Learning to Decompose and Organize Complex Tasks

Yi Zhang[†,*] **Sujay Kumar Jauhar**[§], **Julia Kiseleva**[§],
Ryen W. White[§] and **Dan Roth**[†]
[†]University of Pennsylvania, Philadelphia, PA, USA
[§]Microsoft Research, Redmond, WA, USA
[†]{yizhang5, danroth}@cis.upenn.edu, [§]{sjauhar, jukisele, ryenw}@microsoft.com

Abstract

People rely on digital task management tools, such as email or to-do apps, to manage their tasks. Some of these tasks are large and complex, leading to action paralysis and feelings of being overwhelmed on the part of the user. The micro-productivity literature has shown that such tasks could benefit from being decomposed and organized, in order to reduce user cognitive load. Thus in this paper, we propose a novel end-to-end pipeline that consumes a complex task and induces a dependency graph from unstructured text to represent sub-tasks and their relationships. Our solution first finds nodes for sub-tasks from multiple 'how-to' articles on the web by injecting a neural text generator with three key desiderata – relevance, abstraction, and consensus. Then we resolve and infer edges between these subtask nodes by learning task dependency relations. We collect a new dataset of complex tasks with their sub-task graph to develop and evaluate our solutions. Both components of our graph induction solution are evaluated in experiments, demonstrating that our models outperform a state-of-the-art text generator significantly. Our generalizable and scalable end-to-end solution has important implications for boosting user productivity and assisting with digital task management.

1 Introduction

People today increasingly rely on digital modalities and applications to organize, track and complete tasks from their work and life. They depend on email to structure their communications as a way of tracking pending tasks (Bellotti et al., 2003), issue commands to their digital assistants [1] for timely task reminders (Brewer et al., 2017), and use task management applications (Bellotti et al., 2004) [2] to organize grocery runs, thesis writing and everything in between.

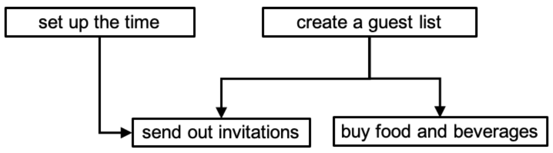

Figure 1: The subtask graph of *"plan a birthday party"*, where nodes are subtasks, and the directed edges represent the dependencies between subtasks.

In this work, we focus on tasks that are complex (Hassan Awadallah et al., 2014), and – which research in micro-productivity has shown (Kirsh, 2000; Teevan et al., 2016a) – may benefit from thoughtful organization. For example, consider Figure 1, where we show how the complex task *"plan a birthday party"*[3] can be broken down into more manageable pieces and structured by mutual temporal dependencies, in order to create an actionable plan that is simpler and more effective.

In this paper we propose to help automate generating such actionable plans in order to reduce cognitive load on users. While prior research (Cheng et al., 2015; Teevan et al., 2016b) has shown the benefits of tracking and acting on micro-tasks, little effort has been expended on finding automated solutions for actually breaking down complex tasks into tractable sub-tasks. Thus we design a novel end-to-end solution that is capable of decomposing complex tasks and structuring sub-task dependencies.

We model our end-to-end solution as a graph induction problem, in which we first find nodes to represent sub-tasks, then infer the temporal dependency edges between them, yielding a flow diagram like the one in Figure 1. All of this is done from unstructured text ubiquitously found on the web, making our approach general and scalable.

[*] Most of this work was done while the first author was an intern at Microsoft Research.

[1] Siri, Cortana, Google Assistant etc.

[2] Google Keep, Todoist, Microsoft To Do etc.

[3] Contrast this with a simple task such as *"send the PPT deck to Sarah"*.

In the first component (that of finding nodes), we learn to synthesize information from multiple 'how-to' articles across the web and generate text fragments for sub-tasks. In particular, we extend a state-of-the-art neural text generator by injecting it with three desiderata for these fragments: **relevance** (to the complex task), **abstraction** (by summarizing content in articles), and **consensus** (for appearing across multiple sources). In the second component (that of finding edges), we infer temporal dependencies between sub-tasks.

Existing corpora of 'how-to' articles (most notably WikiHow (Koupaee and Wang, 2018)) do not contain this latent dependency structure. Moreover, articles in these corpora are structured and formatted consistently and uniformly, making them ill-suited to our approach, which seeks to synthesize the content of multiple heterogeneous web pages. We therefore devise a simple annotation framework through which we gather a new dataset of complex tasks, and their associated subtasks and mutual dependencies, from multiple 'how-to' web articles using non-expert crowd workers. Finally, we use this data to fine-tune our augmented neural text generator, as well as predict dependency edges between the sub-tasks it generates.

