Text Modular Networks: Learning to Decompose Tasks in the Language of Existing Models

Tushar Khot   Daniel Khashabi   Kyle Richardson
Peter Clark   Ashish Sabharwal
Allen Institute for AI, Seattle, WA, U.S.A.
{tushark,danielk,kyler,peterc,ashishs}@allenai.org

Abstract

We propose a general framework called Text Modular Networks (TMNs) for building interpretable systems that learn to solve complex tasks by decomposing them into simpler ones solvable by existing models. To ensure solvability of simpler tasks, TMNs learn the textual input-output behavior (i.e., *language*) of existing models through their datasets. This differs from prior decomposition-based approaches which, besides being designed specifically for each complex task, produce decompositions independent of existing sub-models. Specifically, we focus on Question Answering (QA) and show how to train a next-question generator to sequentially produce sub-questions targeting appropriate sub-models, without additional human annotation. These sub-questions and answers provide a faithful natural language explanation of the model's reasoning. We use this framework to build MODULARQA,[1] a system that can answer multi-hop reasoning questions by decomposing them into sub-questions answerable by a neural factoid single-span QA model and a symbolic calculator. Our experiments show that MODULARQA is more versatile than existing explainable systems for DROP and HotpotQA datasets, is more robust than state-of-the-art blackbox (uninterpretable) systems, and generates more understandable and trustworthy explanations compared to prior work.

1 Introduction

An intuitive way to solve more complex tasks, such as multi-hop question-answering (Yang et al., 2018; Khashabi et al., 2018; Khot et al., 2020) and numerical reasoning (Dua et al., 2019), would be to decompose them into already solved simpler problems, e.g., single-fact QA (Rajpurkar et al., 2016). Besides allowing reuse of existing simpler models, this approach would yield an *interpretable system* that provides a faithful explanation (Jacovi and Goldberg, 2020) of its reasoning as a composition of simpler sub-tasks, as shown in Fig. 1. Motivated by this, we ask the following question:

Given a set of existing QA models, can one leverage them to answer complex questions by communicating with these existing models?

We propose a general framework, **Text Modular Networks (TMNs)**, that answers this question by learning to decompose complex questions (of any form) into sub-questions that are answerable by existing QA models—symbolic or neural (henceforth referred to as *sub-models*).[2] Unlike previous approaches (Talmor and Berant, 2018; Min et al., 2019a), the decompositions are not based on splits of the complex questions and aren't built independent of the sub-model. Instead, our framework learns to generate sub-questions in the scope

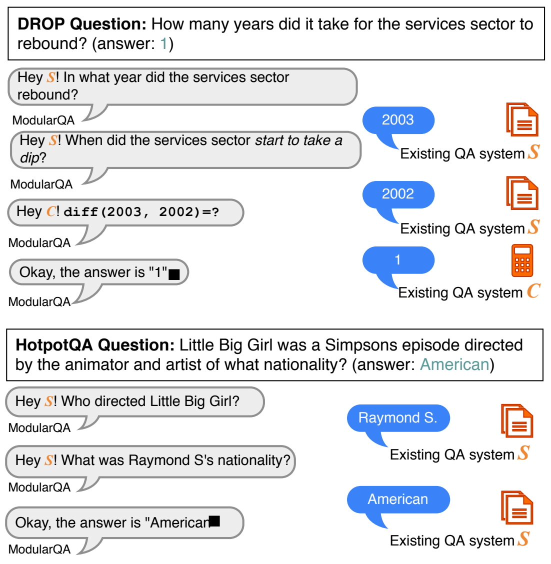

DROP Question: How many years did it take for the services sector to rebound? (answer: 1)
Hey S! In what year did the services sector rebound?
ModularQA
2003
Existing QA system S
Hey S! When did the services sector *start to take a dip*?
ModularQA
2002
Existing QA system S
Hey C! diff(2003, 2002)=?
ModularQA
1
Existing QA system C
Okay, the answer is "1"■
ModularQA
HotpotQA Question: Little Big Girl was a Simpsons episode directed by the animator and artist of what nationality? (answer: American)
Hey S! Who directed Little Big Girl?
ModularQA
Raymond S.
Existing QA system S
Hey S! What was Raymond S's nationality?
ModularQA
American
Existing QA system S
Okay, the answer is "American"■
ModularQA

Figure 1: MODULARQA learns to ask sub-questions to existing simple QA models, including a symbolic calculator, to answer a given complex question. Notably, the approach does not rely on annotated decompositions. Despite this, the system learned to add "start to take a dip" in the DROP dataset question.

[1] https://github.com/allenai/modularqa

[2] TMNs, in fact, treat sub-models as blackboxes, and can thus use *any* model or function as a module.

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1264–1279
June 6–11, 2021. ©2021 Association for Computational Linguistics

of existing models. For instance, the second sub-question in the DROP dataset example in Fig. 1 requires the *introduction of a new phrase*, "start to take a dip", which is beyond the scope of standard decomposition approaches. Additionally, the final sub-question targets a *symbolic calculator*, which operates over a different input language.

The core of our TMN framework is a *next-question generator* that sequentially produces the next sub-question to ask as well as an appropriate sub-model for answering it. The resulting sequence of sub-questions and their answers provides a human-interpretable description of the model's *neuro-symbolic* reasoning (McCarthy, 1988; Smolensky, 1988), as illustrated in Fig. 1. Notably, TMNs learn to produce these decompositions using only distant supervision, *without* the need for any explicit human annotation.

One of our key insights is that the capabilities of existing sub-models can be captured by training a text-to-text system to generate the questions in the sub-model's training dataset (e.g., SQuAD), given appropriate *hints*. In our case, we train a BART model (Lewis et al., 2020) to generate questions given the context, answer, and preferred vocabulary as hints. We then use these sub-task question models to generate sub-questions (and identify appropriate sub-models) that could lead to the likely intermediate answers extracted for each step of the complex question ("Raymond S." and "American" in the HotpotQA example in Fig. 1). The resulting sub-questions, by virtue of our training, are in the language (i.e., within-scope) of the corresponding sub-models. These sub-question sequences can now be used to *train* the next-question generator to sequentially produce the next sub-question. We use this trained generator, along with existing QA models, to answer complex questions, without the need for any intermediate answers.

We use the TMN framework to develop **ModularQA**, a modular system that explains its reasoning in natural language, by decomposing complex questions into those answerable by two sub-models: a *neural factoid single-span QA model* and a *symbolic calculator*. ModularQA's implementation[1] covers multi-hop questions that can be answered using these two sub-models via five classes of reasoning found in existing QA datasets: *composition, conjunction, comparison, difference,* and *complementation.*[3]

We evaluate ModularQA on questions from two datasets, DROP (Dua et al., 2019) and HotpotQA (Yang et al., 2018), resulting in the first cross-dataset decomposition-based interpretable QA system. Despite its interpretability and versatility, ModularQA scores only 3.7% F1 lower than NumNet+V2 (Ran et al., 2019), a state-of-the-art *blackbox* model designed for DROP. ModularQA even outperforms this blackbox model by 2% F1 in a limited data setting and demonstrates higher (+7% F1) robustness (Gardner et al., 2020). ModularQA is competitive with and can even outperform task-specific Neural Module Networks (Gupta et al., 2020; Jiang and Bansal, 2019) while producing textual explanations. Further, our human evaluation against a split-point based decomposition model trained on decomposition annotation (Min et al., 2019b) for HotpotQA finds our explanations to be more trustworthy, understandable, and preferable in 67%-78% of the cases.

Contributions. (1) Text Modular Networks (TMNs), a general framework that leverages existing simpler models—neural and symbolic—as blackboxes for answering complex questions. (2) ModularQA,[1] an interpretable system that learns to automatically decompose multi-hop and discrete reasoning questions. (3) Experiments on DROP and HotpotQA demonstrating ModularQA's cross-dataset versatility, robustness, sample efficiency and ability to explain its reasoning in natural language.

2 Related Work

Many early QA systems were designed as a combination of distinct modules, often composing outputs of *lower-level* language tasks to solve *higher-level* tasks (Moldovan et al., 2000; Harabagiu and Hickl, 2006). However, much of this prior work is limited to pre-determined composition structures (Berant et al., 2013; Seo et al., 2015; Neelakantan et al., 2017; Roy and Roth, 2018).

