TaskLAMA: Probing the Complex Task Understanding of Language Models

Quan Yuan*, Mehran Kazemi*, Xin Xu*, Isaac Noble, Vaiva Imbrasaite, Deepak Ramachandran

Google Research

{yquan, mehrankazemi, xxujasmine, isaacn, vimbrasaite, ramachandrand}@google.com

Abstract

Structured Complex Task Decomposition (SCTD) is the problem of breaking down a complex real-world task (such as *planning a wedding*) into a directed acyclic graph over individual steps that contribute to achieving the task, with edges specifying temporal dependencies between them. SCTD is an important component of assistive planning tools, and a challenge for commonsense reasoning systems. We probe how accurately SCTD can be done with the knowledge extracted from pre-trained Large Language Models (LLMs). We introduce a new high-quality human-annotated dataset for this problem and novel metrics to fairly assess performance of LLMs against several baselines. Our experiments reveal that LLMs are able to decompose complex tasks into individual steps effectively, with a relative improvement of 15% to 280% over the best baseline. We also propose a number of approaches to further improve their performance, with a relative improvement of 7% to 37% over the base model. However, we find that LLMs still struggle to predict pairwise temporal dependencies, which reveals a gap in their understanding of complex tasks.

Introduction

In their daily lives, people are involved in executing multiple tasks of different temporal granularity in order to achieve their varied goals. There is abundant evidence that consciously decomposing a complex task into smaller sub-tasks leads to more efficient and reliable execution. For example, Kokkalis et al. (2013) show that people tend to achieve tasks better and faster when given a concrete plan with actionable steps. Cheng et al. (2015) show that breaking a macro-task into micro-tasks for workers results in more accurate overall quality and allows for easier recovery from interruption. Chilton et al. (2013) show that breaking down a task into sub-tasks is beneficial for taxonomy creation. Teevan, Iqbal, and Von Veh (2016) show that breaking a task into sub-tasks can be beneficial for collaborative writing (e.g., writing a description of a shared project). Allen and Peikoff (2001) argue in their book that *"there is an inverse relationship between things on your mind and those things getting done"*.

Given a complex task, the goal of structured complex task decomposition (SCTD) is to automatically find a complete set of necessary steps for achieving the task, and specify temporal dependencies between these steps (i.e. which steps should be done before which). The output can be described as a directed acyclic graph, which we term *Task Graph*, where the nodes represent the steps and the edges represent temporal dependencies. Any ordering of the nodes that respects the graph edges is a possible ordering of the steps to accomplish the task. Figure 1 demonstrates an example task graph. The importance of this problem has led to an extensive classic and modern literature for developing solutions based on the latest AI technologies available (Newell, Simon et al. 1972; Kokkalis et al. 2013; Awadallah et al. 2014; Zhang et al. 2021). These works were also motivated by the feature of SCTD being a quintessential or (informally speaking) AI-Complete problem, involving many features of human-level reasoning such as logical/defeasible reasoning, uncertainty and high context-dependence. The existing solutions in the literature are typically based on crowd-sourcing (Kokkalis et al. 2013; Zhou et al. 2022b), based on similarity or co-occurrence of user search queries (Awadallah et al. 2014; Mehrotra and Yilmaz 2017; Zhang et al. 2015), or based on summarizing the content of relevant web-pages found through web search (Zhang et al. 2021).

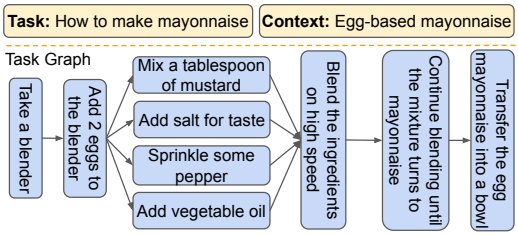

Figure 1: An example of a task graph for a complex task from the *TaskLAMA* dataset. Nine steps are outlined. The four steps with horizontal texts can be done in any order.

In this paper, we probe the extent to which such knowledge can be extracted from large language models (LLMs). Previous work has shown that LLMs contain a large amount of different types of knowledge (Petroni et al. 2019; Sung et al. 2021; West et al. 2021) and can do various types of reasoning (Wei et al. 2022; Nye et al. 2021; Kazemi et al. 2023a). Our work extends this line of work by probing

*Contributed equally.

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

LLMs for their SCTD knowledge and reasoning abilities, and demonstrating their current strengths and limitations.

We create a high-quality human-annotated dataset for the SCTD problem. Following the current convention of naming language model probes as *xLAMA*, we name the dataset *TaskLAMA*. Specifically, we gather a set of 1612 tasks and ask human annotators to 1) write their assumptions to provide context for the task, 2) write the required steps for those tasks under the provided context, and 3) specify the temporal dependencies between the steps. This gives us a total of 12118 steps and 11105 temporal dependencies.

We identify a potential problem with the metrics used in previous work to measure the quality of the generated nodes: one can arbitrarily improve the metric by simply adding duplicate sub-steps. To solve this issue, we propose robust metrics and report results with them providing the first fair comparison of methods for this problem. We also develop novel metrics for measuring the quality of the generated temporal dependencies. These metrics are potentially generally applicable in other settings where annotated/labeled graphs produced by generative models must be compared.

