SUMMARIZATION PROGRAMS: INTERPRETABLE ABSTRACTIVE SUMMARIZATION WITH NEURAL MODULAR TREES

Anonymous authors
Paper under double-blind review

ABSTRACT

Current abstractive summarization models either suffer from a lack of clear interpretability or provide incomplete rationales by only highlighting parts of the source document. To this end, we propose the *Summarization Program (SP)*, an interpretable modular framework consisting of an (ordered) list of binary trees, each encoding the step-by-step generative process of an abstractive summary sentence from the source document. A Summarization Program contains one root node per summary sentence, and a distinct tree connects each summary sentence (root node) to the document sentences (leaf nodes) from which it is derived, with the connecting nodes containing intermediate generated sentences. Edges represent different modular operations involved in summarization such as sentence fusion, compression, and paraphrasing. We first propose an efficient best-first search method over neural modules, SP-SEARCH that identifies SPs for human summaries by directly optimizing for ROUGE scores. Next, using these programs as automatic supervision, we propose seq2seq models that generate Summarization Programs, which are then executed to obtain final summaries. We demonstrate that SP-SEARCH effectively represents the generative process behind human summaries using modules that are typically faithful to their intended behavior. We also conduct a *simulation study* to show that Summarization Programs improve the interpretability of summarization models by allowing humans to better simulate model *reasoning*. Summarization Programs constitute a promising step toward interpretable and modular abstractive summarization, a complex task previously addressed primarily through blackbox end-to-end neural systems.

1 INTRODUCTION

Recent progress in pre-trained language models has led to state-of-the-art abstractive summarization models capable of generating highly fluent and concise summaries (Lewis et al., 2020; Zhang et al., 2020; Raffel et al., 2020). Abstractive summarization models do not suffer from the restrictive nature of extractive summarization systems that only copy parts of the source document. However, their ability to generate non-factual content (Cao et al., 2018; Maynez et al., 2020) and their lack of clear interpretability makes it harder to debug their errors and deploy them in real-world scenarios. Towards interpretable summarization models, Jing & McKeown (1999; 2000) show that human summaries typically follow a cut-and-paste process, and propose a modular architecture involving separate operations that perform sentence extraction, sentence reduction, sentence fusion, etc. Most recent efforts on explainable abstractive summarization follow an extractive-abstractive framework that only provides supporting evidence or 'rationales' for the summary (Hsu et al., 2018; Gehrmann et al., 2018; Liu & Lapata, 2019; Zhao et al., 2020; Li et al., 2021). These models highlight words or sentences from the source document but are not able to explicitly capture the generative process of a summary, i.e., the reasoning steps performed in order to generate each summary sentence from the source document sentence(s), like sentence compression, fusion, etc. In this work, we seek to bridge this gap by proposing a novel *Summarization Program* framework for explaining abstractive summarization, that views summarization as a systematic reasoning process over document sentences.

A Summarization Program (SP) is a modular executable program that consists of an (ordered) list of binary trees, each encoding the generative process of an abstractive summary sentence from the

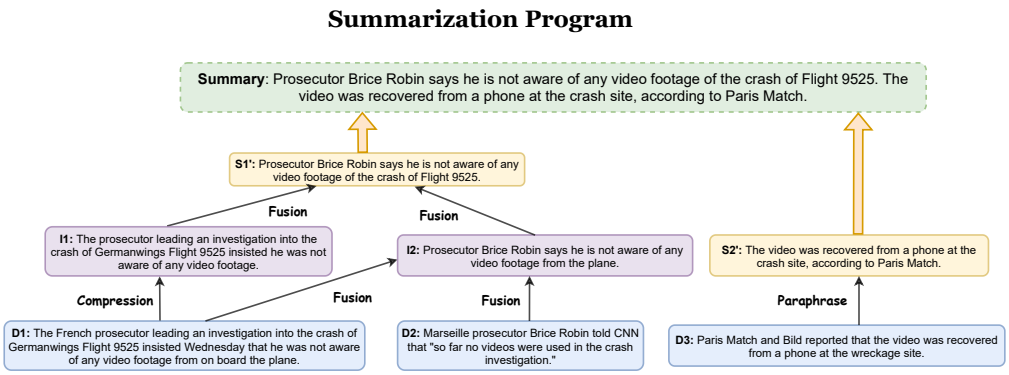

Figure 1: Example of a Summarization Program showing the generative process of two summary sentences (marked with labels S1' and S2' in yellow) from the three document sentences (marked with labels D1, D2, and D3 in blue) using compression, paraphrase and fusion neural modules. Edges directed from leaves to roots, and the intermediate generations are labeled with I1 and I2.

source document (§3). Fig. 1 shows an example. The leaves in an SP are the source document sentences (typically, only a small subset that are relevant for generating the summary). Each intermediate node represents a generation from a neural module (shown with labeled edges) which are then composed to derive the final summary sentences at the roots of the trees. We develop three neural modules for building SPs – sentence compression, paraphrasing, and fusion (Jing & McKeown, 1999; 2000) that finetune a pre-trained language model on task-specific data. We evaluate Summarization Programs by asking the following two research questions (see Fig. 2 for an overview).

1. **RQ1.** *Given a human-written abstractive summary, can we develop an algorithm for identifying a Summarization Program that effectively represents the generative process of the summary?*
2. **RQ2.** *Using the SPs identified in RQ1 as supervision, can we develop models that generate Summarization Programs as interpretable intermediate representations for generating summaries?*

We answer the first research question by automatically identifying SPs for human summaries (§4). Specifically, we develop an efficient best-first search algorithm, SP-SEARCH that iteratively applies different neural modules to a set of extracted document sentences in order to generate newer sentences, such that the ROUGE (Lin, 2004) score with respect to the gold summary is maximized by these new sentences. SP-SEARCH achieves efficiency through important design choices including maintaining a priority queue that scores, ranks, and prunes intermediate generations (appendix A). We conduct experiments to show that SP-SEARCH outputs SPs that effectively reproduce human summaries, significantly outperforming several baselines and obtaining a high ROUGE-2 of 40 on the CNN/DailyMail dataset (Hermann et al., 2015) (§6.1). Moreover, human evaluation of the Summarization Programs shows that our neural modules are highly faithful,[1] performing operations they are supposed to and also generating outputs that are mostly factual to their inputs (§6.2).

We leverage SP-SEARCH to obtain oracle programs for human summaries that also serve as supervision for answering our second research question. In particular, we propose two seq2seq models for Summarization Program generation from a source document (§5, Fig. 2). In our first Extract-and-Build SP generation model, a state-of-the-art extractive summarization model first selects a set of document sentences, which are then passed to another program-generating model. In our second Joint SP generation model, sentence extraction and SP generation happen as part of a single model. We generate SPs by finetuning a pre-trained language model, BART (Lewis et al., 2020) which are then executed to obtain the final summaries.

As a way to evaluate whether Summarization Programs improve the interpretability of summarization models, we conduct a small-scale *simulation study* (Doshi-Velez & Kim, 2017; Hase & Bansal, 2020; Zhou et al., 2022) where we ask humans to simulate model's *reasoning* by writing programs for model summaries (§6.4). We observe that after seeing model-generated programs, humans are

[1] 'Neural module faithfulness' refers to whether the modules perform their expected operations. This is different from 'summary faithfulness' that evaluates whether a summary contains only factual claims from the source document. The latter will be referred to as 'summary factuality'.

able to better predict SPs for unseen samples, such that the executed summaries match more closely with the model summaries. In summary, our contributions are as follows:

- We introduce the *Summarization Program*, an interpretable modular framework for explainable abstractive summarization. Summarization Program consists of an ordered list of binary trees, each encoding the step-by-step generative process of an abstractive summary sentence through the use of different neural modules (sentence compression, paraphrasing, and fusion).
- We propose an efficient best-first search method, SP-SEARCH that identifies Summarization Programs for human-written summaries, obtaining a high ROUGE-2 of 40 on the CNN/DailyMail dataset with neural modules that are highly faithful to their intended behavior.
- We present initial Summarization Program generation models that generate SPs from a source document, which are then executed to obtain final summaries. We demonstrate that SPs improve the interpretability of summarization models by allowing humans to better *simulate* model behavior.

2 RELATED WORK

Pre-trained Models for Summarization. State-of-the-art pre-trained language models like BART (Lewis et al., 2020), T5 (Raffel et al., 2020), and PEGASUS (Zhang et al., 2020) generate summaries in an end-to-end manner. Lacking complete mechanistic understanding of transformers (Elhage et al., 2021), it remains difficult to understand the reasoning process behind the generated summary. In particular, we cannot determine whether generation follows a process similar to that of humans, e.g., using operations like selection of important information, abstraction through vocabulary generalization, sentence fusion, etc (Kintsch & van Dijk, 1978; Brown & Day, 1983; Jing & McKeown, 1999; 2000). Moreover, it is hard to determine the source of errors in summaries without knowing the reasoning path to the summary. Recent work on interpretable summarization models highlight words or sentences as rationales for the generated summaries (Hsu et al., 2018; Gehrmann et al., 2018; Liu & Lapata, 2019; Zhao et al., 2020; Li et al., 2021), but textual highlights do not provide a complete explanation of the summary generation process. Thus, Summarization Program is a step towards more explicit interpretability in abstractive summarization.

