine-tuning approaches like TravelPlanner [20] showed improved planning performance through targeted training. World models have been used to enable more structured planning [21], while continual learning approaches [22] demonstrated better exploration in reinforcement learning settings.



Academic Editors: Mads Sloth Vinding, Ivan Maximov and Christoph Stefan Aigner

Received: 21 February 2025

Revised: 14 March 2025

Accepted: 14 March 2025

Published: 15 March 2025

Citation: Maher, G. LLMPC: Large Language Model Predictive Control. *Computers* **2025**, *14*, 104. <https://doi.org/10.3390/computers14030104>

Copyright: © 2025 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

A key underpinning of the previous approaches is the ability of LLMs to act as general-purpose problem solvers after they have undergone instruction fine-tuning [23,24]. However, progress has been hampered by a number of limitations of instruction fine-tuned LLMs. While the LLM will produce a proposed solution to a given input task, there are no guarantees on the optimality of the solution. Indeed, the distribution of solutions produced by LLMs is still not well understood, but empirical evidence indicates there are limits to the complexity of tasks that can be solved by LLMs. Additionally, limits on the input sequence length prevent the use of LLMs on problems that require large amounts of input context, such as, e.g., developing a large codebase.

While planning prompts and search algorithms have been successful in extending the problem domains that can be solved using LLMs, a number of open questions remain. The first is that more explanation is needed as to why planning algorithms and formulations can improve the performance of LLMs on complex tasks. Second, the key factors and tradeoffs that influence the quality of LLM planning algorithms could be better understood. Finally, it is necessary to explore how LLM planning algorithms can overcome the limitations of problems that require large input and output token sequences. We show that integration of LLMs as planners in an MPC framework explains how planning algorithms help to boost LLM performance, illustrates what the key design considerations are, and provides a natural way to handle large input and output spaces.

Our work makes the following key contributions:

- We formulate a novel framework, LLMPC, that integrates Large Language Models with model predictive control principles to enable iterative solving of complex planning problems.
- We highlight that planning methods improve LLM performance by breaking large complex problems down into simpler sub-tasks that can be solved by the LLM due to instruction fine-tuning.
- We show that sampling multiple plans from LLMs and selecting the best according to a cost function significantly improves planning performance as problem complexity increases.
- We empirically validate LLMPC on three diverse planning benchmarks, showing improved performance over few-shot prompting and better computational efficiency than Monte Carlo Tree Search approaches.

In this paper, we first discuss the model predictive control framework and how it is used to make control problems with long horizons tractable in Section 2. Thereafter, in Section 3 we show our proposed method of integrating LLMs with MPC to create an improved LLM planning procedure. In Section 4, we evaluate our method on several problems, including a dynamical spring–mass system, a trip-planning problem, and a meeting scheduling problem. In particular, we compare LLMPC against a few-shot prompting approach and against MCTS [4]. Finally, we discuss the results in Section 5.

The code for our experiments is available at <https://github.com/gmaher/llmpc> (accessed on 14 March 2025).

2. Model Predictive Control

Model predictive control is a control framework that was developed to enable autonomous agents to solve control problems with long or even infinite horizons [25–27]. In the model predictive control setting, an agent must navigate a state space $s_t \in S \subset \mathbb{R}^d$ by selecting actions from an action space, $a_t \in A \subset \mathbb{R}^m$. The agent's state is then updated based on the current state and the selected action according to a state transition model $s_{t+1} = f(s_t, a_t)$. In stochastic formulations of MPC, the state is instead sampled from a distribution conditioned on the the current state and selected action $s_{t+1} \sim P(S|s_t, a_t)$.

Typically, the control problem will have constraints on feasible states and actions as well as a goal state and notion of cost to indicate preferred states and actions. Given a maximum horizon length T and initial state s_0 , these conditions can be modeled through the use of a cost function $C : S^H \times A^T \rightarrow \mathbb{R}$ and a set of equality $g_i : S^T \times A^T \rightarrow \mathbb{R}$ and inequality $h_j : S^T \times A^T \rightarrow \mathbb{R}$ constraint functions. The exact solution of the planning problem is obtained by solving the following constrained optimization problem:

$$\begin{aligned} a_0^*, \dots, a_{T-1}^* &:= \arg \min_{a_0, \dots, a_{T-1}} C(s_0, \dots, s_T, a_0, \dots, a_{T-1}) \\ \text{subject to } s_{t+1} &= f(s_t, a_t) \quad t = 0, \dots, T-1 \\ g_i(s_0, \dots, s_T, a_0, \dots, a_{T-1}) &= 0 \quad i = 1, \dots, n_g \\ h_j(s_0, \dots, s_T, a_0, \dots, a_{T-1}) &\leq 0 \quad j = 1, \dots, n_h \end{aligned} \quad (1)$$

However, for most problems of interest solving (1) is intractable, as the horizon may be infinite or large and typically one or more of the cost function, state update function, and constraint functions are nonlinear.

MPC relaxes the exact planning problem by introducing a shorter finite horizon H and the concept of replanning. In the MPC framework, the relaxed version of (1) is solved from the current state s_t to obtain the next H actions. A number of actions are executed to obtain the next states, and then the relaxed problem is again solved from the new state to obtain the next actions, and so forth. As such, MPC approximately solves (1) through iterated replanning over a shorter horizon. Furthermore, MPC enables linearized approximations of any nonlinear functions in (1) to be used, as replanning around the current state with a short horizon reduces the magnitude of state and action changes. This increases the likelihood the linearizations will remain accurate.

Given the horizon H and current state s_t , the MPC relaxation is formulated as

$$\begin{aligned} \hat{a}_t \dots \hat{a}_{t+H-1} &:= \arg \min_{a_t, \dots, a_{t+H-1}} \tilde{C}(s_t, \dots, s_{t+H}, a_t, \dots, a_{t+H-1}) \\ \text{subject to } s_{k+1} &= \tilde{f}(s_k, a_k) \quad k = t, \dots, t+H-1 \\ \tilde{g}_i(s_t, \dots, s_{t+H}, a_t, \dots, a_{t+H-1}) &= 0 \quad i = 1, \dots, n_g \\ \tilde{h}_j(s_t, \dots, s_{t+H}, a_t, \dots, a_{t+H-1}) &\leq 0 \quad j = 1, \dots, n_h. \end{aligned} \quad (2)$$

where the tilde indicates potential linearized forms of any of the relevant functions. In contrast to (1), (2) can often be solved exactly or to numerical precision limits through the use of constrained optimization algorithms.

3. LLM as MPC Plan Sampler

As previously discussed, instruction fine-tuned LLMs produce sample solutions to a given input text description of a task $q \in Q^L := \{1, 2, \dots, M\}^L$. However, they suffer from limitations when the task is sufficiently complex or the input and output sequences are long. We note the striking similarity to the exact form of the MPC problem, where the exact form of MPC is intractable due to the long horizon. We propose that, similar to MPC relaxation, complex problems can be tractably solved by LLMs by iteratively solving finite horizon relaxed versions of the problem instead of trying to solve the full problem in one step. Furthermore, this highlights why planning algorithms can enhance LLM performance: the iterative relaxation bounds the complexity at each step, increasing the likelihood that the LLM can solve the problem at that step. Here, we describe a precise framework for integrating LLMs with MPC to allow them to iteratively solve complex long-horizon problems.

