Can Sequence-to-Sequence Transformers Naturally Understand Sequential Instructions?

Xiang Zhou[1*] **Aditya Gupta**[2] **Shyam Upadhyay**[2] **Mohit Bansal**[1] **Manaal Faruqui**[2]
[1]UNC Chapel Hill [2]Google
{xzh, mbansal}@cs.unc.edu
{gaditya, shyamupa, mfaruqui}@google.com

Abstract

While many real-life tasks require reasoning over multi-step sequential instructions, collecting fine-grained annotations for each intermediate step can be prohibitively expensive. In this work, we study how general pretrained sequence-to-sequence transformers perform under varying types of annotation for sequential instruction understanding. We conduct experiments using T5 (Raffel et al., 2020) on a commonly-used multi-step instruction understanding dataset SCONE (Long et al., 2016) that includes three sub-tasks. First, we show that with only gold supervision for the final step of a multi-step instruction sequence, depending on the sequential properties of different tasks, transformers may exhibit extremely bad performance on intermediate steps, in stark contrast with their performance on the final step. Next, we explore two directions to relieve this problem. We show that with the same limited annotation budget, using supervision uniformly distributed across different steps (instead of only final-step supervision), we can greatly improve the performance on intermediate steps with a drop in final-step performance. Further, we explore a contrastive learning approach to provide training signals on intermediate steps with zero intermediate gold supervision. This, however, achieves mixed results. It significantly improves the model's bad intermediate-step performance on one subtask, but also shows decreased performance on another subtask.

1 Introduction

Transformer-based sequence-to-sequence models (Vaswani et al., 2017; Raffel et al., 2020) have shown remarkable performance on many natural language understanding tasks including semantic parsing (Yu et al., 2018), dialog state tracking (Budzianowski et al., 2018), procedure text understanding (Dalvi et al., 2018) etc. However, much of this success relies on fine-grained annotations. For example, many instruction-following datasets (Long et al., 2016) contain the corresponding parse or label for every single instruction showing their immediate effects. However, such data can be hard to collect in practice because even seemingly simple and straightforward tasks can involve multiple steps,[1] making the collection of detailed annotations expensive and time-consuming.

For these scenarios, many earlier works applied task-specific methods to provide additional inductive biases about the sequential nature of these instructions (Suhr and Artzi, 2018; Muhlgay et al., 2019). These methods need substantial prior knowledge and can be harder to generalize to new domains.[2] In this work, first, we provide a case study to explore whether transformer-based seq2seq models trained only using end-step supervision (i.e., gold supervision is given only at the very end of the entire sequence) can naturally handle these sequential instructions without task-aware specific architecture changes. We conduct experiments on the SCONE dataset (Long et al., 2016) including three different subtasks. The input of each example contains a sequence of instructions. During training, the model *only* observes the final state (label) after executing all the instructions, while for evaluation, the model needs to predict both the final states and all the intermediate states. We use T5 (Raffel et al., 2020) as our baseline model. Interestingly, we observe mixed trends on the three different subtasks of SCONE depending on their different sequential properties. On two out of three tasks (SCENE and TANGRAMS), T5 models demonstrate good performance on the intermediate steps.

*Work partially conducted during an internship at Google.

[1]For example, map instructions for how to reach the closest supermarket may involve a number of turns, cooking instructions may involve adding multiple different spices, etc.

[2]The prior knowledge is usually injected by either knowing the exact parses or grounded actions of each instruction, or by using a world simulator that can execute the instructions and facilitates RL-based approaches.

*Proceedings of the The 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 527–534

On ALCHEMY, however, the performance on intermediate steps is extremely bad, in stark contrast with their decent performance on final steps. Such behavior reveals that the model does not learn to understand the instructions sequentially and is not maintaining a correct intermediate state. Therefore, while these models may do well on instructions similar to the training examples, they can also fail miserably on instructions shorter or longer than the instructions they are trained on.

Hence, we next explore two potential mitigations to this problem. We first study an alternative labeling schema. We find that if the *same amount* of labels are uniformly sampled across multiple steps instead of only coming from the last step, the model can have substantially better performance at intermediate steps, despite a drop of the performance on the final step. This can be a favorable behavior if the target application has more focus on intermediate steps. However, re-collecting labels may not always be practical. Therefore, for scenarios where only final-step labels are accessible, we also explore a contrastive learning based approach to improve the intermediate-state performance without additional gold labels. Specifically, we use a contrastive learning loss to encourage an alignment between the change in the predicted states and the most recent instruction, and provide useful training signals on the intermediate steps. However, we see mixed results from this approach. While it can significantly improve the low intermediate-step performance on ALCHEMY, it decreases performance on SCENE and does not further improve other models that already have decent performance. Finally, we discuss the limitation of this approach and point out that the lack of precise regularization to capture the fine-grained state differences may be the reason behind the mixed results, which makes it hard to further improve strong baselines already showing sequential understanding abilities (as in SCENE).