Various **modular network** architectures have been proposed to exploit compositionality (Rosenbaum et al., 2018; Kirsch et al., 2018). The closest models to our work are based on *neural module networks* (NMN) (Andreas et al., 2016) which compose task-specific simple neural modules. We

[3]Composition and conjunction questions are also referred to as 'bridge' questions. Complementation refers to questions such as 'What percentage of X is *not* Y?' ModularQA can be easily extended to other reasoning types by defining the corresponding hints (§4.3).

compare against formulations of NMNs for HotpotQA (Jiang and Bansal, 2019) and DROP (Gupta et al., 2020), both of which target only one dataset and do not reuse existing QA systems. Moreover, they provide attention-based explanations whose interpretability is unclear (Serrano and Smith, 2019; Brunner et al., 2020; Wiegreffe and Pinter, 2019).

Question decomposition has been pursued before for ComplexWebQuestions (Talmor and Berant, 2018) and HotpotQA. Both approaches (Talmor and Berant, 2018; Min et al., 2019b) focus on directly training a model to produce sub-questions using question spans—an approach not suitable for DROP questions (as illustrated in Fig. 1). Our next-question generator overcomes this limitation by generating free-form sub-questions in the language of existing models. Perez et al. (2020) also use a text-to-text model to generate sub-questions for HotpotQA. However, they generate simpler questions without capturing the requisite reasoning, and hence use them mainly for evidence retrieval.

BREAK (Wolfson et al., 2020) follows an alternative paradigm of collecting full question decomposition meaning representations (QDMR) annotations. While this can be effective, it relies on costly human annotation that may not generalize to domains with new decomposition operations. Its decompositions are generated in a model-agnostic way and still need QA systems to answer the sub-questions, e.g, high-level QDMR questions such as "Which is earlier?" and "Which is longer?" would need special systems that can map these to symbolic comparisons. In contrast, TMNs start with pre-determined models and learn to generate decompositions in their language.

While many **multi-hop QA** models exist for HotpotQA and DROP, these are often equally complex models (Tu et al., 2020; Fang et al., 2020; Ran et al., 2019) focusing on *just one* of these datasets. Only on HotpotQA, where supporting sentences are annotated, can these models also produce post-hoc explanations, but these explanations are often not faithful and shown to be gameable (Trivedi et al., 2020). TMNs are able to produce explanations for multiple datasets without needing such annotations, making it more generalizable to future datasets.

3 Text Modular Networks

TMNs are a family of architectures consisting of *modules* that communicate through *language* learned from these modules, to accomplish a certain goal (e.g., answering a question). Figure 2 illustrates this general idea in the context of answering a DROP question. The core of our system is a *next-question generator* $\mathbb{D}$, a component in charge of generating and distributing sub-tasks among sub-models $\mathbb{A}_s$. The system alternates between using $\mathbb{D}$ to produce the next question (*NextGen*) and using the corresponding sub-model to answer this question. Formally, solving a complex question qc is an alternating process between the following two steps:

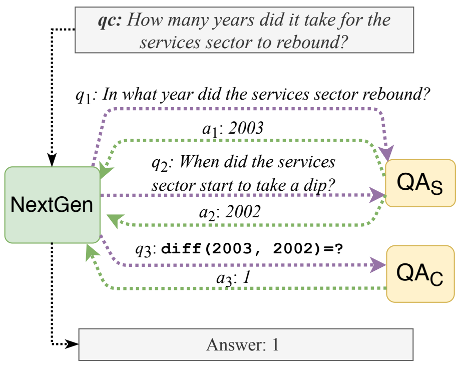
Generate the next question q_i for submodel t_i:

$$\langle t_i, q_i \rangle = \mathbb{D}(qc, q_1, a_1, \ldots, q_{i-1}, a_{i-1})$$

Find answer a_i by posing q_i to submodel t_i:

$$a_i \quad = \mathbb{A}_{t_i}(q_i, p)$$

where q_i is the i^{th} generated sub-question and a_i is the answer produced by a sub-model t_i based on a given context paragraph p. This simple iterative process ends when q_{i+1} equals a special end-of-sequence symbol (denoted throughout as `[EOQ]`) with the final output answer a_i.



qc: How many years did it take for the services sector to rebound?
q1: In what year did the services sector rebound?
a1: 2003
q2: When did the services sector start to take a dip?
a2: 2002
q3: diff(2003, 2002)=?
a3: 1
NextGen
QA_S
QA_C
Answer: 1

Figure 2: A sample inference on a DROP question using TMNs with the text-to-text interactions between the next-question generator $\mathbb{D}$ and existing QA models $\mathbb{A}$.

Building a Text Modular Network. The key challenge in building a Text Modular Networks is developing the next-question generator model. Training this model requires a next-question prediction dataset where each example is a step in the iterative progression of sub-question generation. For example, the second step in Fig. 2 is:

In {
qc: How many years did it take for the services sector to rebound?
q_1: In what year did the services sector rebound?
a_1: 2003

Out { $\langle t_2, q_2 \rangle$= ⟨*SQuAD, "When did the services sector start to take a dip?"*⟩

While it may be possible to collect task-specific datasets or design a task-specific next-question generator (Min et al., 2019b; Talmor and Berant, 2018), our goal is to build a framework that can be easily extended to new complex QA tasks reusing existing QA sub-models. To achieve this, we present a *general* framework to *generate the next-question training dataset* by: (1) Modeling *the language* of sub-models; (2) Building decompositions in the language of these sub-models using minimal distant supervision *hints*.

3.1 Modeling QA Sub-Models

To ensure the sub-questions are answerable by existing sub-models, we train a text-to-text *sub-task question model* on the original sub-task to generate a plausible q_i conditioned on hints, e.g., a BART model trained on SQuAD to generate a question given the answer. We can view this utility as characterizing the question language of the sub-model. For example, such a model trained on the SQuAD dataset would produce factoid questions—the space of questions answerable by a model trained on this dataset.

While an unconditional text generation model can also capture the space of questions, it can generate a large number of possibly valid questions, making it hard to effectively train or use such a model. Instead, we scope the problem down to conditional text generation of questions given hints z. Specifically, we use the context p, answer a and question vocabulary v as input conditions to train a question generator model $\mathbb{G} : z \rightarrow q$ where $z = \langle p, a, v \rangle$. Such a generator, $\mathbb{G}_{\mathcal{S}}$, produces the first two sub-questions in the example in Fig. 4, when using a=2003 (or 2002, resp.) and v=$\Phi(qc)$={"service", "sector", "year", "rebound"} as hints.

3.2 Training Decompositions via Distant Supervision

To generate training decompositions for a complex question using a sub-task question model, we extract distant-supervision hints z corresponding to each reasoning step. This is akin to the distant supervision approaches used to extract logical forms in semantic parsing (Liang et al., 2013; Berant et al., 2013) and the intermediate entities in a reasoning chain (Gupta et al., 2020; Jiang and Bansal, 2019).

In our running DROP example, under the definition of $z = \langle p, a, v \rangle$, we would need to provide the context, answer, and question vocabulary for each reasoning step. We can derive intermediate answers by finding the two numbers whose difference is the final answer (see Fig. 4). We can use words from the input question as vocabulary hints.[4]

As shown in Fig. 4, we generate the training sub-questions in the language of appropriate systems for each step i using the question generation model $\mathbb{G}_{t_i}$: $q_i = \mathbb{G}_{t_i}(z_i)$ where $z_i = \langle p_i, a_i, v_i \rangle$ and the model t_i is determined by the answer type (or can be a hint too).

Note that our framework does not depend on the specific choice of z. Our key idea is to train the sub-task question model conditioned on the same z that we can provide for the complex task. The hint z could be very general (just the context) or very specific (exact vocabulary of the question), trading off the ease of extracting hints with the quality of the generated decomposition. Similarly, these hints don't have to be 100% accurate as they are only used to build the training data and play no role during inference.