Due to instruction fine-tuning, to integrate LLMs with MPC we can formulate an input task to the LLM that incorporates the state s_t , the cost function $\tilde{C}$, and the constraints $\tilde{g}_i, \tilde{h}_j$; and tasks the LLM with producing the sequence of H next actions $\hat{a}_t, \dots, \hat{a}_{t+H-1}$. Since the LLM operates on a length L sequence of input tokens $q_t \in Q^L := \{1, 2, \dots, M\}^L$, we can use a prompting function $E : S \times Q^L \rightarrow Q^L$ to use the sequence of tokens generated up to step t and the current state s_t to produce the next LLM input q_{t+1} (Figures 1 and 2). The cost function and constraints can be incorporated through the prompting function as well.

The LLM will then produce a probability vector over the possible M tokens for the next token in the sequence (Figure 1). Various methods are used for sampling the actual next token value [28]. For example, Beam Search is a common approach, where several likely token sequences are sampled by selecting the top most likely tokens at each step and using these to sample subsequent tokens, with repeated pruning to remove unlikely sequences. Repeatedly evaluating the LLM with the new output tokens produces a sequence of output tokens $y \in \{1, 2, \dots, M\}^{L_o}$, where L_o is the length of the output sequence (Figure 1). We can represent the process of sampling the output token sequence as a function $F : \{1, 2, \dots, M\}^L \rightarrow \{1, 2, \dots, M\}^{L_o}$.

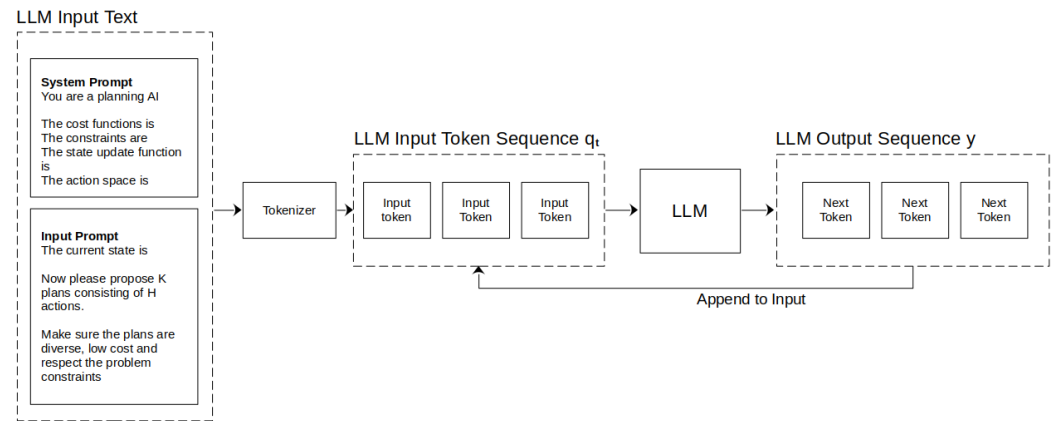


Figure 1. Diagram illustrating sampling the output token sequence y from the input token sequence q_t using the LLM. The input prompt describes the task for the LLM in text form. A tokenizer converts the input text to the sequence of input tokens q_t . The LLM function then produces an output probability distribution from which the next token is sampled. The generated tokens are repeatedly added to the input sequence to obtain the subsequent tokens.

To close the MPC control loop, we then need a way to extract the action sequence from the LLM output token sequence. This can be carried out using a mapping function $\phi : \{1, 2, \dots, M\}^{L_o} \rightarrow A^H$, similar to the Tool-Use scenario [29]. We now see that the LLM plan approximately solves the relaxed MPC problem through

$$\hat{a}_t \dots \hat{a}_{t+H} = \phi(y) := \phi(F(q_t)) := \phi(F(E(s_t, q_{t-1}))) \quad (3)$$

In the MPC framework, we thus see that generating a plan using the LLM is equivalent to approximately solving the constrained minimization of the MPC cost function. Indeed, due to instruction fine-tuning LLMs can be used as approximate cost function minimizers, but there is no guarantee on the optimality of their proposed solutions [30,31]. Since the LLM output y is a sequence of random variables, the LLM-generated plan will also be random. We propose that the generated LLM plan can be improved by using the LLM to sample a number of control sequences and then using the cost function to evaluate and rank each plan. The best sample is selected as the control sequence to apply.

That is, we sample a number of plans $A_t^i := \{a_t^i, \dots, a_{t+H}^i\}$ from the LLM. For each plan, we simulate the system to obtain the states $S_t^i := \{s_t^i, \dots, s_{t+H}^i\}$. We then pick the plan that minimizes the cost function among the sampled plans

$$\hat{A}_t := \arg \min_{i=1, \dots, K} C(S_t^i, A_t^i) \quad (4)$$

We then execute up to H steps from the plan $\hat{A}_t$ and then replan with the new state information. A diagram for the full LLMP controller algorithm is shown in Figure 2.

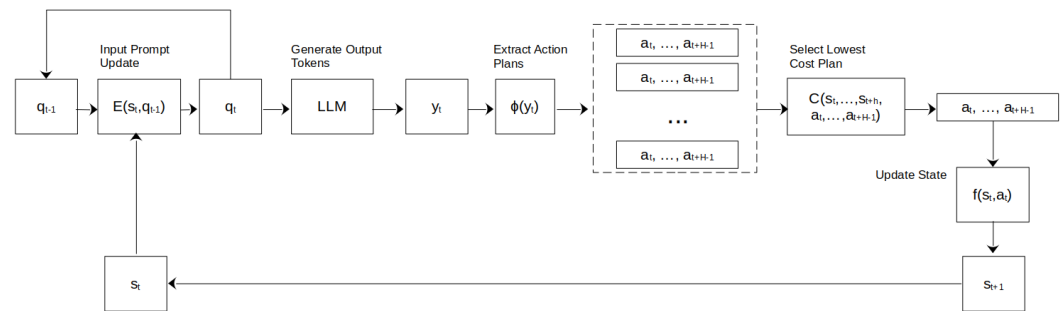


Figure 2. LLMP controller diagram. The new LLM input q_t is obtained from the current state and previous prompt through $q_t = E(s_t, q_{t-1})$. Evaluating the LLM on q_t produces the output token sequence y , containing the plans proposed by the LLM. The plans are extracted from the output sequence using the extraction function $\phi(y)$. Using the cost function, the lowest cost plan is selected. The next state is then obtained through the state update function $s_{t+1} = f(s_t, a_t)$. The process then repeats for the next state.

We note that sampling multiple plans can be carried out in a performant and computationally efficient manner by asking the LLM to produce multiple diverse plans in the input prompt rather than naively evaluating the same prompt K times. Asking for diverse plans enables the space of plans to be sampled more effectively, thus increasing the likelihood that low-cost plans will be found. Additionally, using modern LLM inference methods such as Key-Value caching [32], the input prompt only needs to be evaluated once to sample multiple plans rather than once for each plan, substantially reducing the computational cost.

Finally, we note that in the constrained MPC case, a plan generated by the LLM may not necessarily respect the constraints of the problem and may potentially lead to unsafe or invalid control trajectories [10,33–35]. Here too, sampling multiple plans can be used to improve constraint satisfaction; each sampled plan can be evaluated using the constraint functions $\hat{g}_i, \hat{h}_j$ and invalid plans discarded, leaving only feasible plans to select from.