We compare the performance of LLMs on TaskLAMA against various baselines. Our results reveal that the steps generated by an off-the-shelf LLM have higher quality than the baselines, offering $15\% - 280\%$ relative improvement compared to the best baseline in terms of different metrics; they also understand the context and can adapt the generated steps based on the context in which the complex task is to be done. We then propose a number of approaches to improve the performance of off-the-shelf LLMs even further, using the specialized structure of the SCTD problem. The combination of these solutions result in $7\% - 37\%$ improvement over the base model, depending on the metric we use. We also measure the quality of the temporal dependencies produced by LLMs and observe that while LLMs are good at generating good sequences of steps, their ability in predicting pairwise temporal dependency remains unsatisfactory.

A summary of our main contributions follow: 1- we create TaskLAMA, a high-quality probe specifically focused on complex real-world task understanding, 2- we develop metrics for measuring model performance for SCTD, 3- we propose various LLM-based approaches for SCTD and compare against a number of baselines that do not leverage LLMs, 4- we conduct a comprehensive set of experiments showing that LLMs perform well at decomposing a complex task into a sequence of steps, but their understanding of temporal dependencies between these steps remains unsatisfactory.

Related Work

We categorize the related work as follows:

Crowd-Sourced SCTD: One line of work uses crowdsourcing for obtaining the steps and their temporal order for complex tasks. Kokkalis et al. (2013) develop a framework where users can find information about various tasks: if a similar task (measured using natural language processing tools) already exists in their database, the information about that task is shown to the user; otherwise, the task is sent for crowd-sourcing. Zhou et al. (2022b) combine the information from the WikiHow website to produce hierarchical task trees. While crowd-sourcing may lead to high-precision task graphs, it is costly and may suffer from low-recall as new task graphs cannot be built on-the-fly for novel tasks.

Query-based SCTD: Another commonly used approach for SCTD is by leveraging user search queries. Awadallah et al. (2014) create sessions from the search queries of a commercial search engine and propose steps for complex tasks by finding the queries that frequently co-occurred with the complex task in different sessions. Mehrotra and Yilmaz (2017) propose a hierarchical clustering approach for search queries where the queries higher in the hierarchy correspond to tasks and their children represent steps to those tasks. Zhang et al. (2015) map queries to demands using external knowledge and mine frequent demand patterns. In this work, we compare against a number of query-based approaches.

Summarization-based SCTD: Zhang et al. (2021) proposed an approach to SCTD where for a given complex task, first a web search is done to identify relevant web pages, and then a language model is trained to summarize the contents of those web-pages into task graphs. In this work, we take a different approach by measuring how much of the information can be directly obtained from the LLM itself.

Hierarchical Task Networks (HTNs): HTNs provide an approach to automated planning where the action dependencies can be given in the form of a hierarchically structured network (Erol 1995; Humphreys 2019; Guan et al. 2023). They are typically used for planning problems where the world is observable, and the set of actions, pre-conditions, and effects can be explicitly defined. In our setting, however, plans should be constructed for many different tasks (many of which only observed at the test time) and with a very partial view of the world.

LLM Knowledge Probing: Previous work has shown that LLMs contain a large amount of different types of knowledge. This includes factual (Petroni et al. 2019; Jiang et al. 2020), commonsense (Zhou et al. 2020; Davison, Feldman, and Rush 2019; Yin et al. 2022), biomedical (Sung et al. 2021), numerical (Lin et al. 2020), scale (Zhang et al. 2020), and many other types of knowledge. Most related to our work, it has been shown that LLMs perform well in breaking a simple goal into specific low-level actions a robot needs to take to achieve the goal (Huang et al. 2022) (e.g., providing the low-level steps for a goal such as *throw away garbage)*; they also perform well in simple, advice-seeking scripts (Sakaguchi et al. 2021; Madaan et al. 2022; Brahman et al. 2023) (e.g., *go out with friends*, *live somewhere warmer*, etc.). Our work is in the same vein with these works, but extends them by measuring the amount of information one can extract from LLMs for complex tasks requiring multiple (potentially complex) steps to be completed (see Table 1 for a sample of such tasks).

The TaskLAMA Probe

To measure the performance of LLMs for task graph generation, we create a dataset of task graphs for 1630 complex tasks. Following the **LA**nguage **M**odel **A**nalysis (LAMA) naming convention, we call our dataset *TaskLAMA*. Crucially, *TaskLAMA* is created directly by human annotators as opposed to being extracted from public websites (as, e.g., in

Complex Task	Assumption / Context
Build a curved retaining wall	Using concrete
Start a property management company	In Florida
Write a grant proposal	For non-profit
Cook lobster tails at home	Grilled
Install a light switch	–
Get a real estate license	In Texas
Recover deleted photos	From iPhone
Plan a wedding	In Italy
Become a travel agent	Online agent

Table 1: Sample complex tasks and extra assumptions (context) from *TaskLAMA* (– means no extra assumption).

(Wang et al. 2022; Zhang, Lyu, and Callison-Burch 2020)) to allow for evaluating LLMs on previously unobserved data. Before describing the dataset creation process, we start with defining our notation. We represent a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as a tuple with $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ representing the nodes and $\mathcal{E} \subset \mathcal{V}^2$ represent edges. Throughout this paper, we work with directed graphs where, for an edge (v_i, v_j), the order of the nodes is important.