Neural Module Networks. Our work also draws inspiration from Neural Module Networks (NMN) that execute programs as learned functions composed of neural modules (Andreas et al., 2016; Hu et al., 2018; Jiang & Bansal, 2019; Gupta et al., 2020; Subramanian et al., 2020; Saha et al., 2021a; Le et al., 2022). Typically, the modules in an NMN provide attention-based explanations whose interpretability has been debated (Serrano & Smith, 2019; Wiegreffe & Pinter, 2019). Khot et al. (2021) alternatively propose Text Modular Networks that decompose multi-hop questions into sub-questions to be solved by simpler QA models. We also follow this text-in text-out paradigm for our modules but equipped to perform diverse sentence-level operations in document summarization. We evaluate our modules for 'neural module faithfulness' (Subramanian et al., 2020) to demonstrate that the SPs mostly provide a faithful interpretation of the generated summaries.

Multi-step Reasoning over Text. Step-by-step reasoning has recently received much interest as a way to explain various reasoning tasks including explainable QA via Entailment Trees (Dalvi et al., 2021; Ribeiro et al., 2022), natural language deduction via statement compositions (Bostrom et al., 2022), and proof generation for rule-bases (Saha et al., 2020; 2021b; Tafjord et al., 2021; Gontier et al., 2020). Summarization Programs similarly encode the reasoning steps in a summary generation process. Our SP-SEARCH method follows a forward chaining method that tries to reach a hypothesis from a set of premises (Bostrom et al., 2022; Tafjord et al., 2021), unlike backward chaining methods that do the opposite (Arabshahi et al., 2021; Gontier et al., 2020; Dalvi et al., 2022; Kalyanpur et al., 2022). In another recent line of work, chain-of-thought prompting (Wei et al., 2022; Nye et al., 2021; Wang et al., 2022) encourages LMs to generate intermediate reasoning steps before producing a final answer to a problem. However, the lack of explicit chaining between the reasoning steps and the final output potentially compromises the faithfulness of those steps, which have also not yet been evaluated as explanations of model behavior per se (Creswell & Shanahan, 2022).

3 SUMMARIZATION PROGRAM

We assume that we have a document $\mathcal{D} = \{D_i\}_{i=1}^{d}$ consisting of d sentences and a corresponding abstractive summary $\mathcal{S} = \{S_i\}_{i=1}^{s}$ consisting of s sentences. A Summarization Program

$\mathcal{P} = \{T_i\}_{i=1}^{s}$ is defined as an (ordered) list of s binary trees where each tree $T_i = (V_i, E_i)$ is a structured representation of the generative process of each summary sentence $S_i \in \mathcal{S}$. Fig. 1 shows an example of an SP with two trees for two summary sentences. The set of nodes V_i in each tree consists of single sentences, and the edges E_i are labeled with one of the neural modules $m \in \{paraphrase(\cdot), compression(\cdot), fusion(\cdot,\cdot)\}$. These modules represent operations over sentences wherein $compression(X) \rightarrow Y$ and $paraphrase(X) \rightarrow Y$ are unary operations and $fusion(X, Y) \rightarrow Z$ is a binary operation. The leaf nodes in each tree are sentences from the document $D_i \in \mathcal{D}$, and the root is a summary sentence $S_i \in \mathcal{S}$. All other nodes are intermediates sentences generated by executing a neural module (referred to as I1 and I2 in Fig. 1). An edge from a node $u \in V_i$ to a node $v \in V_i$ labeled with the module m means that v is generated by executing m on u. The summary $\mathcal{S}$ is obtained by concatenating the root nodes of the trees in order. We hypothesize that the generative process of each summary sentence can be captured by composing different neural modules that operate over sentences. Following prior work on modular approaches to abstractive summarization (Jing & McKeown, 1999; 2000; Lebanoff et al., 2019; 2020), we define the following three neural modules for building Summarization Programs.

Fusion Module. Sentence fusion involves combining information from multiple sentences. We train a BART-large (Lewis et al., 2020) based fusion module, which takes two sentences as input and outputs one fused sentence. Existing sentence fusion datasets either aim to improve discourse connections (Geva et al., 2019) or aim to fuse similar sentences from multiple documents (Brook Weiss et al., 2022). Instead, we want to fuse two disparate sentences into one sentence, which requires the model to merge related pieces and remove unimportant information. To obtain training data for such model, we follow Lebanoff et al. (2019) to align each summary sentence from CNN/DailyMail (Hermann et al., 2015) with one to many similar and non-redundant document sentences. As our training data, we only use examples that align one summary sentence with two source sentences.

Compression Module. The compression module generates a compressed output of a single sentence. It involves generating a shorter sentence by preserving the essential content and the syntactic structure of the input. We finetune a BART-large model (Lewis et al., 2020) on a large parallel corpus of uncompressed and compressed sentences (Filippova & Altun, 2013).

Paraphrase Module. The paraphrase module generates a sentence that involves syntactic transformations or lexical paraphrasing of the input sentence. We use a publicly available PEGASUS-based (Zhang et al., 2020) paraphrase model from HuggingFace (Wolf et al., 2020).[2] In practice, we observe paraphrased outputs to frequently involve some compression as well, which we analyze in detail as part of the 'Neural Module Faithfulness' evaluation (§6.2).

4 RQ1: SUMMARIZATION PROGRAM SEARCH

Our first research question of interest is whether given a document $\mathcal{D}$ and a human-written summary $\mathcal{S}$, we can generate a Summarization Program $\mathcal{P}$ that best explains the generative process of the summary (see the left part of Fig. 2). We achieve this by developing an efficient best-first search method, named SP-SEARCH, outlined in Algorithm 1 in the Appendix. Conceptually, it is similar to a forward chaining algorithm in logic programming (Russell & Norvig, 2009), in which we start from a set of premises (equivalently, a small set of document sentences) and iteratively apply neural modules on them to generate newer deductions (equivalently, intermediate sentences) until the goal hypothesis (equivalently, the summary) is generated. SP-SEARCH processes each summary sentence separately, thereby generating a unique tree for each summary sentence. We optimize the trees for their ROUGE-L (R-L)[3] scores with respect to the gold summary sentences.

SP-SEARCH Algorithm. SP-SEARCH operates as follows (see Algorithm 1). For each summary sentence, it initializes a priority queue, each element (s_1, s_2, m, h) of which represents a module m, the sentences s_1 and s_2 on which m is defined (s_2 is empty for 'compression' and 'paraphrase' modules) and the corresponding tree height h. The initial elements in the queue represent possible operations only on the document sentences. Next, at each step of the search, SP-SEARCH pops an element from the queue and executes the operation over the corresponding operands to generate

[2]Model available at https://huggingface.co/tuner007/pegasus_paraphrase.

[3]We conducted some initial experiments to observe that optimizing for R-L leads to more consistent improvements across all ROUGE measures. See Appendix C for details.

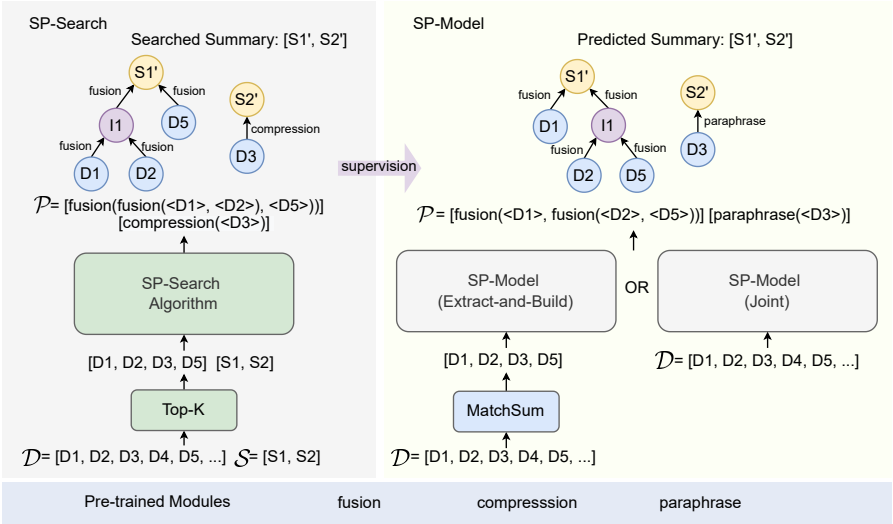

Figure 2: Overview of the two RQs aimed at evaluating SPs. On the left, we show our SP-SEARCH algorithm that takes a source document and a human summary as input and identifies an SP that best explains the summary (RQ1). On the right, we show our SP generation models that take a document as input and output a program, which is then executed to generate the summary (RQ2). S1 and S2 denote human summary sentences while S1' and S2' represent identified or generated summaries.

a new sentence. Each module generates top-5 outputs using beam search and the one with the highest R-L score is chosen. Using this new sentence, SP-SEARCH creates new elements for the priority queue by considering all potential operations involving the new sentence and other available sentences. Then, these new elements may be explored in the next steps of the search. As a best-first search method, SP-SEARCH ranks the elements in the queue with the following scoring function.

$$\text{Score}(s_1, s_2, m, h) = \max\big(\text{R-L}(s_1, S),\ \text{R-L}(s_2, S)\big) \tag{1}$$

The scoring function considers the maximum of the R-L scores with respect to generations s_1 and s_2 such that if the module m is executed, it can possibly lead to a sentence that is of an even higher R-L score than its children. Intuitively, it encourages prioritizing elements in the queue that can potentially lead to higher scores. Whenever a new sentence is generated, its R-L score with the gold summary sentence is computed and accordingly, the best possible root node of the tree is updated. Upon completion of the search (when the queue becomes empty), the node with the maximum score is chosen as the root node and the tree rooted at that node is the final tree. Since performing SP-SEARCH exhaustively can be prohibitive, it is made efficient through the following design choices.