4. Experiments

4.1. Control of Spring-and-Mass System

Here, we compare LLMP against MPC on the problem of applying force to a spring-and-mass system to arrive at a particular goal state. The equations of motion for the system are

$$a_t = \frac{1}{m}(u - k(x - x_0 - l)),$$

$$v_{t+1} = v_t + dt a_t,$$

$$x_{t+1} = x_t + dt v_t.$$

Here, u is a force applied by the controller to control the spring. The objective is to bring the spring to the goal state x^* with zero velocity. The cost function is

$$c_t = Q_x(x_{t+H} - x^*)^2 + Q_v(v_{t+H} - v^*) + \sum_{k=t}^{t+H} Q_u u_k^2 \quad (5)$$

For both MPC and LLMPC, we set $H = 3$ and execute two of every three steps from the returned action sequence. For MPC, we use CVXPY to solve the planning problem. For LLMPC, we use GPT-4o-mini, a temperature of 0.1, and a topp of 1.0. The LLM is asked to suggest five plans at every step using a templated prompt (listing A1 in Appendix A). Each of the suggested action sequences is evaluated using (5), and the plan with the lowest cost value is selected.

We solved the control problem with both MPC and LLMPC with $x_0 = 1$, $v_0 = 0$, $x^* = 2$, $v^* = 0$, $m = 1$, and $k = 5$ (Figure 3). Both MPC and LLMPC produce control sequences that control the spring to the goal state. As expected, the cost values from the plans produced by LLMPC are higher than when solving the problem exactly with MPC, highlighting that LLMs are approximate planners.

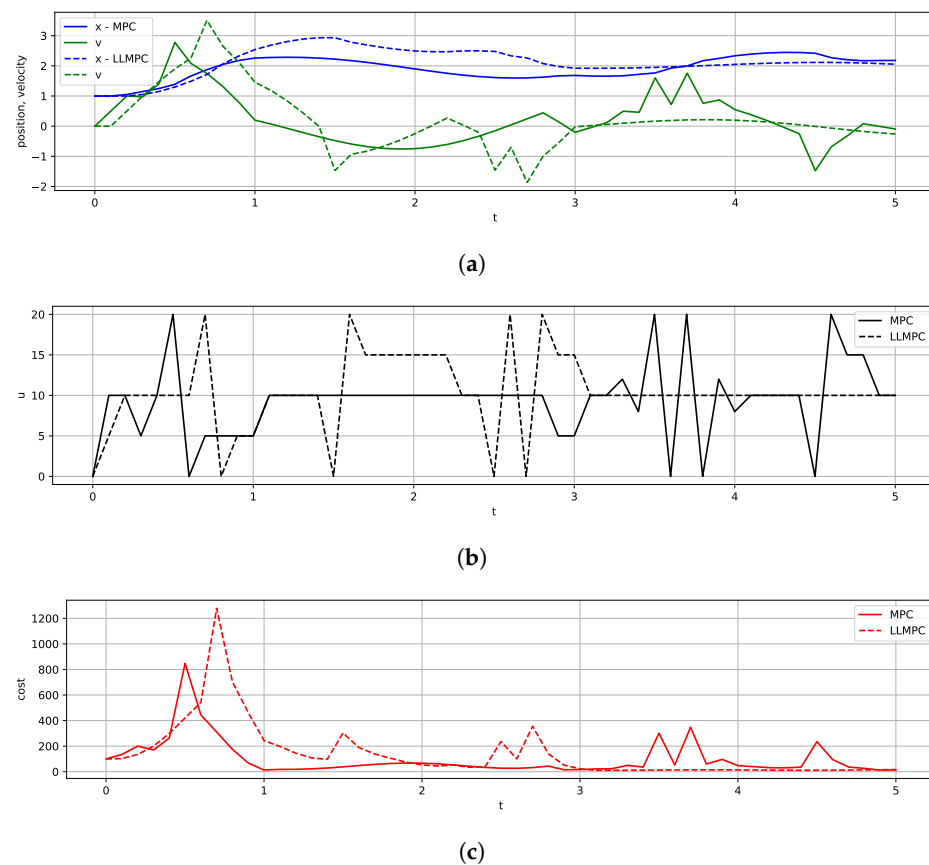
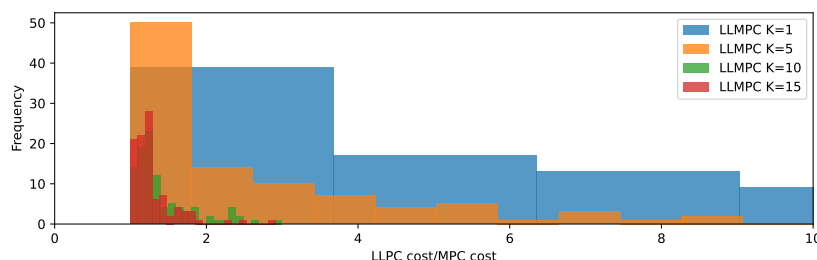


Figure 3. State, control, and cost trajectories for MPC and LLMPC algorithms on spring-mass problem: (a) position and velocity trajectory; (b) control trajectory; and (c) cost trajectory.

To further illustrate the approximate planning capacities of LLMs, we sample a range of state vectors s_t for the mass-spring system and solve the planning problem using both LLMPC and MPC at each sampled state. We then calculate the ratio of the optimized cost value returned by LLMPC against that of MPC (Table 1, Figure 4). We see that sampling only a single plan from an LLM produces poor cost values when compared to MPC (cost ratio of 8.21). However, as we increase the number of plans the performance of LLMPC significantly increases, with the ratio decreasing to 1.3 when 15 plans are sampled per iteration.

Table 1. Mean cost ratio for LLMPC vs MPC across range of states and number of plans sampled.

	$K = 1$	$K = 5$	$K = 10$	$K = 15$
Ratio LLMPC cost/MPC cost	8.21	2.89	1.42	1.30

**Figure 4.** Ratio of LLMPC optimized planning cost value to MPC optimized planning cost value for mass-spring system over a range of states.

4.2. Trip Planning

We evaluate LLMPC on the trip-planning test case from the Natural Plan benchmark [36]. The trip-planning problem involves proposing a travel itinerary to visit multiple cities, with given constraints on the number of days for visiting each city and the available flights between cities (see listing A2 in Appendix B for an example). We compare LLMPC with increasing iteration budgets against a single-round few-shot prompt solution and MCTS. For both LLMPC and the few-shot prompting approach, we used GPT-4o, a temperature of 0.2, and a *topp* of 1.0. For MCTS, we used a temperature of 0.7 to enhance its ability to explore the solution space and simulation budgets of $N = 20$ and $N = 40$.

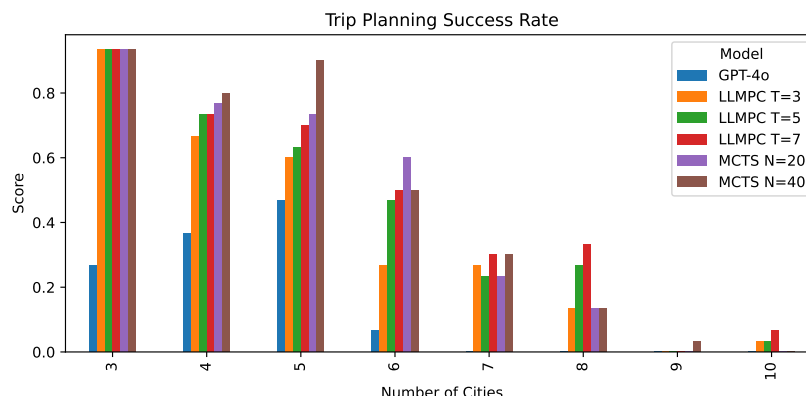
We construct a sequential planning system prompt and instruction prompt (listings A3 and A4 in Appendix B) that provides the problem description to an LLM along with few-shot examples and feedback on the previously proposed plan. To provide feedback on the previous plan, we use an evaluation function that checks whether the constraints from the problem description are met and lists any unmet constraints. If no plan has been proposed yet, the feedback section is omitted. The LLM is asked to then propose a new plan to address any issue and solve the problem. With these prompts, the trip-planning problem can be solved using LLMPC by proposing an initial plan without feedback and then sequentially iterating and improving the proposed plan.