A task graph is defined as follows:

Definition 1 (Task Graph)**.** *A task graph for a complex task* T *is a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *where each node* $v_i \in \mathcal{V}$ *represents a step required for accomplishing* T *and each edge* $(v_i, v_j) \in \mathcal{E}$ *represents a temporal dependence between* v_i *and* v_j*, indicating that* v_i *should be done before* v_j*.*

The problem we study in this paper is the following: Given a complex task T (and sometimes an extra context about the task) as input, generate a task graph $\mathcal{G}$ as output. TaskLAMA provides a probe for this problem with $(\mathsf{T}_i, \mathcal{G}_i)$ pairs. The creation of TaskLAMA involves four main components: 1- selecting a set $\{\mathsf{T}_1, \ldots, \mathsf{T}_\tau\}$ of complex tasks, 2- gathering steps $\mathcal{V}_i$ involved in the execution of these tasks, 3- gathering temporal dependencies $\mathcal{E}_i$ between the steps, and 4- splitting the dataset into train, validation, and test sets. In what follows, we explain each component in detail [1].

Selecting a Set of Complex Tasks

We obtain a varied and representative set of complex tasks performed by humans from the following two sources.

The MSComplexTasks dataset (Zhang et al. 2021): There are 711 distinct tasks in this dataset coming from the logs of Wunderlist, a popular task management application. These tasks are selected from a bigger pool of tasks using a number of filters, most notably filtering simple tasks.

Popular *How To* search queries: From the logs of a commercial search engine, we extracted popular search queries that start with *How To*. To ensure anonymity, we removed any query issued by fewer than 1000 unique users. We then deduplicated these tasks and applied a number of other filters to remove tasks involving sensitive and harmful topics, tasks requiring medical advice, or tasks that did not deem complex (e.g., tasks that did not involve multiple steps). We labeled the remaining tasks based on topic and sampled from each topic to avoid over-presence or under-presence from certain topics. At the end, we obtained 901 tasks.

[1]The full dataset can be downloaded from https://storage.googleapis.com/gresearch/tasklama/tasklama.zip

Gathering Steps for the Tasks

Complex tasks can be typically accomplished in multiple different ways, with a potentially different set of steps involved each time depending on the context (e.g., *Make a burger* can have different steps depending on whether we do it *On a charcoal grill* or *In an air fryer*). To gather the steps $\mathcal{V}$ for a task T while taking the context into account, we instructed annotators to do the following: for each task, write down the assumptions they are making and then the set of steps for the task under those assumptions. Some examples of tasks and assumptions are presented in Table 1.

The annotators were allowed to search and learn about the task, but were required to write the steps in their own words and based on their understanding. The annotators were also instructed to make sure that each step starts with a verb, corresponds to exactly one action, is meaningful as a standalone sentence/does not contain anaphora (e.g., avoid *put it on the grill*), is actionable as opposed to general advice, and is applicable in the context of the assumptions made. If the annotator was unfamiliar with the task, they were instructed to skip it; the task was then sent to another annotator.

We trained the annotators over three rounds of pilot study, each time having them annotate a small number of tasks and then explaining to them the mistakes they made. Some of the dominant mistakes in the initial rounds included directly copying search results, failing to follow one of the rules mentioned above, and/or misunderstanding how to provide assumptions. These issues were mostly resolved over the three rounds of training. In the final round, the workers spent an average of 892 seconds on each task. Through the above process, we gathered a total of 12118 steps for our 1612 tasks, with an average of 7.5 steps per task.

Gathering Temporal Dependencies

Once we gathered the assumptions and steps for each task, a separate set of annotators were asked to specify the order dependencies $\mathcal{E}$ for the steps of each task. The annotators were instructed to first draw the graph on a piece of paper and then submit the edges one by one. Similar to the previous case, we trained the workers over three rounds of pilot studies. The mistakes in the initial rounds ranged from producing a linear sequence instead of a graph, only providing a partial graph (i.e. only a subset of the edges), and misunderstanding the concept of temporal dependence. These issues were mostly resolved over the three rounds of training. In the final round, the workers spent an average of 1138 seconds on each task. Through this process, we gathered a total 11105 temporal dependencies for our 1612 tasks, with an average of 6.9 dependencies per task.

Dataset Splitting

We split the data into train, validation, and test sets in such a way that the tasks are conceptually different in the three sets. Towards this goal, we first grouped the 1612 tasks into

621 clusters based on their textual similarity and then randomly split the clusters into train, validation, and test sets. This splitting strategy ensures some amount of difference in the tasks in each set. Following this splitting strategy, we ended up with 965 examples in the training set, 169 in the validation set, and 478 in the test set.

Method

To generate task graphs for a given complex task, a model needs to 1) generate the steps, and 2) decide the order dependency between the steps.

Generating the Steps

To generate the steps for a task, we experiment with the following strategies:

In-Context Learning (ICL): In ICL (Brown et al. 2020), one provides a few demonstrations each containing an input and the expected output, followed by the query for which the model has to generate the output. The model learns the relation between the input and the output in context, and uses that to generate an output for the provided query.

