Learning to Reason and Memorize with Self-Questioning

Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, Sainbayar Sukhbaatar
Meta AI

Abstract

Large language models have been shown to struggle with limited context memory and multi-step reasoning [1]. We propose a simple method for solving both of these problems by allowing the model to ask questions and answer them. Unlike recent scratchpad approaches, the model can deviate from the input context at any time for self-questioning. This allows the model to recall information and perform reasoning on the fly as it reads the context, thus extending its memory and enabling multi-step reasoning. Our experiments on two synthetic tasks demonstrate that our method can successfully generalize to more complicated instances from their training setup by performing self-questioning at inference time.

1 Introduction

Transformers [2] and similar variants have shown impressive results on sequence-based tasks [3]. Notably, large language models (LMs) such as GPT-3 [3] use transformers and are capable of solving various NLP tasks such as question answering (QA). When a LM is used for a QA task, it is usually fed a context prompt along with a question, and then the model generates the answer directly, as shown in Fig. 1 (top). However, this autoregressive "one-step" approach struggles with multi-step reasoning tasks [1]. Recently, Nye et al. [4] proposed the use of a scratchpad that allows the model to generate reasoning tokens before answering the question, but *after* it has read the full context and question, illustrated in Fig. 1 (middle). Similarly, chain-of-thought prompting methods [5–7] push the model to explain their answer one step at a time, leading to more coherent final answers. In addition to the "one-step" problem, transformers as a feed-forward model lack memory for state-tracking and solving highly nonlinear tasks [8], something that recurrent predecessor models such as the LSTM [9] are well equipped for. Modifications to the feed-forward transformer architecture that use a recurrent mechanism have been shown to improve state-tracking results [8, 10, 11].

In this paper, we propose an approach that simultaneously makes the challenges in multi-step reasoning and state-tracking memory more tractable. Our method allows the LM to deviate from the context prompts on the fly to generate a question and answer it by itself, as demonstrated in Fig. 1 (bottom). Such "*self-QAs*" can act as both explicit intermediate reasoning steps as well as memory for state-tracking. Specifically, if the self-question requires combining multiple facts, the resulting answer which consolidates multiple facts into one can then be used for future QAs, thus acting as an intermediate reasoning step. For example, given "`Alice has the box.`" and "`Alice is at the park.`" one can infer "`The box is at the park.`", which can be further combined with a later statement "`The key is in the box.`" to conclude that "`The key is at the park.`". Additionally, the self-QA can act as a form of memory because the question can ask about some entity mentioned in the past and the generated answer will write its latest state as new tokens in the current context position. For example, assume `x=5` initially, and then `x` gets incremented by 1. If the model correctly writes `x=6`, it can safely remove the original `x=5` statement from its context.

The main difference between our proposed method and prior work such as scratchpad [4], chain-of-thought [5], or inner monologue [7] is that we allow the model to explicitly write out questions and

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

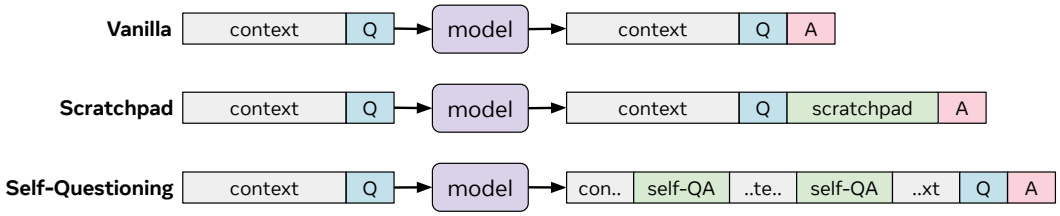

Figure 1: **(top)** Baseline vanilla LM directly generates the answer (A) given the context and the question (Q). **(middle)** Scratchpad allows the model to generate intermediate reasoning tokens before answering the question but after it has seen the context. **(bottom)** Our Self-Questioning method allows the model to deviate from the input context at any time to ask a question and answer it.

answers *as it reads* each context statement sequentially. Prior methods allow the model to ruminate after it reads the full context, forcing it to do a large chunk of reasoning at the end, rather than while it's reading. Furthermore, such post-context reasoning cannot act as memory because earlier context tokens may already be out of the model's context window before the reasoning starts.