Finally, we convert the decompositions into training data for the next-question generator. For each question q_i generated using the sub-task question model $\mathbb{G}_{t_i}$, we create the training example:

Input: $qc, q_1, a_1, \ldots, q_{i-1}, a_{i-1}$
Output: $\langle t_i, q_i \rangle$

Training Data Generation Summary. Fig. 3 illustrates the complete process for generating the training data for the next-question generator. For each complex question, we extract a set of possible hints for each potential reasoning chain (e.g., all number pairs that lead to the final answer). For each step, we use the corresponding sub-task question models to generate potential sub-questions that lead to the expected answer. Finally we use these generated sub-question decompositions as the training data for the next-question generator model.

4 ModularQA System

We next describe a specific instantiation of the Text Modular Network: ModularQA – a new QA system that works across HotpotQA and DROP. To handle these datasets, we first introduce the two QA sub-models(§4.1), the sub-task question models for these models(§4.2), our approach to build training data (§4.3), and the inference procedure used for question-answering(§4.4).

[4]As mentioned before, these are soft hints and the model can be trained to handle noise in these hints.

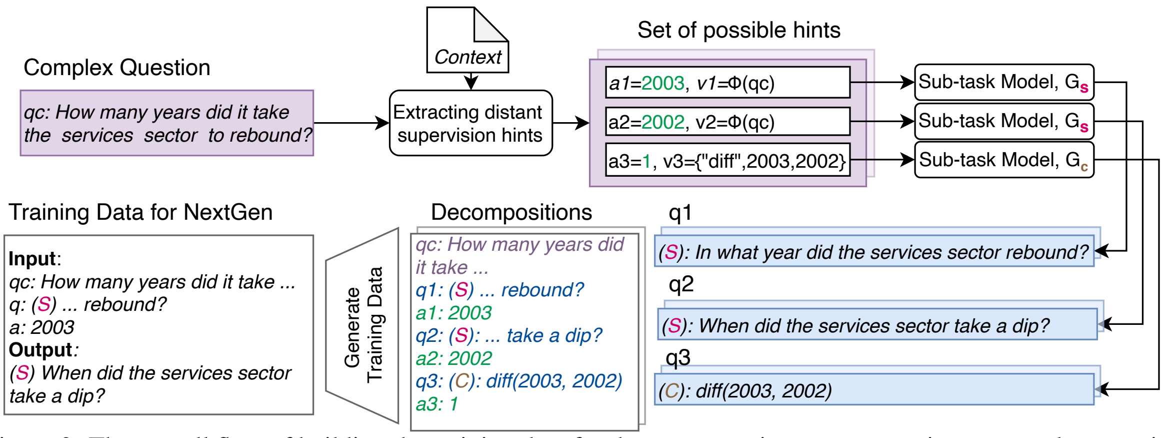

Complex Question
qc: How many years did it take the services sector to rebound?
Context
Extracting distant supervision hints
Set of possible hints
a1=2003, v1=Φ(qc)
a2=2002, v2=Φ(qc)
a3=1, v3={"diff",2003,2002}
Sub-task Model, Gs
Sub-task Model, Gs
Sub-task Model, Gc
q1
(S): In what year did the services sector rebound?
q2
(S): When did the services sector take a dip?
q3
(C): diff(2003, 2002)
Decompositions
qc: How many years did it take ...
q1: (S) ... rebound?
a1: 2003
q2: (S): ... take a dip?
a2: 2002
q3: (C): diff(2003, 2002)
a3: 1
Generate Training Data
Training Data for NextGen
Input:
qc: How many years did it take ...
q: (S) ... rebound?
a: 2003
Output:
(S) When did the services sector take a dip?

Figure 3: The overall flow of building the training data for the next-question generator, given a complex question.

Para, p: ... The sector decreased by 7.8 percent in 2002, before rebounding in 2003 ...
Question, qc: How many years did it take for the services sector to rebound? **Answer** a: **1**
Hints → Sub-Questions
$\langle a_1$=**2003**$, p_1{=}p, v_1{=}\phi(\text{qc})\rangle \rightarrow q_1{=}\mathbb{G}_{\mathcal{S}}(p_1, a_1, v_1)$: *In what year did the services sector rebound?*
$\langle a_2$=**2002**$, p_2{=}p, v_2{=}\phi(\text{qc})\rangle \rightarrow q_2{=}\mathbb{G}_{\mathcal{S}}(p_2, a_2, v_2)$: *When did the services sector start to take a dip?*
$\langle a_3$=**1**$, p_3{=}p, v_3{=}\{\text{diff}, 2003, 2002\}\rangle \rightarrow q_3{=}\mathbb{G}_{\mathcal{C}}(p_3, a_3, v_3)$: *diff(2003, 2002)*
■ → q_4 = *[EOQ]*

Figure 4: An example decomposition generated for a DROP example using hints and sub-question generators $\mathbb{G}$. ϕ(qc) indicates words in the input question qc.

4.1 QA Sub-Models, $\mathbb{A}$

We use two QA models with broad coverage on the two datasets:

SQuAD model, $\mathbb{A}_{\mathcal{S}}$, A RoBERTa-Large model trained on the entire SQuAD 2.0 dataset including the no-answer questions; and

Math calculator model, $\mathbb{A}_{\mathcal{C}}$, a symbolic Python program that can perform key operations needed for DROP and HotpotQA, namely:

`diff`(X, Y, Z) that computes the difference between X and Y in unit Z (days/months/years);

`not`(X) that computes the complement % of X, i.e., 100 - X;

`if_then`(X <op> Y, Z, W) that returns Z if X <op> Y is true, otherwise returns W.

4.2 Sub-task Question Models, $\mathbb{G}$

We define two sub-task question models corresponding to each of our QA sub-models.

SQuAD Sub-task Question Model, $\mathbb{G}_{\mathcal{S}}$. We train a BART-Large model on the answerable subset of SQuAD 2.0 to build our sub-task question model for SQuAD. We use the gold paragraph and answer from the dataset as the input context and answer. For the estimated question vocabulary, we select essential words[5] from the gold questions (referred as the function Φ) with additional irrelevant words sampled from other questions.[6]

To train the text-to-text $\text{BART}_{\mathcal{S}}$ model, we use a simple concatenation of the passage, vocabulary, and answer (with markers such as "H:" and "A:" to indicate each field) as the input sequence and the question as the output sequence. While a constrained-decoding approach (Hokamp and Liu, 2017; Hu et al., 2019a) could be used here to further promote the use of the vocabulary hints, this simple approach was effective and more generally applicable to other hints in our use-case.

Once this model is trained, we use it with nucleus sampling (Holtzman et al., 2020) to generate k sub-questions, Q, and filter out those that lead an incorrect or no answer using $\mathbb{A}_{\mathcal{S}}$:

$$\mathbb{G}_{\mathcal{S}}(p, a, v) = \{q \in Q \mid \text{overlaps}(\mathbb{A}_{\mathcal{S}}(p, q), a)\}$$

Math Sub-task Question Model, $\mathbb{G}_{\mathcal{C}}$. Given the symbolic nature of this solver, rather than training a neural generator, we simply generate all possible numeric questions given the context. Similar to $\mathbb{G}_{\mathcal{S}}$, we first generate potential questions Q and then filter down to those that lead to the expected answer using $\mathbb{A}_{\mathcal{C}}$:

$$\mathbb{G}_{\mathcal{C}}(p, a, v) = \{q \in Q \mid \mathbb{A}_{\mathcal{C}}(p, q) = a\}$$

4.3 Generating Training Decompositions

We broadly identify five classes of questions in HotpotQA and DROP dataset that can be answered using our two models.[7] These question classes, how

[5] $\Phi(q)$ = Non-stopword tokens with pos tags ∈ {NOUN, VERB, NUM, PROPN, ADJ, RB}

[6] More details in Appendix A

[7] Other questions require a QA model that can return multiple answers or a Boolean QA model, as discussed in §6.