We measure the overall success rate and success rate segmented by the number of cities the problem description lists. Adding more cities to the problem increases its complexity and difficulty. The results show that LLMPC significantly outperforms single-round GPT-4o planning, with success rates improving from 14.5% to 44.6% as we increase the number of iterations (Table 2). The performance gains are particularly notable for problems with more cities to visit (Figure 5). Increasing the iteration budget for LLMPC improves the performance as problem complexity increases. However, as the number of cities increases beyond eight cities, LLMPC is also not able to solve the itinerary.

We also compare LLMPC against Monte Carlo Tree Search (MCTS), a search algorithm that has shown strong performance on sequential decision problems. With a search budget of $N = 40$ simulations, MCTS achieves a success rate of 45%, slightly outperforming our best LLMPC configuration (44.6%). However, LLMPC uses significantly fewer iterations, 7 for LLMPC vs 40 for the best MCTS configuration, showing that LLMPC is computationally more efficient for the trip-planning problem while still being competitive with MCTS. Furthermore, LLMPC performs better as problem complexity increases (Figure 5).

Table 2. Trip planning success rate for single-round GPT-4o and LLMPC.

	GPT-4o	LLMPC $T = 3$	LLMPC $T = 5$	LLMPC $T = 7$	MCTS $N = 20$	MCTS $N = 40$
Success Rate	0.145	0.363	0.413	0.446	0.425	0.45

**Figure 5.** Trip planning success rate grouped by number of cities.

4.3. Meeting Planning

We evaluate LLMPC on the meeting planning test case from the Natural Plan benchmark [36]. In this problem, a user must plan meetings with multiple friends in different locations across a city while satisfying temporal and spatial constraints. Each friend is available at a specific location during a fixed time window and requires a minimum meeting duration. The planner must account for travel times between locations, meeting duration requirements, and friend availability windows to construct a valid schedule that maximizes the number of successful meetings.

The meeting planning problem presents several key challenges that make it an ideal test case for LLMPC. There are complex temporal dependencies between travel times, meeting durations, and availability windows. Additionally, the problem contains spatial constraints from the city travel network that limit possible meeting sequences. Finally, the problem involves multiple competing objectives around maximizing meetings while satisfying minimum durations.

For LLMPC, we again construct a system prompt and instruction prompt that provides the problem description, few-shot examples, and plan feedback as input (listings A5 and A6 in Appendix C). We again use an evaluation function to assess any unmet constraints of the proposed plan and provide these as feedback.

We compare LLMPC against single-round few-shot prompting and evaluate different configurations:

1. Varying the number of iterations T to refine plans;
2. Sampling multiple plans per iteration ($K > 1$);
3. Combinations of iteration and sampling to balance exploration and refinement.

We also compare it to MCTS. For both LLMPC and the few-shot approach, we used GPT-4o, a temperature of 0.5, and a $topp$ of 1.0. For MCTS, we used a temperature of 0.7 to enhance its ability to explore the solution space and simulation budgets of $N = 20$ and $N = 40$.

We measure the overall success rate and success rate segmented by number of friends that are listed in the problem description. More meetings increase the complexity and difficulty of the problem.

The results demonstrate that LLMPC provides consistent improvements over single-round planning (Table 3). Increasing both the number of iterations T and plans per iteration

K leads to better performance, with the best results achieved using $T = 9$ and $K = 3$ (67% success rate vs 52.5% for single-round planning). These results highlight the fact that the largest improvements come from both sampling multiple plans and iterating on previous solutions. Figure 6 shows the performance breakdown, highlighting LLMP’s ability to better handle the temporal and spatial constraints of the meeting planning problem. For the meeting planning problem, as complexity increases it becomes increasingly necessary to both iterate and sample multiple plans.

When comparing LLMP to Monte Carlo Tree Search (MCTS), we observe that our best LLMP configuration ($T = 9, K = 3$) outperforms MCTS with both $N = 20$ and $N = 40$ simulation budgets. MCTS achieves success rates of 60.3% and 60.5%, respectively, compared to LLMP’s 67%. This again highlights the improved computational efficiency of LLMP vs MCTS. Similar to the trip-planning problem, LLMP outperforms MCTS, particularly as problem complexity increases (Figure 6).

Table 3. Meeting planning success rate for single-round GPT-4o and LLMP.

	GPT-4o	LLMP $T = 5$	LLMP $T = 10$	LLMP $T = 15$	LLMP $T = 3, K = 3$	LLMP $T = 9, K = 3$	MCTS $N = 20$	MCTS $N = 40$
Success Rate	0.525	0.555	0.565	0.595	0.56	0.67	0.603	0.605

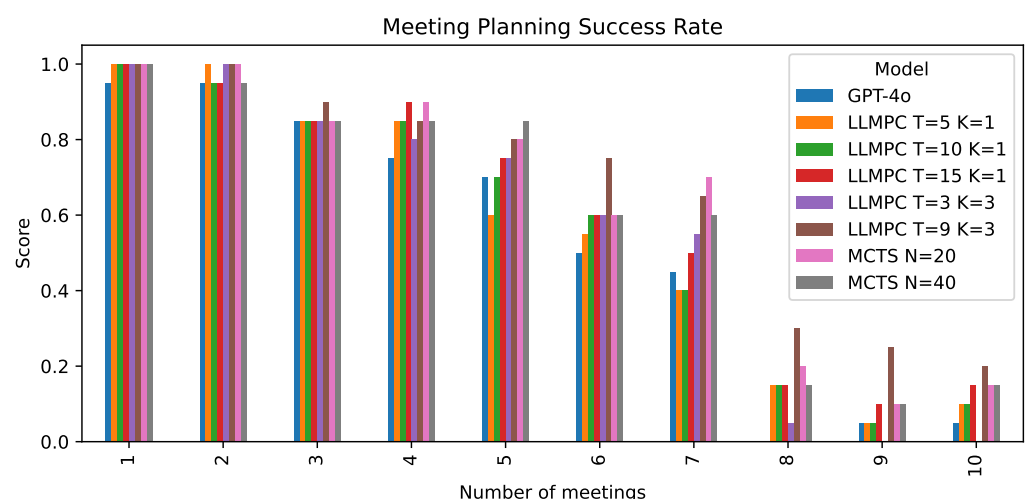


Figure 6. Comparison of LLMP against GPT-4o with few-shot prompt on meeting planning problem, success rate segmented by number of meetings.

5. Discussion

This paper introduced LLMP—an MPC control framework for improving the ability of LLMs to solve complex problems. The framework relaxes complex problems into an iterative stepwise replanning formulation, allowing the LLM to solve a simpler problem at each planning step. We showed that with this formulation, LLMs act as implicit optimization algorithms when generating plans and that making this optimization explicit through MPC improves performance. Furthermore, sampling multiple plans from LLMs and selecting the best according to a cost function significantly improves planning performance.