Multiple Sequences (MultSeq): We notice that when we generate multiple sequences of steps for a task using the ICL approach, the sequences sometimes have complementary steps between them. To leverage this intuition, we generate k sequences of steps using the ICL approach, setting the decoding temperature to 0.5 to allow for diverse generations. Then we deduplicate the steps and combine the remaining steps to obtain the final set of steps. The deduplication procedure is explained in the Appendix.

Sample and Filter (S&F): We notice that when we generate multiple sequences of steps for a task using the ICL approach, some of the sequences have higher quality than the others. To leverage this, we first train a separate model that scores the sequences generated by the ICL approach. Then, we generate multiple sequences of steps and select the one with the highest score according to the trained model. To train a model that can score the generated sequences, we first generate 16 sequences per task for the tasks in our training set, then we evaluate each of the generated sequences with respect to the golden sequence and obtain a single number indicating how good that sequence is. This gives us a dataset of (Task, Sequence of Steps, Score). We then train a model that given a task and a sequence of steps predicts the score.

Soft-Prompt Tuning (SPT): In the case of ICL, the in-context demonstrations we provide as input get mapped to the corresponding token embeddings that are then fed into the LLM. Recently, it has been shown that instead of using a fixed set of token embeddings as the in-context demonstrations, one can learn those embeddings based on training data to enable better in-context examples. This technique is typically referred to as *soft-prompt tuning* (Lester, Al-Rfou, and Constant 2021). We learn the prompt embedding based on our training data and decide the size of the prompt based on performance on our validation set.

MultSeq + S&F: We generate k' sequences of steps using the ICL approach, then select and combine the top k sequences ranked by the S&F model.

MultSeq + SPT: We combine k sequences from the SPT model instead of the ICL model.

S&F + SPT: We use the S&F model to score the sequences of steps generated by SPT and then select the best.

MultSeq + S&F + SPT: This is similar to S&F + SPT except that we combine the steps from the top k sequences.

Generating the Order Dependencies

While task graphs are directed acyclic graphs (e.g., see Figure 1), when using an LLM to generate the steps for a task we get a sequence of steps. We compare a few approaches that can turn the sequence of steps generated by the LLM into a task graph.

Linear: We use the linear order of steps produced by the LLM as the final task graph.

ICL: We provide multiple examples as demonstrations each containing a task, two of its steps, and the label indicating whether the first step should be done before the second one. We then provide a new task and two of its steps and ask for the label.

ICL with Chain-of-Thought: Chain-of-thought (CoT) prompting (Wei et al. 2022) is a technique where besides providing the input and the label, the demonstrations also provide a rationale for the label. We test a version of ICL with CoT where the rationale for the demonstrating examples are written manually.

SPT: We soft-prompt tune the LLM on the training data to learn to predict the label given a task and two of its steps.

LLM Scoring: Given the initial linear order produced by the LLM, we generate m sequences by randomly swapping the order of two steps and use the LLM to score the sequences[2]. We then sort the sequences descendingly based on their LLM score and select the highest scoring sequences. Then, for two steps v_i and v_j, if we see v_i before v_j in some sequences and v_j before v_i in the other sequences, we assume v_i and v_j can be done in any order; otherwise, if v_i always appears before v_j (or vice versa), we assume v_i has to be done before v_j (or vice versa). We turn the sequences into a graph following the above strategy.

In the case of cycles, i.e. if a model predicts that A should be done before B, B should be done before C, and C should be done before A, we remove the dependencies assuming that each of the steps can be done before the other one so they can be done in any order.

Metrics

For evaluation, we need two sets of metrics: one for measuring the quality of the steps (nodes) and one for measuring the quality of the temporal dependencies (edges). We discuss each of these separately.

Node Metrics: Let $\mathcal{V}_G = \{v_1, \ldots, v_n\}$ be the steps in the golden graph, $\mathcal{U}_M = \{u_1, \ldots, u_m\}$ be the steps in the generated graph, and S be a pairwise similarity function of the steps (we use the cosine similarity of the universal sentence encodings (Cer et al. 2018) of the steps). Previous work has

[2]Note that LLMs can be used both to generate an output and also to score a provided output.

-	Rouge1		Rouge2		RougeL		Hungarian		Relaxed Hung.	
Model	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2
Repeat Task	12.2	10.3	1.7	1.5	12.1	10.2	34.6	33.0	41.3	37.7
Repeat Sim	11.6	10.2	1.5	1.3	10.8	9.5	33.	33.4	40.1	38.4
Co-occur	16.4	15.0	2.7	2.5	13.6	12.5	34.8	34.7	41.7	40.0
Hierarchical	14.6	12.6	2.3	2.0	12.7	11.0	34.3	34.1	41.2	39.3
ICL	33.1	31.7	10.0	9.5	24.0	23.0	40.1	40.3	48.1	47.3
MultSeq	32.5	37.2	10.5	12.1	22.9	26.3	35.3	42.8	47.5	50.4
S&F	36.6	34.3	11.3	10.6	25.6	24.1	42.7	41.9	50.2	49.1
SPT	38.7	36.3	13.2	12.3	**26.7**	25.1	43.6	42.4	51.5	50.3
MultSeq + S&F	37.6	38.3	12.1	12.3	25.2	25.7	41.7	44.4	50.5	51.1
MultSeq + SPT	36.8	**41.3**	13.4	**15.0**	25.3	**28.4**	39.6	**46.2**	51.3	**53.2**
S&F + SPT	**39.0**	38.2	13.4	13.1	**26.7**	26.2	**43.8**	43.2	51.5	50.4
MultSeq + S&F + SPT	38.7	40.0	**13.7**	14.1	25.8	26.7	42.5	44.5	**51.6**	51.9