Design Choices for Efficient SP-SEARCH. Here we discuss the choices briefly and provide more details in Appendix A. (1) *Top-k Document Sentences.* We rank each document sentence by computing ROUGE-1 with the summary and build the SP with only the top-k document sentences as eligible source sentences. (2) *Filtering Queue Elements.* To limit the search space, SP-SEARCH defines certain heuristics for choosing the elements to be added to the queue like a compressed sentence is not compressed again, each document sentence is used at most once in a tree, etc. (3) *Priority Queue and Pruning.* SP-SEARCH maintains a fixed queue size and a ranked list of the elements according to Eq. 1 such that only the top-ranked elements are kept in the queue and the rest are pruned. (4) *Parent has a higher ROUGE than children.* A branch of the search is only expanded further if the new generation (parent) has a higher R-L score than the source sentence(s) (children). This greedy approach ensures that every reasoning step is a step closer to the summary. (5) *Maximum Tree Height.* SP-SEARCH chooses a maximum height of the trees. (6) *Batching Module Executions.* SP-SEARCH executes all operations in the queue together by batching at each depth of the search.

5 RQ2: ABSTRACTIVE SUMMARIZATION VIA SUMMARIZATION PROGRAMS

Given that we have proposed SP-SEARCH for identifying SPs for human summaries, we now want to leverage the algorithm to generate training data for building SP generation models, as part of RQ2.

In particular, we use SP-SEARCH to identify Summarization Programs for all training samples in the CNN/DailyMail dataset (Hermann et al., 2015).[4] Our second research question now asks whether these identified SPs can be used as supervision for developing abstractive summarization models via the generation of SPs (see the right part of Fig. 2). Hence, we define a supervised learning problem $f : \mathcal{D} \rightarrow \mathcal{P}$ that generates a Summarization Program $\mathcal{P}$ from document $\mathcal{D}$.

Summarization Program Generation Models. We propose two initial models for our task. The SPs are encoded as strings such that they can be generated by a seq2seq generative model. We first label each sentence in the document with unique identifiers like '<D1>', '<D2>', etc. The Summarization Program is then represented using a nested bracketing structure composed of the modules and the sentence identifiers as shown in Fig. 2. For our first **Extract-And-Build Summarization Program Generation Model**, we hypothesize that a document can contain a large number of sentences but only a small fraction of those are typically useful for generating a summary. Given a training corpus of samples ($\mathcal{D}_k$, $\mathcal{P}$) where $\mathcal{D}_k$ represents the set of k document sentences on which the SP $\mathcal{P}$ is built, we finetune BART such that it can generate a Summarization Program given a small number of document sentences. During inference, given a document, we first extract a set of relevant sentences using a state-of-the-art extractive summarization model, MatchSum (Zhong et al., 2020), which are then used to generate the program. While this model simplifies the learning problem by separating out the sentence extraction phase from the program building phase, relevant sentence extraction is an essential first step towards generating good SPs. Our next **Joint Summarization Program Generation Model** aims to solve sentence extraction and SP generation as part of a single model. We finetune BART on the parallel corpus of document and Summarization Programs and the model learns to directly generate the program given an entire document.

Summary Generation from an SP. We parse and execute a generated SP through iterative inference over the neural modules to obtain the final summary. We check for well-formedness of the SP by ensuring that the generated sequence (1) has balanced parentheses, (2) does not contain any out-of-vocabulary token (besides module names, sentence identifiers, and parentheses), and (3) has consistent number of operands for an operation. During inference, we decode up to top-k SPs using beam search and execute the first well-formed one. When none of the top-k outputs is well-formed ($<$1% of the samples), we generate the corresponding extractive summary.

6 EXPERIMENTS

We conduct experiments on an English single-document sumamrization dataset, CNN/DailyMail (Hermann et al., 2015)[5] to demonstrate the promise of Summarization Programs pertaining to the two research questions.

6.1 RQ1 RESULTS: CAN SP-SEARCH REPRESENT THE SUMMARIZATION PROCESS?

Our first set of experiments is aimed at answering our first RQ (as discussed in §4) where the goal is to analyze how well the Summarization Programs identified by SP-SEARCH represent the human-written summaries. In particular, we answer whether SP-SEARCH can produce summaries similar to human summaries. We consider two variants of SP-SEARCH – (1) SP-SEARCH Top-1 in which each module generates only one output, and (2) SP-SEARCH where each module generates Top-5 outputs (via beam search) and one with the best R-L is chosen.

Evaluation Metrics. We measure similarity with ROUGE (Lin, 2004) scores between the summaries generated by the SP-SEARCH programs and the human summaries. We also compute the factuality of the searched summaries with respect of the gold summaries using a state-of-the-art factuality metric, QuestEval (Scialom et al., 2021). Note that while factuality metrics are typically computed against the source document, RQ1 requires us to evaluate how factual the SP-SEARCH summary is to the reference summary (that the SPs are aiming to reproduce).

Experimental Design. We conduct experiments with 1000 random validation samples.[6] We set the number of initial document sentences (k) to 4, the maximum queue size (Q) to 20, maximum

[4]Please refer to Appendix B for an analysis of the SPs in our training corpus.

[5]Our method can be easily applied to other text summarization datasets, which is left as future work.

[6]Due to the expensive nature of SP-SEARCH, we experiment with a random subset of the validation set.

height of the trees (H) to 2 and the decoding strategy for each module to beam search with beam size of 5. Our choice of hypeparameters is based on a comprehensive study of the trade-off between ROUGE scores and average search time, as presented in Appendix C. As baselines, we consider the following extractive and abstractive oracles. (1) **Random SP.** For each summary sentence in the gold summary, we randomly sample a tree structure (from those obtained through SP-SEARCH) and execute it with randomly chosen leaf nodes from the set of top-k sentences. (2) **Top-4 Sentences.** Our next baseline is an extractive summarization system with the top-4 document sentences ranked according to ROUGE-1 scores with the gold summary. (3) **BART-Oracle.** We also compare with two BART-oracle models. In one, we generate top-10 summaries using beam search. In another, we sample 10 summaries with multinomial sampling and the summary with the highest ROUGE-L score with the gold summary is chosen. (4) **SP-SEARCH Leaves.** Our final baseline is an SP-SEARCH variant where we only consider the leaf nodes (document sentences). This represents an extractive summary with *upto*[7] top-4 sentences. Since the trees are built on top of these nodes, a better ROUGE score for SP-SEARCH will indicate that these leaf nodes or document sentences are being composed in a way such that the resultant summary is more similar to the human summary. (5) **Ablations of Modules.** These are SP-SEARCH variants where each module is removed.

Results. As shown in Table 1, SP-SEARCH generates summaries that better emulate gold summaries by obtaining an R-2 of 40, a significant[8] 20 points improvement ($p < 0.001$) over the SP-SEARCH leaves. The random SP baseline uses the same pre-trained modules as SP-SEARCH but only obtains an R-2 of 15, demonstrating that arbitrary composition of these modules is much less effective than the searched programs. The BART-oracle models exhibit less diversity between the generations, as reflected by their lower ROUGE scores. This demonstrates the utility of a tree structure in an SP. Finally, we observe that each of our neural modules plays a significant role in constructing more accurate SPs. Removing the fusion module leads to the highest drop in ROUGE which suggests that fusing information from multiple sentences is one of the core operations in human summaries. SP-SEARCH also obtains the best QuestEval score, even though it does not specifically optimize for it, suggesting that the summaries are not only similar to the human summaries (as measured by ROUGE) but also have high factual overlap with the human summaries. In summary, SP is an effective way of representing the generative process of abstractive summaries, outperforming random SPs, extractive oracles and unstructured abstractive oracles.