In experiments, we demonstrate that our optimal solution – which encodes relevance, abstraction and consensus – yields significant improvements over a state-of-the-art text generator on both sub-task generation and dependency prediction.

The focus of this paper is on Complex Tasks; however, our research has impact beyond intelligent task management. For example, learning to decompose complex natural language expressions could have impact on complex question answering (Chali et al., 2009; Luo et al., 2018), where question decomposition, multi-hop reasoning, information synthesis, and implicit knowledge all play an important role. More generally, the ability to model mappings between short text fragments and elements in multiple documents could benefit research in areas such as topic-focused multi-document summarization (Wan et al., 2007) and event timeline extraction of evolving news stories (Do et al., 2012).

In summary, our key contributions are (i) building an end-to-end pipeline for complex task decomposition as a graph induction problem from unstructured text; (ii) constructing a new dataset for complex tasks that contain sub-tasks as well as the temporal dependencies between them; and (iii) extending a neural text generator by injecting signals for relevance, abstraction and consensus, thereby making it more capable at tackling task decomposition.

2 Problem Definition

We begin by defining some key concepts. We refer to a **task** as a text fragment that represents a goal people want to track, remind themselves of, or learn how to do; for example, *"buy a Christmas present"*, *"eat healthier"* or *"change a tire"*. In order to disambiguate the intent of tasks (consider the fragment *"Harry Potter"*, which could equally refer to *"read [the book]"* or *"watch [the movie]"*), we scope our investigation to tasks that contain at least one verb.

A task is considered as a **complex task** when two or more individual steps – themselves also worth tracking, remembering or learning how to do – need to be performed in its completion. Therefore, *"plan a birthday party"*, which involves *creating a guest list* and *buying food and beverages* (see Figure 1) is a complex task. While *"throw out the trash"* is not such a complex task, even though it may involve *opening the front door* and *walking to the trash bins*. We refer to the individual steps of a complex task as **sub-tasks**.

Sub-tasks may sometimes depend on other sub-tasks being completed before they can be tackled. Consider the example from Figure 1 again, which illustrates how one must "set up a time" and "make a guest list" before "send(ing) out invitations". We refer to these relations as **temporal dependencies**, and are pair-wise notated as sub-task B depending on ($\leftarrow$) A.

Given these key concepts, we define a **complex task graph** as follows. Let the sub-tasks of a complex task t be denoted by $ST(t) = \{s_i\}_{i=1}^{n}$. Then define $G_s(t) = (V, E)$ as the complex task graph of t. Here $G_s(t)$ is a directed graph, where $V = ST(t)$ is the set of sub-task nodes, and E represents the set of temporal dependency edges such that $(s_i, s_j) \in E, s_i \leftarrow s_j$.

2.1 Modeling Overview

Given these definitions, the problem of decomposing and organizing a complex task becomes inducing a graph $G_s(t) = (V, E)$ from a complex task input t[4]. To construct the graph, the key steps are (1) generating the sub-task nodes V, and (2) inferring the temporal dependency edges E between nodes. We propose to do both from unstructured text.

In particular, the web has made a large number of instructional texts on a variety of topics and activities freely available for public consumption; some of them are in purpose-built websites, such as WikiHow (Koupaee and Wang, 2018), while others appear in personal blogs, social fora, educational portals and a number of other heterogeneous sources. We leverage these resources to find relevant information for complex tasks. Specifically, given a task t, we query a search engine with the term "how to t" ("plan a birthday party" becomes "how to plan a birthday party"), and store the k most relevant results in a collection $D_k(t)$. Our graph induction problem then becomes finding the optimal graph $G_s(t) = (V, E)$ given the evidence in $D_k(t)$. We elaborate on solutions for node generation and edge inference in what follows.

Sub-task Generation Formally, given a complex task t and a collection of relevant articles $D_k(t)$, we attempt to generate the sub-tasks $ST(t)$. We argue that the text fragments for sub-task nodes we generate must satisfy three desiderata:

(1) **Relevance**, so that generated sub-tasks are directly related to the complex task t.

(2) **Abstraction**, because 'how-to' articles often explain and expand on sub-tasks.

(3) **Consensus**, since sub-tasks that are cited across multiple sources are more likely to be important.

Our model for sub-task generation builds on BART (Lewis et al., 2019), a state-of-the-art sequence-to-sequence model for text generation, and injects it with our three desiderata. Concretely, we make BART capable of handling multi-source input, design a custom *relevance-aware cross attention* layer and implement a *cluster encoding* technique to guide the generation process. Details for the model are presented in Section 4.1.