To teach the model to generate self-QAs, during training we provide the language model with ground truth self-QAs as part of the input context. During inference, the model can deviate from the context and generate a self-QA if it generates a special query token learned during training. When the model finishes generating self-QAs, the original context tokens will continue to be fed. This allows the model to reason and create memory while processing input tokens, not just at the end. We test our method on two text datasets designed to evaluate multi-step reasoning and state-tracking: an Algorithmic task [8], and a proposed Toy-story task. Our method outperforms both fine-tuned language model and scratchpad baselines. In particular, our method outperforms a fine-tuned language model, which does not do any explicit rumination, by 46 and 39 absolute percentage points on each dataset, respectively.

2 Method: Self-Questioning

In autoregressive language modeling, the task is to predict the next token x_{S+1} given the previous S tokens $x_1, x_2, ..., x_S$. For question answering, we denote each context statement, or independent fact, as C_t, the question as Q, and the answer as A. The task is to generate the answer A given the context and question: $C_1, C_2, ..., C_T, Q$. Note that each of the variables may contain several tokens, but we condense them for brevity.

In addition to the original T context statements, we also consider access to intermediate question-answer pairs available throughout the context, which we call *self-QAs*. That is, each of the T context statements can be followed by 0 or more self-QAs. For example, if each context statement is followed by a single self-QA pair, the new sequence will look like: $C_1, SQ_1, SA_1, C_2, SQ_2, SA_2, ..., C_T, SQ_T, SA_T$, where "$SQ$" indicates a self-question, and "SA" indicates a self-answer. The task is still to answer the final question Q. These QA pairs can act both as intermediate reasoning steps, aggregating multiple steps of information into a concise statement, as well as a form of explicit memory where past events are written out into the current context, recalling old information.

Directly providing the model with self-QA pairs makes it easier to do multi-step and long-term reasoning. However, we want the model to generate these QA pairs on its own. Therefore, we need mechanisms to train the model to generate them, and allow it to do so during testing.

During training, we give the model a set of "ground truth" self-QA pairs interspaced within the context, and train it to predict the next token as typically done in LM. We prepend to the self-QA a special query token, e.g. "`SQ:`", to signal to the model that this is a self-question, and not a context fact. During testing, we want the model to generalize to samples where we don't have the ground truth self-QA pairs. Therefore, after a context fact C_t is processed by the model, we check to see if the model wants to ask a question. That is, if the most likely next token is the special query token, "`SQ:`", then we allow it to generate a question and answer. Once the end-of-answer token (e.g. "`.`") is generated to indicate the end of the QA, we resume feeding the original context facts starting with

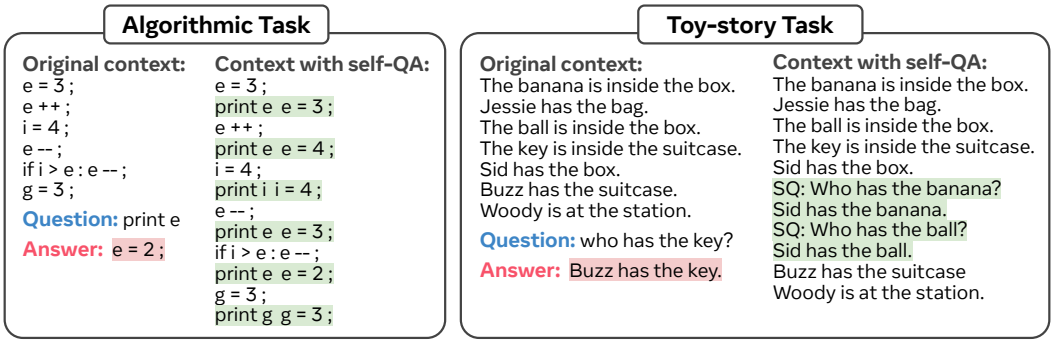

Figure 2: Task samples. Green tokens are the self-QA pairs. Red tokens are the final answer. "`print`" and "`SQ:`" are special tokens to generate a self-QA for the Algorithmic and Toy-story, respectively. In our Self-Questioning model, if the model predicts the special token as the next token, it is allowed to generate a question and answer before returning to the original context.