Table 2: The performance of different models for task step generation measured in terms of multiple metrics. Bold indicates winner. ICL stands for In-Context Learning, MultSeq for Multiple Sequences, S&F for Sample and Filter, and SPT for Soft Prompt Tuning.

proposed to compute precision as $\sum_i \frac{\max_j S(u_i,v_j)}{|\mathcal{U}_M|}$ and recall as $\sum_j \frac{\max_i S(u_j,v_i)}{|\mathcal{V}_G|}$ (Zhang et al. 2021). We find, however, that with these metrics, one can arbitrarily increase the precision without sacrificing recall. Consider the case where $\mathcal{V}_G = \{v_1, v_2\}$, $\mathcal{U}_{M_1} = \{u_1, u_2\}$, and $\mathcal{U}_{M_2} = \{u_1, u_1', u_2\}$ and let $S(u_1, v_1) = 0.6$ and zero for other pairs, and u_1' be a near-duplicate of u_1. Ideally, the output of M_1 should be preferred to the output of M_2 because they provide the same information but M_1 has no duplicates. However, using the above formulae, M_1 will have a precision of $^{(0.6+0.0)}/_2 = 0.3$ and a recall of $^{(0.6+0.0)}/_2 = 0.3$, whereas M_2 will have a precision of $^{(0.6+0.6+0.0)}/_3 = 0.4$ and a recall of $^{(0.6+0.0)}/_2 = 0.3$. We observed this issue in our experiments too: combining the steps from multiple sequences (without deduplication) increased both precision and recall.

The problem described above is due to the one-to-many mapping formulation of precision and recall. To solve this issue, we use a Hungarian matching (Kuhn 1955) of the steps that enforces a one-to-one mapping. We note, however, that in some cases, one step in the golden graph (e.g., *add salt and pepper*) may correspond to more than one steps in the generated graph (e.g., *add salt* and *add pepper*) and vice versa, in which case a one-to-one mapping may be too restrictive. To account for this, we also report a relaxed version of Hungarian matching that allows a one-to-two mapping[3].

Once the precision and recall are computed using (relaxed) Hungarian matching, we compute the $F1$ score and report it. We note that some of the generated steps that do not appear in the golden steps may still be good steps (e.g. because they provide more detail that is not in the golden graph). For this reason, recall might be a more informative metric than precision. To account for this, we also report an $F2$ score where we weigh recall twice as much as precision.

Following previous work (Zhang et al. 2021; Madaan et al. 2022), we also concatenate the steps for each task and create a single document, and then report Rouge (F1 and F2) scores for these documents.

Edge Metrics To compare the generated edges with those of the ground truth graph, we first match the nodes from the generated graph to the nodes from the golden graph using Hungarian matching. If a node in one graph does not have a match in the other graph, then we connect it to a dummy singleton node. Then, for each pair of matched nodes (v_i, u_j), let P_i and C_i be the parents and children of v_i respectively, and P_j, C_j be the parents and children of u_j respectively. We measure the amount of overlap between P_i and P_j as well as the amount of overlap between C_i and C_j, both computed in terms of Rouge score. Then we report two metrics: 1- **In-Degree:** the average overlap between P_i and P_j over all matched pairs of steps, 2- **Out-Degree:** the average overlap between C_i and C_j over all matched pairs of steps. Intuitively, by measuring the overlap between P_i and P_j, we measure the amount of overlap between the (immediate) preconditions of the two matched nodes. Moreover, by measuring the overlap between C_i and C_j, we measure the amount of overlap between the steps that become executable (immediately) after we execute the matched nodes.

Furthermore, we also report another metric, which we term **step proximity**, computed as the average overlap between $P_i \cup C_i$ and $P_j \cup C_j$. Note that this metric does *not* evaluate the order of the temporal dependencies; instead, it evaluates whether the steps that should be done in close proximity to each other are indeed placed close to each other in the generated graph.

Task Decomposition Results

We compare against a number of baselines outlined below.

Repeat Task: This baseline repeats the task m times; m is a hyperparameter that we tune on the validation set.

Repeat Similar: This baseline works similarly as the previous baseline, but for the i-th step, we randomly select one of the (non-stop word) tokens in the step and replace it with a semantically similar token. The similarity is computed based on GloVe embeddings (Pennington, Socher, and Manning

[3]We could also report more relaxed versions such as one-to-three, but we observe that the cases where a single step corresponds to more than two steps are rare.