[4]We assume complex tasks as input, since the focus of our work is on their understanding and decomposition. We leave the problem of distinguishing complex tasks from simple ones to future work.

Dependency Inference Given the generated set of sub-tasks $V = ST(t)$, the next step in our end-to-end graph induction solution consists of inferring the temporal dependency edges E. We formulate this as a binary classification problem[5], where we attempt to predict the existence of a dependency edge $\forall(s_i, s_j) \in V$.

Specifically, we use the concatenated intermediate representations for sub-tasks (s_i, s_j) from our enhanced BART model and add a final linear layer to learn a binary classifier. We train this classifier on a new dataset of complex tasks that contains pairwise temporal dependency information (see Section 3). More details are given in Section 4.2.

3 Data Collection

To build and evaluate our solutions, we need data. The most relevant existing dataset is WikiHow (Koupaee and Wang, 2018), which is derived from the popular how-to website. However WikiHow, while very useful for parts of our modeling paradigm, is ill-suited to others. Namely, its articles are manually curated, with consistent structure and format, making them a mis-match to the heterogeneous, noisy and free-form articles we expect to encounter on the web. Moreover, they contain no dependency information between sub-tasks beyond a simple numbered ordering.

Thus, to support our problem, we need a dataset which (1) contains complex task and their sub–tasks; (2) encodes dependencies between sub-tasks; and (3) the sub-tasks come from a variety of webpages. Note that our goal is to create a dataset that enables model generalization, rather than constructing a comprehensive knowledge base. Therefore, rather than exhaustively annotating sub-tasks and their dependencies, we seek only to gather labels for the most important ones. In what follows we will describe the step-by-step construction of our dataset, and how these steps encode our three fundamental desiderata for task-sub-task relationships (see Section 2).

The dataset we collect, which we call **MSComplexTasks** [6] is being released freely to the research community.

[5]While edge prediction is technically a structured prediction problem, we demonstrate in this paper that even a simple approach works well; we leave more complex modelling solutions to future work.

[6]`https://github.com/microsoft/MSComplexTasks`

Collecting Complex Tasks and How-to Articles We begin with logs from the popular – now defunct – task management application Wunderlist[7]. These logs are privately and respectfully handled by passing them through an enterprise grade, legal- and trust-approved pipeline, which anonymizes, aggregates and scrubs all personally identifiable information. This yields a collection of task strings, some of which have associated sub-task *metadata* (not sub-tasks themselves). We retain those tasks which have at least one sub-task, and contain at least one verb [8] (to avoid issues with disambiguating task intent) as a candidate seed pool of complex tasks. It may be noted that while the logs are not publicly available, they play a minimal role in our end-to-end solution. Their only purpose is to seed the initial set of complex how-to queries.

In order to find relevant articles for each complex task we trawl through a month's worth of logs from a commercial search engine using the 'how-to' query expansion described in Section 2.1. Further, in order to protect user privacy we discard queries that were issued by fewer than 5 distinct users. To each remaining complex task query, we associate the top-10 clicked URLs across all users for the entire month. Text in these webpages satisfy our notion of **relevance**.

Finding Candidate Sentences for Sub-tasks Next, we create a pool of candidate sentences that we hypothesize might contain sub-tasks. Specifically, we use an in-house webpage parser to extract section headings and list items from the set of URLs previously collected. These types of text fragments often represent very short summaries that are then elaborated on in 'how-to' articles; we thus attempt to restrict our candidate pool to sub-tasks that capture the notion of **abstraction**.

Finally, we also care about **consensus** across articles, since this allows us to retain only those sub-task which are cited in different sources and are therefore more reliably important. Because the same sub-task can be expressed differently in text we perform clustering on the BERT (Devlin et al., 2018) embeddings of candidate sentences and discard those clusters that only contain a single source URL. The remaining set of sentences form our pool of candidate sub-tasks.

[7]`https://en.wikipedia.org/wiki/Wunderlist`

[8]Detected using `https://spacy.io/`.

Labeling Sub-tasks and Dependencies Given the set of candidate complex task queries and their associated sub-task sentences, we ask crowd-workers to label them. Specifically, we guide annotators through a series of questions: (1) Is the candidate query about a task? A complex task? (2) If it is a complex task, which candidate sentences represent sub-tasks? (3) Does the ordering of sub-tasks matter? If so, assign pairwise temporal dependency labels to sub-tasks. We ask three workers to label each HIT, aggregating annotations by majority vote. Table 1 shows some examples of aggregate judgments from our annotation study.