DROP

Example 1: **How many days passed between the Sendling Christmas Day Massacre and the Battle of Aidenbach?**
» Q: *When was the Battle of Aidenbach?* A: 8 January 1706 Q: *When was the Sendling Christmas Massacre?* A: 25 December 1705 Q: `diff`*(8 January 1706, 25 December 1705, days)* A: **14**

Example 2: **Which ancestral group is smaller: Irish or Italian?**
» Q: *How many of the group were Irish?* A: 12.2 Q: *How many Italian were there in the group?* A: 6.1 Q: `if_then`*(12.2 < 6.1, Irish, Italian)* A: **Italian**

Example 3: **How many percent of the national population does not live in Bangkok?**
» Q: *What percent of the national population lives in Bangkok?* A: 12.6 Q: `not`*(12.6)* A: **87.4**

HotpotQA

Example 4: **12 Years a Slave starred what British actor born 10 July 1977)**
» Q: *Who stars in 12 Years a Slave?* A: Chiwetel Ejiofor Q: *Who is the British actor born 10 July 1977?* A: **Chiwetel Umeadi Ejiofor**

Example 5: **How many children's books has the writer of the sitcom Maid Marian and her Merry Men written ?**
» Q: *What writer was on Maid Marian and her Merry Men?* A: Tony Robinson Q: *How many children's books has Tony Robinson written?* A: **sixteen**

Example 6: **Did Holland's Magazine and Moondance both begin in 1996?**
» Q: *When did Holland's Magazine begin?* A: **1876** Q: *When did Moondance begin?* A: **1996** Q: `if_then`*(1876≠1996, no, yes)* A: **no**

Table 1: Sample Reasoning Explanations generated by MODULARQA. Note that the system learns to generate such explanations without relying on manually designed rules such as "smaller" $\Rightarrow x < y$.

they are identified and how we extract hints for each question type is described next. Note that similar rules for extracting distant supervision hints have been used by prior work for DROP (Gupta et al., 2020) and HotpotQA (Jiang and Bansal, 2019) too.
1. **Difference** (*How many days before X did Y happen?*): We identify these questions based on the presence of term indicating a measurement :"how many" and terms indicating difference such as "shorter", "more", "days between", etc. Also we check for two dates or numbers in the context such that their difference (in all units) can lead to the final answer. If these conditions are satisfied, for every pair n_1, n_2 where the difference (in units u) can lead to the final answer, we generate the hints:
$p_1 = p; a_1 = n_1; v_1 = \Phi(qc)$
$p_2 = p; a_2 = n_2; v_2 = \Phi(qc)$
$p_3 = \varepsilon; a_3 = a; v_3 = [\texttt{diff}, n_1, n_2, u]$
where ε refers to the empty string.

2. **Comparison** (*Which event happened before: X or Y?*): We identify the two entities e_1 and e_2 in such questions and find dates/numbers that are mentioned in documents. For every n_1, n_2 number/date mentioned close to e_1 and e_2 respectively, we create the hints:
$p_1 = p; a_1 = n_1; v_1 = \Phi(qc) \setminus e_2$
$p_2 = p; a_2 = n_2; v_2 = \Phi(qc) \setminus e_1$
$p_3 = \varepsilon; a_3 = a; v_3 = [\texttt{if\_then}, n_1, n_2, e_1, e_2]$
The final set of hints are for use by the calculator generator to create the questions: $\texttt{if\_then}(n_1 > n_2, e_1, e_2)$ and $\texttt{if\_then}(n_1 < n_2, e_1, e_2)$.

3. **Complementation** (*What percent is not X?*): We identify these questions mainly based on the presence of ".* not .*" in the question and a number n_1 such that the $a = 100 - n_1$. The hints are:
$p_1 = p; a_1 = n_1; v_1 = \Phi(qc)$
$p_2 = \varepsilon; a_2 = a; v_2 = [\texttt{not}, n_1]$

4. **Composition** (*Where was 44th President born?*): For such questions(only present in HotpotQA), we need to first find an intermediate entity e_1 that would be the answer to a sub-question in qc (e.g. Who is the 44th President?). This intermediate entity is used by the second sub-question to get the final answer. Given the two gold paras d_1 and d_2, where d_2 contains the answer, we use the mention of d_2's title in d_1 as the intermediate entity.[8] While we could use the entire complex question vocabulary to create hints, we can reduce some noise by removing terms that appear exclusively in the other document. So the final hints are:
$p_1 = d_1; a_1 = e_1; z_1 = \zeta(qc, d_1, d_2)$
$p_2 = d_2; a_2 = a; z_2 = \zeta(qc, d_2, d_1) + e_1$
where $\zeta(q, d_1, d_2)$ indicates the terms in $\Phi(q)$ that appear in d_2 but not in d_1.[9]

5. **Conjunction** (*Who acted as X and directed Y?*): These class of questions do not have any intermediate entity but have two sub-questions with the same answer e.g. "Who is a politician and an actor?". If the answer appears in both supporting paragraphs, we assume that it is a conjunction question. The hints for such questions are:
$p_1 = d_1; a_1 = a; z_1 = \zeta(qc, d_1, d_2)$
$p_2 = d_2; a_2 = a; z_2 = \zeta(qc, d_2, d_1)$

While decomposition datasets such as BREAK could be used to obtain more direct supervision for these hints, we focus here on the broader feasibility of distant supervision. We observe that our current approach generates hints for 89% of the questions and can find decompositions that lead to the gold answer for 50% of them. So while the hints cannot

[8]If not found, we ignore such questions.
[9]We use the same for comparison questions in HotpotQA.

be used directly to produce decompositions, the next-question generator is able to generalize from these examples to generate decompositions for all questions with 81% of them leading to the gold answer. App. D provides more details and example of hints for each question class.

As described earlier, given these input hints and our sub-task question models, we can generate the sub-question for each step and the appropriate sub-model (based on the model that produced this question). We use nucleus sampling to sample 5 questions for each reasoning step. To improve the training data quality, we also filter out potentially noisy decompositions.[10] We train a BART-Large model, our next-question generator, on this training data to produce the next question given the complex question and previous question-answer pairs.

4.4 Inference

We use best-first search (Dijkstra et al., 1959) to find the best decomposition chain and use the answer produced at the end of the chain as our predicted answer. We sample n_0 sub-questions from the next-question generator using nucleus sampling. Each question is then answered by the appropriate QA sub-model (defined by the prefix in the question). This partial chain is again passed to the next-question generator to generate the next n_1 sub-questions, and so on.[11] A chain is considered complete when the next-question generator outputs the end-of-chain marker `[EOQ]`.

We define a scoring function that scores each partial chain u based on the new words introduced in the sub-questions compared to the input question.[12] For a complete chain, we additionally add the score from a RoBERTa model trained on randomly sampled chains (chains that lead to the correct answer are labeled as positive). Concretely, we use the negative class score from this classifier, $\delta(u)$, to compute the final chain score as $\theta(u) + \lambda\delta(u)$, i.e., lower is better.[13]

5 Experiments

To evaluate our modular approach, we use two datasets, DROP and HotpotQA, that contain questions answerable using a SQuAD model and a math calculator. We identify 14.4K training questions in DROP that are within the scope of our system,[14] which forms 18.7% of the dataset.[15] We similarly select 2973 Dev questions (from 9536), and split them into 601 Dev and 2371 Test questions.

We evaluate our system on the entire HotpotQA dataset. Since the test set is blind, we split the Dev set (7405 qns.) into 1481 Dev and 5924 Test questions. For training, we only use 17% of the training dataset containing 15661 questions categorized as "hard" by HotpotQA authors.[16]

5.1 Explanation and Interpretability

A key aspect distinguishing ModularQA is that it can explain its reasoning in a human-interpretable fashion, in the form of simpler sub-questions it creates via decomposition. Table 1 illustrates six sample reasoning explanations; the question context and sub-models are omitted for brevity. We see that ModularQA is able to take oddly phrased questions to create clean sub-questions (example 4), handle yes/no questions (example 6), recognize the unit of comparison (example 1), and map the phrase "smaller" to the appropriate direction of comparison without any manual rules (example 2).

Analyzing such explanations for 40 Dev questions (20 from each dataset), we found that among the 28 questions ModularQA answered correctly, it produced a valid reasoning chain in as many as 93% of the cases, attesting to its strong ability to provide understandable explanations.

To further assess the human readability of ModularQA's explanations, we compared them with those produced by DecompRC (Min et al., 2019b), the only decomposition-based system for the considered datasets. We identified 155 questions that are within the scope of ModularQA and for which both systems produce a decomposition.[17] We then asked crowdworkers on Amazon Mechanical Turk to annotate them along three dimensions: (1) given the two explanations, which system's answer do they **trust** more; (2) which system's explanation do they **understand** better; and (3) which system's explanation do they generally **prefer**.