2 Background and Baseline Performances

Problem and Evaluation Setup. We focus on sequential instruction following tasks, more specifically, state tracking or state prediction with multi-step instructions. Given an initial state and a sequence of instructions, the model needs to predict the states after the execution of each instruction. Formally, the training set $D^{train} = \{(inst^i_{j=1..m}, state^i_0, state^i_m)|^n_{i=1}\}$ contains n examples. Each example consists of a sequence of m instructions and two states, the initial state $state_0$, and the final state $state_m$ after executing all the m instructions. The training objective is to predict the final state $state_m$ given the initial state and all the previous instructions. The evaluation sets $D^{eval} = \{(inst^i_{j=1..m}, state^i_{j=0..m})|^{n'}_{i=1}\}$ contain not only the initial and the final state, but also all the intermediate states $state_j$ after every instruction $inst_j$. This allows us to evaluate the models' performance in two ways: (1) the exact-match accuracy at the final state (**acc$_{\text{final}}$**), similar to the training setup; and (2) the exact-match accuracy at all the states from $state_1$ to $state_m$ (**acc$_{\text{all}}$**).

Dataset. We use the SCONE dataset as it contains three different subtasks: ALCHEMY, SCENE and TANGRAMS (Long et al., 2016), and covers a diverse set of different states and instructions. For every example in these three subtasks, the instruction contains 5 steps. See Appendix A for examples and a detailed dataset introduction.

Baseline Performances with Final-Step Supervision. We use T5-base (Raffel et al., 2020) as our main model.[3] At each step, to get the prediction of $state_i$, the model will receive an input containing the concatenation of the initial state $state_0$ and all the instructions from $inst_1$ till $inst_i$. More hyperparameter and preprocessing details are in Appendix A and B. The performance is shown in Table 1. First, if we follow the traditional setup for previous papers to use gold labels across all the steps (the first row), fine-tuned T5 models without any task-specific tricks can already achieve strong performance on **acc$_{\text{final}}$**, reaching competitive performance on all three subtasks compared to all previous methods (including Shi et al. (2022) who also uses pre-trained Transformer-based models) using similar supervision, and the performance on ALCHEMY is even higher. By using all the gold labels across steps, the performances are substantially higher than the results only using final-step supervision. This observation is also connected to the observation in Wies et al. (2023) and Yu et al. (2023), where they notice the decomposition of complex tasks makes learning easier. When we only use final-step supervision (the second row), both **acc$_{\text{all}}$** and **acc$_{\text{final}}$** decrease substantially. However, the trends on different subtasks are different. On SCENE and TANGRAMS, the **acc$_{\text{all}}$** is equal or

[3] Preliminarily, we also conduct our experiments on other scales of T5 (i.e., T5-small and T5-large), but they do not show better performance on our tasks.

Models	Supervision	ALCHEMY		SCENE		TANGRAMS	
		$\textbf{acc}_{\textbf{final}}$	$\textbf{acc}_{\textbf{all}}$	$\textbf{acc}_{\textbf{final}}$	$\textbf{acc}_{\textbf{all}}$	$\textbf{acc}_{\textbf{final}}$	$\textbf{acc}_{\textbf{all}}$
T5-base	All steps	77.0±0.9	86.6±0.5	72.9±1.8	84.9±0.7	60.1±2.4	79.0±0.8
T5-base	Final step	70.0±1.7	58.0±3.8	60.5±2.9	71.7±3.8	14.2±4.5	22.3±6.2
+CL	Final step	70.7±2.4	72.4±4.4	62.5±2.3	60.8±2.0	14.7±6.7	29.0±12.1
T5-base	Uniformly sampled steps	62.8±3.0	80.0±1.3	54.3±1.5	75.2±0.8	23.7±3.2	60.2±2.5
Shi et al. (2022)	All steps	75.4	-	72.3	-	60.0	-
Suhr and Artzi (2018)	All steps	62.7	-	62.0	-	62.4	-
Yeh and Chen (2019)	All steps+Annotated programs	76.1	-	75.1	-	72.5	-

Table 1: Model performance on the SCONE dataset. The numbers in this table are mean and std over 10 runs.

