Graph-enhanced Large Language Models in Asynchronous Plan Reasoning

Fangru Lin [1] **Emanuele La Malfa** [1 2] **Valentin Hofmann** [1 3 4] **Elle Michelle Yang** [1]
Anthony G. Cohn [2 5] **Janet B. Pierrehumbert** [1]

Abstract

Planning is a fundamental property of human intelligence. Reasoning about asynchronous plans is challenging since it requires sequential and parallel planning to optimize time costs. Can large language models (LLMs) succeed at this task? Here, we present the first large-scale study investigating this question. We find that a representative set of closed and open-source LLMs, including GPT-4 and LLaMA-2, behave poorly when not supplied with illustrations about the task-solving process in our benchmark AsynCHow. We propose a novel technique called *Plan Like a Graph* (PLaG) that combines graphs with natural language prompts and achieves state-of-the-art results. We show that although PLaG can boost model performance, LLMs still suffer from drastic degradation when task complexity increases, highlighting the limits of utilizing LLMs for simulating digital devices. We see our study as an exciting step towards using LLMs as efficient autonomous agents. Our code and data are available at `https://github.com/fangru-lin/graph-llm-asynchow-plan`.

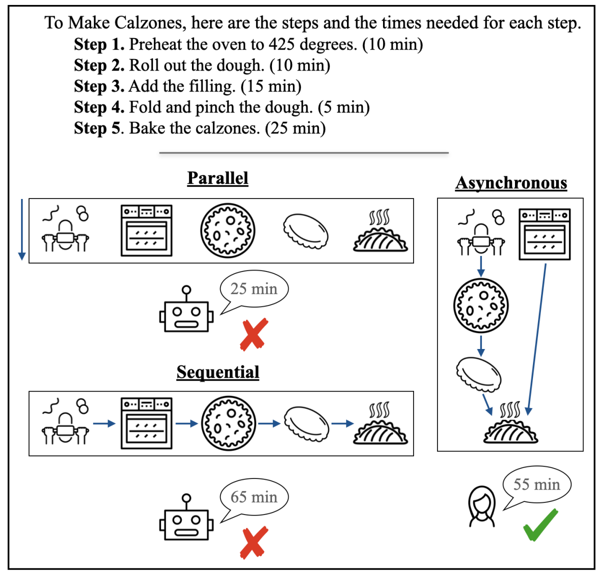

Figure 1. A planning task (top) can be executed sequentially, in parallel, or asynchronously. Blue arrows denote action ordering constraints. Although complete parallelism is logically the most time-efficient strategy, it results in invalid reasoning steps (e.g. `'Baking'` cannot happen at the same time with `'Rolling the dough'`); at the same time, sequentially executing each task negatively affects efficiency. Given infinite resources, an optimal (asynchronous) plan should parallelize actions wherever possible.

1. Introduction

As large language models (LLMs) show unprecedented capabilities, claims surge that artificial general intelligence is close (Bubeck et al., 2023). Planning is an important property of human intelligence (Sternberg, 1984; Colom et al., 2010), and it is also vital in many downstream tasks such as developing autonomous robotic agents (Huang et al., 2022a; Shinn et al., 2023). While symbolic processors have been historically used for handling plan design (Fikes & Nilsson, 1971; McDermott, 2000), LLMs have recently emerged as a relevant complementary approach (Ahn et al., 2022; Dagan et al., 2023; Song et al., 2023). Although LLMs generate reasonable elementary planning steps when informed with appropriate guidance (Huang et al., 2022b; Yuan et al., 2023), they cannot combine those units effectively and develop optimal plans without external processors (Silver et al., 2022; Dagan et al., 2023; Yang et al., 2023). This might be an issue if LLMs are deployed for related tasks.

This work explores the reasoning ability of LLMs in naturalistic asynchronous planning, which we define as complex planning tasks involving both sequential and parallel actions. Given a set of steps for a task, the time required for each step, and step ordering constraints, we ask whether LLMs can compute the shortest possible time needed for an optimal plan for the task (Figure 1). We note that asynchronous planning problems involve (i) time summation (correctly adding time durations), (ii) time comparison (correctly mak-

[1]University of Oxford [2]Alan Turing Institute [3]Allen Institute for AI [4]LMU Munich [5]University of Leeds. Correspondence to: Fangru Lin <fangru.lin@ling-phil.ox.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

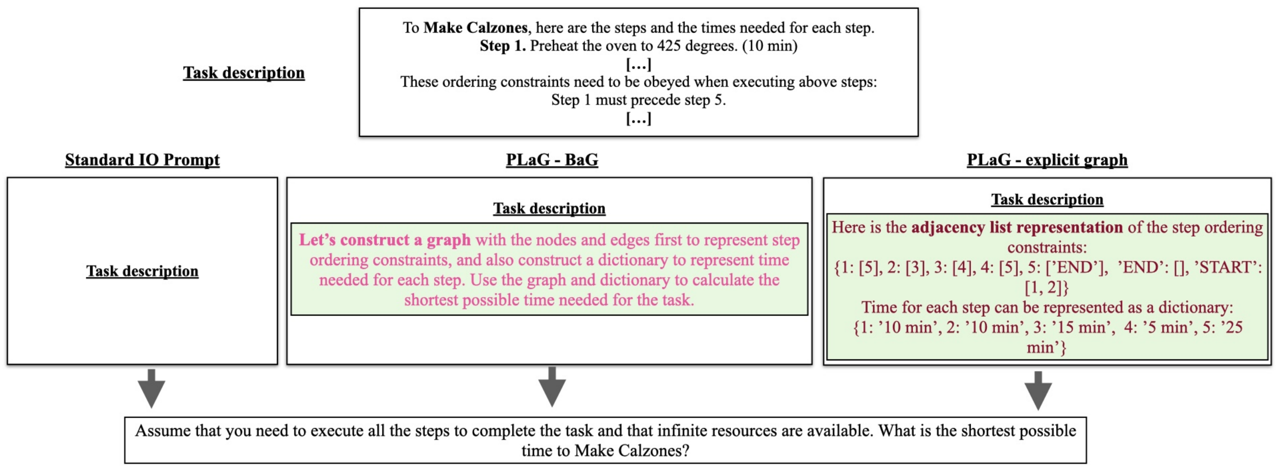

Figure 2. Comparing standard Input-Output (IO) prompting with our method (PLaG). Here, we illustrate PLaG (explicit graph) with an adjacency list, but it can be of any graph type in practice. The standard IO method is similarly deployed in zero-shot, zero-shot + CoT, k-shot, k-shot + CoT in this paper. Please refer to Appendix A.8 for more details.

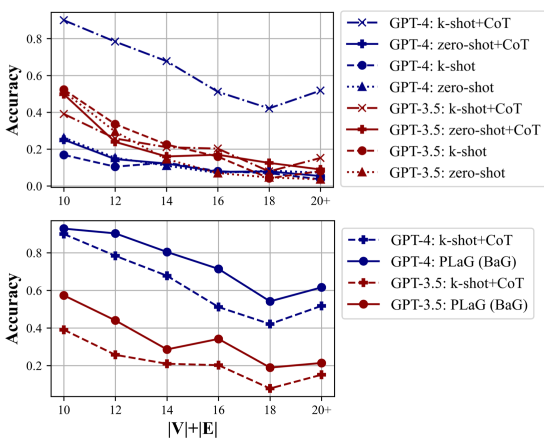

Figure 3. GPT-3.5 and GPT-4 accuracy as a function of asynchronous planning task complexity $|V| + |E|$ (see Section 2), after binning results by width of 2. The upper figure plots the performance of methods without PLaG (our method), and the lower plot displays the best method with/without PLaG.

ing time duration comparisons), and (iii) constrained reasoning (correctly solving constrained optimization problems) — this compositionality of skills makes asynchronous planning a challenging task, and it is yet unclear whether LLMs are capable of solving it. To enable a large-scale evaluation of LLMs, we automatically generate a new benchmark, **Async**hronous Wiki**How** (AsyncHow), with 1.6K high-quality instances for real-life tasks.

We use AsyncHow to evaluate GPT-3.5-turbo (GPT-3.5), GPT-4 (OpenAI, 2023), Cohere Command[1], LLaMA-2-70B-chat (Touvron et al., 2023), and Mistral-7B-Instruct (v0.2; Jiang et al., 2023) on asynchronous planning. We find that while GPT-4 with few-shot task solution illustrations dominates other models in terms of accuracy, all models perform poorly without illustrations about how to solve the task. However, even with few-shot illustrations, model performance is unsatisfactory, with the LLMs failing on instances that are trivial for humans. To remedy this, we propose a novel prompting technique, namely **P**lan **L**ike **a G**raph (PLaG; Figure 2), to instruct models to represent a planning problem like a graph. By converting naturalistic questions to equivalent graph problems, we find that our method boosts the performance of all tested models. Moreover, it can be applied **off the shelf** to models such as GPT-4 to achieve new state-of-the-art (SOTA) results and consistently improve on all task complexity levels (Figure 3, lower). Nonetheless, we find that the improved models still suffer from drastic performance degradation on complex planning tasks.

In summary, the main contributions of this paper are:

- We automatically generate a high-quality naturalistic benchmark for asynchronous plan reasoning, AsyncHow, and open-source it.
- We show that LLMs cannot efficiently execute asynchronous plans unless they are supplied with detailed solution illustrations.
- We provide a formalism to define the complexity of naturalistic asynchronous planning tasks, which successfully predicts LLMs' performance trends.
- We propose PLaG, an off-the-shelf method to consistently boost SOTA model performance across all considered task complexities.

[1]https://cohere.com/models/command

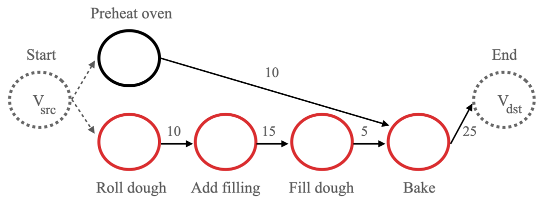

Figure 4. The series-parallel DAG used to solve the planning task in Figure 1. The path for calculating optimal time duration is highlighted in red.

- We show that despite the performance boost, SOTA LLMs suffer from drastic degradation with increasing task complexity, which indicates that there are limits to using LLMs as digital devices.

The paper is structured as follows. Section 2 introduces our asynchronous planning task, and formally defines its complexity as an optimization problem as well as our technique. Section 3 describes how we generate the benchmark. Section 4 lays out the experimental setting and overall results. The results are then analyzed in more detail in Section 5. We review relevant related works and conclude the article with our main contributions in Sections 6 to 7.

2. Preliminaries: Naturalistic Asynchronous Planning

We define our task as follows: assuming infinite resources (e.g., as many agents and tools as needed to achieve optimal parallelism are available) for a naturalistic task with a set of compulsory steps, the time needed for each step, and step ordering constraints, we assess whether LLMs can compute the optimal time needed for the task. Formally, we can cast this as the problem to find the longest path on a Directed Acyclic Graph (DAG, Figure 4). A key advantage of doing so is that we can easily estimate the complexity of our task despite the fact that it is a natural language processing problem. This distinguishes our work from many other studies. We empirically prove that our complexity measure predicts LLM behavior in all prompt settings (Figure 3 and Section 5.1), with variance explained in Section 5.4.

Since our task is essentially similar to DAG search, it sheds light on the limits of LLMs as *digital* devices (La Malfa et al., 2024) and, specifically, as solvers of discrete optimization problems on graphs (Wang et al., 2023). It also serves as (i) an example of an optimal succinct routine that an LLM might be able to implement internally (Weiss et al., 2021) to solve a planning problem and (ii) a baseline to measure the loss induced by specifying a problem in natural language, namely an LLM's *language divide*.