We demonstrated these benefits empirically on three test cases—a mass-spring control problem, trip planning, and meeting scheduling. In all cases, LLMP improved performance over single-round few-shot prompting, with the gains increasing as problem complexity grew. For the spring-mass system, increasing the number of sampled plans reduced the cost gap with exact MPC from $8.21 \times$ to $1.3 \times$. On the Natural Plan

benchmarks, LLMPc improved success rates by 30.1% on trip planning and 14.5% on meeting scheduling.

When comparing LLMPc to Monte Carlo Tree Search (MCTS), we observed several interesting differences. While both approaches iterate through potential solutions, LLMPc achieved comparable or better performance with fewer iterations—particularly on the meeting planning benchmark, where LLMPc with $T = 9$, $K = 3$ outperformed MCTS with a budget of 40 simulations. This suggests that LLMPc's explicit cost function evaluation and iterative refinement provide a more computationally efficient approach to planning than purely search-based methods in these domains. Furthermore, LLMPc demonstrated stronger performance as problem complexity increased. This advantage likely stems from LLMPc's ability to directly incorporate feedback from constraint violations into subsequent planning iterations, whereas MCTS must discover these constraints through exploration. Future work could explore hybrid approaches that combine LLMPc's efficient feedback mechanism with MCTS's structured exploration.

Future work could explore several promising directions. Combining LLMPc with other planning techniques like beam search may yield further improvements. Additional work could explore the use of learned cost functions and state update functions when exact versions are not available. Further work is also needed to address problems of increasing complexity. Here, a promising direction is to develop alternative state and action representations that will work better with an MPC framing.

We note a limitation of LLMPc in the case of planning tasks requiring explicit safety guarantees. LLMPc handles constraints through sampling multiple plans from an LLM, which does not provide strict guarantees on generating a feasible plan. As such, future work should focus on improved methods to guarantee feasibility and constraint satisfaction when planning using LLMs.

The LLMPc framework provides both practical benefits for real-world planning problems and theoretical insights into how LLMs function as planners. We believe this perspective will be valuable for developing more capable AI planning systems that combine the flexibility of LLMs with the rigor of traditional control approaches.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code and data for experiments are available at <https://github.com/gmaher/llmpc> (Accessed on 14 March 2025) The Natural Plan benchmark data are available at <https://github.com/google-deepmind/natural-plan> (Accessed on 6 March 2025)

Conflicts of Interest: The author declares no conflicts of interest.

Appendix A. Mass–Spring System

Listing A1. LLMPc prompt template for mass-spring problem (python v3.12)

```
prompt = f """
Given:
- A mass–spring system with position  $x$  and velocity  $v$ .
- Dynamics:

$$x_{(k+1)} = x_k + dt * v_k$$


$$v_{(k+1)} = v_k + (dt/m)*(u_k - k_{spring}*x_k)$$

- Parameters:  $m=\{m\}$ ,  $k_{spring}=\{k_{spring}\}$ ,  $dt=\{dt\}$ 
- Current state:  $x=\{x_{init}\}$ ,  $v=\{v_{init}\}$ 
```

- Goal position: $x_{\text{goal}} = \{x_{\text{goal}}\}$
- Horizon: $H = \{H\}$
- The current spring force is $\{-k_{\text{spring}} \cdot x_{\text{init}}\}$

You control the force on the spring via the control sequence $u = [u_0, u_1, \dots, u_{H-1}]$.

You must apply forces to get the spring to the goal position. Please propose $\{K\}$ candidate control sequences, each being a list of length H .

- Controls should be between 0 and 20
- Return them as a Python dictionary with keys "sequence_1", "sequence_2", ..., "sequence_{K}", where each value is a list of length H . Example:

```
{
  "sequence_1": [u_0, u_1, ..., u_{H-1}],
  "sequence_2": [u_0, u_1, ..., u_{H-1}],
  ...
}
```

Do not use `'''python` tags, no extra commentary, just return the dictionary.

"""

Appendix B. Trip Planning

Listing A2. Example trip planning problem

TASK:

You plan to visit 10 European cities for 21 days in total. You only take direct flights to commute between cities. You plan to stay in Vienna for 3 days. You want to spend 5 days in Frankfurt. You want to spend 2 days in Oslo. You are going to attend a wedding in Oslo between day 20 and day 21. You want to spend 3 days in Prague. You would like to visit Valencia for 2 days. You want to meet a friend in Valencia between day 17 and day 18. You plan to stay in Dubrovnik for 2 days. You would like to visit Edinburgh for 5 days. From day 1 to day 5, there is a annual show you want to attend in Edinburgh. You plan to stay in London for 2 days. You plan to visit relatives in London between day 12 and day 13. You would like to visit Munich for 3 days. You would like to meet your friends at Munich between day 18 and day 20 to tour together. You would like to visit Budapest for 3 days.

Here are the cities that have direct flights: Valencia and Munich, Vienna and Munich, Vienna and Valencia, London and Budapest, London and Oslo, Edinburgh and Budapest, Frankfurt and Budapest, Frankfurt and London, Prague and Oslo, Edinburgh and Oslo, Edinburgh and Munich, Prague and Munich, London and Prague, Edinburgh and London, Edinburgh and Frankfurt, Dubrovnik and Munich, Dubrovnik and Vienna, Munich and Oslo, Dubrovnik and Oslo, Budapest and Munich, Frankfurt and Prague, Vienna and London, Frankfurt and Vienna, Frankfurt and Oslo, Frankfurt and Munich, Vienna and Oslo, Vienna and Prague, Budapest and Oslo, Budapest and Prague, London and Valencia, London and Munich, Frankfurt and Dubrovnik, Prague and Valencia, Frankfurt and Valencia, Edinburgh and Prague, Vienna and Budapest.

Find a trip plan of visiting the cities for 21 days by taking direct flights to commute between them. SOLUTION: Here is the trip plan for visiting the 10 European cities for 21 days:

SOLUTION:

```
**Day 1-5:** Arriving in Edinburgh and visit Edinburgh for 5 days.
**Day 5:** Fly from Edinburgh to Frankfurt.
**Day 5-9:** Visit Frankfurt for 5 days.
**Day 9:** Fly from Frankfurt to Dubrovnik.
**Day 9-10:** Visit Dubrovnik for 2 days.
**Day 10:** Fly from Dubrovnik to Vienna.
**Day 10-12:** Visit Vienna for 3 days.
**Day 12:** Fly from Vienna to London.
**Day 12-13:** Visit London for 2 days.
**Day 13:** Fly from London to Budapest.
**Day 13-15:** Visit Budapest for 3 days.
**Day 15:** Fly from Budapest to Prague.
**Day 15-17:** Visit Prague for 3 days.
**Day 17:** Fly from Prague to Valencia.
**Day 17-18:** Visit Valencia for 2 days.
**Day 18:** Fly from Valencia to Munich.
**Day 18-20:** Visit Munich for 3 days.
**Day 20:** Fly from Munich to Oslo.
**Day 20-21:** Visit Oslo for 2 days.
```

Listing A3. Trip Planning LLMPC System Prompt Template

You are an expert travel planner assistant. Your goal is to create and refine travel plans that satisfy all given constraints. Your only job is to focus on the constraints around cities to visit, number of days and ordering of the trip. You do not need to investigate activities, accomodation etc, only focus on satisfying the stated trip constraints. If you are revising an existing plan, don't forget to consider changing the starting city of the trip or reordering the cities visited.