4 Models

Recall from Section 2 that we model complex task decomposition as a graph induction problem over unstructured text. In what follows we will first describe our approach for sub-task node construction, followed by our method for sub-task temporal dependency inference.

4.1 Sub-task Construction

As described in Section 2, we treat sub-task finding as a text *generation* problem. While we could ostensibly frame it as a span prediction problem, this is unsuitable for our modeling paradigm. First, our multi-source setting means that we might potentially (and in fact want to) extract more than one text span referring to the same sub-task. Thus resolving identical sub-tasks and picking the best among them would require additional logic, as well as a sub-task coreference module. Moreover, while we could use a webpage parser to build a candidate pool of sub-task text spans (see Section 3), such a parser might be brittle and error prone – or even non-existent. While we have human annotators to refine this pool during dataset construction, no such remedy exists at automatic inference time.

Model Architecture Our model is based on the pre-trained text generation model BART (Lewis et al., 2019). This is a sequence-to-sequence neural summarizer, which consists of a bidirectional encoder over corrupted text and a left-to-right autoregressive decoder. To extend it to a multi-source setting, we encode each source article independently with BART's encoder (a bi-directional Transformer (Vaswani et al., 2017)), and then concatenate the output embeddings of the encoders. These are then fed to the decoder which generates the output sentences autoregressively. We treat all of

	Task?	Complex Task?	Sub-tasks	Ordering Matters?	Dependencies
apply for a green card	Yes	Yes	find out if you re eligible prepare for your interview ...	Yes	prepare for your interview $\leftarrow$ find out if you re eligible
start a grocery store	Yes	Yes	form your LLC finding the best location marketing your grocery store ...	Yes	marketing your grocery store $\leftarrow$ finding the best location

Table 1: Examples of results from the annotation

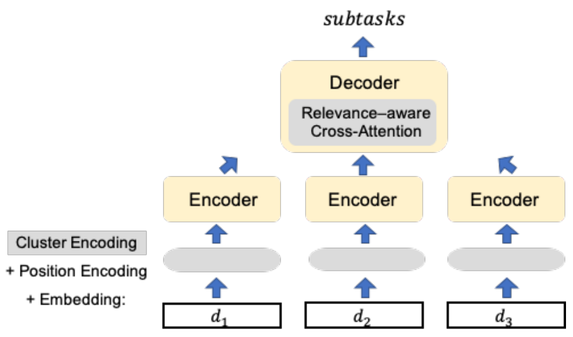

Figure 2: The model architecture of our solution. We extend BART to handle multiple documents (d_1, d_2, d_3) as input, design the relevance-aware cross attention layer and cluster encoding techniques to guide the generation process.

the subtasks of a given complex task as a single target document, and subtasks are thus generated as a sequence of text fragments. A diagram of our architecture is shown in Figure 2. We initialize our model using the parameters of the pre-trained BART-large model released with the HuggingFace Transformer library (Wolf et al., 2019). These parameters are then fine-tuned to our task, using the proposed architecture. In our work, the inputs are the textual content from URLs returned by a 'how-to' complex task query, and the outputs are the set of generated sub-tasks. Because we are learning mappings between 'how-to' articles and short text fragments that summarize their contents, we are implicitly learning the notion of **abstraction**.

Relevance-Aware Cross-Attention As discussed in Section 2, sub-tasks need to also be relevant to complex tasks. In order to encode **relevance** we design a cross-attention mechanism that explicitly captures textual relevance. Specifically, we score each sentence in a set of articles based on its relevance to the complex task, then propose a general mechanism to inject this information into the text generation model.

Given a complex task t, we denote the query expansion "how to do t" as q, and the collection of related articles as $D_k(q)$. To score the relevance of each sentence $s \in D_k(t)$, we consider two factors. The first is how relevant the sentence s is to the query q, and the other is how relevant the article d (with $s \in d$) is to q. We denote the relevance of s as $P(s|q)$, and the relevance of d as $P(d|q)$. To compute $P(d|q)$, we first represent q and d as n-dimensional embedding vectors respectively using BART, denoted respectively as $\vec{q}$ and $\vec{d}$. Then article-query relevance is computed as the softmax:

$$P(d|q) = \frac{\exp f(d,q)}{\sum_{d' \in D_k(q)} \exp f(d',q)} \qquad (1)$$

where $f(d,q) = \vec{q}^T \cdot \vec{d}$.