Task	Instructions
ALCHEMY	Instruction i: *throw out the right most orange chemical* Instruction $i+1$: *throw out 2 units of the purple chemical*
SCENE	Instruction i: *he disappears* Instruction $i+1$: *the person in all orange moves one step right*
TANGRAMS	Instruction i: *remove the first figure* Instruction $i+1$: *swap the first and third figures*

Table 2: Example instructions from the three subsets in SCONE. Due to the heavy use of coreference, changing the order of instructions in SCENE and TANGRAMS can lead to different results, while a larger number of examples in ALCHEMY are interchangeable as they may refer to independent actions for different beakers.

higher than $\textbf{acc}_{\textbf{final}}$, showing that the models already have a tendency to track the semantics on intermediate steps and early steps are easier than later steps. On the contrary, the $\textbf{acc}_{\textbf{all}}$ performance on ALCHEMY is substantially lower than $\textbf{acc}_{\textbf{final}}$. Such low performance indicates that after training on the ALCHEMY, despite the decent $\textbf{acc}_{\textbf{final}}$, the model does not always maintain a correct state in the intermediate steps.

Why are the trends different across subtasks? The three subtasks in SCONE are designed to focus on different linguistic phenomena (Long et al., 2016). Here, we argue these different designs cause T5 to correctly understand the sequential nature of the instructions on SCENE and TANGRAMS and achieve good $\textbf{acc}_{\textbf{all}}$, but not on ALCHEMY. Due to the focus on the coreference across steps (see Table 2 and dataset descriptions in Appendix A), instructions in SCENE and TANGRAMS are more sensitive in their order, because switching the order of instructions can break the coreference and lead to different outcomes. Specifically, only 39% of the instruction pairs in ALCHEMY are non-interchangeable in their orders, compared to 62% in SCENE and 85% in TANGRAMS.[4] These non-interchangeable instructions encourage the model to keep tracking the state change in a correct sequential way. Otherwise, as in ALCHEMY, the model may not have a strong incentive to follow the order of the instructions and understand them sequentially. Nonetheless, our finding here is *not* a dataset design problem, as many real-life instructions can have the same property, but more about analyzing the effect of differing dataset properties. Additionally, these results can be seen as empirical evidence about how or whether fine-tuned seq2seq models form internal meaning representations when only final-supervision is given, complementing the study by Li et al. (2021). Models with internal meaning representations should have higher $\textbf{acc}_{\textbf{all}}$ than $\textbf{acc}_{\textbf{final}}$ as representations at later steps are built on representations at earlier steps so they will be more error-prone. Therefore, our experiments imply that the exact behavior may depend on the nature of the fine-tuning tasks. On ALCHEMY, the model shows no significant evidence of maintaining a reliable meaning representation, while on the other two tasks, the model shows hints of maintaining a meaning representation even with final step supervision.

3 Intermediate State Prediction with Uniformly Sampled Annotations

One of the major reasons behind the poor performance in Sec. 2 is that all the gold labels are at the final step, so for the intermediate steps, there is no strong supervision to ensure a desirable behavior. While for many applications, final-step labels are indeed more natural to collect, in this section, we explore if a better annotation strategy can improve the performance with the *same amount of labeling budget*. Specifically, we replace the final-step-only supervision with the same amount of supervision

[4] These statistics are manually estimated by the authors from 100 randomly-sampled instruction pairs from each task.

distributed uniformly across different steps. Such labels can reduce the gap between training and evaluation, and the model can receive supervision at intermediate steps. The results with such uniformly sampled labels are shown in the fourth row of Table 1. Compared to the final-step supervision results, we notice a substantial improvement on **acc$_{\text{all}}$** on all three subtasks, but there is a drop on **acc$_{\text{final}}$** on two subtasks (ALCHEMY and SCENE). Therefore, the preference between uniformly-distributed labels and end-step only labels depends on the final target. Additionally, there still exist many applications where the intermediate labels are difficult to collect or there is no budget to re-annotate labels. For those cases, we next describe our exploration to improve **acc$_{\text{all}}$** without additional gold labels.

4 Intermediate State Prediction with Contrastive Learning

Method. In Sec. 2, our baseline T5 predicts all the intermediate states independently, similar to the re-translation strategy (Arivazhagan et al., 2020) in streaming MT. However, it ignores the strong correlation of predictions over different steps, which partially leads to weak intermediate-step results. Next, we use contrastive learning to leverage such correlation without gold labels.