2.1. Complexity of Naturalistic Planning

We briefly introduce the complexity measure for our task in this subsection, which we will later show correlates with LLM behavior. With infinite resources, the formalism of a DAG captures the complexity of finding the optimal execution order of compulsory actions a in a plan P to minimize the time cost $TC(P)$. A DAG $G(P)$ representing P can be defined as $G(P) = \langle V, E, w\rangle$, where V is a set of nodes v, each representing an action a in the planning problem, including auxiliary START (v_{src}) and END (v_{dst}). E is a directed set of flow relations e representing ordering constraints, while w is a function that assigns a weight to all edges in the graph $w : E \rightarrow \mathbb{R}^+$. Each flow relation $e_{i,j}$ is associated with a positive number $w(e_{i,j})$ to express that node/action v_i is connected to node/action v_j and requires $w(e_{i,j})$ time to be completed. The edges also represent causal links in that the precondition for an action/node a is met if and only if all actions/nodes linked to and preceding a are performed. For simplicity, we denote $G(P)$ as G in the remaining part of the paper.

In this setting, finding the time cost for an optimal plan P^* in a planning problem is equivalent to finding the longest path G^* on G and can be cast as the following optimization problem on a subgraph $G' = \langle V', E', w\rangle$, $G' \subseteq G$. Exhaustively searching a graph and comparing every path's length can deterministically find the gold answer. On series-parallel graphs (Eppstein, 1992), which are sufficient to describe planning tasks with infinite resources, the average time complexity is $\mathcal{O}(|V| + |E|)$ (Takamizawa et al., 1982), i.e., it is linear with respect to the number of nodes and edges in G.[2] We define our task complexity accordingly.

2.2. Method: Plan Like a Graph

In our work, we propose a novel prompting technique **P**lan **L**ike **a** **G**raph (PLaG, Figure 2). Taking inspiration from Discourse Representation Theory (Wolf et al., 2004) and relevant works on graphical prompt representations (Fatemi et al., 2023; Wang et al., 2023), PLaG includes a graph representation in the prompt, where we give models k-shot illustrations with graphs describing the task and instruct them to either reason based on a given graph (i.e., explicit graph) or to generate a graph themselves and then reason about it (i.e., **B**uild **a** **G**raph/BaG; Wang et al., 2023). We instruct models to produce graph representations of the naturalistic question and then use the information to solve relevant tasks.

[2]While we assume infinite resources to complete a planning task, the natural extension to the case of finite resources (i.e., not all independent actions can be parallelized) is better captured by the formalism of a Petri net (or discrete-time Markov chains with constraints). We introduce the current formalism and Petri net in more detail in Appendix A.1.

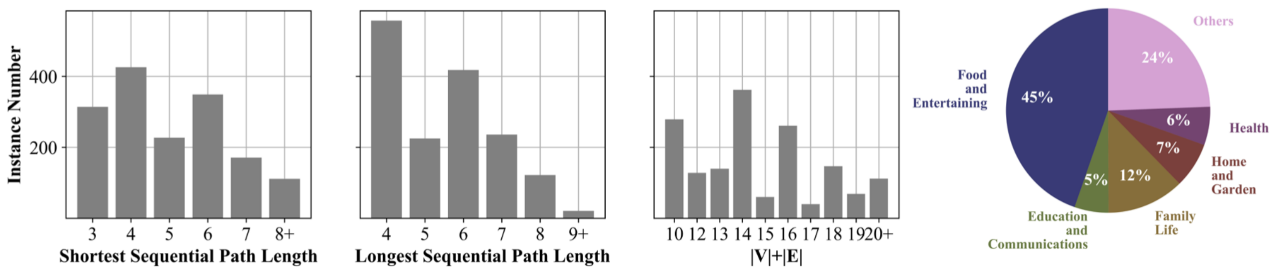

Figure 5. Overview of the AsyncHow benchmark. The three bar charts on the left display the instance numbers for the shortest/longest sequential path length and $|V| + |E|$ in different plans. The pie chart on the right shows the topic distribution in our dataset. See Appendix A.3 for details about the topic assignment.

3. The AsyncHow Benchmark for Planning

Since there is no existing dataset appropriate for our defined task, we generate a new naturalistic asynchronous planning benchmark called **Async**hronous Wiki**How** (AsyncHow). This section describes and validates an *automatic* method for generating this benchmark. With LLMs that consume new benchmarks at an unprecedented pace, our contribution goes beyond AsyncHow and can be used by practitioners to synthesize new datasets.

In addition to the existing data in ProScript (Sakaguchi et al., 2021), an end-to-end human-annotated partial-order plan dataset[3], we use WikiHow (Koupaee & Wang, 2018; Zhang et al., 2020) to collect the planning tasks we need. In line with recent works, we use LLMs as data annotators (Gilardi et al., 2023; Huang et al., 2023). Specifically, we use the GPT models for part of pre-processing, time annotation, and step dependency annotation, as they exhibit impressive annotation capabilities (He et al., 2023). However, we would like to stress that (i) any LLM (or equivalent algorithmic procedure) can be used as an annotator and (ii) the LLM used to annotate is not involved in the ground truth answer generation, where we use deterministic procedures such as the longest path on a DAG. This means that the GPT models used for annotation should not be considered oracles in the benchmarking experiments.

This process culminates in **AsyncHow**, a curated list of 1.6K data points for asynchronous planning. We provide an overview of the benchmark structure in Figure 5. We evaluate the dependency annotation quality automatically and the general generation quality with human annotators (Section 3.1). We do not verify the time annotation because the task time estimation in a less grounded setting such as ours tends not to have a unique gold answer (e.g., `‘finding a gym’` may take five minutes or a week to different people), and GPT-3.5 (the model we use for time annotation) is reported to be a reliable annotator for this task (Jain et al., 2023). Furthermore, our interest is in assessing whether an LLM outputs the optimal plan for a task, and we expect end users to supply different time durations when querying a model.

We now briefly describe the data generation process, with more details in Appendix A.2.

First, we **preprocess** the dataset to collect high-quality plans rated by WikiHow users. Then, given our task definition (e.g., all steps need to be executed, etc.), we filter out plans with optional steps and others that do not fit into our research goal by both matching keywords and prompting GPT-3.5 to answer relevant questions (e.g., `Are all steps needed in this plan?`). Then, we use GPT-3.5 to **estimate the time duration** per step and exclude instances whose step durations cannot be quantified numerically.

Next, we use GPT-4 to **annotate step dependencies** with the dot language. After removing redundant dependencies (e.g., in an answer saying `‘step 1` $\rightarrow$ `step 2’`, `‘step 2` $\rightarrow$ `step 3’`, `‘step 1` $\rightarrow$ `step 3’`, we remove `‘step 1` $\rightarrow$ `step 3’`), we keep data points that have at least four consistent answers that form asynchronous plans and discard the others.

After the above steps, we combine all asynchronous instances with complete time annotation for all meaningful steps in ProScript with our generated asynchronous instances from WikiHow, after which we obtain a collection of 1.6K instances. We then **generate natural language prompts** based on the task information in dot language, as users tend to use natural language descriptions to specify such a task. We have 10 trivially different plausible templates with their succinct use cases in our dataset (e.g. `‘step 1 -> step 2, step 1 -> step 3’` may be expressed as `‘Step 1 must precede step 2, step 1 must precede step 3’`, and succinctly as `‘Step 1 must precede step 2 and 3’`) to allow for relevant paraphrase robustness studies (Elazar et al., 2021).

Last, we **generate equivalent DAGs** representing the work-

[3] ProScript is similar to our dataset, but it is not suitable enough for our task. See Section 6 for discussions.

Table 1. Comparison of our step dependency annotation for the ProScript dev and test set, with mean and standard deviation performance on three randomized experiments (100 instances per experiment).

	dev (in-domain)			test (cross-domain)		
	F1	P	R	F1	P	R
Humans	89.32	89.60	89.21	89.28	89.91	88.86
GPT-4	$89.80_{\pm 1.70}$	$90.65_{\pm 1.36}$	$89.30_{\pm 2.09}$	$85.59_{\pm 2.92}$	$85.95_{\pm 2.43}$	$85.77_{\pm 3.56}$

flow and **compute the optimal time duration** for a plan by calculating the time duration for the longest one. Each planning task is eventually accompanied by four types of graph representations: the adjacency and the edge list, the adjacency matrix, and the compressed sparse row (csr), which can be used to aid LLMs in structural reasoning and assess LLMs' robustness against different graph representations. We do not further vary natural language representations for graphs because relevant investigations can be found in Fatemi et al. (2023).

3.1. Quality Check

On top of the intermediate quality check stages in the above process (e.g., filtering out inconsistent answers and low-scored scripts, etc.), we finally perform two other rounds of quality checks to further ensure high data quality.

First, we assess step dependency annotation quantitatively: in three randomized experiments, we sample 100 instances from the ProScript dev and test sets. Following Sakaguchi et al. (2021), we compare the pair-wise precision, recall, and F1 score of our generated dependency annotations with human performance, which is obtained via asking crowdworkers to annotate partially-ordered scripts for randomly shuffled steps.[4] Our annotation method has near human-level performance, as reported in Table 1.

In addition, we randomly sample 80 instances with a mixture of LLM and human-annotated data and qualitatively survey experts without informing them which data points are human-annotated. We follow the 'prescriptive' approach in Röttger et al. (2021) by instructing them to consider the acceptability of the task time estimations and step ordering constraints. Human-annotated and LLM-generated instances receive similar levels of acceptability.

4. Benchmarking Experiment

We are interested in answering the following questions. First, can a model efficiently solve asynchronous planning tasks with existing prompting techniques such as k-shot (Brown et al., 2020) and Chain of Thought prompting (CoT; Wei et al., 2022)? Second, can we develop a better method to prompt models to improve their performance? Third, how do scale effects manifest when varying problem complexity and model size? Last, is an LLM's performance robust to trivially different linguistic or graphical prompts? We design the experiments accordingly.

4.1. Experimental Setting and Design

We conduct experiments with GPT-3.5, GPT-4, and Command, three closed-source LLMs, as well as LLaMA-2-70B-chat and Mistral-7B-Instruct (v0.2), two open-source LLMs.[5] We first experiment with different language descriptions of our problem in a zero-shot setting with 100 sampled prompts (see details in Appendix A.7) and use the best-performing one for the successive experiments.

We then benchmark our models in full scale in four prompting regimes: (i) **zero-shot**: only prompting models with task descriptions without additional information or training; (ii) ***k*-shot** (Brown et al., 2020): prompting with k in-context instances with desired outputs preceding the task description; (iii) **zero-shot with CoT** (zero-shot+CoT; Kojima et al., 2022): prompting the model with the task description along with the instruction `'Let's think step by step'`, and (iv) ***k*-shot with CoT** (k-shot+CoT; Wei et al., 2022): prompting the model with k in-context instances with CoT illustrations for the problem-solving process and desired outputs preceding the task description with CoT.[6]

Then, we experiment by sampling 100 instances for the adjacency list, edge list, adjacency matrix, and csr in the

[4] Precision, recall, and F1 score are defined as follows: Precision $= \frac{|E \cap \hat{E}|}{|E|}$, Recall $= \frac{|E \cap \hat{E}|}{|\hat{E}|}$, $\mathrm{F}_1 = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}}$, with E being the gold edges and $\hat{E}$ denoting the predictions in each discourse graph, respectively.

[5] In a preliminary experiment, we also tested CodeLlama-34B (Roziere et al., 2023) and Phi-2 (Gunasekar et al., 2023) , but we exclude them from the evaluation due to poor performance. Our observation that the code models perform poorly is in line with La Malfa et al. (2024) and Liu et al. (2024), showing that simulation is more difficult than generation.

[6] We use $k = 3$. We do not conduct full-scale benchmarking on other prompting techniques such as Chain-of-Thought Self-Consistency (CoT-SC; Wang et al., 2022) and Tree of Thought (ToT; Yao et al., 2024) as they primarily use standard IO prompts like CoT, and our method can be deployed in addition to these prompting techniques. We show that our method is superior to using CoT-SC and ToT alone for a more comprehensive comparison in Appendix A.10. See also the latency analysis for cost-performance trade-offs in Appendix A.11.

Table 2. Model accuracy in different settings on the AsyncHow benchmark. Model performances without our method are in plain background, while those with our method are in blue background. We mark the best performance per model in **bold**. Following Dror et al. (2018), we use McNemar's tests (McNemar, 1947) to obtain p-values and Holm-Bonferroni method (Holm, 1979) to correct them for each evaluation to test the statistical significance of performance difference between experiment with and without our proposed method. We denote with † when the performances with PLaG are significantly better ($p < 0.05$) than the best result without.