[10] if an intermediate answer is unused or vocabulary of question chain is too different from the input question. See Appendix A.3 for more details.

[11] To enable early exploration, we use exponential decay on the number of generated questions: $n_i = 10/2^i$.

[12] $\theta(u)$ = #new words/#words in input question

[13] For more details, refer to App. A.4.

[14] See App. D for how this subset is automatically identified.

[15] Previous modular systems (Gupta et al., 2020) have targeted even smaller subsets to develop modular approaches.

[16] Increasing the training set didn't affect performance.

[17] DecompRC failed to produce chains on 6x more questions than our system. See App. C for details on how these questions were selected and how they were normalized.

	DROP F1				HotpotQA F1		
	All	Diff	Comp	Cmpl	All	Br	Comp
Interpretable Cross-Dataset Models (§5.2)							
ModularQA	87.9	85.2	81.0	96.6	61.8	64.9	49.2
WordOverlap	80.5	82.5	58.3	95.8	57.5	61.7	40.5
Greedy	60.2	52.2	52.9	76.3	42.4	44.8	33.0
Limited Versatility (§5.3)							
NMN-D†	79.1*	–	–	–	–	–	–
SNMN†	–	–	–	–	63.1	63.7	60.1
DecompRC†	–	–	–	–	70.3	72.1	63.4
Limited Interpretability (§5.4)							
NumNet+V2	91.6	86.5	94.5	95.5	–	–	–
Quark	–	–	–	–	75.5	78.1	64.9

Table 2: F1 scores on the DROP and HotpotQA questions and the individual classes: Difference(Diff), Comparison(Comp), Complementation(Cmpl) and Bridge(Br). TOP: Comparison to variations of ModularQA that work across datasets. MIDDLE: Comparison to *targeted* interpretable systems. BOTTOM: Comparison to *targeted blackbox* systems. ModularQA is competitive with previous approaches on DROP and mainly lags behind systems on HotpotQA that are able to exploit artifacts.

	Trust	Understand	Prefer
DecompRC	50 (33%)	34 (22%)	49 (32%)
ModularQA	105 (67%)	121 (78%)	106 (68%)

Table 3: Human evaluation of the explanation quality. Across all dimensions, crowdsource workers preferred the explanations of ModularQA over DecompRC.

Table 3 summarizes the aggregate statistic of the majority labels, with 5 annotations per question. Crowdworkers understood ModularQA's natural language explanations better in 78% of the cases, trusted more that it pointed to the correct answer, and generally preferred its explanations.

5.2 Interpretable Cross-Dataset Models

With ModularQA being the first interpretable model for DROP and HotpotQA, there were no comparable existing cross-dataset systems. We instead consider two baselines obtained by modifying ModularQA: (1) only the word-overlap based scoring function $\theta(u)$ for chains (no RoBERTa classifier); and (2) greedy inference, i.e., use the most likely question at each step (no search).

As shown in Table 2 (top rows), ModularQA outperforms the purely word-overlap based approach by 7pts F1 on DROP and 4pts on HotpotQA. A simple coverage-based decomposition is thus not as effective, although HotpotQA suffers less because of decompositions being explicit in it.[18] Performance drops much more heavily (18pts on DROP and 19pts on HotpotQA) when we do not employ search at all. This is primarily because the optimal sub-question can often be unanswerable by the intended sub-model while an alternate decomposition may lead to the right answer.

[18]Recall that our word-overlap based score penalizes missed question words and words introduced during decomposition.

5.3 Comparison to Dataset-Specific Models

To assess the price ModularQA pays for being versatile, we compare it to three interpretable systems that target a particular dataset. Two are Neural Module Networks, with modules designed specifically for a subset of DROP (referred to as NMN-D) (Gupta et al., 2020) and for HotpotQA (referred to as SNMN) (Jiang and Bansal, 2019). The third is DecompRC, whose split-based decomposition, human annotations, as well as answer composition algorithm was specifically designed for HotpotQA.

As seen in Table 2 (middle rows), ModularQA actually substantially outperforms the DROP model NMN-D while being able to produce textual explanations (rather than attention visualization).[19] On the HotpotQA dataset, ModularQA is comparable to S-NMN but underperforms compared to DecompRC. Note that DecompRC can choose to answer some questions using single-hop reasoning and potentially exploit many artifacts in this dataset (Min et al., 2019a; Trivedi et al., 2020).

5.4 Comparison to Black-Box Models

To assess the price ModularQA pays for being interpretable, we compare it to two state-of-the-art black-box systems that not only lack interpretability but are also targeted towards specific datasets: NumNet+V2 (Ran et al., 2019) for DROP

[19]Since NMN-D focuses on a different subset, we report its score on the shared subset, on which ModularQA achieves an F1 score of 92.5 (not shown in the table)

and Quark (Groeneveld et al., 2020) for HotpotQA. Since we use the SQuAD QA system in our model, we first fine-tune the LM in both of these systems on the SQuAD dataset, and then train them on the same datasets as MODULARQA.

As seen in Table 2 (bottom rows), we are competitive with the state-of-the-art model on DROP but underperform compared to the Quark system. Note that Quark relies on supporting fact annotation and trains a single end-to-end QA model, thereby being more likely to exploit dataset artifacts.

Upon analyzing MODULARQA's errors (defined as questions with F1 score under 0.5) on HotpotQA, we found 65% of the errors arise from intermediate questions having multiple or yes/no answers. These are not handled by modules in our current implementation, suggesting a path for improvement.

We also analyzed the errors on the DROP dev set and identified question decomposition (53.3%) and QA models (33.33%) as the main sources of error.[20] Within question decomposition, the key cause of error is higher RoBERTa score for an incorrect decomposition (50% of errors). Both the SQuAD and Math QA models were responsible for errors, with the latter erring only due to out-of-scope formats (e.g., date ranges 1693-99). Appendix E provides more details.

5.5 Additional Benefits of TMNs

The last set of experiments support two distinct benefits (besides interpretability) of our approach even against state-of-the-art black-box models.

Higher Robustness. We evaluate on the DROP *contrast set* (Gardner et al., 2020), a suite of test-only examples created for assessing robustness via minimally perturbed examples. On the 239 (out of 947) questions that are within our scope using the same logic as before, we find that MODULARQA outperforms NumNet+V2 by 7%-10%:

Contrast Test	EM	F1
MODULARQA	**55.7**	**63.3**
NumNet+V2	45.2	56.2

Learning with Less Data. We next evaluate the sample efficiency of MODULARQA by considering training sets of 3 different sizes: 100%, 60%, and 20% (14448, 8782, and 2596 questions, resp.) of the training questions selected for DROP.[21] As shown below, the gap (in F1 score) between MODULARQA and the state-of-the-art model steadily shrinks, and MODULARQA even outperforms it when both are trained on 20% of the data.

Portion of Train set	100%	60%	20%
MODULARQA	87.8	**89.3**	**87.0**
NumNet+V2	**91.6**	88.3	85.4

6 Conclusion & Future Work

We introduced *Text Modular Networks*, which provide a general-purpose framework that casts complex tasks as *textual* interaction between existing, simpler QA modules. Based on this conceptual framework, we built MODULARQA, an instantiation of TMNs that can perform multi-hop and discrete numeric reasoning. Empirically, MODULARQA is on-par with other modular approaches (which are dataset-specific) and outperforms a state-of-the-art model in a limited data setting and on expert-generated perturbations. Importantly, MODULARQA provides easy-to-interpret explanations of its reasoning. It is the first system that decomposes DROP questions into textual sub-questions and can be applied to both DROP and HotpotQA.

Extending this model to more question classes such as counting ("How many touchdowns were scored by X?") and Boolean conjunction ("Are both X and Y musicians?") are interesting avenues for future work. To handle the former class, the first challenge is building models that can return a list of answers—a relatively unexplored task until recently (Hu et al., 2019b; Segal et al., 2020). For Boolean questions, the challenge is identifying good sub-questions as there is a large space of questions such as "Did musicians work for X?" that may have the expected yes/no answer but are not part of the true decomposition. Semantic parsing faces similar issues when questions have a large number of possible logical forms (Dasigi et al., 2019). Finally, end-to-end training of the next-question generator and QA models via REINFORCE (Williams, 1992) can further improve the score and allow for faster greedy inference.