We first introduce the notations. We denote the function learned by the seq2seq transformer as $f(state_0, inst_1, \ldots, inst_i) = p_i^{state}$. Here $state_i$ and $inst_i$ are input tokens representing the state representation at step i and the instruction at step i respectively. p_i^{state} is the model prediction of the state at step i, which is a sequence of categorical distributions. The length of the sequence is the total number of tokens of the state representation, and each categorical distribution is over the vocabulary. For two consecutive steps, the seq2seq model produces two predictions p_i^{state} and p_{i+1}^{state}.

Our main intuition is to leverage the observation that *"There is a strong correlation between **the change in two consecutive states** and **the instruction of that step**."* To implement this idea, we compute two sets of vectors, one for the difference in consecutive states, and the other to represent the instruction. Then we use contrastive learning to encourage matching between these two sets of embeddings. Concretely, we start from the model predicted distribution p_i^{state}. We map the distribution back to the embedding space by computing $e_i^{state} = Ep_i^{state}$ where E is the input embedding matrix of the seq2seq model. Then, we compute a vector h_i^{state} to represent each state by computing the transformer-style multi-layer self-attention between the embeddings e_i^{state} and an additional learnable vector h^s. $h_i^{state} = \text{SelfAtt}([h^s, e_i^{state}])$. Now, with two state embeddings for two consecutive steps, we can compute a difference vector that captures the difference in consecutive states following Conneau et al. (2017): $h_i^{diff} = \text{MLP}(h_i^{state}, h_{i+1}^{state}, |h_i^{state} - h_{i+1}^{state}|, h_i^{state} \odot h_{i+1}^{state})$. For the instruction vector, we directly feed the latest instruction $inst_{i+1}$ to an off-the-shelf sentence-T5 model (Ni et al., 2022).[5] With these two set of embeddings, we compute a contrastive matching loss (Gao et al., 2021): $\mathcal{L}_{cont} = \frac{\exp(\text{sim}(h_i^{diff}, h_{i+1}^{inst}))}{\sum_{\text{all } inst \text{ in the batch}} \exp(\text{sim}(h_i^{diff}, h^{inst}))}$, where sim is the similarity function, and we use all the other in-batch examples as negative examples. An illustration of this idea is at Fig. 1. The total training loss will be the sum of both standard MLE loss and the contrastive loss, $\mathcal{L}_{total} = \mathcal{L}_{MLE} + \mathcal{L}_{cont}$.

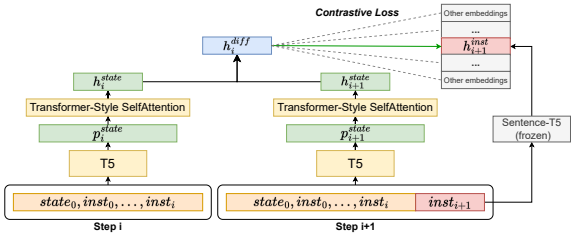

Figure 1: Contrastive learning encourages matching between state differences and corresponding instructions.

Results. The results for our contrastive learning method are in the "+CL" row of Table 1. We see opposite trends on ALCHEMY and SCENE. On ALCHEMY, we can see a substantial increase on **acc$_{\text{all}}$**, improving it from the extremely low accuracy of 58.0% to 72.4%, which is comparable to its **acc$_{\text{final}}$**, and making its behavior similar to other tasks. We can also observe a small improvement on the TANGRAMS subtask. However, such improvement does not translate to other settings where the **acc$_{\text{all}}$** performance is already decent and is comparable to **acc$_{\text{final}}$**. For instance, on SCENE, adding our contrastive learning method does not improve either **acc$_{\text{final}}$** or **acc$_{\text{all}}$**, and leads to a drop on **acc$_{\text{final}}$**. We also do not observe additional gain by combining contrastive learning with the uniformly sampled annotation described in Sec. 3. We conjec-

[5] Preliminarily, we tried to extract embeddings from our model itself, but observe no substantial improvement.

ture such mixed results may result from a lack of more fine-grained control on h_i^{diff}, as the current implementation may allow h_i^{diff} to encode irrelevant features from one of the consecutive steps. This lack of more precise regularization makes it hard to further improve strong baselines already showing sequential understanding abilities (e.g., on SCENE). See Appendix C for more discussions.

