TALM: Tool Augmented Language Models

Aaron Parisi  Yao Zhao  Noah Fiedel

{aarontp,yaozhaoyz,nfiedel}  
@google.com

Abstract

Transformer based language models (LMs) demonstrate increasing performance with scale across a wide variety of tasks. Scale alone however cannot enable models to solve tasks that require access to ephemeral, changing, or private data that was unavailable at training time. Many useful tasks may also benefit from LMs being able to access APIs that read or modify state. In this work, we present Tool Augmented Language Models (TALM), combining a text-only approach to augment language models with non-differentiable tools, and an iterative “self-play” technique to bootstrap performance starting from few tool demonstrations. TALM exhibits strong performance on both a knowledge-heavy QA task and a reasoning oriented math task with simple tools. At a given model scale, TALM significantly outperforms non-augmented LMs. We further demonstrate that TALM successfully performs out-of-distribution inferences on both QA and math tasks, where non-augmented LMs fail. Our results suggest that Tool Augmented Language Models are a promising direction to enrich LMs’ capabilities, with less dependence on scale.

1 Introduction

Language models using the Transformer architecture [Vaswani et al., 2017] demonstrate increasing performance at larger scales, e.g. T5 [Raffel et al., 2019], GPT-3 [Brown et al., 2020], and PaLM [Chowdhery et al., 2022]. Scale related performance gains are observed on a variety of benchmarks, e.g. SuperGLUE [Wang et al., 2019] and BIG-bench [BIG-bench collaboration, 2021].

Scaling up has practical downsides. Large scale models are unwieldy to store, transfer, and deploy. Their costs to train or perform inference can be prohibitively high for many researchers and organizations.

Larger models memorize more world knowledge [Roberts et al., 2020]. While good for many benchmark tasks, relying on memorization alone poses several problems. First, models sometimes generate incorrect outputs that are problematic for some applications. Second, world knowledge is constantly changing. The knowledge from yesterday’s training data might be invalid today. Third, large models can memorize parts of their training data with undesirable consequences [Carlini et al., 2022].

Retrieval based approaches to enhancing LMs can lower the dependence on scale. REALM [Guu et al., 2020] learns retrieval via backpropagation from a fixed corpus. RETRO [Borgeaud et al., 2021] adds an “internet scale” retrieval mechanism. RAG [Lewis et al., 2020] uses a dense vector index of Wikipedia, and retrieves either once per token or once per query. Other works demonstrated that LMs can be enhanced on math reasoning with access to a calculator [Andor et al., 2019].

Looking towards the future utility of language models, it is clear that scale and retrieval cannot solve all useful problems. Many knowledge tasks and desirable applications require access to read live or private data (e.g. weather or a person’s cal-
end or to invoke APIs that modify state. Recent works such as Say Can [Ahn et al., 2022] connect languages models to an environment, though with the model as a recipient of queries. In contrast, TALM’s approach enables models to invoke arbitrary tools with model-generated output, and to attend to tool output to generate task outputs.  

In summary, our contributions are:

- Demonstrating that language models can be augmented with tools via a text-to-text API.
- Demonstrating an iterative self-play technique to bootstrap tool-augmented datasets and subsequent tool-augmented model performance, from few labeled examples.

## 2 Methods

We use pretrained T5 models [Raffel et al., 2019, Roberts et al., 2022] for finetuning, inference and evaluation. To measure the effects of model scaling, we use the base, large, and XL sizes.

### 2.1 Tool Augmented Language Models

![Figure 2: LM and Tool Augmented LMs.](image)

We use a Text-To-Text tool interface given its broad applicability and simplicity, as shown in Fig. 3. TALM first generates a tool input conditioned on the task input text and invokes a tool’s API by generating a delimiter, such as “\text{\texttt{|result}}”. Whenever this delimiter is detected, the tool API is called and its result appended to the text sequence. TALM then continues to generate the final task output.

