Successive Prompting for Decomposing Complex Questions

Dheeru Dua[♣] **Shivanshu Gupta**[♣] **Sameer Singh**[♣,♠] **Matt Gardner**[♡]
[♣]University of California, Irvine, USA [♠]Allen Institute for Artificial Intelligence
[♡]Microsoft Semantic Machines
{ddua,shivag5,sameer}@uci.edu, mattgardner@microsoft.com

Abstract

Answering complex questions that require making latent decisions is a challenging task, especially when limited supervision is available. Recent works leverage the capabilities of large language models (LMs) to perform complex question answering in a few-shot setting by demonstrating how to output intermediate rationalizations while solving the complex question in a single pass. We introduce "Successive Prompting", where we iteratively break down a complex task into a simple task, solve it, and then repeat the process until we get the final solution. Successive prompting decouples the supervision for decomposing complex questions from the supervision for answering simple questions, allowing us to (1) have multiple opportunities to query in-context examples at each reasoning step (2) learn question decomposition separately from question answering, including using synthetic data, and (3) use bespoke (fine-tuned) components for reasoning steps where a large LM does not perform well. The intermediate supervision is typically manually written, which can be expensive to collect. We introduce a way to generate a synthetic dataset which can be used to bootstrap a model's ability to decompose and answer intermediate questions. Our best model (with successive prompting) achieves an improvement of ~5% absolute F1 on a few-shot version of the DROP dataset when compared with a state-of-the-art model with the same supervision.

1 Introduction

Compositional reading comprehension datasets like HotpotQA (Yang et al., 2018) and DROP (Dua et al., 2019) have inspired a range of model architectures that learn to answer complex questions with weak supervision from the final answer. One recent direction is to leverage large language models (LMs) to solve compositional tasks with very few examples by generating latent reasoning steps before answering the question (Wei et al., 2022; Nye et al., 2021; Karpas et al., 2022). Given a complex question, this approach first finds nearest-neighbor training examples from a dataset of (question, reasoning, answer) triples and then concatenates them to create an input for the LM. A large LM is then prompted with this input to generate the intermediate reasoning steps needed, while answering the complex question in a single pass.

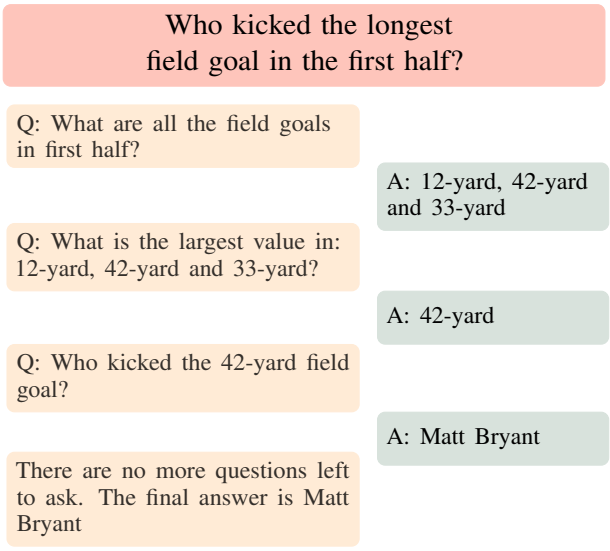

Figure 1: Example decomposition used by Successive Prompting's question decomposition and question answering stage on a DROP example. The model iterates between predicting a simple question to ask and answering the simple question.

While promising, this approach discards many of the benefits of prior approaches to this task (Khot et al., 2021; Karpas et al., 2022) by coupling the supervision for question decomposition to the supervision for performing the intermediate steps. Moreover, its non-modular nature does not allow using alternate symbolic reasoning engines in cases where they perform better than LMs. Additionally, the model gets exposed to only a single set of in-context examples, selected based on their proximity to the complex question, which may not contain

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 1251 - 1265

optimal supervision for the intermediate steps that need to be taken.

We propose "Successive Prompting", where we iteratively decompose the complex question into the next simple question to answer, answer it, and then repeat until the complex question is answered (Figure 1). Each of these steps is performed with separate a query to the LM. Since the decomposition and answering steps are performed separately, we can decouple the supervision of each step, providing two primary benefits. First, when performing in-context learning, we get multiple opportunities to select different in-context examples, which can be tailored to the particular decomposition or answering step being performed, instead of selecting a single set of examples based only on the complex question. Second, when fine-tuning (with or without in-context examples (Chen et al., 2022)), we can provide training examples for each step independently, so the model only has to learn to perform one step at a time.

This decoupling additionally allows us to judiciously inject synthetic data into the learning process, e.g., to help the model answer a particular kind of simple question that it could not previously answer, or a new reasoning composition it did not know how to decompose. Because the steps are separate, we can isolate model failures and develop synthetic approaches to fill in the gaps. It also allows us to replace the LM with other, purpose-built components to perform symbolic reasoning when appropriate (Khot et al., 2021; Segal et al., 2020; Jin et al., 2021).

We demonstrate the utility of successive prompting using a few-shot variant of the DROP dataset (Dua et al., 2019), selecting 300 examples for training (either fine-tuning or in-context example selection). These 300 examples are manually annotated with simple QA pairs as decompositions. We find that performance of all models is quite low in this few-shot setting, so we develop a synthetic data generator that produces complex questions with their decompositions from semi-structured Wikipedia tables (Yoran et al., 2021). This synthetic data provides not just complex question supervision, but also supervision for the intermediate steps. We augment this data with the 300 (complex) training examples and their decompositions from DROP. In this few-shot setting, our best performing successive prompting model shows a ~5% improvement in F1 when compared to state-of-the-art model on DROP. The code and data are available at https://github.com/dDua/succesive_prompting