	R1 / R2 / RL / RLsum	QEval
Random SP	37.54 / 15.18 / 25.65 / 33.42	40.50
Top-4 Sentences	35.55 / 17.77 / 25.02 / 32.42	50.56
BART-Oracle (Beam)	40.07 / 19.57 / 32.04 / 37.06	43.25
BART-Oracle (Sample)	42.07 / 19.44 / 33.56 / 39.04	45.09
SP-SEARCH Leaves	39.15 / 20.01 / 27.94 / 35.58	50.53
SP-SEARCH Top-1	56.92 / 34.34 / 52.15 / 54.80	52.32
SP-SEARCH	**61.88 / 40.11 / 58.46 / 60.32**	**54.48**
- Paraphrase	60.08 / 38.28 / 56.36 / 58.29	53.80
- Compression	59.50 / 37.44 / 55.45 / 57.66	53.85
- Fusion	53.73 / 31.30 / 48.90 / 51.73	48.78

Table 1: RQ1 – Comparison of ROUGE scores for the SP-SEARCH summaries with different oracle extractive and abstractive baselines. The final column compares the factuality of the summaries with respect to the *gold summaries* through a SOTA factuality metric, QuestEval.

6.2 RQ1 RESULTS: NEURAL MODULE FAITHFULNESS EVALUATION

Higher ROUGE and factuality scores obtained by SP-SEARCH with respect to the reference summaries do not guarantee that SPs provide a faithful interpretation of the generated summaries. §6.1 specifically evaluates the final summaries but not the intermediate steps used to generate them. In fact, prior work on Neural Module Networks has shown that faithfulness may not always be guaranteed for neural

	Comp	Para	Fusion	Non-Factual
Comp	0.98	0.05	-	0.00
Para	0.68	0.90	-	0.04
Fusion	0.86	0.45	0.80	0.20

Table 2: RQ1 – Neural Module Faithfulness evaluation of SPs. Entry (i,j) shows how often a module 'i' in a row performs the operation 'j' in a column. The final column shows the fraction of non-factual outputs.

[7] SP-SEARCH can ignore certain top-4 sentences if including them do not lead to higher ROUGE scores.

[8] We use a non-parametric bootstrap test (Efron & Tibshirani, 1994) for significance testing.

modules and that incorrect intermediate outputs can still lead to correct final outputs (Subramanian et al., 2020). Hence, we also evaluate the faithfulness of the three modules by conducting a human study (with 2 experts having knowledge of NLP) for 20 SPs. These constitute a total of 107 samples, including 54 fusion, 25 paraphrase, and 28 compression operations. Table 2 shows the results (with more details in Appendix D). The diagonal entries in the table demonstrate how often a module performs its intended behavior and the high values of 0.8-0.98 suggest that our modules are highly faithful. Interestingly, the values in the last column demonstrate that compression and paraphrase modules almost never generate non-factual outputs while our fusion module is more prone to that (about 20%), indicating room for improvement in the fusion module.

6.3 RQ2 Results: Evaluation of Summarization Program Generation Models

Our next set of experiments addresses our second RQ (as discussed in §5) on evaluating Summarization Program Generation models in terms of their summary generation capabilities.

Experimental Design. We compare our SP generation models (Joint and Extract-and-Build) with the following baselines and oracles on the CNN/DailyMail test set. As baselines, we consider (1) a **SOTA Extractive Summarization Model, MatchSum (Zhong et al., 2020)** that we also use to extract initial document sentences for our Extract-and-Build SP generation model, (2) **SOTA Abstractive Summarization Models, BART (Lewis et al., 2020) and PEGASUS (Zhang et al., 2020)**, and (3) **Random SP Models** that randomly sample the number of summary sentences from {1,2,3,4}, and for each summary sentence, randomly sample and execute a tree structure with randomly chosen leaf nodes from all or top-k document sentences. As oracles, we consider the same set of models introduced in §6.1 for RQ1. They all take the gold summaries into account and hence provide an estimate of the upper bound of SP generation models.

Evaluation Metrics. Besides ROUGE, we also evaluate the factuality of the generated summaries (with respect to the source document) using QuestEval that is shown to correlate well with humans.

Results. Table 3 shows the results. We observe that our models obtain R-2 and R-L scores of 16 and 26 respectively. The Extract-and-Build model performs slightly better than our Joint model, possibly due to the former generating programs over a good initial set of document sentences. Both of our models outperform the Random SP baselines, demonstrating that they learn useful patterns of which neural modules should act on which document sentences. Compared to SOTA abstractive models, our models' interpretability (as discussed in §6.4) comes at the cost of some drop in performance (about 5 points in R-2). However, the oracle results suggest that our models provide good starting points for better SP generation models and that there is substantial room for improvement in future work (e.g., the SP-Search Top-1 model obtains an R-2 of 35, leaving a large gap of 19 R-2 points to oracle SPs). We hope the community will explore and improve upon our SP generation models, as a way to make progress on the important and challenging task of developing interpretable abstractive summarization models. We observe a similar trend for the QuestEval scores where our models largely outperform the random SP baselines, while showing worse performance than BART and PEGASUS, possibly because of the cascading effect of compositional generation via SPs. SP-Search obtains slightly worse factuality scores than the SP models, indicating that the

	R1 / R2 / RL / RLsum	QEval
MatchSum (Zhong et al., 2020)	43.98 / 20.66 / 28.72 / 40.02	63.22
BART (Lewis et al., 2020)	44.19 / 21.29 / 31.06 / 41.04	59.15
PEGASUS (Zhang et al., 2020)	44.16 / 21.55 / 31.38 / 41.01	58.83
Random SP (Joint)	24.47 / 6.48 / 15.60 / 22.07	40.73
Random SP (Extract-and-Build)	31.61 / 12.69 / 21.84 / 28.88	52.83
SP Model (Joint)	37.66 / 16.00 / 25.68 / 34.45	56.88
SP Model (Extract-and-Build)	37.23 / 16.40 / 26.48 / 34.18	55.95
Top-4 Sentences (Oracle)	45.35 / 24.60 / 31.30 / 41.92	60.00
SP-Search Leaves (Oracle)	47.28 / 25.92 / 32.73 / 43.70	59.60
SP-Search Top-1 (Oracle)	58.17 / 35.31 / 51.87 / 55.96	54.72
SP-Search (Oracle)	62.69 / 40.58 / 58.02 / 61.10	54.11

Table 3: RQ2 – Comparison of our SP generation models with state-of-the-art extractive and abstractive summarization models, random SP baselines and oracle systems with our SP-Search variants on the CNN/DM test set. All methods are compared based on ROUGE and a state-of-the-art factuality metric, QuestEval (Scialom et al., 2021).

searched summaries are not quite factual to the document either, and one possible way to improve this is to directly integrate a factuality metric into our search objective.

6.4 RQ2 Results: Interpretability Evaluation of SP via Simulation Study

We ask if SPs improve the interpretability of a summarization model. In particular, we are interested in evaluating model *simulatability*, a measure of whether a person can predict model behavior on new inputs (Doshi-Velez & Kim, 2017). The behavior we are interested in is specifically the model *reasoning*, as represented by the programs that our models generate and execute. The primary motivation for predicting model reasoning is that model reasoning is what we want to better understand by virtue of model explanations, and simulation studies that focus on predicting final outputs do so only to show that users have a good mental model of model reasoning (Hase & Bansal, 2020).

Study Design. With 3 experts (authors) who have prior knowledge in NLP and explainability, we design and carry out a small-scale human study that tests whether presenting humans with Summarization Programs improves their ability to predict SPs on future unseen data. Human subjects are first provided with some *test* documents and corresponding model summaries (from our Explain-and-Predict SP model) and are tasked with predicting the model's Summarization Programs. Next, in a *training* phase, they are shown model-generated SPs for a few *training* samples and are then asked again to perform the same task with the same set of *test* examples. The study is conducted with 10 *training* examples and 31 *test* examples. We say that an SP is more representative of the model summary if executing it with our pre-trained modules generates a summary that is closer to the model summary. This is a measure of program similarity since variance in the SPs may make graph similarity measures inappropriate (e.g., compression followed by paraphrase and vice versa may generate similar outputs).

	R1 / R2 / RL / RLsum
Before Expl.	74.37 / 62.79 / 71.62 / 72.54
After Expl.	**78.03 / 66.80 / 72.37 / 74.69**

Table 4: RQ2 – Comparison of mean ROUGE scores between our model summaries and their human simulations before and after being trained with SPs.

Results. The results are presented in Table 4. In general, high ROUGE scores suggest that, when models are forced to generate summaries via SPs, their reasoning becomes quite predictable a priori. We also see about 4 points improvement in ROUGE-2, with 61% of samples showing an improved R-2 score (statistically significant at $p < 0.001$), 10% of samples with a drop in R-2, and remaining being ties. This suggests that SPs are potentially good explanations of model reasoning, such that humans can generalize across model reasoning patterns after being given the explanations.

7 Discussion and Conclusion

A small-scale manual study of our generated SPs reveal that the two most common forms of errors include (1) redundant or longer paths in an SP, and (2) fusion module generating non-factual sentences or ignoring one of its source sentences. Two other notable issues, arising out of the independence assumption of summary sentences, are (1) final summary sentences having overlapping information, and (2) incoherence between consecutive sentences. One way to improve this is to add a 'coherence' module on top of the root nodes before generating the final summary, which is an interesting topic for future work. Appendix F shows some examples of generated SPs. We build SPs using sentences as the fundamental content unit (nodes) due to the relative ease of defining and training neural modules on sentences and the availability of large-scale training data. Summarization may also involve other text manipulation operations that are not fully captured by our modules but our framework allows easy inclusion of other modules. Finally, we envision that SPs could also help in locating and debugging errors in human and model summaries by tracing back the generative process of summary sentences to document sentences.