Specifically you will be asked to propose a trip plan given constraints on the number of days, flights and order of locations to visit.

Note that when flying from one city to another it counts as a day spent in both cities and will count towards the number of days required to visit both of those cities. Take this into account when making your plan.

For example if we fly from city A to city B on day 7 the visit to city B will start on day 7 and the visit to city A will end on day 7.

Here are example Task descriptions and solutions:

TASK:

You plan to visit european cities for 10 days. You want to spend 5 days in Rome, 4 days in Amsterdam and 3 days in Paris.

You plan to meet a friend in Paris on the 9th day of the trip.

There are direct flights between Rome and Paris, Rome and Amsterdam.

Find a trip plan of visiting the cities for 10 days by taking direct flights to commute between them.

PLAN:

****Day 1-4:**** Visit Amsterdam for 4 days.

****Day 4-8:**** Fly from Rome to Amsterdam.

****Day 4-8:**** Visit Rome for 5 days.

****Day 8-9:**** Fly from Rome to Paris.

****Day 8-10:**** Visit Paris for 3 days, spend the 9th day with your friend as planned.

TASK:

You have been asked to solve the following trip planning task:

You plan to visit 3 European cities for 15 days in total. You only take direct flights to commute between cities. You want to spend 6 days in Athens. You want to spend 4 days in London. You want to spend 7 days in Madrid.

Here are the cities that have direct flights:

Madrid and London, from London to Athens.

Find a trip plan of visiting the cities for 15 days by taking direct flights to commute between them.

Output:

****Day 1-7:**** Visit Madrid for 7 days.

****Day 7-10:**** Fly from Madrid to London.

****Day 7-10:**** Visit London for 4 days.

****Day 10-15:**** Fly from London to Athens.

****Day 10-15:**** Visit Athens for 6 days.

TASK:

You plan to visit 8 European cities for 22 days in total. You only take direct flights to commute between cities. You would like to visit Vilnius for 4 days. You would like to visit Venice for 5 days. You plan to stay in Warsaw for 4 days. You want to meet a friend in Warsaw between day 14 and day 17. You want to spend 5 days in Mykonos. You plan to stay in Salzburg for 5 days. You plan to stay in Amsterdam for 2 days. You would like to meet your friends at Amsterdam between day 17 and day 18 to tour together. You plan to stay in Hamburg for 2 days. You would like to visit Copenhagen for 2 days.

Here are the cities that have direct flights:

Warsaw and Amsterdam, Hamburg and Venice, Hamburg and Warsaw, Venice and Warsaw, Hamburg and Amsterdam, Venice and Copenhagen, Vilnius and Amsterdam, Vilnius and Warsaw, Hamburg and Copenhagen, Salzburg and Hamburg, Copenhagen and Amsterdam, Copenhagen and Vilnius, Copenhagen and Warsaw, Venice and Amsterdam, Amsterdam and Mykonos.

Find a trip plan of visiting the cities for 22 days by taking direct flights to commute between them.

Output:

****Day 1-5:**** Visit Salzburg for 5 days.

****Day 5-6:**** Fly from Salzburg to Hamburg.

****Day 5-6:**** Visit Hamburg for 2 days.

****Day 6-10:**** Fly from Hamburg to Venice.

****Day 6-10:**** Visit Venice for 5 days.

****Day 10-11:**** Fly from Venice to Copenhagen.

****Day 10-11:**** Visit Copenhagen for 2 days.

****Day 11-14:**** Fly from Copenhagen to Vilnius.

****Day 11-14:**** Visit Vilnius for 4 days.

****Day 14-17:**** Fly from Vilnius to Warsaw.

****Day 14-17:**** Visit Warsaw for 4 days.

****Day 17-18:**** Fly from Warsaw to Amsterdam.

****Day 17-18:**** Visit Amsterdam for 2 days.

****Day 18-22:**** Fly from Amsterdam to Mykonos.

****Day 18-22:**** Visit Mykonos for 5 days.

Always format your response using the following template:

PLAN:

<your complete trip plan here>

Listing A4. Trip Planning LLMP Instruction Prompt Template

You have been asked to solve the following trip planning task:

TASK:

{task}

Your current trip plan is:

{current_plan}

{feedback_string}

Appendix C. Meeting Planning

Listing A5. Example meeting planning problem

You are visiting San Francisco for the day and want to meet as many friends as possible. Solve the problem by considering various different schedules and picking the best one to optimize your goals.

Travel distances (in minutes):

Marina District to Alamo Square: 15.
 Marina District to Fisherman’s Wharf: 10.
 Marina District to Union Square: 16.
 Marina District to Embarcadero: 14.
 Marina District to Financial District: 17.
 Alamo Square to Marina District: 15.
 Alamo Square to Fisherman’s Wharf: 19.
 Alamo Square to Union Square: 14.
 Alamo Square to Embarcadero: 17.
 Alamo Square to Financial District: 17.
 Fisherman’s Wharf to Marina District: 9.
 Fisherman’s Wharf to Alamo Square: 20.
 Fisherman’s Wharf to Union Square: 13.
 Fisherman’s Wharf to Embarcadero: 8.
 Fisherman’s Wharf to Financial District: 11.
 Union Square to Marina District: 18.
 Union Square to Alamo Square: 15.
 Union Square to Fisherman’s Wharf: 15.
 Union Square to Embarcadero: 11.
 Union Square to Financial District: 9.
 Embarcadero to Marina District: 12.
 Embarcadero to Alamo Square: 19.
 Embarcadero to Fisherman’s Wharf: 6.
 Embarcadero to Union Square: 10.
 Embarcadero to Financial District: 5.
 Financial District to Marina District: 15.
 Financial District to Alamo Square: 17.
 Financial District to Fisherman’s Wharf: 10.
 Financial District to Union Square: 9.
 Financial District to Embarcadero: 4.

CONSTRAINTS: You arrive at Marina District at 9:00AM. Mary will be at Alamo Square from 3:15PM to 6:30PM. You’d like to meet Mary for a minimum of 60 minutes. Deborah will be at Fisherman’s Wharf from 7:00PM to 10:00PM. You’d like to meet Deborah for a minimum of 45 minutes. Jason will be at Union Square from 11:00AM to 1:15PM. You’d like to meet Jason for a minimum of 75 minutes. Betty will be at Embarcadero from 2:00PM to 6:15PM. You’d like to meet Betty for a minimum of 90 minutes. Anthony will be at Financial District from 12:15PM to 9:30PM. You’d like to meet Anthony for a minimum of 105 minutes.

SOLUTION: You start at Marina District at 9:00AM. You travel to Union Square in 16 minutes and arrive at 9:16AM. You wait until 11:00AM. You meet Jason for 75 minutes from 11:00AM to 12:15PM. You travel to Alamo Square in 15 minutes and arrive at 12:30 PM. You wait until 3:15PM. You meet Mary for 60 minutes from 3:15PM to 4:15PM. You travel to Embarcadero in 17 minutes and arrive at 4:32PM. You meet Betty for 90 minutes from 4:32PM to 6:02PM. You travel to Financial District in 5 minutes and arrive at 6:07PM. You meet Anthony for 105 minutes from 6:07PM to 7:52PM. You travel to Fisherman’s Wharf in 10 minutes and arrive at 8:02PM. You meet Deborah for 45 minutes from 8:02PM to 8:47PM.