5 Related Works

Our work focuses on sequential instruction understanding. Many earlier works in this direction rely on a pre-defined action set or a world simulator that facilitates the inference of the semantics of each sentence (Long et al., 2016; Guu et al., 2017; Suhr and Artzi, 2018; Muhlgay et al., 2019). Neural models can bring additional improvement, especially with specifically designed architectures (Huang et al., 2018; Yeh and Chen, 2019) or training methods (Fried et al., 2018; Shi et al., 2022). Our work advances this direction by examining transformer seq2seq models in limited supervision settings, and providing solutions for undesirable behaviors. Many other tasks (Anderson et al., 2018; Dalvi et al., 2018; Kiddon et al., 2015) also require understanding the sequential relationship between sentences. The contrastive learning component can also be viewed as a way to relieve the reward sparsity problem, similar to the effect of forward modeling (Pathak et al., 2017). Pretrained transformers have been applied to many different tasks. However, it is unclear how they process sequences inherently. Li et al. (2021) study whether language models implicitly build meaning representations. Our empirical results provide evidence of different behaviors in different datasets.

6 Conclusion

We study seq2seq transformers for sequential instruction following. Depending on data properties, if only final-supervision is given, transformers may naturally perform well on intermediate steps, but can also fail miserably. We explore two potential remedies, one with uniformly sampled supervision, and the other with contrastive learning.

Acknowledgments

We thank the reviewers for their helpful comments. This work was partially supported by NSF-CAREER Award 1846185, ONR Grant N00014-18-1-2871, and DARPA Machine Commonsense (MCS) Grant N66001-19-2-4031. The views are those of the authors and not of the funding agency.

Limitations

The main limitation of this work is a limited choice of models and datasets. In our work, we mainly tested with T5-base models. We expect our claims to hold on models with similar scales, but models with significantly more parameters (e.g., GPT-3) may demonstrate different fine-tuning behaviors. Additionally, we observe different behaviors on three subtasks on SCONE, so in order to generalize our finding to other datasets and predict the intermediate step performance, one needs to judge whether the new dataset is more similar to ALCHEMY or more similar to SCENE and TANGRAMS, which may not be straightforward for some applications.

Ethics Statement

Our work studies transformer-based seq2seq models for sequential instruction following tasks. Our experiment is conducted on synthetic domains in the SCONE (Long et al., 2016) dataset, and aims to have a better understanding of transformer models. We do not foresee any ethical risk for this work.

References

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, and George Foster. 2020. Re-translation versus streaming for simultaneous translation. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 220–227.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: Composable transformations of Python+NumPy programs.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural*

Language Processing, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. 2018. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, New Orleans, Louisiana. Association for Computational Linguistics.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified pragmatic models for generating and following instructions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1951–1963, New Orleans, Louisiana. Association for Computational Linguistics.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Hsin-Yuan Huang, Eunsol Choi, and Wen-tau Yih. 2018. Flowqa: Grasping flow in history for conversational machine comprehension. In *International Conference on Learning Representations*.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 982–992, Lisbon, Portugal. Association for Computational Linguistics.

Belinda Z. Li, Maxwell Nye, and Jacob Andreas. 2021. Implicit representations of meaning in neural language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1813–1827, Online. Association for Computational Linguistics.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Berlin, Germany. Association for Computational Linguistics.

Dor Muhlgay, Jonathan Herzig, and Jonathan Berant. 2019. Value-based search in execution space for mapping instructions to programs. 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 1942–1954, Minneapolis, Minnesota. Association for Computational Linguistics.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. 2022. Sentence-t5: Scalable sentence encoders from pretrained text-to-text models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1864–1874, Dublin, Ireland. Association for Computational Linguistics.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. 2022. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*.

Qi Shi, Qian Liu, Bei Chen, Yu Zhang, Ting Liu, and Jian-Guang Lou. 2022. LEMON: Language-based environment manipulation via execution-guided pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 471–485, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Alane Suhr and Yoav Artzi. 2018. Situated mapping of sequential instructions to actions with single-step reward observation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2072–2082, Melbourne, Australia. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Noam Wies, Yoav Levine, and Amnon Shashua. 2023. Sub-task decomposition enables learning in sequence to sequence tasks. In *The Eleventh International Conference on Learning Representations*.

Yi-Ting Yeh and Yun-Nung Chen. 2019. FlowDelta: Modeling flow information gain in reasoning for conversational machine comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 86–90, Hong Kong, China. Association for Computational Linguistics.