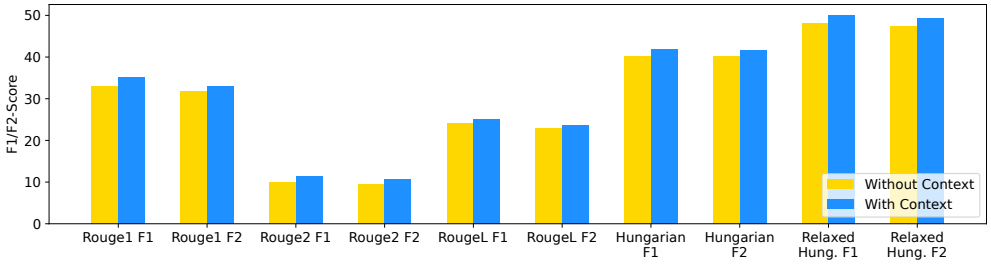

Figure 2: The model performance with and without the context provided as input. When the context is provided to the model, it performs better on all metrics.

2014) of the tokens. The rationale for this substitution is that those words are likely to appear in the steps. For example, for the task *make a smoothie*, some of the most similar words to *smoothie* include *yogurt*, *juice*, *strawberry* and *granola* which are likely to appear in the steps of the task.

Search Query Co-occurrence: We cluster a large set of queries from a commercial search engine by placing near-duplicate queries into the same cluster. Then, inspired by Awadallah et al. (2014), for a given task T we find the cluster $\mathcal{C}_t$ that is most similar to T and we rank the other clusters based on the co-occurrence of their queries with those of $\mathcal{C}_t$ and pick the top k clusters. We select a representative query from each of these cluster to serve as a step for the task T. The hyperparameter k is tuned on the validation set.

Search Query Hierarchy: We perform a second level of clustering on top of the clustering from the previous baseline, where we aim to put similar-intent queries into the same cluster. Then, inspired by Mehrotra and Yilmaz (2017), for a given task T we find the clusters $\mathcal{C}_{L1}$ and $\mathcal{C}_{L2}$ from our level one and level two clustering (note that $\mathcal{C}_{L1}$ is a child of $\mathcal{C}_{L2}$) where T should belong, and then obtain steps by selecting a query from each of the top k sibling clusters of $\mathcal{C}_{L1}$. Here, k is a hyperparameter that is tuned on the validation set.

Results: According to the results in Table 2, even the ICL approach significantly outperforms the other baselines on all metrics. For example, we observe a relative improvement of 280% over the best baseline in terms of Rouge2 F2-Score, and 15% in terms of Hungarian matching F1-score. This establishes LLMs as a powerful source for extracting information about task steps. Our solutions further improve upon the ICL results. In particular, we observe that both S&F and SPT result in improvements compared to ICL across various metrics, with SPT providing more improvement compared to S&F. The *MultSeq* method brings improvement mostly for the F2-scores, due to having higher recall, showing that the steps generated in multiple LLM calls could be complementary as the combination of them improves recall, and hence the F2-score. The approaches that mix two solutions perform better than the individual approaches in isolation in many cases. Among these approaches, *MultSeq + SPT* works best in terms of the F2-score and *S&F+SPT* performs best in terms of F1-score. The combination of all three solutions also works well, but is often dominated by one of the approaches that combines two solutions.

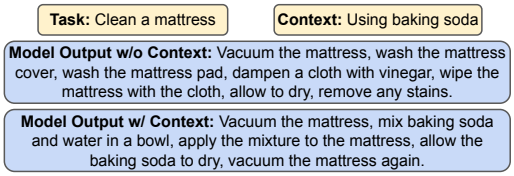

Figure 3: An example of model outputs for a task with and without the context provided as input to the model.

Context Understanding Results

The steps for completing a complex task can be different depending on the context. For example, *recover deleted photos* has different steps depending on the device. We measure the ability of LLMs in providing contextualized steps for a task.

Recall that in our dataset, the steps for a task are written under certain assumptions that provide the context. To measure how well LLMs can providing contextualized steps for a task, we compare their performance with and without the assumption/context being provided to them. We test both settings with ICL: in one case we only provide the task and the steps and in the other we provide the task, assumptions/-context, and the steps.

According to Figure 2, when we provide extra context, the model performs better across all metrics. This shows that LLMs are able to adjust the steps based on the context for a task. In Figure 3, we provide an example model output with and without the context provided to the model. We can see that when no context is provided to the model, the model either provides general steps or selects a specific approach (using vinegar), but when the context is provided the model provides steps that are more specific to the context.

Temporal Dependency Results

We next verify how well LLMs can predict the temporal dependencies between the steps of the tasks (i.e. the edges of the task graphs). Initially, we compare the ICL, ICL+CoT, and SPT approaches on the golden nodes and edges (note that the other two edge prediction approaches are only applicable to generated graphs). In this case, for each pair of nodes in each of the golden task graphs in the test we ask the model to predict if one step should be done before the other one and report the accuracy. The results are reported