We proposed Summarization Programs (SPs), a novel framework for explainable abstractive summarization that encodes the step-by-step generative process of each summary sentence from the source document sentences via neural modules. We showed the effectiveness of our framework by (1) developing SP-Search that effectively identifies Summarization Programs for human summaries with highly faithful neural modules, and (2) proposing SP generation models that generate SPs from source documents which are then executed to generate summaries. Via an initial simulation study, we also showed that SPs improve the interpretability of a summarization model.

ETHICS STATEMENT

Despite the recent success of pre-trained language models in abstractive summarization, their lack of explainability still remains a major concern and we hope that Summarization Programs prove to be an important step in bridging that gap. That said, summarization is inherently a subjective task and existing summarization datasets also significantly vary in terms of their stylistic features like abstractiveness, length, and specificity (Goyal et al., 2021). Hence, more future work is needed to understand the general applicability of our neural modules and how effective they are in encoding different kinds of summaries. Broadly put, Summarization Program is a case study for breaking a complex NLP problem down into sub-problems and then solving them through neural modules without having access to intermediate supervision. In that regard, neural modules can be thought of as models that perform specialized skills and hence should provide users with more control, compared to end-to-end models. Overall, we do not foresee any major risks or negative societal impact of our work, however, like any other machine learning model, the generated programs may not always be completely accurate and hence should be used with caution for real-world applications.

REPRODUCIBILITY STATEMENT

To encourage reproducibility, we provide our source code in the supplementary material. The details of our SP-SEARCH algorithm are shown in Algorithm 1. The hyperparameters for RQ1 and RQ2 are discussed in Appendix C and Appendix E respectively. The CNN/DailyMail dataset we experiment with is also publicly available at https://huggingface.co/datasets/cnn_dailymail.

REFERENCES

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 39–48, 2016.

Forough Arabshahi, Jennifer Lee, Mikayla Gawarecki, Kathryn Mazaitis, Amos Azaria, and Tom Mitchell. Conversational neuro-symbolic commonsense reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4902–4911, 2021.

Kaj Bostrom, Zayne Sprague, Swarat Chaudhuri, and Greg Durrett. Natural language deduction through search over statement compositions. *arXiv preprint arXiv:2201.06028*, 2022.

Daniela Brook Weiss, Paul Roit, Ori Ernst, and Ido Dagan. Extending multi-text sentence fusion resources via pyramid annotations. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1854–1860, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.135. URL https://aclanthology.org/2022.naacl-main.135.

Ann L Brown and Jeanne D Day. Macrorules for summarizing texts: The development of expertise. *Journal of verbal learning and verbal behavior*, 22(1):1–14, 1983.

Ziqiang Cao, Furu Wei, Wenjie Li, and Sujian Li. Faithful to the original: Fact aware neural abstractive summarization. In *thirty-second AAAI conference on artificial intelligence*, 2018.

Antonia Creswell and Murray Shanahan. Faithful reasoning using large language models. *arXiv preprint arXiv:2208.14271*, 2022.

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. Explaining answers with entailment trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7358–7370, 2021.

Bhavana Dalvi, Oyvind Tafjord, and Peter Clark. Towards teachable reasoning systems. *arXiv preprint arXiv:2204.13074*, 2022.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

N Elhage, N Nanda, C Olsson, T Henighan, N Joseph, B Mann, A Askell, Y Bai, A Chen, T Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Katja Filippova and Yasemin Altun. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1481–1491, 2013.

Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4098–4109, 2018.

Mor Geva, Eric Malmi, Idan Szpektor, and Jonathan Berant. DiscoFuse: A large-scale dataset for discourse-based sentence fusion. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3443–3455, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1348. URL https://aclanthology.org/N19-1348.

Nicolas Gontier, Koustuv Sinha, Siva Reddy, and Chris Pal. Measuring systematic generalization in neural proof generation with transformers. *Advances in Neural Information Processing Systems*, 33:22231–22242, 2020.

Tanya Goyal, Nazneen Fatema Rajani, Wenhao Liu, and Wojciech Kryściński. Hydrasum: Disentangling stylistic features in text summarization using multi-decoder models. *arXiv preprint arXiv:2110.04400*, 2021.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. Neural module networks for reasoning over text. In *ICLR*, 2020.

Peter Hase and Mohit Bansal. Evaluating explainable AI: Which algorithmic explanations help users predict model behavior? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5540–5552, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.491. URL https://aclanthology.org/2020.acl-main.491.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 132–141, 2018.

Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable neural computation via stack neural module networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 53–69, 2018.

Yichen Jiang and Mohit Bansal. Self-assembling modular networks for interpretable multi-hop reasoning. 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)*, pp. 4474–4484, 2019.

Hongyan Jing and Kathleen McKeown. Cut and paste based text summarization. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 2000.

Hongyan Jing and Kathleen R McKeown. The decomposition of human-written summary sentences. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 129–136, 1999.

Aditya Kalyanpur, Tom Breloff, David Ferrucci, Adam Lally, and John Jantos. Braid: Weaving symbolic and neural knowledge into coherent logical explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Text modular networks: Learning to decompose tasks in the language of existing models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1264–1279, 2021.

Walter Kintsch and Teun van Dijk. Cognitive psychology and discourse: Recalling and summarizing stories. *Current trends in text linguistics*, pp. 61–80, 1978.

Hung Le, Nancy Chen, and Steven Hoi. Vgnmn: Video-grounded neural module networks for video-grounded dialogue systems. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3377–3393, 2022.

Logan Lebanoff, Kaiqiang Song, Franck Dernoncourt, Doo Soon Kim, Seokhwan Kim, Walter Chang, and Fei Liu. Scoring sentence singletons and pairs for abstractive summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2175–2189, 2019.

Logan Lebanoff, John Muchovej, Franck Dernoncourt, Doo Soon Kim, Lidan Wang, Walter Chang, and Fei Liu. Understanding points of correspondence between sentences for abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pp. 191–198, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-srw.26. URL https://aclanthology.org/2020.acl-srw.26.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, 2020.

Haoran Li, Arash Einolghozati, Srinivasan Iyer, Bhargavi Paranjape, Yashar Mehdad, Sonal Gupta, and Marjan Ghazvininejad. Ease: Extractive-abstractive summarization with explanations. *arXiv preprint arXiv:2105.06982*, 2021.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. 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)*, pp. 3730–3740, 2019.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1906–1919, 2020.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

Danilo Ribeiro, Shen Wang, Xiaofei Ma, Rui Dong, Xiaokai Wei, Henry Zhu, Xinchi Chen, Zhiheng Huang, Peng Xu, Andrew Arnold, et al. Entailment tree explanations via iterative retrieval-generation reasoner. In *NAACL*, 2022.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Pearson, 3 edition, 2009.

Amrita Saha, Shafiq Joty, and Steven CH Hoi. Weakly supervised neuro-symbolic module networks for numerical reasoning. *arXiv preprint arXiv:2101.11802*, 2021a.

Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. PRover: Proof generation for interpretable reasoning over rules. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 122–136, 2020.

Swarnadeep Saha, Prateek Yadav, and Mohit Bansal. multiPRover: Generating multiple proofs for improved interpretability in rule reasoning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3662–3677, 2021b.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang, and Patrick Gallinari. Questeval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6594–6604, 2021.

Sofia Serrano and Noah A Smith. Is attention interpretable? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2931–2951, 2019.

Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. Obtaining faithful interpretations from compositional neural networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5594–5608, 2020.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 3621–3634, 2021.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. 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)*, pp. 11–20, 2019.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pp. 11328–11339. PMLR, 2020.

Yao Zhao, Mohammad Saleh, and Peter J Liu. Seal: Segment-wise extractive-abstractive long-form text summarization. *arXiv preprint arXiv:2006.10213*, 2020.

Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuan-Jing Huang. Extractive summarization as text matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6197–6208, 2020.

Tree Structure	H	Freq (in %)
compression (·)	1	8
compression (fusion (·, ·))	2	8
fusion (fusion (·, ·) fusion (·, ·))	2	7
(·)	0	7
fusion (compression (·) fusion (·, ·))	2	6
paraphrase (compression (·))	2	6
paraphrase (fusion (·, ·))	2	6
fusion (fusion (·, ·) compression (·))	2	5
fusion (fusion (·, ·))	2	5
fusion (compression (·))	2	5

Table 5: Top 10 tree structures for the human-written summaries in the training corpus of CNN/DM. For clarity, each tree structure is accompanied with the corresponding tree height and the frequency (in percentage). An empty structure of "(·)" represents a singleton node.

Yilun Zhou, Marco Tulio Ribeiro, and Julie Shah. ExSum: From local explanations to model understanding. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5359–5378, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.392. URL https://aclanthology.org/2022.naacl-main.392.

A Design Choices for Efficient SP-Search

SP-Search is outlined in Algorithm 1. It is made efficient through important design choices as discussed below. For clarity, we do not show all of these choices in the Algorithm outline.