Listing A6. Meeting Planning LLMPC System Prompt Template

You are an expert meeting planner assistant. Your goal is to create and refine plans to meet friends at different places in the city, taking into account travel times and meeting time constraints.
 Your job is to iteratively create and modify a plan that meets with all the listed friends for the duration and location specified in the constraints.

You will be given a limited number of iteration to make a full plan, these will be indicated by STEP X/TOTAL_STEPS.
 For each step, propose {PLANS_PER_ITERATION} different possible plans. Make the plans meaningfully different from each other (e.g. meeting friends in different orders, starting with different meetings).

When no plan is given you should start by planning meetings with the first few required people.

When an existing plan is given you should consider:

1. Adding more meetings
2. Modifying meeting orders
3. Adjusting meeting durations
4. Trying completely different meeting sequences

If you believe you have the best possible plan make no modifications and output the existing plan {PLANS_PER_ITERATION} times.

Do not propose meeting with imaginary friends.

Only propose meetings with friends mentioned in the task description.

Do not propose multiple meetings with the same friend.

If the plan already includes all mentioned friends, do not make any modifications or propose additional meetings.

PLAN FORMAT:

A plan includes a start, meeting, traveling and waiting steps.

- * The start of a plan should be phrased as ‘You start at <location> at <start_time>’
- * A meeting step should be phrased as ‘You meet <friend_name> for <time_spent> minutes from <start_time> to <end_time>.’
- * A travel step should be phrased as ‘You travel to <location> in <travel_time> minutes and arrive at <arrival_time>.’
- * A waiting step should be phrased as ‘You wait until <end_time>.’

Only use the above phrasing, e.g. do not mention ‘You travel back to...’ only mention ‘You travel to...’

EXAMPLES:

Here are example input task descriptions and output plans:

You are visiting San Francisco for the day and want to meet as many friends as possible. Solve the problem by considering various different schedules and picking the best one to optimize your goals.

Travel distances (in minutes):

Marina District to Alamo Square: 15.
 Marina District to Fisherman’s Wharf: 10.
 Marina District to Union Square: 16.
 Marina District to Embarcadero: 14.
 Marina District to Financial District: 17.
 Marina District to Nob Hill: 12.
 Alamo Square to Marina District: 15.
 Alamo Square to Fisherman’s Wharf: 19.
 Alamo Square to Union Square: 14.
 Alamo Square to Embarcadero: 17.
 Alamo Square to Financial District: 17.
 Alamo Square to Nob Hill: 11.
 Fisherman’s Wharf to Marina District: 9.

Fisherman’s Wharf to Alamo Square: 20.
 Fisherman’s Wharf to Union Square: 13.
 Fisherman’s Wharf to Embarcadero: 8.
 Fisherman’s Wharf to Financial District: 11.
 Fisherman’s Wharf to Nob Hill: 11.
 Union Square to Marina District: 18.
 Union Square to Alamo Square: 15.
 Union Square to Fisherman’s Wharf: 15.
 Union Square to Embarcadero: 11.
 Union Square to Financial District: 9.
 Union Square to Nob Hill: 9.
 Embarcadero to Marina District: 12.
 Embarcadero to Alamo Square: 19.
 Embarcadero to Fisherman’s Wharf: 6.
 Embarcadero to Union Square: 10.
 Embarcadero to Financial District: 5.
 Embarcadero to Nob Hill: 10.
 Financial District to Marina District: 15.
 Financial District to Alamo Square: 17.
 Financial District to Fisherman’s Wharf: 10.
 Financial District to Union Square: 9.
 Financial District to Embarcadero: 4.
 Financial District to Nob Hill: 8.
 Nob Hill to Marina District: 11.
 Nob Hill to Alamo Square: 11.
 Nob Hill to Fisherman’s Wharf: 11.
 Nob Hill to Union Square: 7.
 Nob Hill to Embarcadero: 9.
 Nob Hill to Financial District: 9.

CONSTRAINTS: You arrive at Marina District at 9:00AM. Deborah will be at Alamo Square from 11:15AM to 1:30PM. You’d like to meet Deborah for a minimum of 45 minutes. Jason will be at Fisherman’s Wharf from 11:00AM to 1:15PM. You’d like to meet Jason for a minimum of 75 minutes. Betty will be at Union Square from 2:00PM to 6:15PM. You’d like to meet Betty for a minimum of 90 minutes. Anthony will be at Embarcadero from 12:15PM to 9:30PM. You’d like to meet Anthony for a minimum of 105 minutes. Daniel will be at Financial District from 7:00AM to 10:15AM. You’d like to meet Daniel for a minimum of 120 minutes. Jessica will be at Nob Hill from 6:00PM to 10:00PM. You’d like to meet Jessica for a minimum of 105 minutes.

SOLUTION: You start at Marina District at 9:00AM. You travel to Fisherman’s Wharf in 10 minutes and arrive at 9:10AM. You wait until 11:00AM. You meet Jason for 75 minutes from 11:00AM to 12:15PM. You travel to Alamo Square in 20 minutes and arrive at 12:35PM. You meet Deborah for 45 minutes from 12:35PM to 1:20PM. You travel to Union Square in 14 minutes and arrive at 1:34PM. You wait until 2:00PM. You meet Betty for 90 minutes from 2:00PM to 3:30PM. You travel to Embarcadero in 11 minutes and arrive at 3:41PM. You meet Anthony for 105 minutes from 3:41PM to 5:26PM. You travel to Nob Hill in 10 minutes and arrive at 5:36PM. You wait until 6:00PM. You meet Jessica for 105 minutes from 6:00PM to 7:45PM.

OUTPUT FORMAT:

For your response only output the keyword SOLUTION followed by {PLANS_PER_ITERATION} plans separated by ‘---’, do not number the plans or add headers, only output the text of the {PLANS_PER_ITERATION} plans, make sure the plans are different from each other, do not output anything other than this format:

SOLUTION:
 <insert first plan>

 <insert second plan>

 ...

Listing A7. Meeting Planning LLMP Instruction Prompt Template

```
STEP {step}/{total_steps}

You have been asked to solve the following meeting planning task, pay particular attention the constraints and propose 3 different possible plans according to the format described in the system prompt:

TASK:
{task}

Your current best meeting plan is:
{current_plan}

{feedback_string}

OUTPUT FORMAT:

For your response only output the keyword SOLUTION followed by {num_plans} plans separated by ‘---’, do not number the plans or add headers, only output the text of the {num_plans} plans, make sure the plans are different from each other, do not output anything other than this format:

SOLUTION:
<insert first plan>
---
<insert second plan>
---
...
```

References

1. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv* **2022**, arXiv:2201.11903.
2. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv* **2022**, arXiv:2203.11171.
3. Wang, A.; Song, L.; Tian, Y.; Peng, B.; Yu, D.; Mi, H.; Su, J.; Yu, D. LiteSearch: Efficacious Tree Search for LLM. *arXiv* **2024**, arXiv:2407.00320.