The sentence-query relevance $P(s|q)$ is similarly computed using the softmax over embeddings $\vec{q}$ and $\vec{s}$. Notably, these embeddings are generated from a BART model fine-tuned on an auxiliary sequence classification task, where positive samples are sub-tasks of complex task t_i and negative samples are randomly samples sub-tasks from other complex tasks $t_{j(j \neq i)}$.

Given $P(s|q)$ and $P(d|q)$, we need a way of injecting them into our generation model. The BART decoder performs cross-attention over the final hidden layer of the encoder to determine how much focus to place on other parts of the input sentence, as words in specific positions are encoded. Our model should not only pay attention to the inputs, but should additionally pay more attention to those are more relevant. We therefore define our relevance-aware cross-attention as follows:

$$\sigma\left(\frac{Q(x)K(w)^T + \alpha p(s|q) * p(d|q)}{\sqrt{d_k}}\right) V \qquad (2)$$

where $\sigma(\cdot)$ represents the softmax function, x is a token in the output sub-task, $w \in s \in d$ is a token in the input article, Q, K and V are query, key and value representations, d_k is the dimension of the key vector, and α is a learnable parameter which controls the importance of our relevance injection.

Cluster Encoding The final desiderata for our model is **consensus**, or the ability to be able to recognize and reward sub-tasks that are mentioned across multiple articles. To encode this signal we create a *cluster embedding*, which identically represents sentences that refer to the same sub-task across sources.

To generate the new cluster embeddings, we first embed the query q as well as the set of sentences $s \in D_k(t)$ into n-dimension vectors using BART. Then we cluster the embeddings of all sentences s with KMeans, ranking the clusters by the proximity of their centroid to the embedding of q. Finally, using a formulation similar to the positional encodings from Vaswani et al. (2017) we define cluster encoding as:

$$\begin{aligned} PE^c_{(s,2j)} &= \sin\left(\frac{r_i}{10000^{2j/d_{model}}}\right) \\ PE^c_{(s,2j+1)} &= \cos\left(\frac{r_i}{10000^{2j/d_{model}}}\right) \end{aligned} \tag{3}$$

where r_i is the rank of the cluster that s belongs to, and the symbols j s.t. $1 \leq j \leq n$ are dimensional indices; specifically $2j$, and $2j+1$ represent indices into the clusters' embeddings. For instance, if the cluster embedding were defined as a vector of length 512, then $2j$ (resp. $2j+1$) represent the 0th, 2nd, 4th, ..., 510th (resp. 1st, 3rd, 5th, ..., 511th) index positions of the vector. These notations are identical to the ones used in the original by Vaswani et al. (2017) in their definition of positional encoding.

This formulation allows our model to identify tokens that belong to similar sentences, and are injected into the extended BART model as an additional input.

4.2 Dependency Inference

In our work, we treat inferring dependencies between sub-tasks as a binary classification problem. Specifically, we learn a classifier capable of predicting the existence (or not) of a temporal dependence $s_i \leftarrow s_j$ between all possible ordered pairs of generated sub-tasks $(s_i, s_j) \in E$.

We use the same extended BART architecture previously proposed in Section 4.1, and concatenate the intermediate representations for s_i and s_j to yield an output representation for a pair of sub-tasks. Then we add a linear layer on top of this output to predict a binary label for the existence of a temporal dependency edge.

Similar to sub-task generation, we hypothesize that finding consensus across articles may also be helpful in determining the dependencies between sub-tasks. Therefore, we also use cluster encodings in infering edge dependencies.

5 Experimental Evaluations

In this section we aim to answer the following research questions:

RQ1 Can we accurately and automatically generate sub-tasks given an input complex task?

RQ2 Can we correctly identify the temporal dependencies between sub-tasks?

We begin by describing datasets, baselines and metrics used in our evaluation. [9]

5.1 Evaluation Methodology

In Section 3, we introduced a new Complex Task Dataset (CTD). In addition to this data, we also develop a new variant of WikiHow (Koupaee and Wang, 2018) dataset better suited to our modeling paradigm (which we call WKH-R), for larger scale development and evaluation. We describe both in what follows.

5.1.1 WKH-R

Recall that WikiHow (Koupaee and Wang, 2018) can be interpreted as an abstractive summarization dataset, where the source are the textual content webpage bodies, and section headings are the summaries. In our problem, webpage titles for articles can be treated as complex tasks, and the section headings as subtasks. Even though WikiHow does not encode dependencies between subtasks (other than strict numerical ordering of sections), we could ostensibly exploit the relationship between page titles and section headings to learn to generate sub-tasks from complex tasks. However, as we have previously argued (see Section 2.1), directly using WikiHow articles as the only source for training our model would make it brittle, and prone to fail on free-form content found in other heterogeneous sources on the web. Therefore, we propose an extension to WikiHow, which notably does not require human annotation and is created from information already present in WikiHow.