Top-k Document Sentences. The search space grows exponentially at each depth (because of the fusion operation), but one way to limit the growth is to ensure that SP-Search starts with a small number of document sentences. A summary is typically constructed by using information from only a small number of sentences from the document. Hence, we rank each document sentence by computing the ROUGE-1 score with the summary and build the Summarization Program with only the top-k document sentences ('k' being a small number) as eligible source sentences.

Filtering Queue Elements. The search space is also dependent on how many elements are added to the queue at each step and how many of those are expanded further. Note that whenever a new sentence is generated via a module, it can potentially fuse with all previous generations to create new queue elements. Since doing this exhaustively increases the search space, SP-Search defines certain heuristics for choosing the elements that will be added to the queue: (1) a sentence that has been compressed once is not compressed again, (2) a sentence that has been paraphrased once is not paraphrased again, (3) each document sentence is used at most once in a tree, (4) two sentences are not fused if they are intermediate generations from the same sentence, and (5) since fusion is not a symmetric operation and can lead to different generations based on the order, sentence fusion happens keeping the temporal order of the sentences in the document intact.

Priority Queue and Pruning. SP-Search performs an additional step of pruning for the elements that are added to the queue. It maintains a fixed queue size and while expanding the elements in the queue in a best-first manner, it maintains a ranked list of the elements according to Eq. 1 such that only the top-ranked elements are kept in the queue and the rest are pruned.

Parent has a higher ROUGE than children. Each module generates multiple outputs through beam search and SP-Search ensures that the best output has a higher R-L score than the nodes on which the module is defined. If this is not the case, the corresponding branch of the search is not expanded further. This constraint generalizes to the property that every node in a Summarization Program will have a higher R-L score (with respect to the summary sentence) than all other nodes in the subtree rooted at that node. Conceptually, this greedy approach ensures that every reasoning step in a Summarization Program is a step closer to the summary (according to a scoring function).

Algorithm 1: SP-SEARCH Algorithm

Input: Top-k Document Sentences $\mathcal{D}_k$, Summary $\mathcal{S}$, Modules $\{\mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_f\}$, Maximum Height $\mathcal{H}$, Maximum Queue Size $\mathcal{Q}$, Number of Generations $\mathcal{G}$, Scoring Function $Score$
Output: Summarization Program $\mathcal{P}$

function SP-SEARCH($\mathcal{D}_k, \mathcal{S}, \{\mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_f\}, \mathcal{H}, \mathcal{Q}, \mathcal{G}, Score$)
  $\mathcal{P}$ = [] ▷ Initialize Summarization Program.
  forall $S \in \mathcal{S}$ **do**
    $queue \leftarrow$ [], $height \leftarrow 1$ ▷ Initialize queue of items and tree height.
    $maxRouge \leftarrow 0$, $bestRoot \leftarrow \phi$ ▷ Initialize a tree with maximum ROUGE.
    forall $\mathcal{D}_1 \in \mathcal{D}_k$ **do**
      $currRouge \leftarrow$ ComputeRouge($\mathcal{D}_1$, S) ▷ Compute ROUGE with leaf nodes.
      if $currRouge > maxRouge$ **then**
        $maxRouge \leftarrow currRouge$, $bestRoot \leftarrow$ Node($\mathcal{D}_1$) ▷ Update ROUGE and root.
      $queue \leftarrow queue \cup \{(\mathcal{D}_1, \phi, \mathcal{M}_c, height)\}$ ▷ Add compression items.
      $queue \leftarrow queue \cup \{(\mathcal{D}_1, \phi, \mathcal{M}_p, height)\}$ ▷ Add paraphrase items.
      for $\mathcal{D}_2 \in \mathcal{D}_k$ **do**
        $queue \leftarrow queue \cup \{(\mathcal{D}_1, \mathcal{D}_2, \mathcal{M}_f, height)\}$ ▷ Add fusion items.
    $prevHeight \leftarrow 0$, $visited \leftarrow$ []
    while $|queue| > 0$ **do**
      $height \leftarrow queue[0].height$ ▷ Get height of the top queue item.
      if $height > \mathcal{H}$ **then**
        continue ▷ Stop expanding beyond max height.
      if $height \neq prevHeight$ **then**
        $prevHeight \leftarrow height$
        $queue \leftarrow \arg\max_{q \in queue}(queue, \mathcal{Q}, Score)$ ▷ Rank and prune the queue.
        $gens \leftarrow$ Execute($queue, \{\mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_f\}, \mathcal{G}$) ▷ Execute all operations in queue.
      $newNode \leftarrow queue$.pop() ▷ Pop the top item from the queue.
      $newGen \leftarrow gens[newNode]$ ▷ Get the newly generated sentence.
      $newRouge \leftarrow$ ComputeRouge($newGen$, S) ▷ Get rouge with new sentence.
      if $newRouge > maxRouge$ **then**
        $maxRouge \leftarrow newRouge$, $bestRoot \leftarrow$ Node($newGen$) ▷ Update root.
      if $newGen \notin visited$ **then**
        $visited \leftarrow visited \cup \{newGen\}$, $height \leftarrow height + 1$
        $queue \leftarrow queue \cup \{(newGen, \phi, \mathcal{M}_c, height)\}$ ▷ Add compression items.
        $queue \leftarrow queue \cup \{(newGen, \phi, \mathcal{M}_p, height)\}$ ▷ Add paraphrase items.
        forall $prevGen \in visited$ **do**
          $queue \leftarrow queue \cup \{(newGen, prevGen, \mathcal{M}_f, height)\}$ ▷ Add fusion items.
    T_S = ConstructTree($bestRoot$) ▷ Get Tree by backtracking from best root.
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{T_S\}$ ▷ Append new tree to the Summarization Program.
  return $\mathcal{P}$

Maximum Tree Height. SP-SEARCH chooses a maximum height of the trees, beyond which the nodes are not expanded further during the search.

Batching Module Executions. Instead of executing each module separately, which requires a forward pass over a neural model and can be time-consuming, SP-SEARCH executes all operations in the queue together by batching at each depth of the search.

B ANALYSIS OF SP-SEARCH SUMMARIZATION PROGRAMS FOR CNN/DM

We observe that 68% of the summaries in CNN/DM have 3 or 4 summary sentences and, equivalently, the corresponding Summarization Programs have 3 or 4 trees. Note that while we initialize SP-SEARCH with the Top-4 document sentences, a Summarization Program may choose to ignore some of these sentences if including them does not lead to higher ROUGE scores. Upon analysis, we

k	Q	D	H	ROUGE (R1/R2/RL)	T
3	20	beam (5)	2	59.36 / 37.38 / 55.59	21.09
4	20	beam (5)	2	61.88 / 40.11 / 58.46	29.58
5	20	beam (5)	2	63.72 / 41.99 / 60.46	35.80
4	10	beam (5)	2	61.21 / 39.06 / 57.46	15.38
4	20	beam (5)	2	61.88 / 40.11 / 58.46	29.58
4	30	beam (5)	2	62.32 / 40.57 / 58.97	41.61
4	20	beam (5)	1	56.45 / 34.16 / 51.34	13.58
4	20	beam (5)	2	61.88 / 40.11 / 58.46	29.58
4	20	beam (5)	3	65.00 / 43.81 / 62.36	34.68
4	20	greedy	2	52.87 / 30.12 / 47.08	26.85
4	20	beam (1)	2	56.91 / 34.34 / 52.15	28.56
4	20	beam (5)	2	61.88 / 40.11 / 58.46	29.58

Table 6: Analysis of ROUGE scores (R1/R2/RL) and average search time (in seconds) for SP-SEARCH on 1000 validation samples in CNN/DM with different search hyperparameters. k = Number of extracted document sentences. Q = Maximum queue size. D = Decoding Strategy from the modules. H = Maximum height of the tree. T = Average search time in seconds. Search times are computed on a single RTX 2080 Ti GPU. Beam(5) refers to beam search with beam size 5 and best of top-5 generations from each module.

Metric	R1	R2	RL	RLsum
R-1	**64.28**	38.40	54.36	57.68
R-2	59.11	**41.85**	53.28	55.97
R-L	61.88	40.11	**58.46**	**60.32**

Table 7: Comparison of ROUGE scores for the summaries generated by SP-SEARCH based on the optimization metric used. Optimizing for R-L leads to more consistent results across all ROUGE metrics.

find that 73% of the programs are constructed using all four initial sentences, 23% are constructed with three sentences and 3% have two sentences. We also note that the trees can have any height up to the maximum defined height of 2. A tree of height 0 represents a singleton node with a single document sentence. Thus, a Summarization Program with only singleton nodes represents an extractive summary. Overall, we observe that our trees have as many as 20 different structures. Table 5 shows the top 10 tree structures in our corpus. As an example, a tree structure of "compression (fusion $(\cdot, \cdot)$)" represents a tree of height 2 in which two document sentences are first fused and then the resultant sentence is compressed.