	Without PLaG				With PLaG	
Model	zero-shot	zero-shot + CoT	k-shot	k-shot + CoT	PLaG (explicit graph)	PLaG (BaG)
GPT-4	0.130	0.129	0.107	0.657	0.730†	**0.777**†
GPT-3.5	0.199	0.224	0.248	0.226	0.290†	**0.355**†
Command	0.078	0.015	0.050	0.078	**0.100**	0.050
LLaMA-2-70B-chat	0.039	0.038	0.053	0.076	**0.101**†	0.069
Mistral-7B-Instruct	0.078	0.070	0.098	0.149	**0.161**	0.146

setting of PLaG (explicit graph). We use the best type for full-scale PLaG experiments (explicit graph/BaG). Prompt examples are given in Appendix A.8.

4.2. Experiment Results

We evaluate each model's performance by the accuracy of correctly reporting the shortest time needed for different plans. Main results are in Table 2.[7]

The strongest performance is obtained by GPT-4 with PLaG (BaG). This is surprising given that GPT-3.5 does better than GPT-4 (though not very well) on the zero-shot, zero-shot+CoT, and k-shot settings, which lack explicit illustrations. A solid performance gap divides open-source models from GPT models, although Mistral-7B-Instruct performs better despite being much smaller than LLaMA-2-70B-chat.

PLaG (our method) successfully boosts the performance of all models. This is particularly interesting considering that while natural language prompts in the vanilla setting essentially represent the same information as the graphs, explicitly providing prototypical graph-structured data enhances LLM performance and highlights its inherent limitation of reasoning at a conceptual level. Furthermore, many other real-world tasks in natural language such as dialogue state tracking (Lin et al., 2021) can be abstracted as graphs. Thus, our finding is relevant for future research on enhancing conceptual representations in LLMs.

Surprisingly, PLaG with BaG, which does not require external processing to supply new graphs explicitly in every task description, improves the performance of the most capable models (GPT-3.5 and GPT-4) across all complexity levels **off the shelf**.[8] These results suggest that PLaG benefits LLMs by adding graph information and indicates the potential of boosting capable models' performance in planning without external processors.

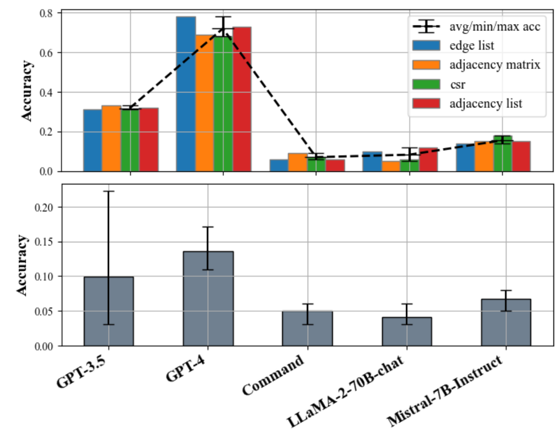

Figure 6. The upper plot refers to average model accuracy with 100 instances of PLaG (explicit graph) in different graphs. Colored bars refer to model performance with different graph types. Black dashed lines refer to average accuracy with different graph types. The lower plot refers to the average zero-shot accuracy in 100 instances of different text prompts (without economic usage). Error bars in both plots refer to worst/best performance per model.

Next, we report our experiment results on different text prompts and graph types. As shown in Figure 6, different text prompts and graph types induce variations in model performance, and models have different preferences for these variables. In general, using more succinct and more natural expressions (see Appendix A.2.4 for details) tends to downgrade model performance (results are reported in Appendix A.13), a hint that models cannot adapt to slight variations of the same prompts.

[7] If a closed-source model does not return anything due to content filtering, we consider the answer to be false. In Appendix A.9, we provide analyses where we exclude such invalid instances.

[8] We emphasize that the superior performance of BaG does not result from noise in sampling (see Appendix A.12 for discussion). We hypothesize that the superior performance of the explicit graph in other models results from their incapability of generating accurate graph representations.

5. Further Analysis of GPT-3.5/4 Results

This section investigates which factors influence the most potent models in our task, namely GPT-3.5 and GPT-4. We first relate an LLM's accuracy with task complexities, then provide an ablation study to identify the salient characteristics that make a planning problem inherently complex. Next, we use a synthetic dataset that covers and goes beyond the distribution of AsyncHow to estimate model performance in the potential scenarios that might fall out of the distribution of our benchmark. Last, we perform a qualitative analysis to provide further rationales for some surprising phenomena observed in our results.

5.1. Accuracy vs. Complexity

In Figure 3 (Section 1), we plot the accuracy of GPT-3.5 and GPT-4 as a function of the task complexity $|V| + |E|$.[9] Generally, the accuracy negatively correlates with the complexity of the task for all models and all settings, with graphs of complexity $|V| + |E| \geq 18$ already resulting in challenging problems for the most capable model and best setting. The complexity measure also predicts the model accuracy trends well in settings without graphs. We notice a little jump at complexity $|V| + |E| \geq 20$, for which we will discuss possible reasons in Section 5.4.

Without our method, GPT-4 with k-shot + CoT consistently outperforms any other settings by a solid margin, while all the other settings have comparable performances independently from the number of illustrations provided or the model employed (Figure 3, upper).

Our method (PLaG) consistently improves over k-shot+CoT, the best method without PLaG, among tasks of all complexities (Figure 3, lower) for both GPT-3.5 and GPT-4. In line with what was observed before, the accuracy drops significantly with complex planning tasks, once again proving that LLMs are not yet robust enough to be deployed as generally intelligent agents in planning.

5.2. Ablation Study

Solving our planning task requires a certain degree of compositionality in combining *time comparison*, *time summation*, and *constraint reasoning* correctly. We perform an ablation study to identify which skills LLMs lack. We sample 200 sequential planning tasks (i.e., the optimal time calculation only requires summation) and fully parallel tasks (i.e., the optimal time calculation only requires comparison) from the non-asynchronous part in our generated dataset. We sample equal numbers of plans per step with a minimum of three steps and a maximum of seven, as smaller plans are trivial while higher ones are sparse in AsyncHow, which can potentially lead to sampling bias. We experiment with k-shot + CoT and compare model performance across step numbers in different plan types in Figure 7.

[9] We show in Appendix A.14 that $|V|$ and $|E|$ equally contribute to the complexity of a planning task, with no clear dominance of one over the other.

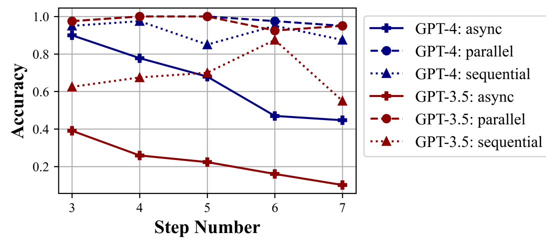

Figure 7. Comparison of parallel/sequential plan execution accuracy with asynchronous plans. All experiments are done in the setting of k-shot + CoT. Blue and red lines refer to GPT-4 and GPT-3.5 results, respectively.

While GPT-3.5 and GPT-4 have a similar accuracy in parallel tasks requiring time comparison, GPT-4 outperforms GPT-3.5 on sequential tasks, i.e., at time summation. Our results suggest a performance gap between parallel/sequential and asynchronous plans for both models. We conclude that reasoning about task constraints adds special difficulty on top of time comparison and summation for LLMs.

5.3. Out-of-distribution Probing

We estimate LLMs' performance on out-of-distribution data points whose complexities fall out of the data-rich part (i.e., complexity $|V| + |E| < 20$) of AsyncHow. As our naturalistic planning problem can be cast into longest-path graph search (refer to formalism in Section 2), we generate a synthetic dataset of 2,000 data points evenly distributed between complexity 10 to 40 for *prototypical* shortest-path graph search (we can cast the longest-path search to shortest-path search by negating edge weights). By *prototypical*, we refer to formulating prompts for dynamic programming problems where an LLM is queried to compute the longest path on a graph with numerical edge weights to simulate time durations for each node the edge starts from. For consistency, we sample the graphs as similar to AsyncHow data as possible (see details in Appendix, Section A.14). We prompt GPT-3.5 and GPT-4 in zero-shot + CoT with their respective best graph representations found in Section 4.2. We compare its accuracy with the accuracy in zero-shot + CoT in the naturalistic experiment.

As shown in Figure 8, graph search accuracy shows a downgoing trend similar to the in-domain naturalistic data, which indicates that model performance in naturalistic planning tasks is likely to follow the pattern of synthetic data and continue to drop with complexity further increasing. This outcome strengthens other findings that LLMs can be unre-

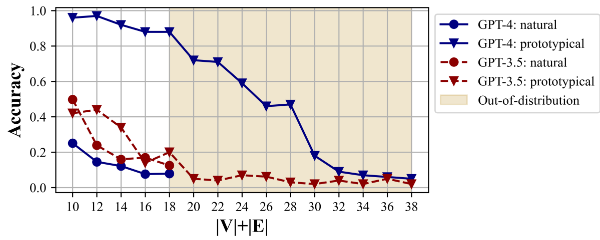

Figure 8. GPT-3.5 and GPT-4 performance on *prototypical* longest path search problem and on *natural* AsyncHow task with zero-shot + CoT. Models' respective best graph types found in Section 4.2 are used in prototypical probing.

liable routine simulators (La Malfa et al., 2024).

Interestingly, although solving essentially the same task, the performance of GPT-4 is much higher in the prototypical setting than the naturalistic one. In comparison, GPT-3.5 derives little benefit from the prototypical setting. We impute this gap to several concurrent factors. First, computing the optimal plan with durations expressed as numbers is easier than naturalistic time conversion (we will discuss this in the next subsection). Second, naturalistic planning requires turning language into an effective procedure, which adds to the difficulty of processing *prototypical* graphs. The results also shed light on the reasons behind the boost of performance caused by PLaG: PLaG points a model to a setting it is already familiar with and better masters.

5.4. Qualitative Study

We qualitatively overview some failures and successes of LLMs, which shed light on edge cases that are of interest to understanding their capabilities and limitations.

Wrong answers in easy problems. Even for low-complexity planning instances, GPT-4 may incur trivial errors. It emerges that errors tend to fall into a few macro-categories: (i) **parallelism error** where LLMs cannot efficiently parallelize as many steps as possible: e.g., when step 3 (10 min) can be done together with step 1 (5 min) and 2 (15 min), the model only parallelizes 1 and 3 but schedules step 2 to follow them; (ii) **time unit conversion error** where LLMs cannot efficiently convert time units to common measures for calculation: e.g., 3 weeks and 1 hour is wrongly converted to be 5,041 hours (30 weeks and 1 hour) in the final answer. Our findings are in line with Dziri et al. (2023) and La Malfa et al. (2024) in that LLMs tend to prefer linear pattern matching and are prone to mistakes in time carries (Wang & Zhao, 2023).

Correct answers in hard problems. For graphs of complexity $|V|+|E| \geq 20$, GPT-3.5 and GPT-4 perform slightly better than that at $|V|+|E|=18$, which have lower-class complexity. We impute this phenomenon to (i) the sparsity of graphs for higher complexities and (ii) an implicit bias of our benchmark towards easier data conversions for more complex planning tasks. See a more in-depth discussion of this phenomenon in Appendix A.15.

Why is BaG better than explicit graph? A symbolic processor deterministically generates correct graph representations, while BaG prompts a model to generate its internal representation of the problem with no promise of complete correctness. However, among PLaG methods, BaG performs slightly better than explicit graphs (i.e., generated algorithmically). We sample some instances and find that BaG-generated graphs are of the same format as provided in the *k*-shot prompt, and we thus impute the performance gap to a sub-optimal positioning of the graph in the former setting (Liu et al., 2023b; Mao et al., 2023): the explicit graph prompt setting expresses the prompt in the form `'[Task description with graph] Answer:'`, with the graph appearing in the middle of the context, which can be easily ignored by the model. For BaG, by contrast, the prompt has the form `'[Task description] Answer: [Graph]'`, where the graph is generated at a successive step and easier for models to take into account.