C_{t+1} unless the model generates another question. After all T context facts are read, the model will be given the final question Q, where it has to generate an answer.

3 Experiments

We compare against two baseline methods and consider several variants of our method, outlined in Table 1. The first baseline is the pretrained GPT-2 base model [12] from Hugging Face [13] fine-tuned to predict answer tokens given only the context and question. In the second baseline, we train the same GPT-2 model to write self-QA pairs after it has seen the context and question, similar to "Scratchpad" [4]. For our proposed Self-Questioning method, we examine different percentages of total training samples that get self-QA supervision: 100%, 75%, 50%, and 25%. During testing, no self-QAs are provided, but both Scratchpad and Self-Questioning models are allowed to generate tokens. We also perform two Self-Questioning ablations. The first is an upper bound where we provide 100% self-QA supervision to the model during both training and testing (rather than just training). The second is where we give 100% self-QA supervision during training, but don't allow the model to use or generate self-QAs during testing. This baseline analyzes whether self-QA-augmented training data can still help the model learn about the task in its weights even without using self-QA at test time. We use the tasks outlined in the following subsection, which are specifically designed for both multi-step reasoning and state-tracking. For each task, we train on 10K samples for 30 epochs and test on a heldout set of 1K samples. We fine-tune the GPT-2 model with a learning rate of $3e$-5 and batch size of 16.

3.1 Tasks

Algorithmic. We adopt the algorithmic task from [8], which involves printing the state, or value, of variables given algorithmic statements such as increment, decrement, and conditionals. While the original task has separate input and label tokens, we unify them into a single sequence to make it fit the language modeling task. Figure 2 shows an example of the data, with two different types of context: the original context, and context with self-QA. The final question is to print the value of one of the variables. In this dataset, the self-QAs are print statements specifying the intermediate value of a certain variable, so the special query token is "`print`"[1].

Toy-story. We introduce a new synthetic task for testing multi-step reasoning ability. It resembles a short story involving multiple people, items, and places. A story is a sequence of observed facts, each stating a simple relation such as "`Alice is at the park.`". The challenge in this dataset is that by applying pragmatic principles, unseen relations can be inferred from observed relations. For example, given the text "`Alice is at the park. Bob is with Alice.`", we can infer that "`Bob is at the park.`". Furthermore, an unseen inferred relation can lead to another unseen relation, thus

[1]The choice of the special token is arbitrary, and can be anything we train it to predict, as long as it is consistent.

Table 1: Test Accuracy (in %) for the Algorithmic and Toy-story tasks. The train and test columns show what percentage of samples in the split have access to ground truth self-QAs, or if they are generated by the model itself. Here * indicates out-of-distribution harder test settings.

Method	**Train self-QA**	**Test self-QA**	**Algorithmic**			**Toy-Story**	
			$\leq$50	**$\leq$100**	**$\leq$200***	**3-hop***	**4-hop***
GPT-2 fine-tuned	none	none	69.5	53.2	40.0	59.8	39.4
Scratchpad	100%	generated	**100.0**	61.0	30.0	98.6	95.4
Self-Questioning	100%	generated	**100.0**	**100.0**	**98.8**	**99.7**	**98.6**
	75%		99.5	99.6	98.3	98.9	98.5
	50%		98.8	98.2	95.0	98.9	96.8
	25%		95.8	93.5	85.7	97.1	95.1
Ablation	100%	100%	100.0	100.0	100.0	100.0	100.0
	100%	none	25.0	16.3	15.9	39.7	32.5

requiring multiple steps of inference. We call a question k-hop if it requires k observations combined through k-1 reasoning steps. For this dataset, the special query token is "`SQ:`". Following such a token, the model can ask and answer a question, e.g., "`SQ: Where is Bob? Bob is at the park.`".