C SP-SEARCH HYPERPARAMETERS

In order to analyze the effect of different hyperparameters in SP-SEARCH, we compare the trade-off between ROUGE[9] scores and average search time per sample (on a single RTX 2080 Ti GPU) by varying the number of initial document sentences (k), the queue size (Q), decoding strategy (D) and the maximum height of the trees (H) in Table 6. We observe that increasing the number of document sentences k beyond 4 leads to some improvements in ROUGE scores. However, this comes at the cost of increased search time. Increasing the queue size (Q) also leads to minor improvements but again suffers from increased search time. While increasing the maximum tree height (H) to 3 results in better ROUGE scores, on inspection, we observe that it happens primarily due to low-quality fusion of two arbitrary sentences that may not always lead to better programs. Finally, beam search performs better than greedy decoding, and leveraging the best of the top-5 generations from each

[9]We use the HuggingFace implementation of ROUGE at https://github.com/huggingface/datasets/blob/main/metrics/rouge/rouge.py.

	Compression	Paraphrase	Fusion	Non-Factual
Compression	0.98±0.01	0.05±0.02	-	0.00±0.00
Paraphrase	0.68±0.04	0.90±0.05	-	0.04±0.00
Fusion	0.86±0.01	0.45±0.03	0.80±0.03	0.20±0.05

Table 8: RQ1 – Neural Module Faithfulness evaluation of Summarization Programs. Each entry (i,j) shows the fraction of time a module 'i' in a row performs the operation 'j' in a column. The final column shows how often a module generates non-factual outputs. We report the mean and variance of the scores between the two annotators.

module improves the results further with almost no increase in search time. Overall, at a moderate search time of less than 30 seconds/sample on a single RTX 2080 Ti GPU, SP-SEARCH obtains a 20 point increase in R-2 compared to our extractive baseline with leaf nodes (document sentences).

SP-SEARCH builds Summarization Programs by optimizing for ROUGE-L because of its better overall performance across all ROUGE metrics. As shown in Table 7, optimizing instead for R-1 and R-2 leads to slightly better R-1 and R-2 respectively but overall R-L performs better. In general, we find that the different ROUGE metrics may not correlate well with each other when optimizing for one of these.

D RQ1 RESULTS: NEURAL MODULE FAITHFULNESS EVALUATION ON CNN/DM (HERMANN ET AL., 2015)

We discuss the faithfulness evaluation of our modules below.

Study Design. We would attribute each sentence to exactly one of the modules, but since our modules are generative, we do not know if each module performs only and exactly its named function. For example, fusing two sentences into a fluent sentence may in practice also involve some paraphrasing or compression. Hence, we evaluate module faithfulness by analyzing how often a module m_i performs the role of a module m_j. Two expert annotators, with knowledge of NLP, annotate each intermediate generation (from a module) and assign a binary label against each of the modules that could lead to that output. Additionally, the annotators also label each generation for whether it contains non-factual content (that cannot be verified from the sentence(s) used to generate it). The study is conducted with 20 Summarization Programs, consisting of a total of 107 samples, with 54 fusion, 25 paraphrase, and 28 compression operations.

Results. Table 8 shows the results. The diagonal entries in the table demonstrate how often a module performs its intended behavior and the high values of 0.8-0.98 suggest that our modules are highly faithful. In about 20% of the cases, a fusion operation may not involve any fusion. We also observe that some form of compression frequently happens as part of the paraphrasing and fusion modules, as shown in the first column of the table. Similarly, the fusion outputs also tend to involve some compression and paraphrasing operations. Interestingly, the values in the last column of the table demonstrate that compression and paraphrase modules almost never generate non-factual outputs while our fusion module is more prone to that (about 20%), indicating room for improvement in the fusion module.

E HYPERPARAMETERS FOR SP GENERATION MODELS

We build our models on top of the HuggingFace transformers library (Wolf et al., 2020). All models are trained for 40000 steps with a batch size of 16, learning rate of $3 * 10^{-5}$ and warmup steps of 500. We set the maximum input length to 512 and maximum generation length to 100. During inference, we generate up to Top-10 Summarization Programs with beam search and output the first well-formed program. We also set the minimum generation length to 10 to prevent the model from generating too short sequences and repetition penalty to 2. Program execution from the SPs is performed with the same set of hyperparamaters for each module as used during SP-SEARCH.

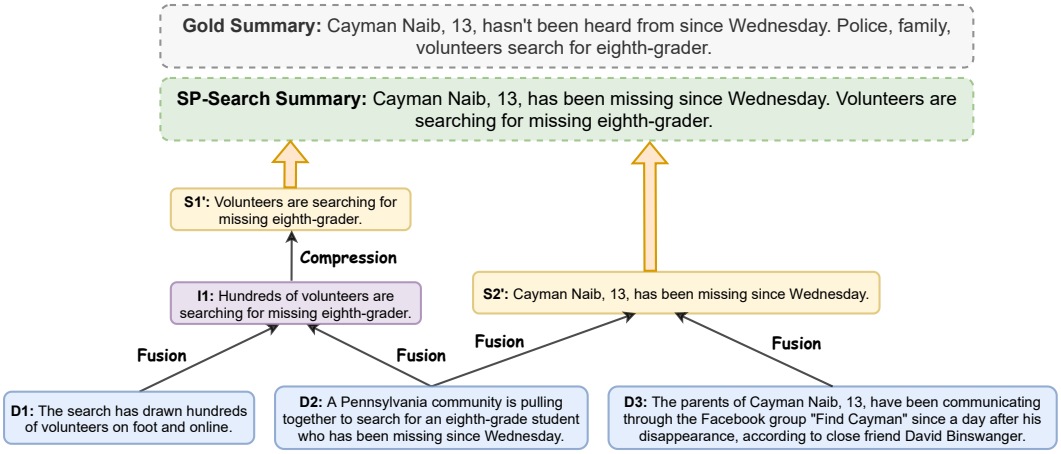

Figure 3: Example of a Summarization Program identified by SP-SEARCH. The summary identified by SP-SEARCH matches closely with the gold summary.

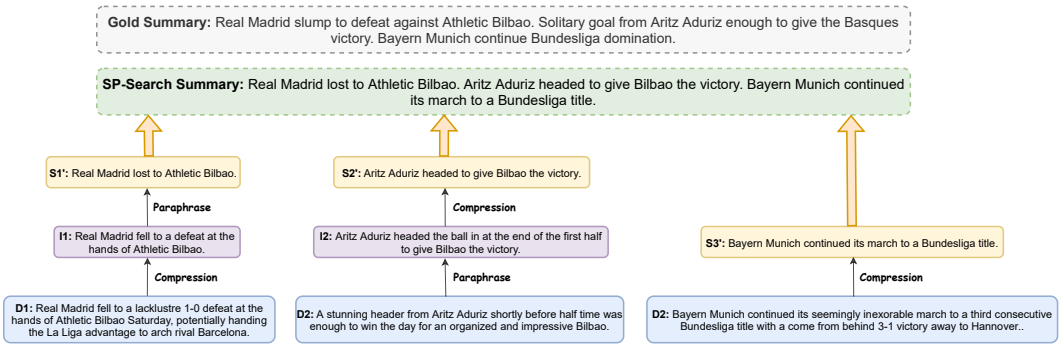

Figure 4: Example of a Summarization Program identified by SP-SEARCH. The summary identified by SP-SEARCH matches closely with the gold summary.

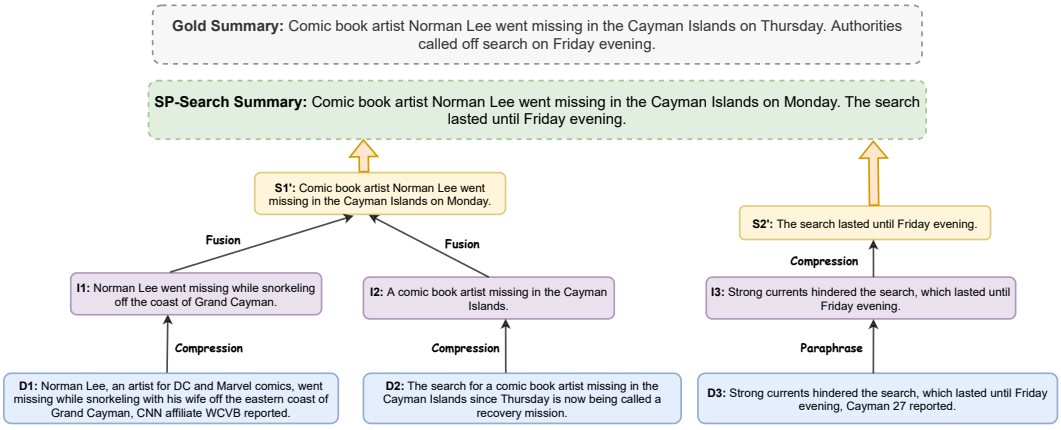

Figure 5: Example of a Summarization Program identified by SP-SEARCH. The summary identified by SP-SEARCH contains a factually incorrect part "on Monday". The Summarization Program lets us identify the exact step in which the error was introduced.