**An abstract task:**

| task input text | tool-call | tool input text | result tool output text | output task output text |
|-----------------|-----------|-----------------|-------------------------|------------------------|

**A weather task:**

| how hot will it get in NYC today? | weather lookup region=NYC | result precipitation chance: 10, high temp: 20°C, low-temp: 12°C | output today's high will be 20°C |

![Figure 3: TALM text-to-text interface example.](image)

TALM learns two subtasks at the same time: calling a tool and generating an answer based on tool results. TALM is architecturally agnostic and can be implemented as Seq2Seq, left-to-right LM or prefix LM. We chose the Seq2Seq family for its high finetuning performance at modest scale [Raffel et al., 2019].

### 2.2 Iterative self-play

When introducing new tools to solve existing tasks, there are often a limited number of demonstrations of tool interactions. However, there is typically plenty of supervised task data consisting of input and target pairs, and automated metrics for evaluating the correctness of a generated output. Inspired by Decision Transformer [Chen et al., 2021], we use a self-play approach to iteratively bootstrap examples of tool-use with progressively higher quality. In this work, we refer to a model interacting with a tool API as self-play rather than adversarial play among models.

**Algorithm 1 Iterative Self-Play Algorithm.**

\begin{algorithm}
  \begin{align*}
  1: T & = \{x_i, y_i\}_T \quad \text{# task set} \\
  2: D & = \{x_j, t_j, r_j, y_j\}_D \quad \text{# tool-use set} \\
  3: P_\theta & \leftarrow \text{pretrained LM} \\
  4: \text{for } t \in \{0, 1, \ldots, R\} \text{ do} \quad \text{# self-play rounds} \\
  5: & \text{# finetune LM} \\
  6: & \theta \leftarrow \text{argmax} \prod_{D} P_\theta(y_j | x_j, t_j, r_j) P_\theta(t_j | x_j) \\
  7: & \text{for } x_i, y_i \in T \text{ do} \quad \text{# iterate task set} \\
  8: & \text{for } n \in \{0, 1, \ldots, N\} \text{ do} \quad \text{# sample tool query} \\
  9: & t_n \leftarrow P_\theta(t|x) \\
  10: & r_n \leftarrow \text{Tool}(t_n) \quad \text{# call tool API} \\
  11: & y_n \leftarrow P_\theta(y|x, t_n, r_n) \quad \text{# get task output} \\
  12: & \text{if } |y_n - y_i| < \text{th} \quad \text{# filter wrong output} \\
  13: & D \leftarrow D \cup \{x_i, t_n, r_n, y_n\} \quad \text{# update tool-use set} \\
  \end{align*}
\end{algorithm}

The iterative self-play pipeline starts with a small tool-use bootstrapping set \( \{x_j, t_j, r_j, y_j\}_D \). In each round of self-play, the TALM is fine-tuned on the tool-use set \( D \). Next, for every example in the task set \( T \), the TALM samples tool inputs, calls a tool API, and samples task outputs based on the tool results. If the TALM generated task output matches the target within some threshold \( \text{th} \), the tool-use sequence led to the result is added to the tool-use set \( D \) for the next round of self-play.

To explore diverse tool API invocations and answers during self-play, the TALM decodes using random sampling with temperature=1.0, and top-
To grow the dataset during self-play, the TALM generates up to \( N = 600 \) tool-use sequences per example. At evaluation time, the model uses beam decoding with 4 beams to generate a single output.

We note that this iterative self-play pipeline represents a special case of a policy-gradient RL algorithm, where the LM is the policy network and is trained by policy gradient with a binary reward signal. Iterative self-play is related to expert iteration [Anthony et al., 2017], which has been demonstrated to work well in tasks with extremely weak supervision [Christiano et al., 2018]. While our tasks are currently single-hop, this formulation can be extended further into RL: modelling multi-hop tool-use tasks as markov decision processes (MDPs), or integrating algorithms like Decision Transformer [Chen et al., 2021].