Hangyeol Yu, Myeongho Jeong, Jamin Shin, Hyeongdon Moon, Juneyoung Park, and Seungtaek Choi. 2023. Towards zero-shot functional compositionality of language models.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

A Dataset

In this section, we provide a detailed description of the dataset we use in our experiments and the preprocessing steps. The experiment in this paper focuses on the SCONE (Long et al., 2016) dataset. The SCONE dataset contains three subtasks: ALCHEMY, SCENE, and TANGRAMS. Each subtask focuses on a different domain and highlights different linguistic properties. The ALCHEMY task includes instructions about mixing colored chemicals in 7 beakers, and focuses on the ellipsis phenomenon. The SCENE task includes descriptions about people's movement in a scene, and focuses on object coreference. The TANGRAMS task includes instructions to manipulate tangram pieces, and focuses on action coreference. Table 3 shows the dataset statistics of all three datasets. The only data filtering we used in this work is that we removed a few examples from the TANGRAMS task where it does not contain 5 complete instructions. Other than that, we use all the examples in the original dataset.

Task	Train	Dev	Test
ALCHEMY	3567	245	899
SCENE	3352	198	1035
TANGRAMS	4159	198	990

Table 3: Dataset statistics for the SCONE (Long et al., 2016) dataset. The numbers in this Table are the number of examples. Each example will contain 5 steps.

State Representation Linearization Pretrained Transformers, including T5s used in this paper, are shown to be sensitive to the output format. Therefore, we convert the original output format in SCONE into a more readable text description. An example for each subtask can be seen in Table 4.

B Implementation Details

All the models used in this work are implemented using JAX (Bradbury et al., 2018) and the T5x (Roberts et al., 2022) framework. For all the experiments, we finetune the T5-v1.1-base model. We use a batch size of 128, a constant learning rate of 0.0001, and a dropout rate of 0.1. For the ALCHEMY task, we finetune for 100k steps. For the SCENE and TANGRAMS tasks, we notice the model converges faster, so we finetune for 50k steps. For the contrastive learning experiments, the instruction embeddings are extracted using sentence-T5-base (Ni et al., 2022) models. We use cosine similarity as the similarity function in the contrastive loss. All our experiments are conducted on Google v3 TPUs.

C More Discussions about Contrastive Learning Results

In Sec. 4, we notice that while our contrastive learning approach can improve the low $\mathbf{acc_{all}}$ on ALCHEMY, it fails to consistently improve in other settings, especially when the baseline performance is already decent on the SCENE subtask. One of our observations that may prevent contrastive learning from further improvement is the tendency for the contrastive loss to overfit during the training. In our experiments, we often observe a significant gap between the contrastive matching accuracy on the training set and on the development set. This problem is very likely to be caused by the lack of regularization in the current implementation of the

Task	Example state	Example instructions
ALCHEMY	1: empty empty empty empty, 2: empty empty empty empty, 3: empty empty empty empty, 4: empty empty empty empty, 5: orange empty empty empty, 6: orange orange orange empty, 7: green green green green	Instruction 1: pour the last orange beaker into beaker two Instruction 2: then into the first ...
SCENE	1: red empty, 2: empty empty, 3: empty empty, 4: empty empty, 5: green empty, 6: green orange, 7: yellow orange, 8: empty empty, 9: yellow empty, 10: empty empty	Instruction 1: the man in the red hat takes a step to the right Instruction 2: he's joined on his left by a person wearing a blue shirt ...
TANGRAMS	1: two, 2: one, 3: four, 4: zero, 5: three	Instruction 1: delete the second object from the left Instruction 2: undo that ...

Table 4: Example linearized states and instructions used in this work for three subtasks of SCONE. For graphic demonstrations of these states and instructions, please visit https://nlp.stanford.edu/projects/scone/

difference vector. In our current implementation, the only constraint the difference vector have is that it needs to be a function of consecutive states, i.e. $h_i^{diff} = f(h_i^{state}, h_{i+1}^{state})$. While this implementation can capture the difference between the states, and can help when the model's performance is bad (as empirically verified on ALCHEMY), it can also capture many irrelevant features, which helps reduce the contrastive matching loss, but does not help the model to correct its prediction on intermediate steps. In our experiments, we have also tried several approaches to resolve this problem, including having hard negatives in contrastive learning, having an auto-encoder style reconstruction loss, etc. But none of these methods solves this problem effectively. Hence, we leave a more in-depth exploration of this direction for future work.