Acknowledgements

We thank the Aristo team at AI2 for helpful input, Beaker team for their support with experiments, Dirk Groeneveld for providing the output of the Quark system for evaluation, and Jonathan Berant, Matt Gardner, and Hanna Hajishirzi for invaluable feedback on initial drafts of this paper.

[20]Remaining errors are due to dataset and scope issues.

[21]For simplicity, we train MODULARQA on the DROP questions only here. To obtain sufficient examples, we increase the number of questions sampled for each decomposition step. See App. A.6 for more details.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *NAACL-HLT*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP*.

Gino Brunner, Y. Liu, Damián Pascual, Oliver Richter, and Roger Wattenhofer. 2020. On identifiability in transformers. In *ICLR*.

Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. 2019. Iterative search for weakly supervised semantic parsing. In *NAACL*.

Edsger W Dijkstra et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL*.

Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. 2020. Hierarchical graph network for multi-hop question answering. In *EMNLP*.

Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hanna Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Quan Zhang, and Ben Zhou. 2020. Evaluating NLP models via contrast sets. In *Findings of EMNLP*.

Dirk Groeneveld, Tushar Khot, Ashish Sabharwal, et al. 2020. A simple yet strong pipeline for hotpotqa. In *EMNLP*.

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

Sanda Harabagiu and Andrew Hickl. 2006. Methods for using textual entailment in open-domain question answering. In *ACL*.

Chris Hokamp and Qun Liu. 2017. Lexically Constrained Decoding for Sequence Generation using Grid Beam Search. In *ACL*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *ICLR*.

J Edward Hu, Huda Khayrallah, Ryan Culkin, Patrick Xia, Tongfei Chen, Matt Post, and Benjamin Van Durme. 2019a. Improved Lexically Constrained Decoding for Translation and Monolingual Rewriting. In *NAACL*.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dong sheng Li. 2019b. A multi-type multi-span network for reading comprehension that requires discrete reasoning. In *EMNLP*.

Alon Jacovi and Y. Goldberg. 2020. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In *ACL*.

Yichen Jiang and Mohit Bansal. 2019. Self-assembling modular networks for interpretable multi-hop reasoning. In *EMNLP-IJCNLP*.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *NAACL*.

Tushar Khot, Peter Clark, Michal Guerquin, Paul Edward Jansen, and Ashish Sabharwal. 2020. QASC: A dataset for question answering via sentence composition. In *AAAI*.

Louis Kirsch, Julius Kunze, and David Barber. 2018. Modular networks: Learning to decompose neural computation. In *NeurIPS*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In *ACL*.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

John McCarthy. 1988. Epistemological challenges for connectionism. *Behavioral and Brain Sciences*, 11(1):44–44.

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019a. Compositional questions do not necessitate multi-hop reasoning. In *ACL*.

Sewon Min, Victor Zhong, Luke S. Zettlemoyer, and Hannaneh Hajishirzi. 2019b. Multi-hop reading comprehension through question decomposition and rescoring. In *ACL*.

Dan I. Moldovan, Sanda M. Harabagiu, Marius Pasca, Rada Mihalcea, Roxana Girju, Richard Goodrum, and Vasile Rus. 2000. The structure and performance of an open-domain question answering system. In *ACL*.

Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. In *ICLR*.

Ethan Perez, Patrick Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. 2020. Unsupervised question decomposition for question answering. In *EMNLP*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. Numnet: Machine reading comprehension with numerical reasoning. In *EMNLP-IJCNLP*.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. 2018. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *ICLR*.

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. *TACL*, 6:159–172.

Elad Segal, Avia Efrat, Mor Shoham, Amir Globerson, and Jonathan Berant. 2020. A simple and effective model for answering multi-span questions. In *EMNLP*.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *EMNLP*.

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In *ACL*.

Paul Smolensky. 1988. On the proper treatment of connectionism. *Behavioral and brain sciences*, 11(1):1–23.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *NAACL*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2020. Is multihop qa in dire condition? measuring and reducing disconnected reasoning. In *EMNLP*.

Ming Tu, Kevin Huang, Guangtao Wang, Jui-Ting Huang, Xiaodong He, and Bufang Zhou. 2020. Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. In *AAAI*.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *EMNLP/IJCNLP*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *TACL*, 8:183–198.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.

A Model Settings

Each BART-Large model (406M parameters) is trained with the same set of hyper-parameters – batch size of 64, learning rate of 5e-6, triangular learning rate scheduler with a warmup of 500 steps, and training over 5 epochs. Each RoBERTa model is trained with the same set of hyper-parameters but a smaller batch size of 16. We selected these parameters based on early experiments and did not perform any hyper-parameter tuning thereafter. All the baseline models are trained with their default hyper-parameters provided by the authors. All the experiments are performed on single-GPU machines (either V100 GPUs or RTX 8000s).

We always used nucleus sampling to sample sequences from the BART models. To sample the sub-question using the SQuAD sub-question generator, we sampled 5 questions for each step with p=0.95 and max question length of 40. To sample the question decompositions during inference, we additionally set k=10 to reduce the noise in these questions.

A.1 Training SQuAD Question Generator

We use the SQuAD 2.0 answerable questions to generate the training data for our SQuAD question generator. We use the nouns, verbs, nouns, adjectives and adverbs (pos tags=[NOUN, VERB, NUM, PROPN, ADJ, RB]) from the question to define the vocabulary hints (after filtering stop words). To simulate the noisy vocabulary, we also add distractor terms with similar pos tags from other questions from the same paragraph. We sample $j \in [2, ..., 7]$ distractor terms for each question and add them to the vocabulary hints.

A.2 Generating sub-questions

For every step in the reasoning process, we generate 5 questions using nucleus sampling. We select the questions that the corresponding sub-model is able to answer correctly. For each sub-question, we generate 5 questions in the next step (and so on). At the end, we select all the successful question chains (i.e each sub-question was answered by the sub-model to produce the expected answer at each step).

A.3 Selecting Question Decompositions

It is possible that some of these sub-questions, while valid answerable questions, introduce other words mentioned in the paragraph. However, these may not be valid decompositions of the original question. E.g., for the complex question: "When was the 44th US President born?", the sub-question may state "Who was the 44th President from Hawaii?". While this a valid question with the expected intermediate answer, it introduces irrelevant words that would not be possible for the next-question generator to learn.

To filter out such potentially noisy decompositions, we compute three statistics based on non-stopword overlap. We compute the proportion of new words introduced in a decomposition $u = \{..., q_i, a_i, ...a_n\}$ that were not in the input question or any of the previous answers, that is:

$$\theta(u) = \frac{\left|\bigcup_i \{w \in q_i \mid w \notin qc \text{ and } \forall j < i \; w \notin a_j\}\right|}{\left|\{w \in qc\}\right|}$$

We also compute the number of words from the input question not covered by the decomposition:

$$\mu(u) = \frac{\left|\{w \in qc \mid \forall i \; w \notin q_i\}\right|}{\left|\{w \in qc\}\right|}$$

Lastly, we compute the number of answers ν that were not used in any subsequent question, i.e., the sub-question associated with this answer is irrelevant:

$$\nu(u) = \big|\{a_i \mid \neg(\exists w \in a_i \text{ s.t. } w \in q_j \text{ where } j > i \text{ or } w \in a_n)\}\big|$$

We only select the decompositions where $\theta < 0.3$, $\mu < 0.3$, $\theta + \mu < 0.4$, and $\nu = 0$. To prevent a single question from dominating the training data, we select upto 50 decompositions for any input question. These hyper-parameters were selected early in the development and gave reasonable results. Minor variations did not have a substantial impact and hence were not tuned on the target set.