4. Jiang, J.; Chen, Z.; Min, Y.; Chen, J.; Cheng, X.; Wang, J.; Tang, Y.; Sun, H.; Deng, J.; Zhao, W.X.; et al. Enhancing LLM Reasoning with Reward-guided Tree Search. *arXiv* **2024**, arXiv:2411.11694.
5. DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv* **2025**, arXiv:2501.12948.
6. Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv* **2022**, arXiv:2204.01691.
7. Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; Anandkumar, A. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv* **2023**, arXiv:2305.16291.
8. Song, C.H.; Wu, J.; Washington, C.; Sadler, B.M.; Chao, W.L.; Su, Y. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 2–3 October 2023; pp. 2998–3009.
9. Zhu, Y.; Qiao, S.; Ou, Y.; Deng, S.; Zhang, N.; Lyu, S.; Shen, Y.; Liang, L.; Gu, J.; Chen, H. KnowAgent: Knowledge-Augmented Planning for LLM-Based Agents. *arXiv* **2024**, arXiv:2403.03101.
10. Yin, S.; Pang, X.; Ding, Y.; Chen, M.; Bi, Y.; Xiong, Y.; Huang, W.; Xiang, Z.; Shao, J.; Chen, S. SafeAgentBench: A Benchmark for Safe Task Planning of Embodied LLM Agents. *arXiv* **2024**, arXiv:cs.CR/2412.13178.
11. Singh, I.; Traum, D.; Thomason, J. TwoStep: Multi-agent Task Planning using Classical Planners and Large Language Models. *arXiv* **2024**, arXiv:cs.AI/2403.17246.
12. Zhou, Z.; Song, J.; Yao, K.; Shu, Z.; Ma, L. ISR-LLM: Iterative Self-Refined Large Language Model for Long-Horizon Sequential Task Planning. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024. [\[CrossRef\]](#)
13. Zhang, C.; Deik, D.G.X.; Li, D.; Zhang, H.; Liu, Y. Planning with Multi-Constraints via Collaborative Language Agents. *arXiv* **2024**, arXiv:cs.AI/2405.16510.
14. Hao, S.; Gu, Y.; Ma, H.; Hong, J.J.; Wang, Z.; Wang, D.Z.; Hu, Z. Reasoning with Language Model is Planning with World Model. *arXiv* **2023**, arXiv:cs.CL/2305.14992.
15. Putta, P.; Mills, E.; Garg, N.; Motwani, S.; Finn, C.; Garg, D.; Rafailov, R. Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents. *arXiv* **2024**, arXiv:2408.07199.
16. Light, J.; Wu, Y.; Sun, Y.; Yu, W.; Liu, Y.; Zhao, X.; Hu, Z.; Chen, H.; Cheng, W. Scattered Forest Search: Smarter Code Space Exploration with LLMs. *arXiv* **2024**, arXiv:cs.SE/2411.05010.
17. Wang, E.; Cassano, F.; Wu, C.; Bai, Y.; Song, W.; Nath, V.; Han, Z.; Hendryx, S.; Yue, S.; Zhang, H. Planning In Natural Language Improves LLM Search For Code Generation. *arXiv* **2024**, arXiv:cs.LG/2409.03733.
18. Gu, Y.; Zheng, B.; Gou, B.; Zhang, K.; Chang, C.; Srivastava, S.; Xie, Y.; Qi, P.; Sun, H.; Su, Y. Is Your LLM Secretly a World Model of the Internet? Model-Based Planning for Web Agents. *arXiv* **2024**, arXiv:cs.AI/2411.06559.
19. Chae, H.; Kim, N.; iunn Ong, K.T.; Gwak, M.; Song, G.; Kim, J.; Kim, S.; Lee, D.; Yeo, J. Web Agents with World Models: Learning and Leveraging Environment Dynamics in Web Navigation. *arXiv* **2024**, arXiv:cs.CL/2410.13232.
20. Chen, Y.; Pesaranghader, A.; Sadhu, T.; Yi, D.H. Can We Rely on LLM Agents to Draft Long-Horizon Plans? Let’s Take TravelPlanner as an Example. *arXiv* **2024**, arXiv:2408.06318.
21. Xiong, S.; Payani, A.; Yang, Y.; Fekri, F. Deliberate Reasoning for LLMs as Structure-aware Planning with Accurate World Model. *arXiv* **2024**, arXiv:cs.CL/2410.03136.
22. Paul, S.K. Continually Learning Planning Agent for Large Environments guided by LLMs. In Proceedings of the 2024 IEEE Conference on Artificial Intelligence (CAI), Singapore, 25–27 June 2024; pp. 377–382. [\[CrossRef\]](#)
23. Peng, B.; Li, C.; He, P.; Galley, M.; Gao, J. Instruction Tuning with GPT-4. *arXiv* **2023**, arXiv:cs.CL/2304.03277.
24. Ghosh, S.; Evuru, C.K.R.; Kumar, S.; S, R.; Aneja, D.; Jin, Z.; Duraiswami, R.; Manocha, D. A Closer Look at the Limitations of Instruction Tuning. *arXiv* **2024**, arXiv:cs.CL/2402.05119.
25. Mayne, D.Q.; Rawlings, J.B.; Rao, C.V.; Sckaert, P.O.M. Constrained model predictive control: Stability and optimality. *Automatica* **2000**, *36*, 789–814. [\[CrossRef\]](#)
26. Schwenzer, M.; Ay, M.; Bergs, T.; Abel, D. Review on model predictive control: An engineering perspective. *Int. J. Adv. Manuf. Technol.* **2021**, *117*, 1327–1349. [\[CrossRef\]](#)
27. Shi, H.; Zuo, L.; Wang, S.; Yuan, Y.; Su, C.; Li, P. Robust predictive fault-tolerant switching control for discrete linear systems with actuator random failures. *Comput. Chem. Eng.* **2024**, *181*, 108554. [\[CrossRef\]](#)
28. Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; Choi, Y. The Curious Case of Neural Text Degeneration. *arXiv* **2020**, arXiv:cs.CL/1904.09751.
29. Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv* **2023**, arXiv:2307.16789.
30. Guo, P.F.; Chen, Y.H.; Tsai, Y.D.; Lin, S.D. Towards Optimizing with Large Language Models. *arXiv* **2024**, arXiv:cs.LG/2310.05204.

31. Jiang, C.; Shu, X.; Qian, H.; Lu, X.; Zhou, J.; Zhou, A.; Yu, Y. LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch. *arXiv* **2025**, arXiv:cs.AI/2410.13213.
32. Li, H.; Li, Y.; Tian, A.; Tang, T.; Xu, Z.; Chen, X.; Hu, N.; Dong, W.; Li, Q.; Chen, L. A Survey on Large Language Model Acceleration based on KV Cache Management. *arXiv* **2025**, arXiv:cs.AI/2412.19442.
33. Hafez, A.; Akhormeh, A.N.; Hegazy, A.; Alanwar, A. Safe LLM-Controlled Robots with Formal Guarantees via Reachability Analysis. *arXiv* **2025**, arXiv:cs.RO/2503.03911.
34. Wu, Y.; Xiong, Z.; Hu, Y.; Iyengar, S.S.; Jiang, N.; Bera, A.; Tan, L.; Jagannathan, S. SELP: Generating Safe and Efficient Task Plans for Robot Agents with Large Language Models. *arXiv* **2025**, arXiv:cs.RO/2409.19471.
35. Hua, W.; Yang, X.; Jin, M.; Li, Z.; Cheng, W.; Tang, R.; Zhang, Y. TrustAgent: Towards Safe and Trustworthy LLM-based Agents. *arXiv* **2024**, arXiv:cs.CL/2402.01586.
36. Zheng, H.S.; Mishra, S.; Zhang, H.; Chen, X.; Chen, M.; Nova, A.; Hou, L.; Cheng, H.T.; Le, Q.V.; Zhou, D. Natural Plan: Benchmarking LLMs on Natural Language Planning. *arXiv* **2025**, arXiv:2406.04520.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.