[9] Resources are organized in `http://cogcomp.org/page/publication_view/939`.

	ROUGE-1		ROUGE-2		ROUGE-L	
	WKH-R	CTD	WKH-R	CTD	WKH-R	CTD
T5	14.22	18.08	2.326	3.978	12.97	17.19
BART	27.61	24.69	6.190	7.223	17.27	18.33
MSBART	31.52	26.11	7.397	9.167	18.46	19.00
MSBART-R	32.74	26.29	7.795	9.458	19.13	19.16
MSBART-C	**33.62**	26.36	7.880	9.816	18.77	19.95
MSBART-F	33.03	**27.29**	**7.947**	**10.30**	**19.27**	**20.85**

Table 2: The performance of generating sub-tasks under the Rouge score metrics

In the extended WKH-R dataset, we do not treat the original WikiHow article as the solitary source. Instead, we conjecture that the WikiHow article is itself written by compiling information from multiple resources, and gather those sources directly instead. Concretely, many WikiHow articles contain a set of references that are cited by authors as sources, and we use these webpages as a collection of multiple sources instead of the content of WikiHow article itself. As references link to a diverse range of URLs on the web, our model learns to be more robust to structural and stylistic variety.

The new extension of WikiHow in this form is fully capable of supporting learning and evaluation of sub-task generation from complex task queries, as modeled with the architecture outlined in Section 4.1. Specifically we use WikiHow page titles as complex tasks, section headings as sub-tasks, and reference webpages in WKH-R as multi-source articles.

In the construction of WKH-R, we only retain articles that have more than one valid reference URL. Overall, we compile a dataset consisting of 7832 webpages, each corresponding to a distinct complex tasks, its own sub-tasks and reference articles. On average, each complex task has 12.9 individual sub-tasks, while citing 2.9 different references. In our experiments, we use 3916 instances for training, 1566 for validation, and set aside the remaining 2350 for testing.

5.1.2 Complex Task Dataset (CTD)

In Section 3, we already described how we created the MSComplexTasks dataset, containing among other signals, the temporal dependency relation between sub-tasks. Here, we briefly summarize some of the characteristics of this dataset. Overall, we collected sub-tasks and their dependencies for 430 complex tasks from an initial candidate pool of 2000 tasks. Many tasks were discarded, either because they were deemed not complex, or no sub-tasks candidates were confirmed by a majority of annotators. While this set may appear small, it may be noted that each instance in the data is a rich, structured object. On average, each complex task has 7.3 sub-tasks, references 2.4 webpages, and encodes 11 pairs of temporal dependencies between tasks. We use 215 instances for training, 86 for validation and 129 for testing in our experimental evaluation.

5.2 Baselines and Metrics

We compare our full modeling approach (see Section 4) against several strong baselines. These include the base **BART** model, – itself a state-of-the-art text generator – which we use as a black-box single-document summarizer since all our variants are build on top of it. We also include a different state-of-the-art text-to-text model, **T5** (Raffel et al., 2019) in our comparison, in order to demonstrate our that our modeling technique not only improves over the base BART variant, but other approaches to text generation in the literature. In terms of our modeling variants we compare against an extension of BART that additionally capture multiple sources (**MSBART**). We also include two variants that inject the MSBART model with our custom relevance and consensus encodings respectively, which yield the MSBART-R and MSBART-C baselines. Finally, we denote our full model, consisting of multiple sources, relevance and consensus as MSBART-F.

In our evaluation of sub-task generation we report Rouge-1, Rouge-2, Rouge-L (Lin, 2004) and pairwise BERTScore (Zhang et al., 2019) metrics to compare performance across models. When reporting Rouge scores, we treat the sub-tasks as a generated summaries, and the reference sub-tasks as the target summaries. In addition to the document level Rouge summarization metric, we also leverage BERTScore to compute a sentence level evaluation number. Specifically, we first compute the best mapping between generated sub-tasks (GS) and the target subtasks (TS) via BERTScore (BS), then report their corresponding precision and recall. The BERTScore based precision and recall are computed as follows:

$$Pr(TS, GS) = \sum_{s \in GS} \frac{\max_{s' \in TS}(BS(s, s'))}{|GS|}$$
$$Rc(TS, GS) = \sum_{s \in TS} \frac{\max_{s' \in GS}(BS(s, s'))}{|TS|} \quad (4)$$

Meanwhile, in our evaluation of dependency inference we compare our full modeling solution MSBART-F against the single-source BART baseline and the multi-source MSBART variant. In this experiment we report accuracy as the sole evaluation metric.