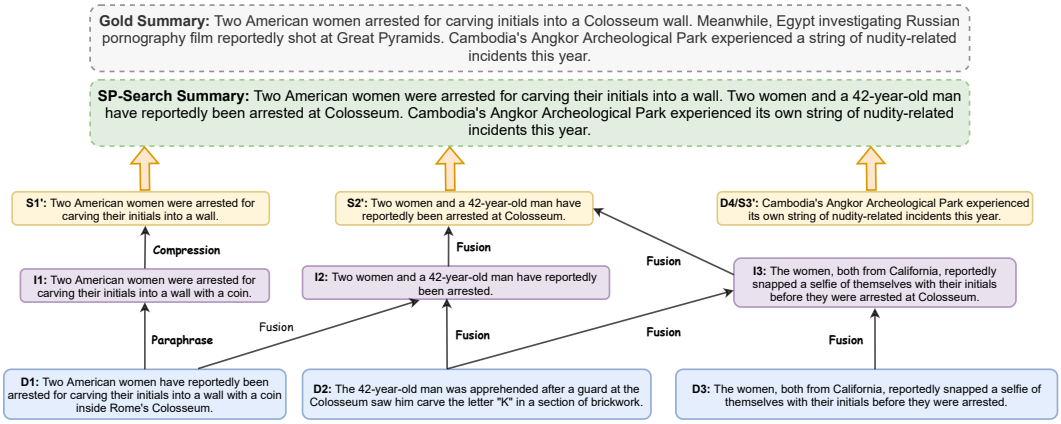

Figure 6: Example of a Summarization Program identified by SP-SEARCH. It contains a singleton node (D4/S3). The first two summary sentences have overlap in information, originating from the independence assumption in the generative process of each summary sentence.

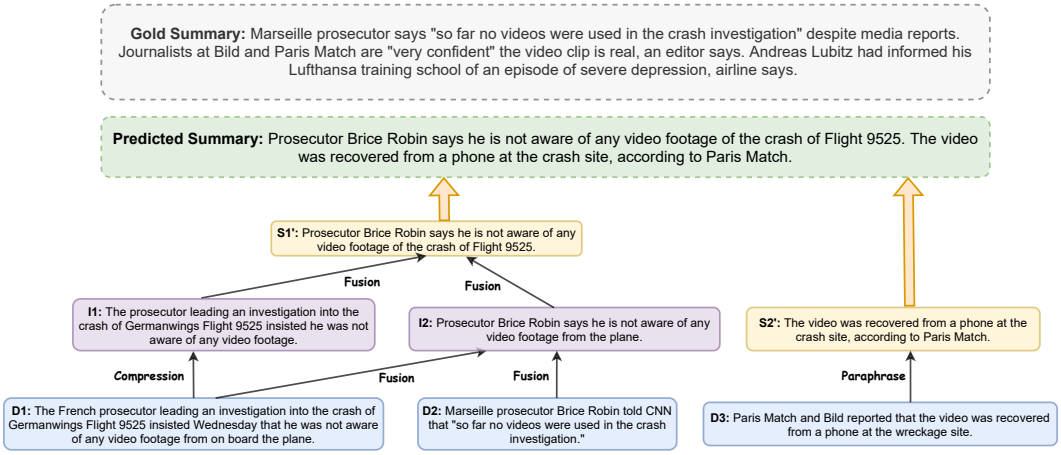

Figure 7: Example of a Summarization Program and summary generated by the Extract-and-Build SP generation model.

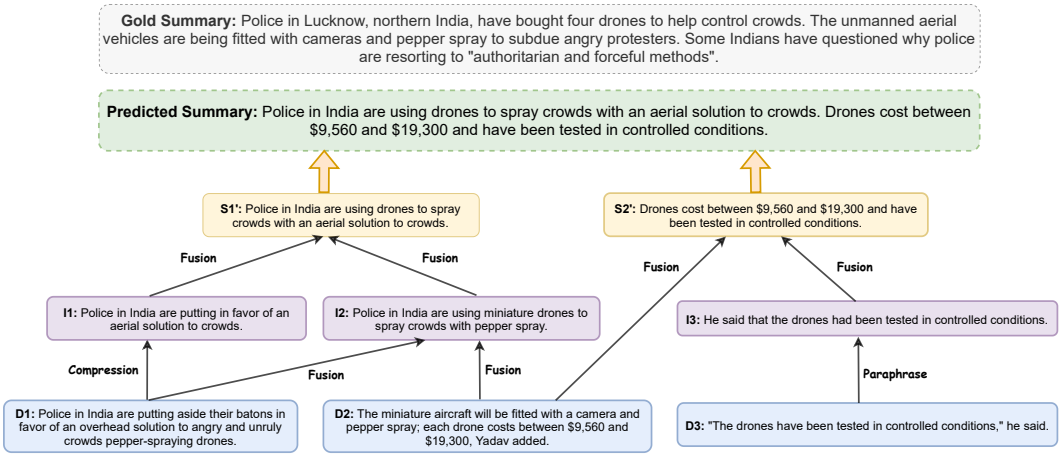

Figure 8: Example of a Summarization Program and summary generated by the Extract-and-Build SP generation model.

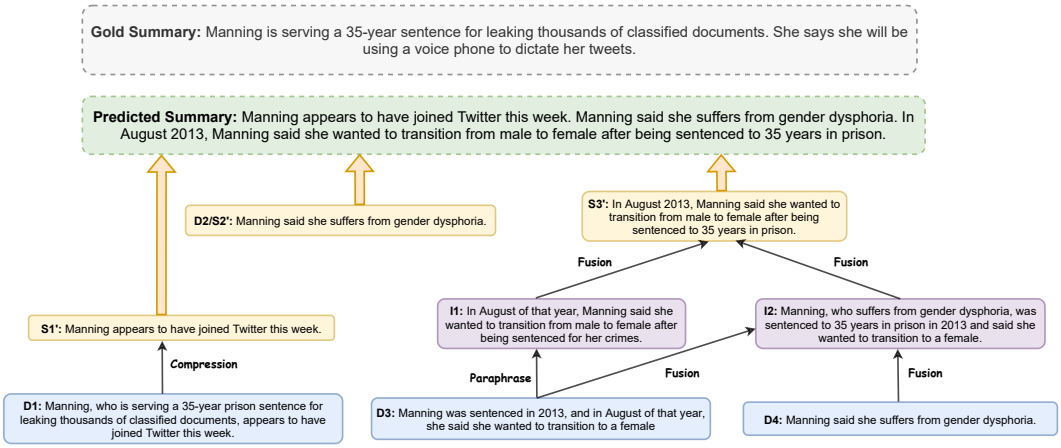

Figure 9: Example of a Summarization Program and summary generated by the Extract-and-Build SP generation model. The generated summary contains a factual error that Manning was sentenced to "35 years in prison".

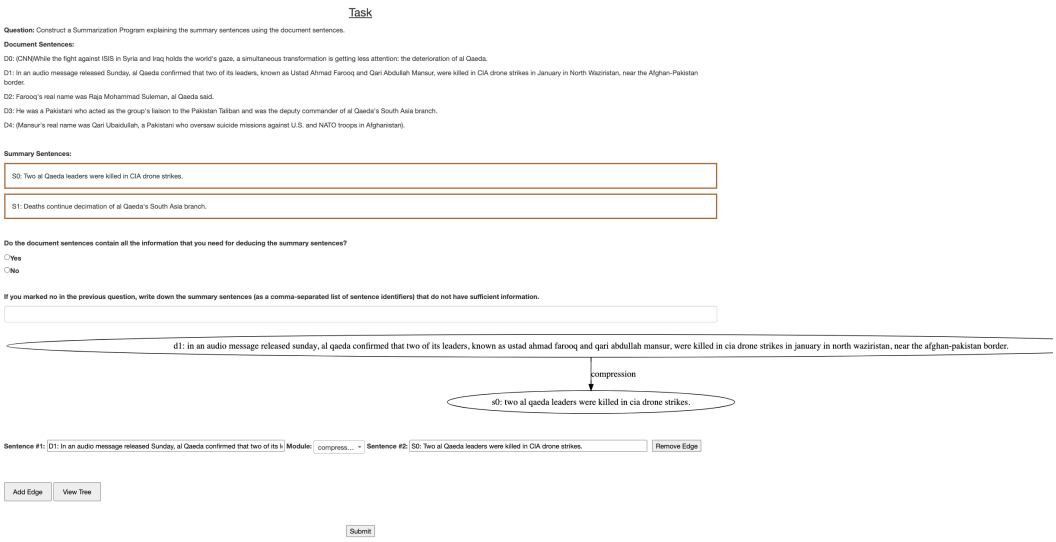

Figure 10: Interface for simulating Summarization Programs given a source document and a model summary. Annotators construct trees for all summary sentences by creating one edge at a time.

F EXAMPLES OF SUMMARIZATION PROGRAMS

Figures 3, 4, 5, and 6 show some examples of Summarization Programs identified by SP-SEARCH for human-written summaries. Figures 7, 8, and 9 show examples of Summarization Programs and corresponding summaries generated by our Extract-and-Build SP generation model. For compactness, we show the Summarization Programs by merging the common nodes between the trees.

G SIMULATABILITY INTERFACE OF SUMMARIZATION PROGRAMS

In Figure 10, we show the interface for our simulatability study in which annotators are asked to write Summarization Pograms for model-generated summaries.