6. Related Work

LLMs for planning. Works focusing on automatically generating plausible plans for daily tasks show that LLMs can be used to develop reasonable and ordered actions or goals (Madaan et al., 2022; Xie et al., 2023; Yuan et al., 2023). The work most similar to ours is Sakaguchi et al. (2021): they collected 6.4K ordered plans via crowdsourcing for ProScript. However, the dataset is insufficient to serve as a benchmark for asynchronous planning as relevant data points are sparse and lack diversity.

Another line of work focuses on finding the optimal plan for domain-oriented tasks such as robotics. Although LLMs can be readily deployed to parse natural language into logical elements, they alone cannot develop optimal plans to accomplish a given goal without external symbolic processors (Collins et al., 2022; Valmeekam et al., 2022; Lawless et al., 2023; Lin et al., 2023; Liu et al., 2023a; Yang et al., 2023). While these works focus on domain-oriented tasks using external symbolic processors, we close the gap between structured and naturalistic tasks and show the potential of solely using LLMs for these tasks.

LLMs for graph reasoning. Two complementary lines of work inform LLMs with graphs and can be categorized into *implicit* and *explicit* methods. Implicit methods help decompose task goals into atomic steps (Huang et al., 2022a; Valmeekam et al., 2022; Sakib & Sun, 2024) and help explain complex reasoning processes (Madaan et al., 2021; Saha et al., 2021; Besta et al., 2023; Dziri et al., 2023). Works on explicit graphs incorporate external knowledge and reason about more complex problems such as multi-hop question answering (Chen et al., 2023b; Park et al., 2023; Ye et al., 2023). Our work shows that instructing LLMs to consider problems like graphs can improve their performance in planning. It also complements recent discoveries suggesting that LLMs' performances negatively correlate with the complexity of graph problems (Fatemi et al., 2023; Guo et al., 2023; Wang et al., 2023), showing that the conclusion also holds in relevant naturalistic tasks.

Discourse Representation Theory. Humans produce and understand language in a structured way. For instance, when writing a paragraph, people can have a main point and then elaborate on the supporting elements of the discourse (Flower et al., 1992; Limpo & Alves, 2018). Graphs offer a structured representation of the discourse, with elements as nodes and relations as edges representing elaboration and parallel or temporally/causally linked actions (Wolf et al., 2004; Presutti et al., 2012; Ma et al., 2022). Recent works suggest that LLMs do not possess identical linguistic representations as humans since they do not compositionally process language and perform tasks in a human-like way (Bertolini et al., 2022; Press et al., 2022; Chen et al., 2023a; Dziri et al., 2023). We find in our work that enriching natural language prompts in a structured manner helps LLMs in relevant tasks.

7. Conclusion

In this paper, we automatically generate a benchmark, AsyncHow, and assess LLMs for their performance in asynchronous plan reasoning. We find that if not provided with a detailed illustration of the task solution process, all models behave extremely poorly in our task. We propose a formalism to classify naturalistic asynchronous planning tasks, which successfully predicts LLMs' performance patterns. We propose PLaG, a method that consistently boosts SOTA model performance across all task complexity levels off the shelf. Despite this, we find that model performance still drastically downgrades with increasing task complexity, which calls into question using them as digital devices or generally intelligent agents.

Limitations and Future Work

Some limitations of this work are as follows. We assume that infinite resources are available in our benchmarking, while only finite resources may be available for tasks in real life. Second, we only consider time cost in plan optimization, while realistically, other restrictions, such as preferences, should be considered. Regarding future work, it will be interesting to further elaborate on our benchmark with the proposed techniques and our dataset to add more elements such as resource constraints, multimodality, multilingualism, or other robotics/reinforcement learning features. Practitioners can also scale up the complexity of the benchmark to more complicated tasks. Another promising avenue of research is to compare the performance patterns of LLMs to those of humans (i.e. are LLMs likely to make the same mistakes as humans in asynchronous plan reasoning).

Data Access Statement

The dataset used in this paper can be found in `https://github.com/fangru-lin/graph-llm-asynchow-plan`.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work. It not only unveils the limitations of SOTA LLMs but can also potentially influence many downstream tasks such as job scheduling with the wide application of such technologies. Ethically, since we generate part of our benchmarking dataset from WikiHow, a web data source, users may find relevant content unsafe or uncomfortable. We try our best in the data generation process to leverage the metadata in GPT model outputs to filter out all instances in which either the prompt itself or the reply is flagged as not completely safe concerning hate, self-harm, violence, or sexual aspects. Furthermore, unlike social media for general usage, WikiHow is designed for gaining tips for real-life tasks, which makes it less likely to contain harmful content.

Acknowledgement

We thank all the bodies who have provided funding for the authors and for the associated project. FL is supported by Clarendon and Jason Hu studentship. ELM is supported by the Alan Turing Institute. AGC is supported by the Economic and Social Research Council (ESRC) under grant ES/W003473/1, by the Fundamental Research priority area of The Alan Turing Institute, and by the Turing Defence and Security programme through a partnership with the UK government in accordance with the framework agreement between GCHQ and the Alan Turing Institute. JBP is supported by the Engineering and Physical Sciences Research Council (EP/T023333/1). The support of Microsoft under their Accelerating Foundation Models Research initiative in providing Azure credits to AGC is gratefully acknowledged. This work was also supported by compute credits from a Cohere For AI Research Grant to FL. These grants are designed to support academic partners conducting research with the goal of releasing scientific artifacts and data for good projects. Last, we are grateful to the people who offered invaluable feedback and suggestions along the way, and in particular to all reviewers of this paper.

References

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 preprint arXiv:2204.01691*, 2022.

Bertolini, L., Weeds, J., and Weir, D. Testing Large Language Models on compositionality and inference with phrase-level adjective-noun entailment. In Calzolari, N., Huang, C.-R., Kim, H., Pustejovsky, J., Wanner, L., Choi, K.-S., Ryu, P.-M., Chen, H.-H., Donatelli, L., Ji, H., Kurohashi, S., Paggio, P., Xue, N., Kim, S., Hahm, Y., He, Z., Lee, T. K., Santus, E., Bond, F., and Na, S.-H. (eds.), *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 4084–4100, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.359.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Gianinazzi, L., Gajda, J., Lehmann, T., Podstawski, M., Niewiadomski, H., Nyczyk, P., et al. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

Chen, J., Pan, X., Yu, D., Song, K., Wang, X., Yu, D., and Chen, J. Skills-in-context prompting: Unlocking compositionality in large language models. *arXiv preprint arXiv:2308.00304*, 2023a.

Chen, Z., Mao, H., Li, H., Jin, W., Wen, H., Wei, X., Wang, S., Yin, D., Fan, W., Liu, H., et al. Exploring the potential of large language models (LLMs) in learning on graphs. *arXiv preprint arXiv:2307.03393*, 2023b.

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

Colom, R., Karama, S., Jung, R. E., and Haier, R. J. Human intelligence and brain networks. *Dialogues in clinical neuroscience*, 12(4):489–501, 2010.

Dagan, G., Keller, F., and Lascarides, A. Dynamic planning with a LLM. *arXiv preprint arXiv:2308.06391*, 2023.

Dror, R., Baumer, G., Shlomov, S., and Reichart, R. The hitchhiker's guide to testing statistical significance in natural language processing. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1383–1392, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1128. URL https://aclanthology.org/P18-1128.

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jian, L., Lin, B. Y., West, P., Bhagavatula, C., Bras, R. L., Hwang, J. D., et al. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*, 2023.

Elazar, Y., Kassner, N., Ravfogel, S., Ravichander, A., Hovy, E., Schütze, H., and Goldberg, Y. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9: 1012–1031, 2021.

Eppstein, D. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992.

Fatemi, B., Halcrow, J., and Perozzi, B. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*, 2023.

Fikes, R. E. and Nilsson, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Flower, L., Schriver, K. A., Carey, L., Haas, C., and Hayes, J. R. Planning in writing: The cognition of a constructive process. *A rhetoric of doing: Essays on written discourse in honor of James L. Kinneavy*, pp. 181–243, 1992.

Gilardi, F., Alizadeh, M., and Kubli, M. ChatGPT outperforms crowd-workers for text-annotation tasks. *arXiv preprint arXiv:2303.15056*, 2023.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pp. 287–326. Elsevier, 1979.

Grice, H. P. Logic and conversation. In *Speech acts*, pp. 41–58. Brill, 1975.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Guo, J., Du, L., and Liu, H. GPT4Graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*, 2023.

He, X., Lin, Z., Gong, Y., Jin, A., Zhang, H., Lin, C., Jiao, J., Yiu, S. M., Duan, N., Chen, W., et al. AnnoLLM: Making large language models to be better crowdsourced annotators. *arXiv preprint arXiv:2303.16854*, 2023.

Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

Huang, F., Kwak, H., and An, J. Is ChatGPT better than human annotators? Potential and limitations of ChatGPT in explaining implicit hate speech. *arXiv preprint arXiv:2302.07736*, 2023.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022a.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 17–23 Jul 2022b. URL https://proceedings.mlr.press/v162/huang22a.html.

Jain, A. S. and Meeran, S. Deterministic job-shop scheduling: Past, present and future. *European journal of operational research*, 113(2):390–434, 1999.

Jain, R., Sojitra, D., Acharya, A., Saha, S., Jatowt, A., and Dandapat, S. Do language models have a common sense regarding time? Revisiting temporal commonsense reasoning in the era of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6750–6774, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Koupaee, M. and Wang, W. Y. Wikihow: A large scale text summarization dataset. *arXiv preprint arXiv:1810.09305*, 2018.

La Malfa, E., Weinhuber, C., Torre, O., Lin, F., Cohn, A. G., Shadbolt, N., and Wooldridge, M. Code simulation challenges for large language models. *arXiv preprint arXiv:2401.09074*, 2024. URL https://arxiv.org/pdf/2401.09074.

Lawless, C., Schoeffer, J., Le, L., Rowan, K., Sen, S., Hill, C. S., Suh, J., and Sarrafzadeh, B. "i want it that way": Enabling interactive decision support using large language models and constraint programming. *arXiv preprint arXiv:2312.06908*, 2023.

Limpo, T. and Alves, R. A. Effects of planning strategies on writing dynamics and final texts. *Acta psychologica*, 188:97–109, 2018.

Lin, K., Agia, C., Migimatsu, T., Pavone, M., and Bohg, J. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

Lin, W., Tseng, B.-H., and Byrne, B. Knowledge-aware graph-enhanced GPT-2 for dialogue state tracking. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7871–7881, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.620. URL https://aclanthology.org/2021.emnlp-main.620.

Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., Biswas, J., and Stone, P. LLM+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

Liu, C., Zhang, S. D., and Jabbarvand, R. Codemind: A framework to challenge large language models for code reasoning. *arXiv preprint arXiv:2402.09664*, 2024.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023b.

Ma, Y., Zhu, J., and Liu, J. Enhanced semantic representation learning for implicit discourse relation classification. *Applied Intelligence*, pp. 1–13, 2022.

Madaan, A., Rajagopal, D., Tandon, N., Yang, Y., and Hovy, E. Could you give me a hint? Generating inference graphs for defeasible reasoning. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 5138–5147, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.456. URL https://aclanthology.org/2021.findings-acl.456.

Madaan, A., Zhou, S., Alon, U., Yang, Y., and Neubig, G. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*, 2022.

Mao, J., Middleton, S. E., and Niranjan, M. Prompt position really matters in few-shot and zero-shot NLU tasks. *arXiv preprint arXiv:2305.14493*, 2023.

McDermott, D. M. The 1998 AI planning systems competition. *AI magazine*, 21(2):35–35, 2000.

McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. In Calzolari, N., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Hasida, K., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., Piperidis, S., and Tokunaga, T. (eds.), *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://aclanthology.org/L18-1008.

Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. 2023.

OpenAI. GPT-4 technical report. *ArXiv preprint*, abs/2303.08774, 2023. URL https://arxiv.org/abs/2303.08774.