	BS_{Pr}		BS_{Rc}	
	WKH-R	CTD	WKH-R	CTD
T5	87.30	87.64	86.89	84.83
BART	87.40	88.33	87.99	84.74
MSBART	88.41	88.37	88.30	84.82
MSBART-R	88.87	88.56	88.47	84.96
MSBART-C	88.84	88.59	88.37	84.81
MSBART-F	**89.03**	**88.74**	**89.23**	**85.20**

Table 3: The performance of generating sub-tasks under the BERTScore metrics.

5.3 Results

To answer RQ1, we demonstrate the performance of our full modeling solution, MSBART-F, when compared against the set of variant baselines on the problem of generating sub-tasks from a given complex task. In particular we present results on Rouge and pairwise BERTScore metrics for both WKH-R and CTD datasets. The results for Rouge and pairwise BERTScores are summarized in Table 2 and Table 3 respectively. We can observe from the tables that extending the problem setting from single source to multi-source considerably improves performance. Meanwhile injecting signals for relevance and consensus can each further improve upon MSBART, and the full solution achieves the best performance on almost every combination of dataset and evaluation metric (the only exception being Rouge-1 on WKH-R).

Thus in answering **RQ1**, we conclude that the proposed MSBART-F model automatically generates the highest quality sub-tasks when compared against several state-of-the-art variant baselines.

In order to answer RQ2, we report the results of inferring temporal dependencies among sub-tasks, using accuracy as a measure of performance. These results are shown in Figure 3. We observe that MSBART, which leverages information from multiple sources, improves upon the accuracy of the single-source BART model. Meanwhile, our full MSBART-F model achieves the best performance overall.

Thus in response to **RQ2**, we conclude that our proposed MSBART-F model infers dependencies between sub-tasks with an accuracy higher than comparative variants. Notably, the prediction accuracy of 0.779, we believe represents a reasonably strong first attempt at the edge inference component of our graph induction solution.

In answering both RQs 1 and 2, we note particularly that our key insights for injecting our models with the capability for encoding relevance, abstraction and consensus lead to consistently improved results in complex task decomposition and organization.

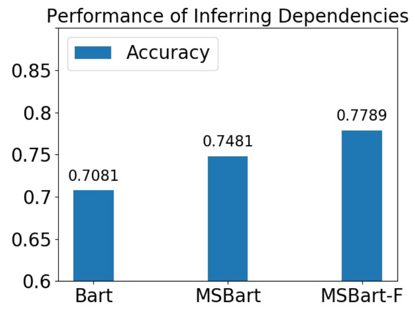

Figure 3: The performance of inferring dependencies between sub-tasks

6 Related Work

Task management applications aim to improve people's productivity by helping *capture, organize, and execute* their daily tasks (Bellotti et al., 2004). Recent advances in natural language understanding techniques (Devlin et al., 2018; Lewis et al., 2019) have sparked rapid progress in facilitating intelligent task organization, beginning with early work on contextual reminders (Kamar and Horvitz, 2011; Graus et al., 2016) to more advanced applications on estimating task duration (White and Hassan Awadallah, 2019), detecting already completed tasks (White et al., 2019), highlighting actionable micro-tasks (White et al., in press), and automatic task extractions from emails (Mukherjee et al., 2020). Meanwhile, task planning remains one of the most challenging and cognitively-demanding activities in task management (Kirsh, 2000). Prior studies have shown that breaking down complex tasks positively influences productivity (Cheng et al., 2015; Teevan et al., 2016a,b). However, to the best of our knowledge, few methods have been proposed to tackle this problem automatically and at scale.

The one exception is Hassan Awadallah et al. (2014) who explore complex search task understanding. Notably, however, their work reasons only over search logs rather than over the unstructured content of webpages. Furthermore, the purpose of their effort is subsequent query recommendation rather than the full complex task decomposition and structuring we propose in this paper.

Thus our work is distinguished from prior research by being the first to attempt automatically decompose and organize complex task from unstructured text, in an end-to-end and scalable manner. One of the primary hurdles for research on complex tasks was the lack of suitable data, particularly with respect to temporal dependency between subtasks. We remedy this by collecting a novel dataset in this paper, which we hope will spur future research in the area.