A.4 Inference Parameters

We sample n_i questions in the i^{th} question decomposition step. To ensure sufficient exploration of the search space, we initially sample a larger number of questions but scale them down every step for efficiency. Due to the pipeline nature of our system, it is difficult for our model to recover from any missed question early in the search. We set the number of sampled questions as $n_i = N * r^i$ where N=15 and $r = \frac{1}{2}$. When we use greedy inference, we just sample one most-likely question using beam search with width=4. For the QA

	E	G	R	DROP All EM | F1	DROP Diff F1	DROP Comp F1	DROP Cmpl F1	HotpotQA All EM | F1	HotpotQA Bridge EM | F1	HotpotQA Comp. EM | F1
ModularQA	✓	✓	✓	86.6 | 87.9	85.2	81.0	96.6	48.5 | 61.8	50.7 | 64.9	39.8 | 49.2
BART	×	✓	×	26.7 | 28.0	7.7	77.5	12.8	38.0 | 49.0	40.4 | 52.9	28.0 | 33.7
NMN-D†	✓	×	×	71.0 | 79.1*	–	–	–	–	–	–
NumNet+V2	×	×	×	90.6 | 91.6	86.5	94.5	95.5	–	–	–
SNMN†	✓	×	×	–	–	–	–	50.0 | 63.1	48.8 | 63.8	54.8 | 60.1
Quark	×	×	×	–	–	–	–	61.7 | 75.5	62.5 | 78.1	58.3 | 64.9

Table 4: Expanded version of the quantitative part of Table 2, reporting both F1 and EM scores in each case. The first three columns, as before, denote qualitative capabilities of each model: whether it can **E**xplain its reasoning, **G**eneralize well to multiple datasets, or **R**e-use existing QA models.

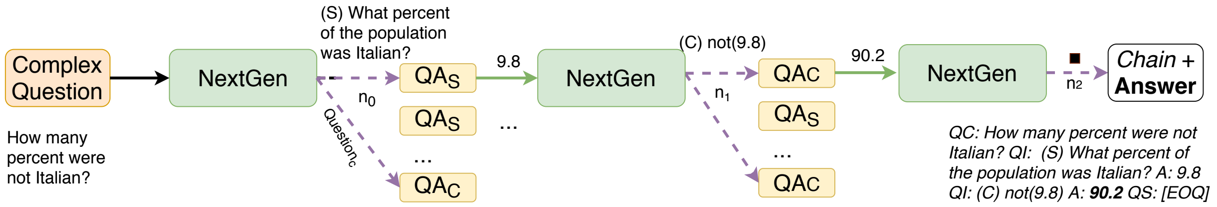

Complex Question
How many percent were not Italian?
NextGen
(S) What percent of the population was Italian?
n_0
Question$_C$
QA$_S$
QA$_S$
...
QA$_C$
9.8
...
NextGen
(C) not(9.8)
n_1
QA$_C$
QA$_S$
...
QA$_C$
90.2
NextGen
■
n_2
Chain + Answer
QC: How many percent were not Italian? QI: (S) What percent of the population was Italian? A: 9.8 QI: (C) not(9.8) A: 90.2 QS: [EOQ]

Figure 5: A sample inference chain scored by ModularQA for a negation DROP question. For each n_0 question generated in the first step, we will explore n_1 questions in the second step (and so on). We use our scoring function w to select the optimal inference chain+answer (and prune incomplete low-scoring chains).

models, we always select the most likely answer. When there are multiple input paragraphs (e.g., HotpotQA), we run the QA model against each paragraph independently and select the most likely answer based on the probability.

To score each generated question, we again rely on the same word-overlap statistic used to filter decompositions. We only use the θ metric that captures the number of new words introduced in a question chain. The other two metrics are non-motonic i.e they could go down depending on future questions and answers in the chain. At the end, we use a chain scorer (described next) to score each decomposition chain. While we use the θ metric to guide the search, we primarily rely on the chain score δ to select the right answer. As a result, the final score for a chain u is a weighted combination of these two metrics with higher emphasis on δ

$$\text{score}(u) = \theta(u) + \lambda\delta(u)$$

where λ=10 (was set initially during development and not fine-tuned). δ can only be computed for a complete decomposition and is set to zero for the intermediate steps. Note that higher this score, the worse the chain i.e. we need to find the chain with the lowest score. This scoring function is monotonically increasing as any continuation of a chain will have the same or higher score. We can thus ignore any partial chains with higher scores once we find a complete chain with the lowest score.

A.5 Chain Scorer

To train the scorer, we first collect positive and negative chains by running inference with just the θ metric. For efficiency, we set the inference parameter N=5 here. For every complete chain, we compute the F1 score of the final answer with the gold answer. If the F1 score exceeds a threshold (0.2 in our case), we assume this chain to be a positive example. We collect such positive and negative chain examples from the training set and then train a RoBERTa model to classify these chains. We use the RoBERTa model's predicted probability for the negative class as the score δ.

A.6 Less Data Training

Since we sample the training data for our next-question generator, we can generate more training data by sampling more questions. When training on 20% of the training data, i.e. only 2600 questions, we sample 15 questions at each step when we are generating the sub-questions(App. A.2). Similarly we increase N=10 to generate more chains for the chain scorer(App. A.5).

B Additional Results

Table 4 expands upon the quantitative results in Table 2 and reports both F1 and EM (exact match)

Operation	QA Model, $\mathbb{A}_C$	Question Generator, $\mathbb{G}_C$
`diff`(X, Y, [Z])	Return absolute difference between X and Y. If $Z \in$ {days, months, years}, find the difference in Z.	Generate questions with all possible date/number pairs as X,Y. If $Z \in$ {days, months, years} is mentioned in the question, add Z
`not`(X)	Return 100 - X	Generate questions for every number ≤ 100 as X.
`if_then`(X [<>≠] Y, Z, W)	If X is [<>≠] Y, return Z else return W	Generate questions with all possible date/number pairs as X and Y. Use pair of entities in the question as Z and W.

Table 5: Set of operations handled by the symbolic calculator model $\mathbb{A}_C$ and the corresponding approach to generate such questions in $\mathbb{G}_C$.

scores in each setting considered.

C Human Evaluation

To identify a subset of questions that *might* be within the scope of our system, we used the human-authored BREAK decompositions (Wolfson et al., 2020) and filtered out questions that require Boolean operations or list operations. We identified the former by the presence of patterns such as "Which is true" and latter by the presence of plural return terms in the sub-questions.

On this resulting subset of 253 questions, MODULARQAis comparable to the DecompRC system with only a 2pt F1 gap. Out of the 253 in-scope dev questions, DecompRC did not produce a chain at all (i.e., relied on single-hop reasoning) for 79 questions, whereas MODULARQA did produce a chain. In contrast, MODULARQA failed on only 12 questions on which DecompRC succeeded in producing a chain (both systems failed on 3 questions).

We used crowdworkers to annotate the generated explanations[22] on 155 questions where both systems produced a chain. The annotation covered three dimensions: (1) given the explanation, which system's answer do they **trust**; (2) which system's explanation do they **understand** better; and (3) overall, which system's explanation do they **prefer** (subjective).

D Hints for Complex QA Tasks

To apply Text Modular Networks to any complex QA dataset, we need to be able to extract the hints needed by the sub-task question model. As mentioned earlier, these need not have full coverage or have 100% precision.

[22]We normalized explanations from both the systems for a fair comparison, e.g., lower-cased our explanations, used the system's answers for both, and converted symbolic terms in both explanations to natural language.

D.1 HotpotQA

The questions qc in HotpotQA have two supporting gold documents: d_1 and d_2. Additionally they are also partitioned into two classes: Bridge and Comparison questions.

D.1.1 Bridge Questions

There are two forms of bridge questions in HotpotQA:

Composition questions: These questions need to first find an intermediate entity e_1 that is referred by a sub-question in HotpotQA. This intermediate entity points to the final answer through the second sub-question. Generally this intermediate entity is the title entity of the document containing the answer. Say d_2 is the document containing the answer and d_1 is the other document. If we are able to find a span that matches the title of d_2 in d_1 and the answer only appears in d_2, we assume it to be a composition question. We set e_1 to the span that matches the title of d_2 in d_1.

For the question vocabulary, we could use the terms from the entire question for both steps. Also the second sub-question will use the answer of the first sub-question, so we add it to the vocabulary too. However, we can reduce some noise by removing the terms that are exclusively appear in the other document. The final hints for this question are:

$$p_1 = d_1; a_1 = e_1; z_1 = \zeta(qc, d_1, d_2)$$
$$p_2 = d_2; a_2 = a; z_2 = \zeta(qc, d_2, d_1) + e_1$$

where $\zeta(q, d_1, d_2)$ indicates the terms in q that appear in d_2 but not in d_1.