3 Results

We evaluate TALM on two domains. The first is the knowledge-oriented Natural Questions (NQ) [Kwiatkowski et al., 2019], a diverse QA task. The second is MathQA [Amini et al., 2019], selected to measure general reasoning capability rather than knowledge.

3.1 Natural Questions

Natural Questions (NQ) is a large (≈ 300k training examples) QA dataset collected from real user queries. NQ contains both long and short answer tasks. We selected the short answer task as it is both more challenging as measured with lower baseline performance, and closer to practical use-cases such as assistants. In addition to a question and short-answer pair, examples in the NQ dataset include an “oracle” context (span) of a Wikipedia document containing the answer. We remove boolean questions to avoid inflated performance due to random-chance guesses. We compare TALM against closed-book LM benchmarks.

For TALM experiments, we do not feed the oracle contexts directly to the model, instead using them to populate an index that TALM can access as a retrieval tool. The retrieval system is implemented using a BM25-based index over the union of all NQ oracle contexts.

In Fig. 5, even the 220M base TALM outperforms 3B XL LM. There is also a smaller performance gap between base and XL sized TALMs than between TALM and LM, suggesting that smaller models benefit more from retrieval tools for knowledge intensive tasks.

3.2 MathQA

MathQA [Amini et al., 2019] is a large scale dataset of math word problems (≈ 30k training examples). Each example includes the word problem, a formula generated by crowd-source workers to calculate the answer, and the correct text-form answer among multiple choices.

We implemented a simple solver tool to...
cute formulas and check their results’ correctness against their associated text-form answers. According to our solver tool, approximately 70% of the formulas in MathQA produce results that match their corresponding answers, similar to the findings in [Hendrycks et al., 2021]. Our manual inspections show that mismatched results are due to either wrong formulas or invalid answers. The bootstrap tool-use dataset consists of a random sample of 10% of the training corpus where the formula is valid ($\approx 2k$ examples). The TALM significantly outperforms a non-augmented LM as shown in Fig. 7.

On the math task, we test large number handling, an area where training data is lacking and non-augmented LMs are known to perform poorly [Brown et al., 2020]. See Fig. 9 demonstrating that TALM can handle large numbers, where a LM does not.

Question: A car is driving 535 miles per hour, how many hours does it take to travel 2450 miles?

LM: 8.5
TALM: 4.58

Figure 9: LM vs TALM on a large number operation.

4 Conclusion

In this paper we present TALM, a framework for augmenting language models with arbitrary tools. TALM has two key ideas. First, we model tool-use via a text-to-text interface. Second, we apply an iterative self-play technique to bootstrap high performance on tasks with few tool-use labelled examples. Taken together, this interface and technique make exploring additional tools and tasks possible, without requiring expensive data labeling efforts.

TALM consistently outperforms a non-augmented LM on both a knowledge task (NQ) and reasoning task (MathQA). Ablations show that self-play is key to good performance, and that iterative self-play yields further gains. We conclude that the combination of tool augmentation and iterative self-play enables smaller models to outperform larger non-augmented LMs.

We hope that this work enables further research into tool augmented language models, a promising direction to enhance model capabilities with less dependency on scale than many contemporary approaches.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jaurigue Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, and
5 Appendix

5.1 Acknowledgements

The authors would like to thank Noam Shazeer for early brainstorming on the path towards this work. We also thank Igor Mordatch for discussions and feedback. Finally we thank Mohammad Saleh for his helpful review and feedback improving this manuscript.

5.2 Author Contributions

This section lists the author contributions of each author.

- Aaron Parisi designed and implemented tool-augmentation and self-play pipelines. Aaron ran the vast majority of experiments, and participated in brainstorming and paper writing.

- Yao Zhao participated in brainstorming, experimental setup discussion and paper writing. Yao implemented NQ/mathQA baselines and mathQA solvers.

- Noah Fiedel conceived of the project, participated in brainstorming, led the research group and writing the paper.