Park, J., Patel, A., Khan, O. Z., Kim, H. J., and Kim, J.-K. Graph-guided reasoning for multi-hop question answering in large language models. *arXiv preprint arXiv:2311.09762*, 2023.

Petri, C. A. Kommunikation mit automaten. 1962.

Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N. A., and Lewis, M. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

Presutti, V., Draicchio, F., and Gangemi, A. Knowledge extraction based on discourse representation theory and linguistic frames. In *Knowledge Engineering and Knowledge Management: 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings 18*, pp. 114–129. Springer, 2012.

Röttger, P., Vidgen, B., Hovy, D., and Pierrehumbert, J. B. Two contrasting data annotation paradigms for subjective nlp tasks. *arXiv preprint arXiv:2112.07475*, 2021.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Russell, S. J. S. J. and Norvig, P. *Artificial intelligence : a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, N.J, 1995. ISBN 0131038052.

Saha, S., Yadav, P., Bauer, L., and Bansal, M. ExplaGraphs: An explanation graph generation task for structured commonsense reasoning. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7716–7740, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.609. URL https://aclanthology.org/2021.emnlp-main.609.

Sakaguchi, K., Bhagavatula, C., Le Bras, R., Tandon, N., Clark, P., and Choi, Y. proScript: Partially ordered scripts generation. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 2138–2149, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.184. URL https://aclanthology.org/2021.findings-emnlp.184.

Sakib, M. S. and Sun, Y. Consolidating trees of robotic plans generated using large language models to improve reliability. *arXiv preprint arXiv:2401.07868*, 2024.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Silver, T., Hariprasad, V., Shuttleworth, R. S., Kumar, N., Lozano-Pérez, T., and Kaelbling, L. P. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., and 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, pp. 2998–3009, 2023.

Sternberg, R. J. Toward a triarchic theory of human intelligence. *Behavioral and Brain Sciences*, 7(2):269–287, 1984.

Takamizawa, K., Nishizeki, T., and Saito, N. Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the ACM (JACM)*, 29(3):623–641, 1982.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. 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*, 2022.

Vicentini, A. The economy principle in language. *Notes and Observations from early modern English grammars. Mots, Palabras, Words*, 3:37–57, 2003.

Wang, H., Feng, S., He, T., Tan, Z., Han, X., and Tsvetkov, Y. Can language models solve graph problems in natural language? *arXiv preprint arXiv:2305.10037*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*, 2022.

Wang, Y. and Zhao, Y. Tram: Benchmarking temporal reasoning for large language models. *arXiv preprint arXiv:2310.00835*, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021. URL https://proceedings.mlr.press/v139/weiss21a/weiss21a.pdf.

Wolf, F., Gibson, E., Fisher, A., and Knight, M. Discourse graphbank. *Linguistic Data Consortium, Philadelphia*, 2004.

Xie, Y., Yu, C., Zhu, T., Bai, J., Gong, Z., and Soh, H. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

Yang, Z., Ishay, A., and Lee, J. Coupling large language models with logic programming for robust and general reasoning from text. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 5186–5219, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.321. URL https://aclanthology.org/2023.findings-acl.321.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *Advances in Neural Information Processing Systems*, 36, 2024.

Ye, R., Zhang, C., Wang, R., Xu, S., and Zhang, Y. Natural language is all a graph needs. *arXiv preprint arXiv:2308.07134*, 2023.

Yuan, S., Chen, J., Fu, Z., Ge, X., Shah, S., Jankowski, C., Xiao, Y., and Yang, D. Distilling script knowledge from large language models for constrained language planning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4303–4325, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.236. URL https://aclanthology.org/2023.acl-long.236.

Zhang, L., Lyu, Q., and Callison-Burch, C. Reasoning about goals, steps, and temporal ordering with WikiHow. In Webber, B., Cohn, T., He, Y., and Liu, Y. (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4630–4639, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.374. URL https://aclanthology.org/2020.emnlp-main.374.

A. Appendix

A.1. Extended Preliminaries: Naturalistic Asynchronous Planning

A.1.1. Naive Asynchronous Planning with DAG

Assuming infinite resources (e.g. as many agents and tools as needed to achieve optimal parallelism are available), our task can be considered as finding the optimal time cost TC of a partial-order plan P. A partial-order plan P is classically defined as $P = \langle A, O, C \rangle$, where A is a set of actions a (including start and finish), O is a set of ordering constraints which take the form of $a_i \prec a_j$, meaning a_i has to be performed before a_j, and C as causal links taking the form of $a_i \xrightarrow{p} a_j$, meaning performing a_i meets precondition p needed for a_j (Russell & Norvig, 1995).

Specifically for our task, with infinite resources, the formalism of a DAG captures the complexity of finding the optimal order of execution of actions that can or cannot be parallelized in P to minimize time cost $TC(P)$. A DAG $G(P)$ representing P can be defined as $G(P) = \langle V, E, w \rangle$, where V is a set of nodes, each representing an action a in the planning problem. We serve two auxiliary nodes v_{src} (START) and v_{dst} (END) that connect respectively each initial and final component in G on top of other meaningful nodes but do not impact the optimal solution of the problem (as sketched in Figure 4). E is a directed set of flow relations representing ordering constraints O, while w is a function that assigns weight to all edges in the graph $w : E \to \mathbb{R}^+$. Each flow relation is associated with a positive number (the *weight* of a connection), namely $w(e_{i,j})$, to express that node/action v_i is connected to node/action v_j and requires $w(e_{i,j})$ time to be completed. The edges also represent causal links C in that the precondition p for an action/node a is met if all actions/nodes linked to and preceding a are performed. For simplicity, we denote $G(P)$ as G in the remaining part of the paper.

In this setting, finding the time cost for an optimal plan P^* in a planning problem is equivalent to finding the longest path G^* on G and can be cast as the following optimization problem on a subgraph $G' = \langle V', E', w \rangle,\ G' \subseteq G$:

$$\begin{aligned} P^* \in \underset{P=\langle A,O,C\rangle}{\arg\min}\ & TC(P) \\ \Leftrightarrow G^* \in \underset{G'=\langle V',E',w\rangle \subseteq G}{\arg\max} & \sum_{e_{i,j}\in E'} w(e_{i,j}) \\ \text{s.t. } \forall v'_i \in (V' \setminus v_{dst}), \exists!\ v'_j \in v'_{i.\text{next}} \mid & (v'_j, e'_{i,j}) \in (V', E') \\ & (v_{src}, v_{dst}) \in V'. \end{aligned} \tag{1}$$

In this formulation, $v'_{i.\text{next}}$ are vertices in G that are successors of v'_i.

Consider the example in Figure 4, where we sketch the DAG to solve the planning task of Figure 1. The maximum time to complete the whole task is 65 minutes (i.e. sequentially executing all actions), while parallelizing all actions violates constraints on the preconditions of some actions (e.g. `'Roll dough'` and `'Add filling'` cannot be done simultaneously). Parallelizing `'Roll dough'`, and `'Preheat oven'`, and then executing the other actions allows solving the problem optimally.

A.1.2. Asynchronous Planning with Petri Net

While we assume infinite resources to complete a planning task, the natural extension to the case of finite resources (i.e., not all independent actions can be parallelized) is better captured by the formalism of a Petri net. Consider a task where one needs to make breakfast by grinding coffee (`'Grind-coffee'`, 3 min), boiling coffee (`'Coffee'`, 8 min), making toast (`'Bread'`, 10 min), and frying an egg (`'Egg'`, 7 min). The problem of finding the optimal order of execution of actions that can or cannot be parallelized is fully captured by the formalism of Petri nets (Petri, 1962). A Petri net consists of a tuple $N = (P, T, F)$, where P and T are disjoint finite sets of places and transitions, and F is a directed set of flow relations associated with a positive number (the *weight* of a connection), namely $(i, j, f_{i,j} \geq 0)$. For an initial configuration of N, an action $p \in P$ 'fires', sequentially or simultaneously with other actions, a transaction $t \in T$, if it contains a token, usually represented as a circle that encompasses a single dot.[10] An optimal planning problem is equivalent to finding the shortest transition in a Petri net. Consider the example in Figure 9: while one can execute the actions {`'Grind-coffee'`, `'Bread'`, `'Egg'`} simultaneously or in parallel, with the expected completion time reported on the edge, the action `'Coffee'` must follow `'Grind-coffee'`. On the other hand, one can execute the actions {`'Egg'`, `'Grind-coffe'`} in parallel and start the action `'Coffee'` when `'Grind coffee'` is complete. Similarly to Figure 1, the minimum amount of time required

[10] We assume states can have at most one token and that a transaction is activated if each 'firing' state contains a token.

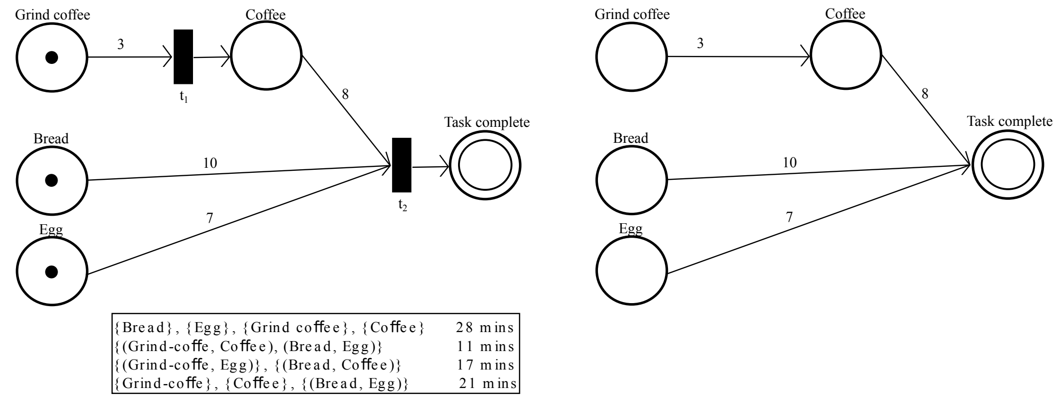

Figure 9. On the left, a Petri net representation of the example of making breakfast. A few admissible *runs* are reported at the bottom with their completion time. Actions executed sequentially (in parallel) are reported in curly (round) brackets. On the right, the DAG used to solve the optimal planning problem.

for the task is 28 minutes, while parallelising $\{$`'Grind-coffee'`, `'Bread'`, `'Egg'`$\}$ and then execute `'Coffee'` when `'Grind-coffee'` is complete allows solving the problem optimally. For a Petri net $N = (P, T, F)$ representing a planning problem, the maximum completion time is the sum of the weights on the longest path (while the minimum is the minimum time to reach the last transition from any parent node).

A.1.3. On the Complexity of a Planning Task with Finite Resources

With infinite resources, i.e., assuming one token per initial node (and thus can 'fire' the transition), finding the optimal plan is equivalent to finding the longest path from the initial to the final state and can thus be computed efficiently. A directed acyclic graph (DAG) is obtainable from a Petri net by discarding the set of transitions and reversing the sign on each edge, then using a search algorithm that is linear in complexity for series-parallel graphs. Formally, a DAG for a planning task has the following formulation: $G = \langle P, W \rangle$, such that $w_{i,j} = -f_{i,j}, (f_{i,j}, w_{i,j}) \in (F, W)$, where P is the set of states and F the set of transitions and weights in the correspondent Petri net. Figure 9 (right) reports the DAG for making breakfast. On the other hand, when resources are finite, finding the optimal planning corresponds to an optimization problem that is generally NP-hard to solve (Graham et al., 1979; Jain & Meeran, 1999) as one has to estimate the number of reachable states (the state space) from a combinatorial number of initial configurations with k resources. While the exact size of the state space depends on the constraints on sequential actions, such a number is combinatorial and upper-bounded by $2^{n(n-1)}$, where n is the number of actions and equivalently the number of states in the correspondent Petri net. Combinatorial optimization algorithms, including genetic algorithms and simulated annealing, are usually applied to search for solutions in NP-hard problems as such since exact methods tend to be too complex.