Conjunction questions: These class of questions do not have any intermediate entity but have two sub-questions with the same answer e.g. “Who is a politician and an actor?”. If the answer appears in both supporting paragraphs, we assume that it is a conjunction question. The hints for such

Complex Q	Input Hints z	Output Sub-Questions
qc: *How many years did it take the services sector to rebound after the 2002 decrease?*	a_1: **2002**, v_1: $\Phi(qc)$, p_1: p	q_1: *When did the services sector take a decrease?*; t_1: $\mathcal{S}$
	a_2: **2003**, v_2: $\Phi(qc)$, p_2: p	q_2: *When did the services sector rebound?*; t_2: $\mathcal{S}$
	a_3: **1**, v_3: [“diff”, “2002”, “2003”], p_3: p	q_3: `diff`*(2002, 2003)*; t_3: $\mathcal{C}$
qc: *Which ancestral group is smaller: Irish or Italian?*	a_1: **12.2**, v_1: $\Phi(qc)$, p_1: p	q_1: *How many of the group were Irish?*; t_1: $\mathcal{S}$
	a_2: **6.1**, v_2: $\Phi(qc)$, p_2: p	q_2: *How many Italian were there in the group?*; t_2: $\mathcal{S}$
	a_3: **1**, v_3: [“`if_then`”, “12.2”, “6.1”], p_3: p	q_3: `if_then`*(12.2 < 6.1, Irish, Italian)*; t_3: $\mathcal{C}$
qc: *How many percent of the national population does not live in Bangkok?*	a_1: **12.6**, v_1: $\Phi(qc)$, p_1: p	q_1: *What percent of the national population lives in Bangkok?*; t_1: $\mathcal{S}$
	a_2: **87.4**, v_2:[“`not`”, “12.6”] , p_2: p	q_1: `not`*(12.6)*; t_2: $\mathcal{C}$
qc: *Little Big Girl was a Simpsons episode directed by the animator and artist of what nationality?*	a_1: **Raymond S Persi**, v_1: $\zeta(qc, d_1, d_2)$, p_1: d_1	q_1: *Who directed “Little Big Girl”?*; t_1: $\mathcal{S}$
	a_2: **American**, v_2: $\zeta(qc, d_2, d_1)+\ a_1$, p_2: d_2	q_1: *What nationality was Raymond S?*; t_2: $\mathcal{S}$

Table 6: Sample hints and the resulting decomposition for DROP and HotpotQA examples. The function Φ selects non-stopword words and $\zeta(q, d_1, d_2)$ selects the words from $\Phi(q)$ that don't exclusively appear in d_2.

questions are simple:

$$p_1 = d_1; a_1 = a; z_1 = \zeta(qc, d_1, d_2)$$
$$p_2 = d_2; a_2 = a; z_2 = \zeta(qc, d_2, d_1)$$

D.2 Comparison Questions

These questions compare certain attribute between two entities/events mentioned in the question. E.g., “Who is younger: X or Y?”. We identify the two entities e_1 and e_2 in such questions and find dates/numbers that are mentioned in documents. For every n_1, n_2 number/date mentioned in the document d_1 and d_2 respectively, we create the following hints:

$$p_1 = d_1; a_1 = n_1; z_1 = \zeta(qc, d_1, d_2)$$
$$p_2 = d_2; a_2 = n_2; z_2 = \zeta(qc, d_2, d_1)$$
$$p_3 = \phi; a_3 = a; z_3 = [\texttt{if\_then}, n_1, n_2, e_1, e_2]$$

The final set of hints would be used by the calculator generator to create the questions: $\texttt{if\_then}(n_1 > n_2, e_1, e_2)$ and $\texttt{if\_then}(n_1 < n_2, e_1, e_2)$.

D.3 DROP

For the questions in DROP, we first identify the class of question that it may belong to and then generate the appropriate hints. Note that one question can belong to multiple classes and we would generate multiple sets of hints in such cases. The questions qc in DROP have only one associated context p.

D.3.1 Difference Questions

We identify these questions based on the presence of term indicating a measurement: "how many" and terms indicating difference such as “shorter’, “more”, “days between”, etc. We remove questions that match patterns indicating counting or minimum/maximum such as “shortest”, “how many touchdown”, etc. Table 7 shows the regexes that must match and ones that must not match for a question to be categorized as a difference question.

Finally we check for two dates or numbers in the context such that their difference (in all units) can lead to the final answer. If these conditions are satisfied, for every pair n_1, n_2 where the difference (in units u) can lead to the final answer, we generate the hints:

$$p_1 = p; a_1 = n_1; v_1 = \Phi(qc)$$
$$p_2 = p; a_2 = n_2; v_2 = \Phi(qc)$$
$$p_3 = \phi; a_3 = a; v_3 = [\texttt{diff}, n_1, n_2, u]$$

D.3.2 Comparison questions

We identify these questions based on the presence of the pattern: “ques: e_1 or e_2”(specifically we match “`([^,]+)[:,](.*) or (.*)\?`”). We handle them in exactly the same way as HotpotQA. Since DROP contexts can have more dates and numbers, we select numbers and dates that are close to the entity mentioned (Gupta et al.,

Must match	Should not match
".*how many (days|months|years).*", ".*how many.*(days|months|years).* between .*", ".*how many.* shorter .+ than .*", ".*how many.* shorter .+ compar.*", ".*how many.* longer .+ than .*", ".*how many.* longer .+ compar.*", ".*how many.* less .+ than .*", ".*how many.* less .+ compar.*", ".*how many.* more .+ than .*", ".*how many.* more .+ compar.*", ".*difference.*"	".*minimum.*", ".*maximum.*" ".*longest.*", ".*shortest.*", ".*highest.*", ".*lowest.*", ".*first.*", ".*last.*", ".*second.*", ".*third.*", ".*fourth.*", ".*how many touchdown.*", ".*how many field goal.*", ".*how many point.*", ".*more touchdown.*", ".*more field goal.*", ".*more point.*"

Table 7: Regexes used to identify difference questions and filter out false positives.

Type	Sub-Type	Description	#Errs
Decomposition	Low Score	RoBERTa chain scorer model returned a lower score for the correct decomposition	5
	Incomplete	The decomposition missed a key part of the complex question (e.g. "When did Killgrew marry?" instead of "When did Killgrew marry Catherine?")	3
	Sampling	A correct and better scoring decomposition exists but was not generated during the search	3
	Long Question	A sub-question exceeded the max token length of 40 set during question generation	2
	Missed Decomp	Valid decomposition was not generated	2
	Noisy Q	Questions in the generated decomposition were ill-formed	1
QA	Incorrect Ans	SQuAD QA model produced an incorrect answer	3
	No Answer	SQuAD QA model produced no answer i	2
	Partial Answer	SQuAD QA model produced a partial answer span	1
	MathQA format mismatch	Math QA model was unable to handle input format (e.g. `if_then(1683-99 > 1591-92,...`	4
Out-of-scope		Question can not be handled by our sub-models	2
Dataset		Question makes assumptions not stated in text	2

Table 8: Break down of errors in 30 DROP questions incorrectly answered (F1 < 0.5) by ModularQA

2020).

$$p_1 = p; a_1 = n_1; v_1 = \Phi(ques) + e_1$$
$$p_2 = p; a_2 = n_2; v_2 = \Phi(ques) + e_2$$
$$p_3 = \phi; a_3 = a; v_3 = [\texttt{if\_then}, n_1, n_2, e_1, e_2]$$

D.3.3 Complementation questions

We identify these questions purely based on the presence of “.* not .*” in the question(specifically we match “`^(.*percent.*)(\Wnot\W|n't\W)(.*)$`“) and a number in the context n_1 such that the $a = 100 - n_1$. The hints are pretty straightforward too:

$$p_1 = p; a_1 = n_1; v_1 = \Phi(qc)$$
$$p_2 = \phi; a_2 = a; v_2 = [\texttt{not}, n_1]$$

E Drop Error Analysis

See Table 8 for the different error types and their counts. Since the search does not explore all possible decompositions, it is possible that there are other decompositions not considered in this analysis. For example, we marked a question to have an error due to “sampling” if running inference again found a higher scoring, valid decomposition that led to the correct answer. However, it is possible that an exhaustive search would find an invalid decomposition with an even lower score. Similarly the error cases due to "Incomplete" decomposition or "No Valid" decomposition could also be due to sampling issues.