2 Decomposing Complex Questions

The goal of compositional question answering is to answer a complex question q in the context of a passage p (together denoted as x) by reasoning through latent sequential decisions $\mathbf{z} = z_1, z_2, ..., z_s$ to reach the final answer, y. Many models have been proposed to accomplish this with varying amounts of supervision and interpretability. In prompting methods like Chain-of-Thought (CoT, Wei et al., 2022) the latent steps are supervised, interpretable sentences; in other models these latent steps might be a program (Gupta et al., 2020; Chen et al., 2020) or even just the (unsupervised) hidden states in the model (Segal et al., 2020; Andor et al., 2019)

We focus on models that take in-context examples and produce a discrete, language-encoded $\mathbf{z}$, with CoT being the primary exemplar. We write the general form for CoT, given an input x, a language model encoder $\mathbb{L}$ and N in-context examples obtained from querying an index $\mathcal{I}$—each containing a triplet of passage with complex question (x^n), latent steps ($\mathbf{z^n}$) and final answer (y^n)—as follows:

$$y, z \leftarrow \mathbb{L}\left(x, \{\, (x^n, y^n, \mathbf{z}^n) \mid n \in [1, N] \,\}\right).$$

2.1 Successive prompting

In successive prompting, we represent each latent step as a pair of simple question and answer, $z_k = (q_k, a_k)$ (see Figure 1 for example QA pairs) unlike CoT which represents each latent step as a declarative sentence. Moreover, CoT queries the index $\mathcal{I}$ for in-context examples and prompts the language model $\mathbb{L}$ for generating output only once. However, in successive prompting, we separate $\mathbf{z}$ into multiple question and answering steps, which gives us many opportunities to prompt $\mathbb{L}$, with potentially different in-context examples that are more tailored to the simple question at each step. It also enables us to re-encode the context given the intermediate state z_k, which can be useful in certain questions that need long chain referencing (e.g., the sort-count example in Figure 3). We can write a general form for successive prompting as follows:

$$q_1 \leftarrow \mathbb{L}\left(x, \left\{\, (x^n, q_1^n) \mid n \in [1, N] \,\right\}\right)$$
$$a_1 \leftarrow \mathbb{L}\left(p, q_1, \{\, (p_*^m, q_*^m, a_*^m) \mid m \in [1, M] \,\}\right)$$

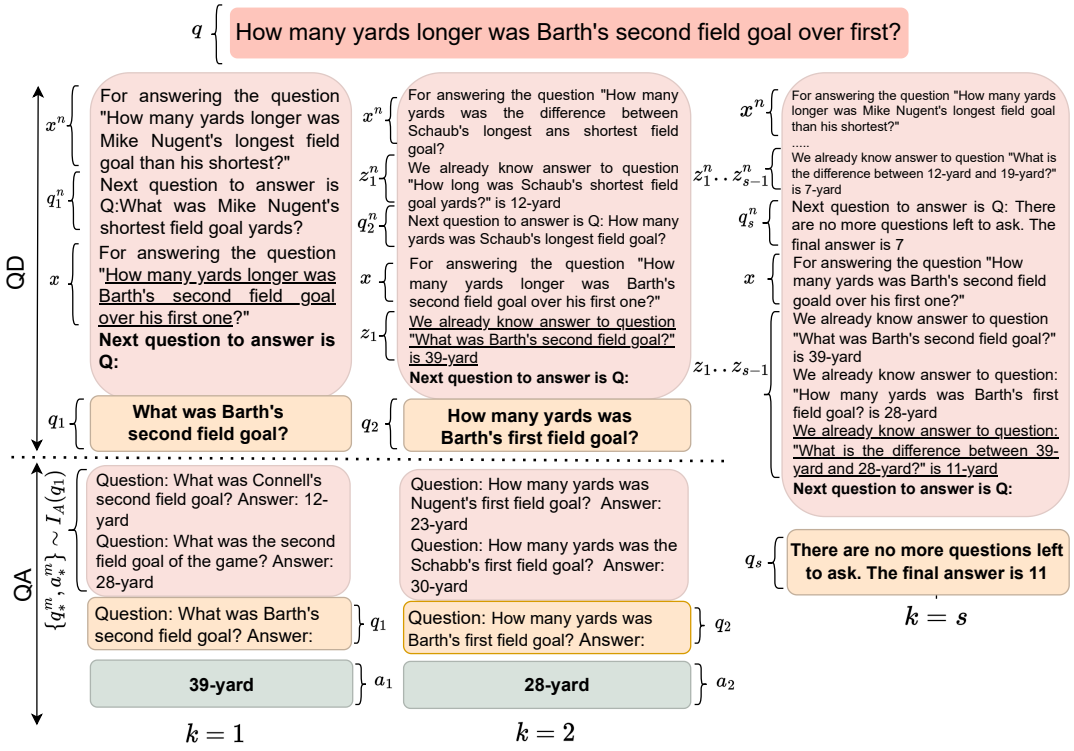

Figure 2: A demonstration of successive prompting with in-context learning. The selected examples for supervision and complex question to be answered pre-pended with the context paragraph (omitted to simplify illustration) are encoded by the model to generate question and answer at QD and QA stage respectively. During fine-tuning, only training supervision is used in an i.i.d manner for learning QD and QA models.

$$q_2 \leftarrow \mathbb{L}\left(x, q_1, a_1, \left\{ (x^n, q_1^n, a_1^n, q_2^n) \mid n \in [1, N] \right\}\right)$$
$$a_2 \leftarrow \mathbb{L}\left(p, q_2, \left\{ (p_*^m, q_*^m, a_*^m) \mid m \in [1, M] \right\}\right)$$
$$\ldots$$
$$y \leftarrow \mathbb{L}\left(x, \mathbf{z}, \left\{ (x^n, y^n, \mathbf{z}^n) \mid n \in [1, N] \right\}\right)$$

There are three kinds of model outputs in this general form: intermediate questions q_k, intermediate answers a_k, and the final answer y. We refer to the first kind of output as *question decomposition* (QD) and the second kind as *question answering* (QA). We treat final answer prediction as a special case of question decomposition, where the model decides that no more decomposition is necessary and outputs a final answer, so we iteratively alternate between question decomposition and question answering until the model terminates.