A.2. Data Generation Details

In addition to existing data in ProScript(Sakaguchi et al., 2021), we use WikiHow (Koupaee & Wang, 2018; Zhang et al., 2020) as a base dataset to derive the planning tasks we need. The data generation strategy is schematized in Figure 10, which consists of five steps: **preprocessing**, **time duration annotation**, **step ordering constraint annotation**, **natural language prompt generation**, and **graph and gold answer generation**. We first leverage GPT-3.5/4 together with keyword-catching algorithms to filter out low-quality examples (Section A.2.1) and estimate the time duration of planning steps (Section A.2.2), and their dependencies for unannotated WikiHow data (Section A.2.3). Then we combine the LLM-annotated WikiHow data (about $1k$) with qualified human-annotated data in ProScript (about $0.6k$), to generate natural language prompts based on pre-defined templates (Section A.2.4) and optimal planning time deterministically by Python, which is by construction correct (Section A.2.5). We perform automatic and human data quality validation and show that our dataset is similar in quality to end-to-end human-annotated data (Section 3.1). This process culminates with AsyncHow, a curated list of 1.6K data points for planning, which enables benchmarking models for asynchronous planning against the gold answer provided by symbolic processors.

We now provide details about the data generation process. WikiHow dataset consists of different script types, namely (i) flat scripts, with solely the task the name and step descriptions; (ii) multi-method scripts, which describe different methods to

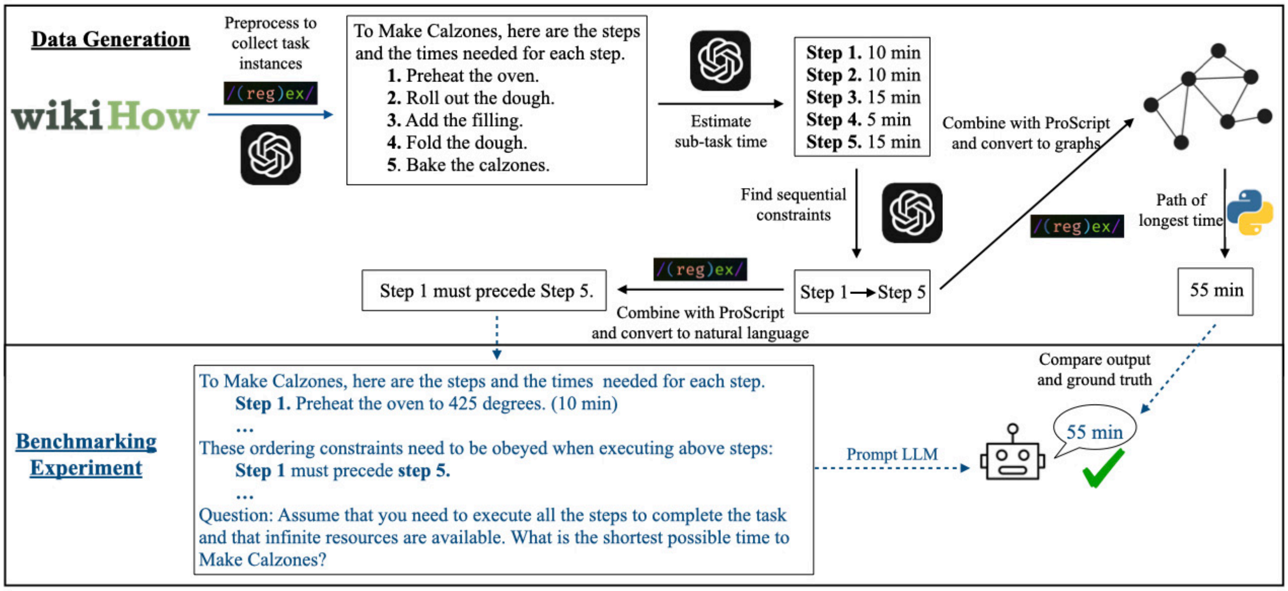

Figure 10. Workflow of data generation and experiment setting. During data generation, we first conduct **preprocessing** to collect high-quality scripts from WikiHow that suit our goal with regular expression and GPT-3.5, then prompt GPT-3.5 to **generate time estimations**. Data points with non-numerical estimations are discarded. The remaining data points are run through GPT-4 to **get step dependencies** in dot language. Then, we combine our dataset with ProScript and pull together information to **generate natural language prompts**. We use an external oracle symbolic processor to **generate graphs** representing given information as well as **generate ground truth answers** for optimal task time duration. In benchmarking experiment, we prompt LLMs with natural language task descriptions and compare models' answers with gold symbolic processor outputs.

solve one task; and (iii) multi-part scripts, which describe complementary parts to solve a task. We exclude script type (i) and (iii) instances entirely if any of their sub-parts do not meet our requirement; we only exclude the unqualified methods in script type (ii) as we can view different methods as different independent scripts. We clarify the qualification criteria below.

A.2.1. Pre-processing

First, we take the scripts with a collaborative rating score (i.e., the helpful percentage in WikiHow) higher than 60% to retain only high-quality plans marked as useful by the users of WikiHow. We leverage GPT-3.5 and keywords methods to filter out plans with optional steps (e.g., `'If you are not happy with the results, proceed to follow steps'`) and others that do not fit into our research goal (we report a list of filtering keywords and descriptions in Appendix A.4). Inspired by Wang et al. (2022), plans that contain unnecessary steps are further filtered with GPT-3.5 in a few-shot setting by sampling three answers (with the temperature set to one) to exclude those in which the majority vote does not agree on all parts/steps being necessary for a script (see prompts and details in Appendix A.5).

A.2.2. Time Annotation

We use GPT-3.5 to estimate the time duration per step. We use a zero-shot setting by sampling three answers (with temperature set to one) and exclude instances whose steps duration an LLM cannot quantify numerically (see prompts and details in Appendix A.5). Empirically, we keep the longest among all the time estimations GPT-3.5 proposes for each step. We note that the longest time estimation is not necessarily always the only acceptable answer. We do not particularly verify time annotation because the task time estimation in a less grounded setting as ours tends not to have a unique gold answer (e.g. `'finding a gym'` may take five minutes or a week to different people), and GPT-3.5 is reported being a reliable annotator for this task (Jain et al., 2023). Furthermore, our interest is in assessing whether an LLM outputs the optimal plan for a task, and we expect end users to supply different time durations when querying a model.

A.2.3. Step Ordering Constraint Annotation

We use GPT-4 in zero shot with the temperature set to one to sample five answers per prompt to annotate dependencies among steps (see more details in Appendix A.5). Specifically, we first shuffle steps in each script and then use *dot* language, an unambiguous syntax (e.g., `‘step 1 must precede step 2’` is expressed as `‘step 1 → step 2’`), to obtain step dependencies to be compliant with the ProScript format. For flat scripts type (i), we annotate dependencies among all steps, while for multi-method scripts type (ii), we annotate dependencies by method; multi-part task dependencies in type (iii) are annotated first among different parts and then among different steps per part, which are then combined to formulate the final dependencies. After removing redundant dependencies, we keep data points that have at least four consistent answers that form asynchronous plans and discard the others (e.g. in an answer saying `‘step 1 → step 2’, ‘step 2 → step 3’, ‘step 1 → step 3’`, we remove `‘step 1 → step 3’`) in a similar vein to self-consistency prompting (Narang et al., 2023).

A.2.4. Natural Language Prompt Generation

We combine all asynchronous instances with complete time annotation for all meaningful steps in ProScript with our generated asynchronous instances from WikiHow. We filter out all instances with unannotated preparation steps in ProScript or those flagged as unsafe by either GPT-3.5 or GPT-4 in WikiHow during the above generation process. By keeping only high-quality asynchronous plans from the WikiHow and the ProScript datasets, we obtain a collection of 1.6K instances.

We then generate natural language prompts based on the task information. Dot language, which we used for ordering constraint generation, provides an unambiguous syntax to formulate a planning problem, yet users tend to use natural language descriptions to specify such a task. We prompt GPT-4 to express the dependency constraint with different linguistic formulations, and we end up with ten plausible templates. We report all templates in Appendix A.6 and include prompts for them in our final dataset.

We further note that people can combine different constraint expressions for succinct usage, a phenomenon widely accepted linguistically known as the principle of quantity/economy (Grice, 1975; Vicentini, 2003). For example, the constraint `‘Step 1 must precede step 2, step 1 must precede step 3’` can be similarly uttered as `‘Step 1 must precede step 2 and 3’`. We, therefore, also include economic usage for these templates by combining the steps following the common preceding step as exemplified above to allow studies about LLMs’ robustness to trivially different natural language prompts.

A.2.5. Graph and Gold Answer Generation

To generate gold answers for a planning task, we parse step dependencies with regular expressions and generate an equivalent DAG representing the workflow. We generate the optimal time duration for a plan by iterating every sequential path and choosing the longest one (the longest path algorithm would produce an equivalent gold label). Each planning task is eventually coupled by four types of graph representations: the adjacency and the edge list, the adjacency matrix, and the compressed sparse row (csr). Such representations can be used to aid LLMs in structural reasoning and assess LLMs’ robustness against different representations of the same graphs.

A.3. Topic Assignment

WikiHow data is coupled with metadata, including category hierarchy. For the WikiHow proportion of our benchmark, we take the top-ranked category for each instance as its topic. To assign topics for datapoints in ProScript, we use fast-text static embedding (Mikolov et al., 2018) trained on 600B Common Crawl data to embed task descriptions for both WikiHow and ProScript data in our benchmark and mean-pooling the representations after removing stop words. We assign a topic its vector by mean-pool all WikiHow task representations for prompts associated with it. We then calculate the cosine similarity and select the highest one as the corresponding category between each topic vector and task vector for each task in ProScript and assign the topic with highest similarity for each task.

A.4. Keywords Excluded From WikiHow Dataset

We exclude instances that contain the following keywords that fall into categories out of our benchmark goal during dataset pre-processing:

As our task is represented without context such as images in the WikiHow webpage, we exclude **context-dependent words**:

`this, above, below`.
We only maintain tasks that can be calculated for their exact time duration so we exclude **words indicating ongoing process or no time duration**: `keep, know, knowing, become, be, stay, repeat`.
We assume all steps in a plan are compulsory so we exclude **words indicating optional procedures**: `opt, if`.
We want all steps in a plan to not overlap each other so we exclude **words indicating parallel constraints**: `when, while`.
We assume steps to have no intervals among them so we exclude **words indicating intervals between steps**: `after, before`.

A.5. Prompts and Settings for Dataset Generation

Table 3. Prompts and settings used to generate AsyncHow dataset.

	Task Prompt	N-shot Example	Shot number	System Prompt	Temperature	Sampling strategy
Necessity Check	Here is a script in [TASK]. [TASK DESCRIPTION] Question: Is this script showing different alternatives to complete this task? Let's think step by step then provide final answer yes or no in double quotes. Answer:	To 'Make a Chicken Sandwich', here is a script in 'Making a Fried Chicken Sandwich'. step1: Done!; step2: Add oil to a large frying pan.; step3: Cut the chicken into thin strips and add toppings of your choice.; step4: Get the necessary ingredients.; step5: Mix the batter.; step6: Batter the chicken.; step7: Put each piece of chicken in the pan. Question: Is this script showing different alternatives to complete this task? Let's think step by step then provide final answer yes or no in double quotes. Answer: The steps as presented are not in a logical sequential order. However, they don't provide alternative methods to make a fried chicken sandwich but rather are parts of a single method that are out of order. To properly make a sandwich, these steps need to be rearranged into a sensible sequence (e.g., gathering ingredients, preparing the chicken and batter, frying the chicken, and assembling the sandwich).	5	You are a helpful plan organizer.	1	Sample 3 answers and take the majority vote.
Time Annotation	Here is a script to [TASK]. [TASK DESCRIPTION] Question: Can you estimate each step in this script for a numerical time duration where possible? Let's think step by step and finally answer in a json format "step index": "duration", each entry a line with double quotes and no description	NONE	0	You are a helpful plan organizer.	1	Sample 3 answers and take the longest time if any. Exclude instances which have unannotatable steps.
Step Dependency Annotation	Here are randomly ordered steps needed to [TASK]. [TASK DESCRIPTION] Assume infinite resources are available and that steps should be parallelized where possible. For each step, does it logically need to follow others considering the nature of the task? Let's think step by step then finally answer in dot language for all necessary constraints, each constraint per line in the format of "preceding step index" $->$ "following step index"	NONE	0	You are ChatGPT.	1	Sample 5 answers and take the 4 consistent answers after removing auxiliary dependencies. Exclude instances that have less than 4 consistent answers.