-	-	In-Degree			Out-Degree			Step Proximity		
Model	#Seqs	Rouge1	Rouge2	RougeL	Rouge1	Rouge2	RougeL	Rouge1	Rouge2	RougeL
Linear Order	1	18.3	8.8	**17.7**	17.3	7.5	**16.7**	**20.2**	**5.9**	**18.2**
ICL	1	13.6	8.0	13.3	13.6	7.6	13.3	15.2	4.0	13.8
ICL with CoT	1	14.0	6.4	13.5	13.7	5.9	13.2	18.2	4.8	16.2
SPT	1	18.0	**8.9**	17.3	**17.4**	**8.2**	**16.7**	19.9	5.6	17.8
SPT + Linear	1	18.1	**8.9**	17.4	17.1	8.1	16.4	20.0	5.7	17.9
LLM Scoring	1	**18.4**	8.8	**17.7**	17.2	7.5	16.6	**20.2**	**5.9**	**18.2**
ICL	2	10.3	4.4	10.0	10.1	4.2	9.8	13.6	3.5	12.3
ICL with CoT	2	10.8	3.8	10.3	10.5	3.9	10.0	14.4	3.7	12.8
SPT	2	**13.1**	**5.8**	**12.7**	**12.8**	**5.5**	**12.3**	**15.1**	**4.2**	**13.5**

Table 3: The performance of different models for edge evaluation on the generated graphs (#seqs=1 corresponds to the SPT model and #seqs=2 to the SPT+MultSeq model). The winner is in bold. ICL stands for In-Context Learning, CoT for Chain-of-Thought, and SPT for Soft Prompt Tuning.

Model	Accuracy
Majority Class	53.8
ICL	47.5
ICL with CoT	49.6
SPT	78.6

Table 4: The performance of different models for edge evaluation on the golden graphs. The *Majority Class* baseline always predicts no dependency.

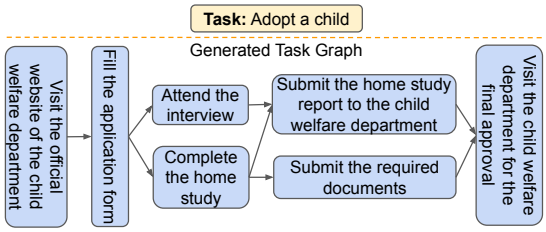

Figure 4: An example of an LLM-generated task graph.

in Table 4. According to the results, both ICL and ICL+CoT perform quite poorly, but the performance improves massively after soft-prompt tuning, thus showing that temporal dependency understanding does not surface out of the box from LLMs but requires tuning on some data.

We next evaluate the performance of the approaches on the generated graphs. To this end, we conduct two experiments in one case we fix the generated steps to those of the SPT model (i.e. there is only one sequence of steps) and in the second we fix the generated steps to those of the SPT+MultSeq model (i.e. there are multiple sequences of steps). We then use the aforementioned approaches to decide the links. The results are reported in Table 3. According to the results, in the case where we use only one sequence, we observe that the linear order produced by the LLM is quite a strong baseline: it outperforms the other models for step proximity (except for LLM Scoring where the two models are on-par) and only slightly underperforms in terms of in-degree and out-degree metrics. We also tried a version of the SPT where we combine it with the linear order of the LLM by making the following assumption: if the LLM produced step A before step B, then we assume either A should be done before B, or A and B can be done in any order (i.e. we rule out the possibility that B should be done before A). Even in this case, we observe that the linear model alone still gives a better performance. Our results show that while LLMs are good at generating the sequence of steps for a task in the right order, they are not particularly good at individually deciding which step of a task should be done before the other or if two steps can be done in any order.

Qualitative Analysis

In Figure 4, we demonstrate an LLM-generated task graph. We can see that many of the steps and the temporal dependencies are sensible; however, there may also be some problems present. For example, one probably needs to complete the home study before attending the interview. More qualitative examples are provided in the Appendix.

Conclusion

We studied the power of large language models (LLMs) for structured complex task decomposition (SCTD), i.e. the problem of decomposing a complex real-world task into multiple steps and determining the temporal dependencies between the steps. We created a probe named TaskLAMA, developed metrics for measuring the performance, tested various task decomposition and temporal dependency prediction models and compared against baselines. Our results indicate that LLMs are strong in task decomposition. For predicting temporal dependencies, however, while they are able to produce a good sequence of steps with the right order of steps, their ability in predicting pairwise temporal dependencies still lags behind. Future work can find ways of improving the temporal dependency understanding of LLMs, or develop new approaches to improve LLM-based SCTD, e.g. by generating the entire task graph at once (see Sakaguchi et al. (2021); Madaan et al. (2022)) or by breaking the complex tasks into simpler tasks and then solving those simpler task (see Zhou et al. (2022a); Kazemi et al. (2023b)).

Acknowledgements

We thank Sid Mittal, Deepti Bhatia, Amr Ahmed, and Tania Bedrax-Weiss for their valuable feedback.

References

Allen, D.; and Peikoff, A. 2001. Getting Things Done: The Art of Stress-Free Productivity.

Awadallah, A. H.; White, R. W.; Pantel, P.; Dumais, S. T.; and Wang, Y.-M. 2014. Supporting complex search tasks. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, 829–838.

Brahman, F.; Bhagavatula, C.; Pyatkin, V.; Hwang, J. D.; Li, X. L.; Arai, H. J.; Sanyal, S.; Sakaguchi, K.; Ren, X.; and Choi, Y. 2023. PlaSma: Making Small Language Models Better Procedural Knowledge Models for (Counterfactual) Planning. *arXiv preprint arXiv:2305.19472*.

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

Cer, D.; Yang, Y.; Kong, S.-y.; Hua, N.; Limtiaco, N.; John, R. S.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Cheng, J.; Teevan, J.; Iqbal, S. T.; and Bernstein, M. S. 2015. Break it down: A comparison of macro-and microtasks. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 4061–4064.