2.2 Training paradigm

We have so far described successive prompting in a setting where only in-context examples are given, so no model training is performed. However, successive prompting can also be used in conjuction with model fine-tuning, where each intermediate output is treated as a training example for $\mathbb{L}$. In this section, we first describe how in-context examples are selected at every step, followed by detailing how these examples are used for model fine-tuning.

In-context Learning During in-context learning, a small number of training examples are provided directly in the prompt that is given to a large LM, before the test input. These examples are selected from an index based on their similarity with the test input. For successive prompting, we create two indices: $\mathcal{I}_D$, for looking-up relevant demonstrations for QD, and $\mathcal{I}_A$, for looking-up relevant demonstrations for QA. The index $\mathcal{I}_D$ contains partially decomposed chains at each step k, demonstrating the next question q_k to be produced for every complex question in the training data. The index $\mathcal{I}_A$ contains all the simple QA pairs in the training data from all the complex questions.

In the QD stage, the index $\mathcal{I}_D$ is queried with the complex test question, q and current step number, k, to select demonstrations regarding how to generate the next question for the held-out example. In the QA stage, the index $\mathcal{I}_A$ is queried with the simple question q_k generated during QD to select relevant simple QA pairs. Figure 2 shows a demonstration

of how in-context learning is executed step-by-step in each stage until QD outputs the special phrase "There are no more questions left to ask", along with a final answer.

Successive prompting allows the QA stage to access simple questions derived from complex questions that would not have been retrieved by Chain-of-Thought prompting because on the surface they are not similar to the held-out complex question, even though they share similar sub-questions.

Model Fine-tuning For model fine-tuning, we use T5 (Raffel et al., 2020) based sequence-to-sequence models. Such models are typically trained with control codes in a multi-task setting (Ma et al., 2021; Rajagopal et al., 2022) to switch between QD and QA tasks with shared model parameters. We adapt and extend the control codes introduced by text modular networks (TMNs, Khot et al., 2021) for training with our synthetic data. TMNs are limited in terms of the operations they can handle as they do not go beyond first order reasoning. We use synthetically generated data, which allows us to deal with higher-order reasoning questions in DROP. Because we are fine-tuning the model, we can use special tokens to denote question decomposition and other separators, instead of the natural language prompts shown in Figure 2, though the content is the same. The specific tokens used for each step are listed in Appendix A.

Specialized Modules Successive prompting also allows us to use specialized sub-modules for solving different QA tasks because we no longer perform QD and QA in an end-to-end manner. Solving arithmetic operations like counting, difference, sorting, etc., can be challenging for language models. As a result, we follow Khot et al. (2021) and construct a simple mathematical sub-module for QA which parses the generated simple question for symbolic operation type and its arguments and then executes them in a deterministic way. If the generated simple question cannot be parsed as a mathematical operation, we apply the language model to solve it.

3 Synthetic Dataset

Any method that prompts LMs to produce intermediate reasoning steps to answer complex questions needs some amount of supervision for those reasoning steps. This kind of annotation can be expensive to collect and often requires expert knowledge.

Round	Date	Opponent	Venue	Attendance
R2 1st Leg	26 Sep 1990	Walsall	A	5,666
QFR	23 Oct 1990	Liverpool	H	18,246
SF 1st Leg	24 Feb 1991	Sheffield Wed.	H	14,074
SF 2nd Leg	27 Feb 1991	Oxford United	A	34,669
QFR	23 Jan 1991	Portsmouth	A	33,861

Table 1: Example table from Wikipedia where rows become sentences and columns are used for question generation (used as context for Figure 3).

Prior work has typically relied on a small handful of manually-written example decompositions. We find that such small collections lead to very poor performance on a dataset as varied as DROP, even for large models.

To mitigate these data issues, we propose a way to synthetically generate complex questions and their decompositions using semi-structured data which is easy to parse. We show that we can bootstrap model learning with this out-of-domain, synthetically generated data so it can adapt better when fine-tuned with limited in-domain supervision.

Generation Process: Inspired by Yoran et al. (2021), we use semi-structured data from tables in English Wikipedia which are available in plenty. We employ curated templates to convert the rows in the tables into paragraphs. We use single column headers to create first order simple questions and a combination of columns for higher order complex questions. We synthesize data for 10 simple operations: `COUNT`, `TOP(k)`, `BOTTOM(k)`, `FILTER`, `SUM`, `COMPARISON`, `DIFFERENCE`, `NEGATION`, `GATHER`, and `INTERSECTION`.

We generate higher order combinations of first-order operations, wherever possible. Figure 3 shows examples of higher order combinations of the atomic operation `COUNT` with a few other simple operations using Table 1 as context. The complete list of all decompositions is provided in Appendix A. Depending on the model, we use either symbolic or natural language version of the arithmetic operations. If we are using an LM to perform arithmetic operations, we output natural language; if we are using a separate symbolic reasoning engine, we output symbolic operations. We generate approximately 141K total complex questions which result in 525K examples for QD and 257K examples for QA. See Appendix A for more dataset statistics.