7 Conclusion and Future Work

In this paper we have tackled the novel problem of decomposing and organizing a complex task from unstructured text. We devised an end-to-end solution that formulated this problem as graph induction in two stages. The first consisted of finding nodes to represent sub-tasks by parsing multiple 'how-to' articles on the web and extracting key text fragments from them. Notably, we framed three desiderata for finding these fragments – relevance, abstraction and consensus – and built a custom neural architecture to encode these properties by extending a state-of-the-art text generation system. In the second stage we designed a crowd-sourcing study to collect a new dataset of complex tasks, – consisting of their sub-tasks and the temporal dependency relations between them – then used this dataset to generate sub-task nodes as well as infer the edges between them. In evaluations of both stages we demonstrated the efficacy of our approach by significantly outperforming the state-of-the-art text generator that we extended.

This work opens several avenues for future research. In this paper, we have assumed a complex task as given input; we plan to extend our pipeline with the ability to distinguish complex from simple tasks. This extension in turn will allow us to expand the scope of our current system by allowing for recursive task decomposition and organization. Meanwhile, although our novel Complex Task Dataset proved a useful resource for modeling sub-task dependency inference, it remains quite small; we hope to increase its size considerably in future work, in order to make it more useful to the broader research community. We also hope to conduct human evaluations of generated sub-tasks in order to gauge their coherence and utility to complex tasks. Finally, we hope to test our system in practical, downstream usage by studying the productivity impact of automated task decomposition and organization on real users in their daily lives.

8 Acknowledgement

The authors would like to thank Asli Celikyilmaz and Chirag Shah for insightful discussions and suggestions on this project during the internship. This work was partly supported by Contract FA8750-19-2-1004 with the US Defense Advanced Research Projects Agency (DARPA), and by the office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA Contract No. 2019-19051600006 under the BETTER Program.

References

Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel G Bobrow, and Nicolas Ducheneaut. 2004. What a to-do: studies of task management towards the design of a personal task list manager. In *CHI*, pages 735–742.

Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. 2003. Taking email to task: the design and evaluation of a task management centered email tool. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 345–352.

RN Brewer, Meredith Ringel Morris, and Siân E Lindley. 2017. How to remember what to remember: exploring possibilities for digital reminder systems. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 1(3):1–20.

Yllias Chali, Shafiq R Joty, and Sadid A Hasan. 2009. Complex question answering: unsupervised learning approaches and experiments. *Journal of Artificial Intelligence Research*, 35:1–47.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Quang Do, Wei Lu, and Dan Roth. 2012. Joint inference for event timeline construction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 677–687.

David Graus, Paul N Bennett, Ryen W White, and Eric Horvitz. 2016. Analyzing and predicting task reminders. In *UMAP*, pages 7–15.

Ahmed Hassan Awadallah, Ryen W White, Patrick Pantel, Susan T Dumais, and Yi-Min Wang. 2014. Supporting complex search tasks. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 829–838.

Ece Kamar and Eric Horvitz. 2011. Jogger: models for context-sensitive reminding. In *AAMAS*, pages 1089–1090.

David Kirsh. 2000. A few thoughts on cognitive overload. *Intellectica*, 30(1):19–51.

Mahnaz Koupaee and William Yang Wang. 2018. Wikihow: A large scale text summarization dataset. *arXiv preprint arXiv:1810.09305*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Kangqi Luo, Fengli Lin, Xusheng Luo, and Kenny Zhu. 2018. Knowledge base question answering via encoding of complex query graphs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2185–2194.

Sudipto Mukherjee, Subhabrata (Subho) Mukherjee, Marcello Hasegawa, Ahmed Hassan Awadallah, and Ryen W. White. 2020. Smart to-do : Automatic generation of to-do list from emails. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Jaime Teevan, Shamsi T Iqbal, Carrie J Cai, Jeffrey P Bigham, Michael S Bernstein, and Elizabeth M Gerber. 2016a. Productivity decomposed: Getting big things done with little microtasks. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 3500–3507. ACM.

Jaime Teevan, Shamsi T Iqbal, and Curtis Von Veh. 2016b. Supporting collaborative writing with microtasks. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2657–2668.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Xiaojun Wan, Jianwu Yang, and Jianguo Xiao. 2007. Manifold-ranking based topic-focused multi-document summarization. In *IJCAI*, volume 7, pages 2903–2908.

Ryen W White and Ahmed Hassan Awadallah. 2019. Task duration estimation. In *WSDM*, pages 636–644.

Ryen W White, Ahmed Hassan Awadallah, and Robert Sim. 2019. Task completion detection: A study in the context of intelligent systems. In *SIGIR*, pages 405–414.

Ryen W White, Elnaz Nouri, James Woffinden-Luey, Mark Encarnacion, and Sujay Kumar Jauhar. in press. Microtask detection. *ACM Transactions on Information Systems (TOIS)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.