3.2 Results

Algorithmic: Table 1 shows the results of all models trained on the Algorithmic dataset with 5 variables, where each sample has $2 \leq T \leq 100$ statements. The Algorithmic columns show the test accuracy for $2 \leq T \leq X$ statements, where X={50, 100, 200}. The vanilla GPT-2 model struggles to track the state of the variables over many statements, and decreases in accuracy with longer sequences. Our proposed Self-Questioning method, which allows the model to generate self-QA statements, achieves a perfect score when trained with 100% of the ground truth self-QA statements, and insignificant changes in performance when trained with 75%, 50%, or 25% of the ground truth self-QA statements. The $\leq$ 200 column shows the ability of methods to generalize to up to 200 statements (out-of-distribution). These results show a significant advantage of our method: as long as the model asks a self-question about a variable, it will keep it in memory by pushing its value to the recent context. The Scratchpad method has to copy the entire context in its scratchpad, often going past the maximum context length, resulting in poor accuracy. The oracle Self-QA train/test model gets perfect accuracy, while the train only model is poor, showing Self-QA must be performed at test time to achieve gains.

Toy-story: The rightmost two columns of Table 1 show the results for the Toy-story task where we train all the models only on 1- and 2-hop questions while testing them on 3- and 4-hop questions. For both the settings, we see that the Self-Questioning models substantially outperform the vanilla GPT-2 model which has to perform multi-step reasoning in "one-step". We see only a minor drop in the Self-Questioning models' performance as we reduce the self-QA supervision from 100% to 25%. The Self-Questioning model trained with supervision as low as 50% of the instances with self-QA pairs outperforms the Scratchpad method on both 3- and 4-hop questions despite Scratchpad having supervision for 100% training instances. However, unlike the Algorithmic task where Scratchpad's failure has to do with the model running out of space to write, for Toy-story instances, that's never the case. We reason that the drop in Scratchpad's performance has to do with the model having to postpone self-QAs after processing the entire input context, which increases the distance between the input context and the self-QAs in comparison to the proposed Self-Questioning approach where the self-QAs are written on the fly as the relevant facts are stated.

4 Discussion

We proposed a general method that allows language models to take internal notes in the form of self-QAs. Unlike scratchpad methods that postpone reasoning until all input tokens are processed, our method can deviate from the input sequence at any time for self-questioning and answering. One advantage of interleaving reasoning with the context in this way is that the reasoning steps can be

closer to their relevant contexts. Another advantage is that it can act as a recurrent memory as the self-QA answers are fed back to the model. Both these advantages make the method scale better to longer sequences unseen during training, as shown in our experiments. In addition, we showed that the amount of self-QA supervision during training can be reduced without a significant performance drop. Future work should explore ways to further reduce the supervision, perhaps using reinforcement learning. Another possible future direction is to combine our method with a scratchpad, which has the advantage of seeing the question and performing backward reasoning to reach the answer.

References

[1] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program Synthesis with Large Language Models. *arXiv*, abs/2108.07732, 2021.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017.

[3] 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. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

[4] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show Your Work: Scratchpads for Intermediate Computation with Language Models. *arXiv*, abs/2112.00114, 2021.

[5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *arXiv*, abs/2201.11903, 2022.

[6] E. Zelikman, Yuhuai Wu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. *arXiv*, abs/2203.14465, 2022.

[7] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *arXiv*, 2022.

[8] Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing Some Limitations of Transformers with Feedback Memory. *arXiv*, 2020.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[10] Da Ju, Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. Staircase Attention for Recurrent Processing of Sequences. *arXiv*, abs/2106.04279, 2021.

[11] DeLesley S. Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-Recurrent Transformers. *arXiv*, abs/2203.07852, 2022.

[12] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

[14] Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring Length Generalization in Large Language Models. *arXiv*, 2022.

5 Appendix

In this section we show one hand-picked test sample from both the Algorithmic task (Table 2) and the Toy-story task (Table 3). In both these examples, the Vanilla and Scratchpad methods fail, but the Self-Questioning method succeeds.

Table 2: Test sample from the Algorithmic task. In this example, the question (Q^*) is "`print d`" and the answer (A^*) is "`d = 3 ;`". The vanilla model fails at tracking the variable(s) and incorrectly predicts "`d = 2 ;`". The scratchpad runs past the GPT-2 context length, since the context window also includes the input text, and thus cannot generate a valid scratchpad end token, so it can't make a prediction. The Self-Questioning method correctly tracks the state of each variable as it sees statements, and successfully predicts "`d = 3 ;`".