Reasoning	Complex Question and Decomposition (Question [Natural Language or Symbolic], Answer)
Count	How many opponents were there? • Q: What are all the opponents? Ans: Walsall; Liverpool; Sheffield Wed.; Oxford United; Portsmouth • Q: count(Walsall; Portsmouth; Sheffield Wed.; Oxford United; Portsmouth) Ans: 5 – Q: How many items are in the list: Walsall, Liverpool, Sheffield Wed. and Oxford United, Portsmouth?
Higher order decompositions	
Sort-Count	Which venue had the most number of opponents? • Q: What are all the venues? Ans: A; H • Q: What are opponents when venue was A? Ans: Walsall; Oxford United; Portsmouth • Q: count(Walsall; Oxford United; Portsmouth) Ans: 3 • Q: What are opponents when venue was H? Ans: Liverpool; Sheffield Wed. • Q: count(Liverpool; Sheffield Wed.) Ans: 2 • Q: top(1, 2;3) Ans: 3 – Q: What is the largest value in: 2 and 3? • Q: Which venue has 3 opponents? Ans: A
Comparison-Count	Which round had more venues: SF 1st Leg or QFR?? • Q: What are the rounds when venue was A? Ans: R2 1st Left; SF 2nd Leg; QFR • count(R2 1st Left; SF 2nd Leg; QFR) Ans: 3 • Q: What are the rounds when venue was H? Ans: QFR; SF 1st Leg • count(QFR; SF 1st Leg) Ans: 2 • if_then(1 > 2; SF 1st Leg; QFR) Ans: QFR – Q: If 1 > 2 then answer is SF 1st Leg else it is QFR

Figure 3: Examples of COUNT operation and some of its higher order combinations, with natural language and symbolic decompositions of the complex question. Underneath the first instance of a symbolic operation we show its corresponding natural language version. See Table 1 for the original table used to generate context and questions.

4 Experiments and Results

The DROP dataset contains a variety of reasoning compositions which are not uniformly distributed. In order to get a fair representation of DROP examples, we first embed the examples using a sentence embedding method trained on the QQP dataset (Reimers and Gurevych, 2019). We then use cosine similarity to get the top-50 nearest neighbor questions for each training example. The connection graph between each training question to its neighbors is then used to obtain 300 questions that cover the majority of the training data, via the vertex cover algorithm. We manually annotate these 300 examples with decomposed QA pairs in the same format as our synthetic data (Figure 3). For synthetic examples, since we know the reasoning types, we uniformly sample example demonstration from each reasoning type.

4.1 In-context Learning

Setup We use faiss[1] index with the QQP-based sentence embedding (Reimers and Gurevych, 2019) for indexing all the questions. We use GPT-J (6B)[2] which is the largest freely available model we could use with prompts containing 6 in-context examples.

[1] https://ai.facebook.com/tools/faiss/
[2] https://github.com/EleutherAI

	Syn-Only	DROP-Only	Syn+ DROP
Standard	22.7	23.8	24.9
CoT	25.3	26.2	27.6
Succ.(w/o calc.)	27.2	29.3	29.9
Succ.(w/ calc.)	28.8	30.8	31.9

Table 2: F1 Performance of in-context prompting on the DROP dev set with and without in-domain annotations.

Results In Table 2, we compare performance of language models without any prompting (Standard), with chain-of-thought prompting (CoT) and successive prompting. We observe that successive prompting performs better than CoT by 3.5% when only synthetic data is available, and 4.3% better with synthetic data and 300 annotations from DROP. The best successive prompting version on the dev set (Synthetic+DROP) has a test set performance of 30.6% F1. We also perform an ablation where the symbolic calculator is replaced by language model and observe that the performance drops by 1.5% F1. This further shows that modular approach is better over a single model that tries to solve all the tasks.

4.2 Model Fine-tuning

Setup We employ a shared question decomposition (QD) and answering model (QA) based on T5-large version of UnifiedQA (Khashabi et al., 2020), trained in a multi-task manner. We use the format described in Appendix A for prompting UnifiedQA. For symbolic questions, we use a simple calculator that parses the operator and arguments in the generated question and executes the discrete operator on the detected arguments.

To deter the model from learning incorrect steps, we use contrastive estimation (Smith and Eisner, 2005). In particular, we first train the model for two epochs with cross-entropy loss while generating the output sequence (simple question or answer). Then we continue training by adding an auxiliary loss term which increases the likelihood of the intermediate sub-question that would produce a correct sub-answer at the cost of one that does not (Dua et al., 2021). We sample up to 3 negative samples at each step. We use HuggingFace transformers[3] to train our models, with a learning rate of 5e-5 and maximum input length of 768.

Due to variance in the types of context tables present in Wikipedia, the synthetic dataset distribution is not uniform across different reasoning types. To have a balanced representation of questions pertaining to different reasoning types, we employ dynamic sampling (Gottumukkala et al., 2020), where at the beginning of each epoch we select 80,000 instances from across all reasoning types in proportion to the drop in their current performance with respect to previous epoch on held-out synthetic data. For the first epoch we sample in proportion to original the size of each reasoning type. During inference, we use beam search with size 5 to generate decompositions, switching between QD and QA stages until QD reaches end of decomposition ("EOQ") or maximum number of steps which we set as 10.

Baseline models We compare against a number of different baselines, both symbolic and non-symbolic. As non-symbolic baselines, we use UnifiedQA (Khashabi et al., 2020), which is pre-trained on a number of existing question answering datasets, and PReasM (Yoran et al., 2021), which is pre-trained on synthetically generated compositional QA pairs. We also include a baseline with symbolic components, TASE (Segal et al., 2020). This model (and others like it (Jin et al., 2021; Andor et al., 2019)) are capable of performing a combination of continuous and discrete operations, which is essential for DROP. TASE does not require expressing decomposition in a specific grammar and can work with natural language. We chose this model as it is close to state of the art on the full DROP dataset and has publicly available code.

Results In Table 3, we use the DROP dev set to compare the performance of different symbolic and non-symbolic models in three settings: (1) using no training data from DROP (0-shot), (2) using only question-answer supervision from the 300 DROP examples, and (3) using both question-answer supervision and the decompositions for the 300 DROP examples. In each of these settings, we can train the model with or without the synthetic data that we generated.

We observe that our out-of-domain synthetic data universally improves model performance, and the improvement is most pronounced in TASE, nearing a 20% absolute improvement. Without synthetic data, PReasM is the best performing baseline, but TASE overtakes PReasM when synthetic data is available. Additionally, and unsurprisingly, increasing the amount of supervision from 0-shot to complex QA pairs to decompositions universally improves model performance.