A.6. Natural Language Prompts for Step Ordering Constraints

We prompt GPT-4 web interface to provide 10 different ways to express dot language constraints with natural language, and randomly sample from them to form our prompts.

Prompt

Suppose you need to execute several steps to complete a task. With dot language, you can represent temporal constraints of the steps such as ”preceding step” $->$ ”following step”. How can you succinctly express the constraint in natural language? Give me 10 different expressions.

Response

To express the temporal constraints between a preceding step and a following step in natural language, you could say:

”Before starting [following step], complete [preceding step].”
”[Preceding step] must precede [following step].”
”Upon completing [preceding step], proceed to [following step].”
”After [preceding step], [following step] should commence.”
”Prioritize [preceding step] before advancing to [following step].”
”[Preceding step] is a prerequisite for [following step].”
”Initiate [following step] subsequent to [preceding step].”
”Ensure [preceding step] is done before [following step].”
”Sequence the tasks: firstly [preceding step], then [following step].”
”[Following step] follows the completion of [preceding step].”

A.7. Varying Natural Language Prompting Details

We first sample 100 prompts for all templates in A.6 to compute model accuracies in these templates **without** succinct usage (e.g. `‘Step 1 must precede step 2, step 1 must precede step 3’`). Then, as we note that the succinct usage of such linguistic utterances is widely accepted as the principle of economy/quantity: do not say more than what’s needed (Grice, 1975; Vicentini, 2003), we choose the best template and test whether a model can achieve better performance **with** succinct usage (e.g. `‘Step 1 must precede step 3’`). Then, we choose the best-performing prompting setting for the rest of the experiment.

A.8. Prompt Examples for Benchmarking Experiment

Zero shot (zero-shot)

To create a video game, here are the steps and the times needed for each step.
Step 1. Learn the basics of programming (180 days)
Step 2. Learn to use a language that is used in games (60 days)
Step 3. Learn to use an existing game engine (30 days)
Step 4. Program the game (90 days)
Step 5. Test the game (30 days)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 2.
Step 1 must precede step 3.
Step 2 must precede step 4.
Step 3 must precede step 4.
Step 4 must precede step 5.

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to create a video game? Answer the time in double quotes.
Answer:

Zero shot+CoT (zero-shot+CoT)

To create a video game, here are the steps and the times needed for each step.
Step 1. Learn the basics of programming (180 days)
Step 2. Learn to use a language that is used in games (60 days)
Step 3. Learn to use an existing game engine (30 days)
Step 4. Program the game (90 days)
Step 5. Test the game (30 days)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 2.
Step 1 must precede step 3.
Step 2 must precede step 4.
Step 3 must precede step 4.
Step 4 must precede step 5.

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to create a video game? Let's think step by step and then answer the time in double quotes.
Answer:

***k*-shot**

###Examples:
To Make Calzones, here are the steps and the times needed for each step.
Step 1. Preheat the oven to 425 degrees. (10 min)
Step 2. Roll out the dough. (10 min)
Step 3. Add the filling. (15 min)
Step 4. Fold and pinch the dough. (5 min)
Step 5. Bake the calzones. (25 min)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 5.
Step 2 must precede step 3.
Step 3 must precede step 4.
Step 4 must precede step 5.

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to Make Calzones? Answer the time in double quotes.
Answer: The shortest possible time to Make Calzones is ”55 min”.
...**[TWO MORE EXAMPLES]**...
###
[ZERO SHOT PROMPT]

***k*-shot+CoT**

###Examples:
To Make Calzones, here are the steps and the times needed for each step.
Step 1. Preheat the oven to 425 degrees. (10 min)
Step 2. Roll out the dough. (10 min)
Step 3. Add the filling. (15 min)
Step 4. Fold and pinch the dough. (5 min)
Step 5. Bake the calzones. (25 min)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 5.
Step 2 must precede step 3.
Step 3 must precede step 4.
Step 4 must precede step 5.

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to Make Calzones? Answer the time in double quotes.
Answer: Since step 1 must precede step 5, step 2 must precede step 3, step 3 must precede step 4, step 4 must precede step 5, we can conclude that we must execute step 2, step 3, step 4, then step 5 sequentially, and since step 1 happens before step 5, it can be done in parallel with step 2, 3, and 4, preceding step 5. Since sequentially executing step 2, 3, 4, and 5 takes 10 + 15 + 5 + 25 = 55 min, while sequentially executing step 1 then step 5 only takes 10 + 25 = 35 min, the shortest possible time to Make Calzones is ”55 min”.
...**[TWO MORE EXAMPLES]**...
###
[ZERO SHOT+COT PROMPT]

PLaG (explicit graph, graph is adjacency list)

###Examples:
To Make Calzones, here are the steps and the times needed for each step.
Step 1. Preheat the oven to 425 degrees. (10 min)
Step 2. Roll out the dough. (10 min)
Step 3. Add the filling. (15 min)
Step 4. Fold and pinch the dough. (5 min)
Step 5. Bake the calzones. (25 min)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 5.
Step 2 must precede step 3.
Step 3 must precede step 4.
Step 4 must precede step 5.

Here is the adjacency list representation of the step ordering constraints:
{’1’: [’5’], ’2’: [’3’], ’3’: [’4’], ’4’: [’5’], ’5’: [’END’], ’END’: [], ’START’: [’1’, ’2’]}
Time for each step can be represented as a dictionary:
{’1’: ’10 min’, ’2’: ’10 min’, ’3’: ’15 min’, ’4’: ’5 min’, ’5’: ’25 min’}

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to Make Calzones? Answer the time in double quotes.
Answer: Since step 1 must precede step 5, step 2 must precede step 3, step 3 must precede step 4,
step 4 must precede step 5, we can conclude that we must execute step 2, step 3, step 4, then step 5 sequentially, and since step 1 happens before step 5, it can be done in parallel with step 2, 3, and 4, preceding step 5. Since sequentially executing step 2, 3, 4, and 5 takes 10 + 15 + 5 + 25 = 55 min, while sequentially executing step 1 then step 5 only takes 10 + 25 = 35 min, the shortest possible time to Make Calzones is ”55 min”.
...**[TWO MORE EXAMPLES]**...
###
To create a video game, here are the steps and the times needed for each step.
Step 1. Learn the basics of programming (180 days)
Step 2. Learn to use a language that is used in games (60 days)
Step 3. Learn to use an existing game engine (30 days)
Step 4. Program the game (90 days)

Step 5. Test the game (30 days)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 2.
Step 1 must precede step 3.
Step 2 must precede step 4.
Step 3 must precede step 4.
Step 4 must precede step 5.

Here is the adjacency list representation of the step ordering constraints:
{'1': ['2', '3'], '2': ['4'], '3': ['4'], '4': ['5'], '5': ['END'], 'END': [], 'START': ['1']}
Time for each step can be represented as a dictionary:
{'1': '180 days', '5': '30 days', '2': '60 days', '4': '90 days', '3': '30 days'}

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to create a video game? Let's think step by step and then answer the time in double quotes.
Answer:

PLaG (BaG, graph is adjacency list)

###Examples:
To Make Calzones, here are the steps and the times needed for each step.
Step 1. Preheat the oven to 425 degrees. (10 min)
Step 2. Roll out the dough. (10 min)
Step 3. Add the filling. (15 min)
Step 4. Fold and pinch the dough. (5 min)
Step 5. Bake the calzones. (25 min)

These ordering constraints need to be obeyed when executing above steps:
Step 1 must precede step 5.
Step 2 must precede step 3.
Step 3 must precede step 4.
Step 4 must precede step 5.

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to Make Calzones? Let's construct a graph with the nodes and edges first to represent step ordering constraints, and also construct a dictionary to represent time needed for each step. Use the graph and dictionary to calculate the shortest possible time needed for the task. Let's think step by step and then answer the time in double quotes.
Answer: Here is the adjacency list representation of the step ordering constraints:
{'1': ['5'], '2': ['3'], '3': ['4'], '4': ['5'], '5': ['END'], 'END': [], 'START': ['1', '2']}
Time for each step can be represented as a dictionary:
{'1': '10 min', '2': '10 min', '3': '15 min', '4': '5 min', '5': '25 min'}
Since step 1 must precede step 5, step 2 must precede step 3, step 3 must precede step 4, step 4 must precede step 5, we can conclude that we must execute step 2, step 3, step 4, then step 5 sequentially, and since step 1 happens before step 5, it can be done in parallel with step 2, 3, and 4, preceding step 5. Since sequentially executing step 2, 3, 4, and 5 takes 10 + 15 + 5 + 25 = 55 min, while sequentially executing step 1 then step 5 only takes 10 + 25 = 35 min, the shortest possible time to make Calzones is "55 min".
...**[TWO MORE EXAMPLES]**...
###

To create a video game, here are the steps and the times needed for each step.
Step 1. Learn the basics of programming (180 days)
Step 2. Learn to use a language that is used in games (60 days)
Step 3. Learn to use an existing game engine (30 days)
Step 4. Program the game (90 days)
Step 5. Test the game (30 days)

These ordering constraints need to be obeyed when executing above steps.
Step 1 must precede step 2.
Step 1 must precede step 3.
Step 2 must precede step 4.
Step 3 must precede step 4.
Step 4 must precede step 5.

Question: Assume that you need to execute all the steps to complete the task and that infinite resources are available. What is the shortest possible time to create a video game? Let's construct a graph with the nodes and edges first to represent step ordering constraints, and also construct a dictionary to represent time needed for each step. Use the graph and dictionary to calculate the shortest possible time needed for the task. Let's think step by step and then answer the time in double quotes.
Answer:

Prototypical task (edge list)

The following lists of nodes [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and edges [[0, 1, 1], [1, 2, 1],[1, 3, 1], [2, 10, 1], ..., [9, 10, 5]] define a directed acyclic graph. Each element in the list of edges is expressed in the form (i,j,w), and specifies that node i connects to node j with weight w. What is the length of the longest path from node 0 to node 10? Think step by step and then reply with the numerical value of the shortest path enclosed by <result><result> tags.
Answer:

A.9. Results after Excluding Invalid Instances

We report results after excluding invalid instances altogether in all models and settings per experiment if an instance is filtered in any setting in an experiment (e.g. if instance indexed 0 is invalid in zero-shot GPT-4 experiment, we remove it from all the test results for all models and settings). General conclusions remain the same as our main content.

Table 4. Model accuracy in different settings on the AsyncHow benchmark. Model performances without our method are in plain background, while those with our method are in blue background. We mark the best performance per model in **bold**. Following Dror et al. (2018), we use McNemar's tests (McNemar, 1947) to obtain p-values and Holm-Bonferroni method (Holm, 1979) to correct them for each evaluation to test the statistical significance of performance difference between experiment with and without our proposed method. We denote with † when the performances with PLaG are significantly better ($p < 0.05$) than the best result without.