Model	Context	Prediction
Vanilla (original context)	d = 6 ; c = 10 ; a = 10 ; d ++ ; if d < 1 : b = 8 ; c - ; d - ; d - ; a - ; b = 7 ; b ++ ; e = 3 ; a - ; b - ; if d < 10 : b - ; d - ; if d < c : a ++ ; e - ; b ++ ; a ++ ; if c > 3 : b ++ ; e - ; if e > d : b ++ ; if a > 8 : e ++ ; b ++ ; if c < 7 : b - ; c - ; b ++ ; d ++ ; e ++ ; a - ; if d < c : c ++ ; if e < 5 : b - ; d - ; if c > e : c ++ ; if b > 7 : b - ; if d < b : d ++ ; c - ; if c < d : e - ; b - ; c ++ ; d ++ ; e ++ ; b ++ ; d ++ ; b ++ ; e - ; d - ; if b < a : b ++ ; e - ; if b < 9 : b - ; a ++ ; if c > 10 : a - ; b ++ ; c - ; d ++ ; b - ; b - ; d ++ ; c ++ ; e - ; if c > b : e ++ ; if c > b : d - ; b ++ ; e - ; c - ; a - ; if c > 1 : e ++ ; if e < d : c ++ ; if e < 1 : c - ; a ++ ; d - ; b - ; c - ; c - ; if a > 4 : b - ; b ++ ; d - ; b ++ ; d - ; e - ; if a < 9 : e ++ ; if c < d : c - ; a - ; c ++ ; b - ; if a > 8 : d ++ ; a - ; d - ; d - ; c - ; b ++ ; a - ; c ++ ;	d = 2 ;
Scratchpad (scratchpad context)	[invalid scratchpad generation]	N/A
Self-Questioning (self-QA context)	c = 10 ; print c c = 10 ; a = 10 ; print a a = 10 ; d ++ ; print d d = 7 ; if d < 1 : b = 8 ; c - ; print c c = 9 ; d - ; print d d = 6 ; d - ; print d d = 5 ; a - ; print a a = 9 ; b = 7 ; print b b = 7 ; b ++ ; print b b = 8 ; e = 3 ; print e e = 3 ; a - ; print a a = 8 ; b - ; print b b = 7 ; if d < 10 : b - ; print b b = 6 ; d - ; print d d = 4 ; if d < c : a ++ ; print a a = 9 ; e - ; print e e = 2 ; b ++ ; print b b = 7 ; a ++ ; print a a = 10 ; if c > 3 : b ++ ; print b b = 8 ; e - ; print e e = 1 ; if e > d : b ++ ; print b b = 8 ; if a > 8 : e ++ ; print e e = 2 ; b ++ ; print b b = 9 ; if c < 7 : b - ; print b b = 9 ; c - ; print c c = 8 ; b ++ ; print b b = 10 ; d ++ ; print d d = 5 ; e ++ ; print e e = 3 ; a - ; print a a = 9 ; if d < c : c ++ ; print c c = 9 ; if e < 5 : b - ; print b b = 9 ; d - ; print d d = 4 ; if c > e : c ++ ; print c c = 10 ; if b > 7 : b - ; print b b = 8 ; if d < b : d ++ ; print d d = 5 ; c - ; print c c = 9 ; if c < d : e - ; print e e = 3 ; b - ; print b b = 7 ; c ++ ; print c c = 10 ; d ++ ; print d d = 6 ; e ++ ; print e e = 4 ; b ++ ; print b b = 8 ; d ++ ; print d d = 7 ; b ++ ; print b b = 9 ; e - ; print e e = 3 ; d - ; print d d = 6 ; if b < a : b ++ ; print b b = 9 ; e - ; print e e = 2 ; if b < 9 : b - ; print b b = 9 ; a ++ ; print a a = 10 ; if c > 10 : a - ; print a a = 10 ; b ++ ; print b b = 10 ; c - ; print c c = 9 ; d ++ ; print d d = 7 ; b - ; print b b = 9 ; b - ; print b b = 8 ; d ++ ; print d d = 8 ; c ++ ; print c c = 10 ; e - ; print e e = 1 ; if c > b : e ++ ; print e e = 2 ; if c > b : d - ; print d d = 7 ; b ++ ; print b b = 9 ; e - ; print e e = 1 ; c - ; print c c = 9 ; a - ; print a a = 9 ; if c > 1 : e ++ ; print e e = 2 ; if e < d : c ++ ; print c c = 10 ; if e < 1 : c - ; print c c = 10 ; a ++ ; print a a = 10 ; d - ; print d d = 6 ; b - ; print b b = 8 ; c - ; print c c = 9 ; c - ; print c c = 8 ; if a > 4 : b - ; print b b = 7 ; b ++ ; print b b = 8 ; d - ; print d d = 5 ; b ++ ; print b b = 9 ; d - ; print d d = 4 ; e - ; print e e = 1 ; if a < 9 : e ++ ; print e e = 1 ; if c < d : c - ; print c c = 8 ; a - ; print a a = 9 ; c ++ ; print c c = 9 ; b - ; print b b = 8 ; if a > 8 : d ++ ; print d d = 5 ; a - ; print a a = 8 ; d - ; print d d = 4 ; d - ; print d d = 3 ; c - ; print c c = 8 ; b ++ ; print b b = 9 ; a - ; print a a = 7 ; c ++ ; print c c = 9 ;	d = 3 ;