Finally, our method, which is a fine-tuned successive prompting model combined with a symbolic reasoning engine, achieves the best performance, giving an improvement of 5.4 F1 over the state-of-the-art model with similar supervision, i.e. TASE+Synthetic w/ decomp. We follow the standard practice of using test set for only our final best performing model (SP w/ decomp). We observe that our best model with a test set performance of 50.2 F1 is better than the state-of-the-art model with similar supervision (45.1 F1) by 5.1% F1.

Overall, methods that learn to decompose complex questions into simple QA pairs adapt well to complex questions in new domain with little (SP w/ decomp: 51.3 F1) to no in-domain supervision for decomposition (SP 0-shot: 49.8). If we have limited complex QA supervision (without any decompositions), un-interpretable symbolic models result in the best performance (TASE + Synthetic w/o decomp: 44.1). This is because of two reasons. First, such models can capture domain specific answer priors which may result it decent held-out performance (Dua et al., 2020; Agrawal et al., 2018).

[3] https://github.com/huggingface/transformers

	0-shot	w/o decomp	w/ decomp
Non-symbolic			
UnifiedQA	24.5	26.7	27.2
+ Synthetic	26.6	30.3	32.6
PReasM	24.9	34.6	37.5
+ Synthetic	30.2	36.2	38.1
Symbolic			
TASE	-	26.1	27.6
+ Synthetic	27.3	44.1	45.9
Succ. Prompting	49.8	-	51.3

Table 3: F1 Performance of various model architectures on DROP dev-set pre-trained on synthetic data and further fine-tuned with 300 DROP examples.

	QA: In-Context	QA: Fine-tuning
QD: In-Context	30.8	40.3
QD: Fine-tuning	31.4	51.3

Table 4: F1 with QD and QA modules from incontext learning and fine-tuning with only DROP annotations

Second, depending on the context, sometimes it may not be straight-forward to decompose the complex questions into QA pairs.

4.3 In-context vs Fine-Tuning

To understand the gap in performance between successive prompting with in-context learning and fine-tuning, we perform ablations across in-context and fine-tuned version of QD and QA modules. We observe that in-context learning is unable to do well on answering simple questions that result in a list of answers—which is especially important for DROP as symbolic aggregations are generally applied on a list of answers. On using a fine-tuned QA model we see an improvement of ~10% in F1 with an in-context QD model. Moreover, since the final answer performance is dependent on how well the QA model performs, using a better QD model (fine-tuned) does not help the overall performance much unless the QA model can handle the decompositions produced by the QD model.

4.4 Qualitative Examples

To evaluate the correctness of decomposed QA pairs, we manually analyze a subset of predictions on the dev set with in-context (DROP-only) learning and model fine tuning (few shot). We do this by randomly sampling 50 correct predictions to determine how often the incorrect decompositions result in correct answer. We observe that QD stage has an accuracy of 88% for in-context and 96% for fine-tuned model. The incorrect decompositions are mainly because the decomposed question is identical to the original question. For instance, "Who made the longest field goal?" can sometimes be answered correctly without decomposing the question if the passage contains a single field goal mention.

We also sample 50 incorrect predictions to ascertain the reason for incorrect predictions in both in-context and fine-tune setup. We observe that the final predictions are incorrect due to three main categories of errors: incorrect QA model prediction, incorrect next question prediction (QD) and out-of-scope reasoning type. The QA model outputs incorrect answers to simple question 40% and 22% of the times for in-context and fine-tuned respectively. The second class of errors, due to incorrect decomposition, occur 30% of the times for both in-context and fine-tuned. The final class of errors, due to compositional questions that are not covered by synthetically generated annotations, occur 28% (in-context) and 46% (fine-tune) of the times.

In Figure 4, we show a few examples of correct and incorrect predictions and point out the strengths and weaknesses of successive prompting. The main strength of successive prompting is that, by breaking down the question, we are able to get improved supervision for QA. As a result, it is able to correctly identify the goals kicked in the first half while answering the question “How many field goals did both teams kick in the first half?", unlike CoT that returns goals for the entire game.

One of the limitations of in-context learning, when compared with fine-tuning (irrespective of the type of prompting), is that examples are chosen based on the question alone, overlooking the context. For instance, DROP has questions like “How many people were not Germans, in terms of percentage?” where we first need to answer “How many people were Germans, in terms of percentage?" and then perform a negation operation (i.e, subtract from 100). The word “not" influences the example lookup to choose decomposition that involves a negation even when the question being answered requires a different operation.

A limitation of successive prompting is that it is sometimes challenging to decompose a question, especially when it involves implicit reasoning from the passage. For instance, for “Which port did the Korean immigrants leave first Chemulpo or Veracruz?”, it is difficult to explicitly define a comparison style decomposition from the sentence,