Model	zero-shot	zero-shot + CoT	k-shot	k-shot + CoT	PLaG (explicit graph)	PLaG (BaG)
GPT-4	0.128	0.128	0.108	0.657	0.728†	**0.771**†
GPT-3.5	0.191	0.217	0.241	0.224	0.284†	**0.348**†
Command	0.079	0.015	0.051	0.078	**0.098**	0.052
LLaMA-2-70B-chat	0.039	0.036	0.053	0.074	**0.101**†	0.069
Mistral-7B-Instruct	0.074	0.070	0.099	0.142	**0.155**	0.144

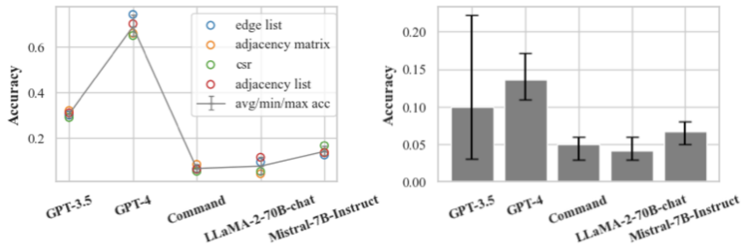

Figure 11. Left plot refers to average model performance accuracy with k shot + CoT in different graphs. Dots refer to model performance with different graph types. Grey lines refer to average accuracy with different graph types. The right plot refers to average zero-shot accuracy in different text prompts. Error bars in both plots refer to worst/best performance

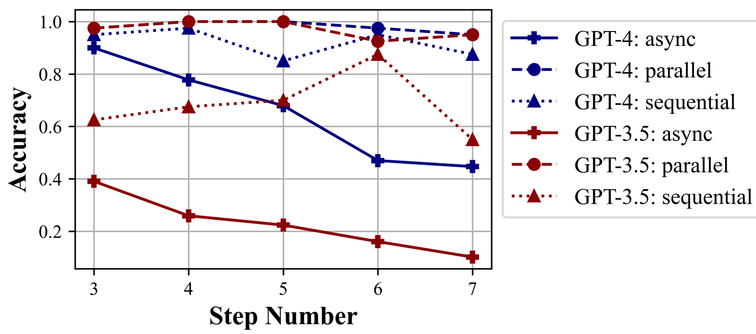

Figure 12. Comparing parallel/sequential plan execution accuracy with asynchronous plans. All experiments are done in the setting of k-shot + CoT. Blue and orange lines refer to GPT-4 and GPT-3.5 results respectively.

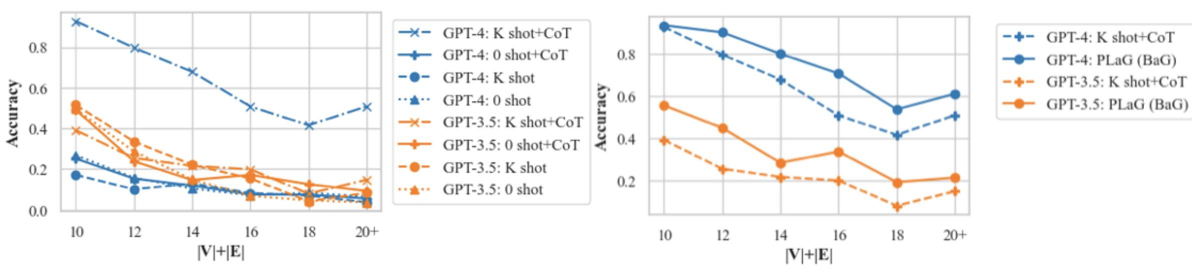

Figure 13. Model accuracy concerning task complexity. The left figure plots model performance without our method (PLaG), and the right plot displays the models' best performance with/without our method.

A.10. Further Comparison with Chain-of-Thought Self-consistency and Tree of Thought

We further added experiments on Chain-of-Thought Self-Consistency (CoT-SC) (Wang et al., 2022) and ToT (Yao et al., 2024) (both for k shot, k=3 as in our main paper). We report results for k-shot CoT-SC and ToT, in comparison to PLaG below (best results are in bold). We see that our method outperforms both methods while inducing much less cost (see Appendix A.11).

Table 5. Comparing k-shot CoT-SC/ToT with PLaG on GPT-3.5 and GPT-4. The best results per model are in bold. Our methods are always superior.

	k-shot CoT-SC	k-shot ToT	PLaG (explicit graph)	PLaG (BaG)
GPT-3.5	0.240	0.263	0.290	**0.355**
GPT-4	0.625	0.624	0.730	**0.777**

A.11. Latency Analysis

We analyze the cost-performance trade-off for PLaG here. PLaG introduces significantly longer inference sequences than the zero-shot setting, but we note that its cost is reasonable compared to other advanced prompting methods. We provide a statistical comparison of the average input/output token count per task for GPT-4 below.

We note that the output token lengths of PLaG (explicit graph) are comparable to that of k-shot CoT, while k-shot CoT-SC and k-shot ToT are much more expensive and underperform both PLaG methods (see Appendix A.10).

Table 6. Latency analysis: comparing PLaG with other prompting methods.

	zero shot	k-shot CoT	PLaG (explicit graph)	PLaG (BaG)	k-shot CoT-SC	k-shot ToT
tokens (input/output)	207/5	1289/135	1698/138	1775/242	1289/407	5212/335

The increased input and output length caused by PLaG may raise concerns about the scaling potential of the technique (i.e., whether it is still applicable when there are hundreds of nodes or more in a graph). First, we consider this to be a problem mainly for LLM context window length, which is universal to all NLP problems in general. In summarization tasks, for example, an LLM can't summarize a book whose length goes beyond the LLM's context length, but this doesn't invalidate summarization as a task. Second, the length of graph representation also depends on the graph format: if a graph is provided as its dependency list, its size will grow linearly with the number of nodes and edges, as opposed to the adjacency matrix, which scales quadratically.

Second, our dataset, which is generated from real-life tasks without specific pruning for complex ones, shows very sparse data points for complexity which goes beyond 20 (Figure 5). This observation motivates us to consider additional graph prompting as a valid technique to improve LLM performance in a wide range of tasks.

A.12. Comparing PLaG Performance on Model-generated and Human-annotated Data

Here, we compare GPT-3.5/4 performance on model-generated and human-annotated data to show that the performance gap between the explicit graph and BaG used in PLaG is not caused by noise in sampling. We perform additional experiments to compare BaG and explicit graph on task complexity 14, which has > 100 data points for both the synthetic and human-annotated parts of our dataset. We find that the BaG framework performs consistently better than the explicit graph as shown below, which means the superior performance of BaG should not be attributed to noise sampling.

Table 7. Compare explicit graph (left) and BaG (right) as PLaG methods in human-annotated data and model-generated data. The best results are in bold.

	Human-annotated data	Generated data
GPT-3.5 (explicit graph/BaG)	0.239/**0.279**	0.315/**0.389**
GPT-4 (explicit graph/BaG)	0.614/**0.701**	0.778/**0.796**

A.13. Model Performance with Economic Linguistic Expressions

We report results comparing LLMs' performance between direct and economic expressions in Figure 8. Generally, models' performance downgrades when using economic expressions except LLaMA-2-70B-chat.

Table 8. Compare model performance with unambiguous direct expression with economic expressions.

Model	Best direct expression performance	Performance difference after using economic expressions
GPT-3.5	0.222	-0.012
GPT-4	0.171	-0.021
Command	0.06	-0.01
LLaMA-2-70B-chat	0.06	+0.06
Mistral-7B-Instruct	0.08	-0.01

A.14. The *Prototypical* Distribution as a Proxy of the Naturalistic Benchmark

In this subsection, we show that $|V|$ and $|E|$ equally contribute to the complexity of a planning task, with no clear dominance of one over the other Figure 14 and the similarity of prototypical and naturalistic graphs Figure 15.

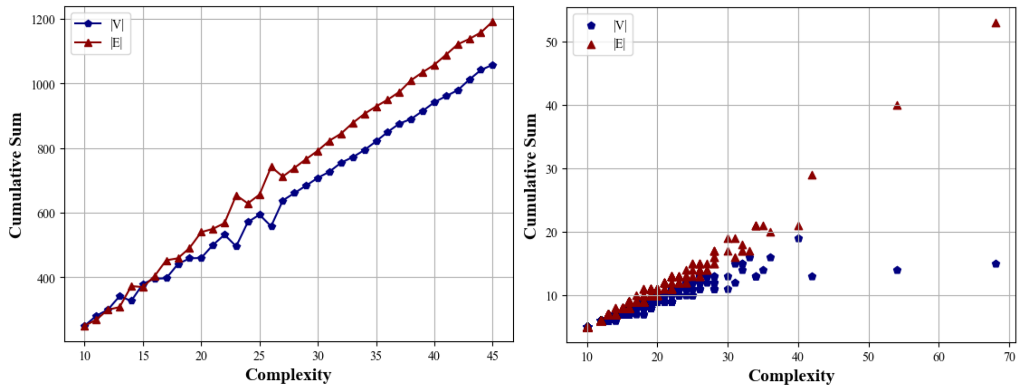

Figure 14. Comparison of the cumulative number of vertices $|V|$ and edges $|E|$ per-complexity task for the prototypical (left) vs. naturalistic datasets.

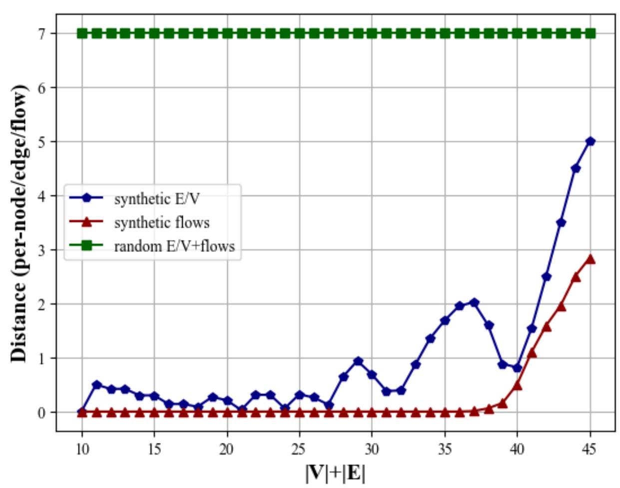

Figure 15. Comparison of the graph distribution of prototypical graphs and that of random graphs. In blue and red, a comparison between the number of nodes/edges and flows of prototypical and naturalistic graphs. In green, comparison of nodes/edges and flows of prototypical and random graphs (baseline). For larger complexities, prototypical graphs become similar to DAG in terms of the number of edges, nodes and flows (i.e., tasks that can be executed simultaneously).

A.15. Analysing Time Units Differences in Complexity Levels $20+$ and 18

We define a list of time units ['sec', 'min', 'h', 'day', 'week', 'month', 'year'] and define time unit distance per instance as the difference between the unit with the highest and lowest index. For instance, a script with steps timed as 5 sec, and 10 min respectively is considered as having a distance of 1, while a script with steps timed as 15 h and 50 h has a distance of 0.

We find that the average time distance over all scripts at 20+ (0.339) is considerably lower than the average distance at complexity 18 (0.801), which partially explains why the accuracy at 20+ has a jump.

A.16. Experiment Details and Hyperparameters

All experiments are performed from December 2023 to May 2024.

For data generation, we use Azure OpenAI API and set temperature=1 for both GPT-35-turbo and GPT-4. In dependency validation, we sample prompts by seed 0, 1, and 2.

During the experiment (i.e. inference stage), we use Azure OpenAI API and set temperature=0 for GPT models to enable as much reproducibility as possible. We use Cohere API to query the Command model and also set temperature = 0. As GPT models and Command filter contents, we query API 3 times to see if the corresponding model is willing to answer the prompt.

We use Huggingface Inference API to query LLaMA-70B-Chat and set do sample=False, max new tokens=4096, and seed=0. We use 2 V100 GPUs and 1 A100 GPU for Mistral-7B-instruct inference, with do_sample=False, temperature=0, max new tokens=4096 and torch manual seed=2024.

A.17. Dataset Information

We use ProScript and WikiHow as our base dataset in data generation. We follow the licensing guide of ACL and determine ProScript to be under CC BY 4.0. WikiHow dataset we use is under MIT License. We follow the licenses used by the existing datasets for our dataset.

A.18. Human Validation

We conduct human validation of WikiHow on a voluntary basis with four experts. Consent was obtained via discussion with them. We do not provide personally identifiable information in the dataset.

A.19. Statement of Contribution

FL wrote the paper, developed the initial idea, developed part of formalism, generated the AsyncHow dataset, ran all experiments, and conducted all analyses unless specified below. EMY helped in polishing ideas, rephrasing prompts, and editing the paper. ELM developed part of formalism, generated a synthetic dataset, ran synthetic experiments, and advised and edited the paper. AGC obtained funding, advised, and edited the paper. VH and JBP advised and edited the paper.