Chilton, L. B.; Little, G.; Edge, D.; Weld, D. S.; and Landay, J. A. 2013. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1999–2008.

Davison, J.; Feldman, J.; and Rush, A. M. 2019. Commonsense knowledge mining from pretrained models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 1173–1178.

Erol, K. 1995. *Hierarchical task network planning: formalization, analysis, and implementation*. University of Maryland, College Park.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. *arXiv preprint arXiv:2305.14909*.

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

Humphreys, T. 2019. Exploring HTN planners through example. *Game AI Pro*, 360: 103–22.

Jiang, Z.; Xu, F. F.; Araki, J.; and Neubig, G. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8: 423–438.

Kazemi, M.; Yuan, Q.; Bhatia, D.; Kim, N.; Xu, X.; Imbrasaite, V.; and Ramachandran, D. 2023a. BoardgameQA: A Dataset for Natural Language Reasoning with Contradictory Information. *arXiv preprint arXiv:2306.07934*.

Kazemi, S. M.; Kim, N.; Bhatia, D.; Xu, X.; and Ramachandran, D. 2023b. Lambada: Backward chaining for automated reasoning in natural language. In *ACL*.

Kokkalis, N.; Köhn, T.; Huebner, J.; Lee, M.; Schulze, F.; and Klemmer, S. R. 2013. Taskgenies: Automatically providing action plans helps people complete tasks. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 20(5): 1–25.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2): 83–97.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Lin, B. Y.; Lee, S.; Khanna, R.; and Ren, X. 2020. Birds have four legs?! numersense: Probing numerical commonsense knowledge of pre-trained language models. *arXiv preprint arXiv:2005.00683*.

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

Mehrotra, R.; and Yilmaz, E. 2017. Extracting hierarchies of search tasks & subtasks via a bayesian nonparametric approach. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, 285–294.

Newell, A.; Simon, H. A.; et al. 1972. *Human problem solving*, volume 104. Prentice-hall Englewood Cliffs, NJ.

Nye, M.; Andreassen, A. J.; Gur-Ari, G.; Michalewski, H.; Austin, J.; Bieber, D.; Dohan, D.; Lewkowycz, A.; Bosma, M.; Luan, D.; et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Petroni, F.; Rocktäschel, T.; Lewis, P.; Bakhtin, A.; Wu, Y.; Miller, A. H.; and Riedel, S. 2019. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*.

Sakaguchi, K.; Bhagavatula, C.; Bras, R. L.; Tandon, N.; Clark, P.; and Choi, Y. 2021. proscript: Partially ordered scripts generation via pre-trained language models. *arXiv preprint arXiv:2104.08251*.

Sung, M.; Lee, J.; Yi, S.; Jeon, M.; Kim, S.; and Kang, J. 2021. Can Language Models be Biomedical Knowledge Bases? *arXiv preprint arXiv:2109.07154*.

Teevan, J.; Iqbal, S. T.; and Von Veh, C. 2016. Supporting collaborative writing with microtasks. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, 2657–2668.

Wang, Z.; Zhang, H.; Fang, T.; Song, Y.; Wong, G. Y.; and See, S. 2022. SubeventWriter: Iterative Sub-event Sequence Generation with Coherence Controller. *arXiv preprint arXiv:2210.06694*.

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

West, P.; Bhagavatula, C.; Hessel, J.; Hwang, J. D.; Jiang, L.; Bras, R. L.; Lu, X.; Welleck, S.; and Choi, Y. 2021. Symbolic knowledge distillation: from general language models to commonsense models. *arXiv preprint arXiv:2110.07178*.

Yin, D.; Bansal, H.; Monajatipoor, M.; Li, L. H.; and Chang, K.-W. 2022. GeoMLAMA: Geo-Diverse Commonsense Probing on Multilingual Pre-Trained Language Models. *arXiv preprint arXiv:2205.12247*.

Zhang, L.; Lyu, Q.; and Callison-Burch, C. 2020. Reasoning about goals, steps, and temporal ordering with WikiHow. *arXiv preprint arXiv:2009.07690*.

Zhang, X.; Ramachandran, D.; Tenney, I.; Elazar, Y.; and Roth, D. 2020. Do language embeddings capture scales? *arXiv preprint arXiv:2010.05345*.

Zhang, Y.; Jauhar, S. K.; Kiseleva, J.; White, R.; and Roth, D. 2021. Learning to decompose and organize complex tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2726–2735.

Zhang, Y.; Zhang, M.; Liu, Y.; Tat-Seng, C.; Zhang, Y.; and Ma, S. 2015. Task-based recommendation on a web-scale. In *2015 IEEE International Conference on Big Data (Big Data)*, 827–836. IEEE.

Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; et al. 2022a. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Zhou, S.; Zhang, L.; Yang, Y.; Lyu, Q.; Yin, P.; Callison-Burch, C.; and Neubig, G. 2022b. Show me more details: Discovering hierarchies of procedures from semi-structured web data. *arXiv preprint arXiv:2203.07264*.

Zhou, X.; Zhang, Y.; Cui, L.; and Huang, D. 2020. Evaluating commonsense in pre-trained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9733–9740.