Example	Chain-of-Thought	Successive (In-context)	Successive (Fine-Tuning)
Who threw the longest touchdown pass?	The touchdown passes are 22-yard and eight-yard. Out of 22 and eight-yard, the longest touchdown pass is 22. Peyton manning threw the 22-yard touchdown pass. The answer is Peyton Manning. *Answer* (✓)	• What are all the touchdown passes? A: 22-yard, eight-yard • What is the largest value in: 22-yard, eight-yard? A: 22-yard • Who threw the 22-yard touchdown pass? A: Peyton Manning. • There are no more questions left to ask. The final answer is Peyton Manning. *Answer* (✓)	• What are all the touchdown passes? A: 22-yard, eight-yard • max(22;eight) A: 22 • Who threw the 22-yard touchdown pass? A: Peyton Manning. *Answer* (✓)
In which two quarters did Stephen Gostkowski only kick one field goal? Count-Argmax type decompositions were not covered in synthetic annotations	The first quarter is the only quarter when Stephen gostowski kick only one field goals. The count becomes 1. The second quarter is the only quarter when Stephen gostoswki only kick two field goals..... The answer is first quarter. *Answer* (✗)	• Which quarter did Gostowski score? A: first, second and fourth • What are the field goals that Gostowski scored? A: 38, 24 and 21 • There are no more question left to ask. The final answer is 38 and 24. *Answer* (✗)	• What are all the field goals when player was Stephen gostowski? A: 38-yard, 24 and 21-yard • What were the quarters when 38-yard field goal was made? A: second *Answer* (✗)
How many field goals did both teams kick in the first half?	The field goals in the game are 20, 24, 53, 30 and 40. The count becomes 7. *Answer* (✗)	• What are the field goals both teams kicked in the first half? A: 20, 24 and 53 • How many items are in the list 20, 24 and 53? A: 3 • There are no more questions left to ask. The final answer is 3. *Answer* (✓)	• What are all the field goals in the first half? A: 20; 24; 53 • count(20; 24; 53) A: 3 *Answer* (✓)
How many of the personnel were not civil servants?	The personnel are 1,513. Since 1,513 is less than 3,183. The answer is not civil servants. *Answer*(✗)	• Q: How many of the personnel were not civil servants? A: 1,176 • Q: What is the difference between 100 and 1176? A: 1111 • There are no more questions left to ask. The final answer is 1111. *Answer* (✗)	• Q: How many personnel were there? A: 2,234 • Q: How many of the personnel were civil servants? A: 1,176 • Q: diff(2234; 1176) A: 1058 *Answer* (✓)
Q: Which port did the Korean immigrants leave first Chemulpo or Veracruz? Passage: In 1905, Korean immigrants departed from the port of Chemulpo.. After which they took a train to Veracruz. In the Veracruz port, another boat was taken to the port of Progreso.	The Korean immigrants left Chemulpo. The answer is Chemulpo. Reasoning (✗) Answer (✓)	• Q: In what year did the Korean immigrants leave Chemulpo? A: 1905 • Q: Which port did the Korean immigrants leave first Chemulpo or Veracruz? A: Chemulpo • Q: If Chemulpo is greater than Veracruz, answer is Chemulpo else it is Veracruz A: Chemulpo. Reasoning (✗) *Answer* (✓)	• Q: When did Korean immigrants leave Chemulpo? A: 1905 • Q: When did Korean immigrants leave Veracruz? A: 1905 • Q: if_then(1905 < 1905; Chemulpo; Veracruz) A: Veracruz. *Answer* (✗)

Figure 4: Generated decompositions depicting strength and weaknesses of Successive Prompting.

"After which they took a train to Veracruz".

5 Related Work

Prompting methods Prompting was introduced as a way to test the reasoning capabilities of large language models (Brown et al., 2020). In follow-up works (Schick, 2022; Chowdhery et al., 2022; Marasović et al., 2021) prompting techniques have been used as a mechanism to supervise the model decision with few demonstrations as a conditioning context to guide its predictions on an unseen example. Works like Chain-of-Thought reasoning (Wei et al., 2022; Zelikman et al., 2022) especially focus on compositional questions where they provide a chain of reasoning as demonstrations. In concurrent work, Least-to-Most prompting (Zhou et al., 2022) takes a similar view as ours to break down the problem into sub-problems. However, in Successive Prompting the question decomposition and answering stages are interleaved, unlike Least-to-Most where the problem is first reduced into sub-problem and then executed in a sequence. In our method, the next question prediction has access to previously answered sub-questions, which is useful in questions that need long chain referencing. Other contemporaneous works (Press et al., 2022; Khot et al., 2022) use very large language models (more than twice the size we used) and show better few-shot generalization. Works like Perez et al. (2021) have shown the importance of having the right in-context examples for downstream performance leading to works that learn to retrieve relevant in-context examples (Rubin et al., 2021).

Non-symbolic methods Most non-symbolic methods are sequence-to-sequence models trained on a large amount of question answering data (Khashabi et al., 2020; Yoran et al., 2021).

Symbolic methods Neural module networks like approaches parse complex questions into a pre-specified grammar and learn neural components to handle symbolic mathematical operations (Gupta et al., 2020; Chen et al., 2020; Nye et al., 2021) which are recursively executed. State-of-the-art models on DROP, however, use a combination of BERT-based contextual models along with a calculator that performs discrete operations (Andor et al., 2019; Segal et al., 2020; Hu et al., 2019). Works like Text Modular networks (Khot et al., 2021) and MRKL (Karpas et al., 2022) are closest to our work. However, they are limited in the terms of types of simple questions they can answer (single-span only) and the complexity of reasoning they can do (single-order only). TMNs, additionally, use a classifier that scores the generated chains module and filters out incorrect question decompositions, while we use contrastive estimation to learn a better question decomposer and as a result do not need a chain scorer.

6 Conclusion

We present a way to successively decompose complex questions into simple QA pairs, which allows for modular QD and QA systems that can be trained and queried independently. When performing in-context learning, we showed that successive prompting yields an improvement of 4.6 F1 over chain-of-thought prompting. When replacing just the in-context QA module with a fine-tuned one, which is adept at handling list type questions, we further improve the overall performance by 9.5 F1. We believe that modular systems that decompose and delegate tasks to the most appropriate model, whether that is a large LM or a tailored component, are more effective at solving complex tasks than trying to have a large LM solve the entire task on its own. Successive prompting shows one way this decomposition and delegation can be done.

Acknowledgements

We would like to thank Anthony Chen, Catarina Belem and the anonymous reviewers for the discussions and feedback. This material is based upon work sponsored in part by the DARPA MCS program under Contract No. N660011924033 with the United States Office Of Naval Research, in part by funding by AI2 and NSF IIS-1817183. We would also like to thank Hasso Plattner Institute(HPI) for supporting the first author through UCI-HPI fellowship. The views in this work are of authors and not the sponsors.