Table 3: Test sample from the Toy-story task. In this example, the question (Q^*) is "`Where is the ball?`" and the answer (A^*) is "`the ball is at the farm.`". The vanilla model fails at multi-step reasoning and incorrectly predicts that the ball is at the "`store`" The Scratchpad starts to incorrectly reason where the ball is early, and cannot correctly predict where it ends. The Self-Questioning method does each intermediate reasoning step correctly, in fact ends up questioning and correctly answering the location of the ball, and finally ends up predicting the correct location of the ball.

Model	Context	Prediction
Vanilla (original context)	Mary is with Daniel. Frank is with Sandra. John has the book. Frank has the suitcase. Daniel is at the station. the banana is inside the basket. Bob has the apple. Bob has the bag. the ball is inside the box. the apple is inside the bag. Alice has the banana. Alice has the key. John is at the farm. Charlie is at the bridge. the book is inside the box. Alice is at the store. Bob is with Alice.	the ball is at the store.
Scratchpad (scratchpad context)	[SQ: Where is Mary? A: Mary is at the station. SQ: Who has the ball? A: Bob has the ball. SQ: Who has the basket? A: Alice has the basket. SQ: Where is the book? A: the book is at the farm. SQ: Where is the bag? A: the bag is at the ball. SQ: Where is the ball? A: the ball is at the bag. SQ: Who has the box? A: John has the box. SQ: Where is the box? A: the box is at the farm. SQ: Who has the key? A: Alice has the key. SQ: Where is the banana? A: the banana is at the store. SQ: Where is the basket? A: the basket is at the store. SQ: Where is the key? A: the key is at the store. SQ: Where is Bob? A: Bob is at the store. SQ: Where is the apple? A: the apple is at the store. SQ: Where is the suitcase? A: the suitcase is at the store. SQ: Where is the basket? A: the basket is at the store.]	the ball is at the ball.
Self-Questioning (self-QA context)	Mary is with Daniel. Frank is with Sandra. John has the book. Frank has the suitcase. Daniel is at the station. SQ: Where is Mary? Mary is at the station. the banana is inside the basket. Bob has the apple. Bob has the bag. the ball is inside the box. the apple is inside the bag. Alice has the banana. SQ: Who has the basket? Alice has the basket. Alice has the key. John is at the farm. SQ: Where is the book? the book is at the farm. Charlie is at the bridge. the book is inside the box. SQ: Who has the box? John has the box. SQ: Where is the box? the box is at the farm. SQ: Who has the ball? John has the ball. SQ: Where is the ball? the ball is at the farm. Alice is at the store. SQ: Where is the banana? the banana is at the store. SQ: Where is the basket? the basket is at the store. SQ: Where is the key? the key is at the store. Bob is with Alice. SQ: Where is Bob? Bob is at the store. SQ: Where is the apple? the apple is at the store. SQ: Where is the bag? the bag is at the store. SQ: Where is the key? the key is at the store.	the ball is at the farm.