Limitations

We propose a way to decompose complex questions into interpretable simple QA pairs as latent steps that get successively asked and answered by large pretrained models. The notion of performing complex tasks by iteratively finding and then filling information needs is very general, but we have only shown the applicability of one specific version of this idea in one specific setting. There are many

potential challenges in applying successive prompting more broadly. The biggest is that it requires at least some decomposition data, which may be hard or even impossible to obtain. Some complex questions are not easily decomposed, and some domains can be very challenging to write synthetic data generators for. We were able to generate synthetic data that covered most of the reasoning types in DROP, but other kinds of complex questions would not be covered by our generator (e.g., questions that require commonsense or causal reasoning).

There is also significant difficulty in choosing a level of granularity for decomposition. If a large pretrained model can directly answer a question as complex as "What was Barth's second field goal?", we should let the model answer the question instead of trying to decompose it further. The right granularity for the decomposition thus depends on the capabilities of the underlying model, and those capabilities are rapidly changing as newer and larger pretrained models are released. There is the possibility that newer model iterations will not need any decomposition to answer the complex questions covered by our synthetic data generator, making that generator obsolete. However, it seems unlikely that pretrained models will be able to handle all complex scenarios in the near future, so the ideas of successive prompting and generating synthetic data to bridge reasoning gaps should still be applicable even when our particular application of them becomes obsolete.

This method also increases the computational requirements for answering complex questions, as instead of making one query to a large LM, successive prompting makes many queries to answer a single question.

Ethics Statement

This work focuses on improving complex question answering with limited data. It uses existing training data and conventional methods of testing model performance. This work does not deal with any social impacts or biases in natural language processing systems.

References

Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. 2018. Don't just assume; look and answer: Overcoming priors for visual question answering. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4971–4980. IEEE Computer Society.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. 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)*, pages 5947–5952, Hong Kong, China. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022. Meta-learning via language model in-context tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 719–730, Dublin, Ireland. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *ArXiv preprint*, abs/2204.02311.

Dheeru Dua, Pradeep Dasigi, Sameer Singh, and Matt Gardner. 2021. Learning with instance bundles for reading comprehension. *ArXiv preprint*, abs/2104.08735.

Dheeru Dua, Sameer Singh, and Matt Gardner. 2020. Benefits of intermediate annotations in reading comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5627–5634, Online. Association for Computational Linguistics.

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 *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)*, pages 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Ananth Gottumukkala, Dheeru Dua, Sameer Singh, and Matt Gardner. 2020. Dynamic sampling strategies for multi-task reading comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 920–924, Online. Association for Computational Linguistics.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. A multi-type multi-span network for reading comprehension that requires discrete 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)*, pages 1596–1606, Hong Kong, China. Association for Computational Linguistics.

Yue Jin, Tianqing Zheng, Chao Gao, and Guoqiang Xu. 2021. Mtmsn: Multi-task and multi-modal sequence network for facial action unit and expression recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3597–3602.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. 2022. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *ArXiv preprint*, abs/2205.00445.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online. Association for Computational Linguistics.

Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2021. 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*, pages 1264–1279, Online. Association for Computational Linguistics.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *ArXiv preprint*, abs/2210.02406.

Xiaofei Ma, Cicero Nogueira dos Santos, and Andrew O. Arnold. 2021. Contrastive fine-tuning improves robustness for neural rankers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 570–582, Online. Association for Computational Linguistics.

Ana Marasović, Iz Beltagy, Doug Downey, and Matthew E Peters. 2021. Few-shot self-rationalization with natural language prompts. *ArXiv preprint*, abs/2111.08284.

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

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True few-shot learning with language models. *Advances in Neural Information Processing Systems*, 34.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *ArXiv preprint*, abs/2210.03350.

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

Dheeraj Rajagopal, Siamak Shakeri, Cicero Nogueira dos Santos, Eduard Hovy, and Chung-Ching Chang. 2022. Counterfactual data augmentation improves factuality of abstractive summarization. *ArXiv preprint*, abs/2205.12416.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *ArXiv preprint*, abs/2112.08633.

Timo Schick. 2022. *Few-shot learning with language models: Learning from instructions and contexts*. Ph.D. thesis, lmu.

Elad Segal, Avia Efrat, Mor Shoham, Amir Globerson, and Jonathan Berant. 2020. A simple and effective model for answering multi-span questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*,

pages 3074–3080, Online. Association for Computational Linguistics.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 354–362, Ann Arbor, Michigan. Association for Computational Linguistics.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Ori Yoran, Alon Talmor, and Jonathan Berant. 2021. Turning tables: Generating examples from semi-structured tables for endowing language models with reasoning skills. *ArXiv preprint*, abs/2107.07261.

Eric Zelikman, Yuhuai Wu, and Noah D Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *ArXiv preprint*, abs/2203.14465.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *ArXiv preprint*, abs/2205.10625.

A Appendix

A.1 Control codes for Model Fine-tuning

To generate a simple question given complex question and previously generated latent steps, we append the input with control code "QS:"

$$\{x^n\}\ \texttt{QI:}\ q_1^n\ \texttt{A:}\ a_1^n\ \cdots\ \texttt{QI:}\ q_{k-1}^n\ \texttt{A:}\ a_{k-1}^n\ \texttt{QS:}$$

To answer the simple question we prompt the model again with only the simple question this time and control code "A:" to generate answer, a_k^n

$$\{p^n\}\ \texttt{QS:}\ q_k^n\ \texttt{A:}$$

We alternate between the two stages till we reach the end of decomposition marker "EOQ"

$$\{x^n\}\ \texttt{QI:}\ q_1^n\ \texttt{A:}\ a_1^n\ \cdots\ \texttt{QI:}\ q_s^n\ \texttt{A:}\ a_s^n\ \texttt{QS: EOQ}$$

A.2 Synthetic Dataset Statistics

Reasoning	Number of Examples
Filter	9634
Count	11019
Comparison	12239
Difference	15433
Negation	2020
Intersection	9089
Sum	18754
Sort	10663
Sort-Filter	5827
Difference-Sort	13872
Sum-Sort	3212
Count-Filter	7382
Gather-Count	1506
Sum-Count	4138
Difference-Count	4216
Sort-Count	6328
Comparison-Count	5998
Total	141,330

Table 5: Synthetic dataset statistics

Round	Date	Opponent	Venue	Attendance
R2 1st Leg	26 September 1990	Walsall	A	5,666
R2 2nd Leg	10 October 1990	Portsmouth	H	10,037
QFR	23 October 1990	Liverpool	H	18,246
SF 1st Leg	24 February 1991	Sheffield Wednesday	H	14,074
SF 2nd Leg	27 February 1991	Oxford United	A	34,669
QFR	23 January 1991	Walsall	A	33,861

(a) Example wikitable where rows become sentences and columns are used for question generation

Reasoning	Complex Question	Decomposition (Question, Answer)
Filter	What are the opponents when date was later than 21 January 1991 and attendance was less than 20000?	• Q: What are the opponents when date was later than 21 January 1991? A: Sheffiedl Wednesday; Oxford United; Walsall • Q: Out of Sheffield Wednesday, Oxford United and Walsall, which opponents have attendance less than 20000? A: Sheffield Wednesday
Count	How many opponents were there?	• Q: What are all the opponents? A: Walsall; Portsmouth; Liverpool; Sheffield Wednesday; Oxford United • Q: count(Walsall; Portsmouth; Liverpool; Sheffield Wednesday; Oxford United) A: 5
Comparison	What round had a higher attendance: SF 2nd Leg or QFR?	• Q: What was the attendance when round was SF 2nd Leg? A: 34,669 • Q: What was the attendance when round was QFR? A: 33,861 • if_then(34,669 > 33,861; SF 2nd Leg; QFR) A: SF 2nd Leg
Difference	What is the difference between attendances when the opponent was Oxford United and Portsmouth?	• Q: What was the attendance when opponent was Oxford United? A: 34,669 • Q: What was the attendance when opponent was Portsmouth? A: 10,037 • diff(34669; 10037) A: 24632 – Q: What is the difference between 34669 and 24632?
Sum	What were the total attendances when opponents were Walsall and Oxford United?	• Q: What was the attendance when opponent was Walsall? A: 5,666; 33,861 • Q: What was the attendance when opponent was Oxford United? A: 34,669 • sum(5666; 33861; 34669) A: 74196
Sort	Which opponent had the second highest attendance?	• Q: What are all the attendances? A: 5,666; 10,037; 14,074; 34,669; 33,861 • Q: top(2, 5,666;10,037;14,074;34,669;33,861 A: 33,861 • What was the opponent when attendance was 33,861? A: Walsall

(b) Single order decompositions

Reasoning	Complex Question	Decomposition (Question, Answer)
Higher-order combinations with SORT		
Sort-Filter	Which opponent had the third lowest attendance after 1 January 1991?	• Q: What are all the attendances after 1 January 1991? A: 14,074; 34,669; 33,861 • Q: bottom(3, 14,074;34,669;33,861 A: 14,074 • What was the opponent when attendance was 14,074 A: Sheffield Wednesday
Difference-Sort	What is the difference between the second highest attendance and lowest attendance?	• Q: What are all the attendances? A: 5,666;14,074;18,246;14,074;34,669;33,861 • Q: top(2, 5,666; 14,074; 18,246; 14,074; 34,669; 33,861 A: 33861 • Q: bottom(1, 5,666; 14,074; 18,246; 14,074; 34,669; 33,861) A:5666 – Q: What is the smallest value in: 33861 and 5666? • diff(33861;5666) A: 28195
Sum-Sort	What was the total of highest attendance and third lowest attendance?	• Q: What are all the attendances? A: 5,666; 14,074;18,246;14,074;34,669;33,861 • Q: top(1, 5,666; 14,074; 18,246; 14,074; 34,669; 33,861 A: 34669 • Q: bottom(3, 5,666; 14,074; 18,246; 14,074; 34,669; 33,861 A:18246 • sum(33861;5666) A: 52915 – Q: What is the sum of 33861 and 5666?
(Additional) Higher-order combinations with COUNT		
Count-Filter	How many rounds had venue as A and attendance greater than 30000?	• Q: What rounds had venue as A? A: R2 1st Left; SF 2nd Leg; QFR • Q: Out of rounds R2 1st Left, SF 2nd Leg and QFR, which had attendance greater than 30000? A: SF 2nd Leg; QFR • Q: count(SF 2nd Leg; QFR) A: 2
Gather-Count	How many opponents were there for each of venue: A and H?	• Q: What are the opponents when venue was A? A: Walsall; Oxford United • Q: count(Walsall; Oxford United) A: 2 • Q: What are the opponents when venue was H? A: Portsmouth; Liverpool; Sheffield Wednesday United • Q: count(Portsmouth; Liverpool; Sheffield Wednesday United) A: 3 • Q: gather(2;3) A: 2 and 3

(c) Higher order decompositions

Figure 5: Examples of decompositions with wikitables

Reasoning	Complex Question	Decomposition (Question, Answer)
Additional Higher-order combinations with COUNT		
Sum-Count	What are the total number of opponents when venue were A and H?	• Q: What are the opponents when venue was A? A: Walsall; Oxford United • Q: count(Walsall; Oxford United) A: 2 • Q: What are the opponents when venue was H? A: Portsmouth; Liverpool; Sheffield Wednesday United • Q: count(Portsmouth; Liverpool; Sheffield Wednesday United) A: 3 • Q: sum(2;3) A: 5
Difference-Count	What is the difference between number of rounds when venue was A and H?	• Q: What are the venues when round was SF 1st Leg? Ans: H • Q: count(A) A: 1 • What are the venues when round was QFR? A: H;A • Q: count(H;A) A: 2 • diff(3; 2) Ans: 1

(d) Higher order decompositions

Figure 5: Examples of decompositions with wikitables