Empowering Language Models with Toolformer: An Implementation

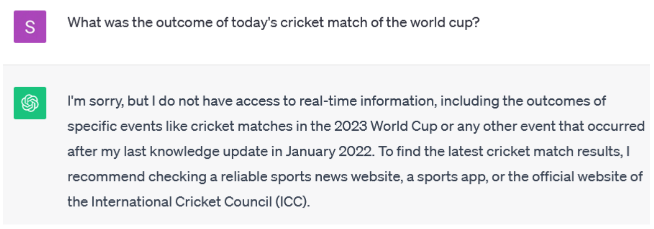
Madhushree Nijagal, Sumedha Joshirao, Tanisha Hegde
University of Wisconsin-Madison

ABSTRACT

Language models (LMs) exhibit a remarkable capacity for problem-solving with limited instructions on a large scale. However, they encounter difficulties with basic arithmetic operations and factual retrieval. In this project, our aim is to implement the Toolformer model using a reference code that is available and extend its capabilities to incorporate other APIs, such as Wolfram Alpha and Calendar APIs. Toolformer is a trained model designed to make decisions regarding API selection, timing, argument choices, and the effective integration of results into future token predictions. Our primary goal is to overcome the limitation of Toolformer's sensitivity to the exact wording of input when deciding whether or not to call an API. Removing this limitation will enable us to evaluate Toolformer on various benchmarks and fine-tune the model for different APIs. The original paper primarily focuses on four essential tools for API calls: a Calculator, Wikipedia, a QA system, and a translation system. Our objective is to enhance the model's functionality by expanding its scope to include additional API tools, beginning with addressing the sensitivity limitation.

1 INTRODUCTION

The integration of artificial intelligence and machine learning has been one of the most transformative advancements in the field of computer science and technology. Among the many breakthroughs in this domain, the emergence of powerful language models has been a game-changer. These models, with their immense capacity to understand and generate human language, have found applications in a wide range of tasks, from natural language processing to content generation and translation. While their proficiency in text-related applications is well-documented, researchers and engineers have been increasingly exploring their capabilities beyond the realm of mere text understanding.
Language models (LMs) are versatile tools for numerous language processing tasks. However, these models come with certain limitations that hinder their capacity to deliver precise information. These limitations arise from their inability to access real-time updates, a tendency to generate fictitious information, limited mathematical capabilities, and a lack of comprehension regarding the temporal advancement. The Toolformer paper [4] aims to address these challenges by empowering the model with the capability to utilize external resources such as search engines, calculators, and text translation services.


S
What was the outcome of today's cricket match of the world cup?

I'm sorry, but I do not have access to real-time information, including the outcomes of specific events like cricket matches in the 2023 World Cup or any other event that occurred after my last knowledge update in January 2022. To find the latest cricket match results, I recommend checking a reliable sports news website, a sports app, or the official website of the International Cricket Council (ICC).

Figure 1: An example output for a real-time question to ChatGPT

Figure 1 illustrates an example where ChatGPT, a language model, encounters difficulty in providing a real-time answer regarding the outcome of the World Cup 2023 cricket match. This response underscores the limitations of language models, which generate answers based on the data they were trained on, making them less adept at handling questions about events that occur after their training.

Consequently, to ensure the accuracy of model responses, it becomes imperative to grant the model access to real-time data through various tools such as search engines and tool APIs. Along with this, it is also important that the model understands which tools to use and when while preserving the model's existing functionality.

The Toolformer model was developed keeping all these requirements in mind. This model learns the usage of these external tools in a self-supervised way without feeling the need of huge amounts of human annotated data and ensures that the generality of the model is not lost. The current implementation of the Toolformer model is trained for these five tools: a question answering system, a Wikipedia search engine, a calculator, a calendar and a machine translation system.

Within the scope of our project, we have the objective of enhancing the model's functionality by delving into additional tools beyond those investigated by the authors. Specifically, we are contemplating the incorporation of tools such as the Weather forecasting system. The subsequent sections delve into the existing research in this domain, perform an experimental analysis on the current codebase, elaborate on the dataset generation for the chosen API, and outline the project schedule that includes important dates for achieving the project milestones.

2 RELATED WORK

2.1 HTLM: Hyper-Text Pre-Training and Prompting of Language Models

In the paper [1], the authors present HTLM, a hypertext language model trained on a massive 23 TB web crawl dataset, specifically using simplified HTML data from the common crawl. They employ a BART-style denoising auto-encoder, allowing them to excel in zero-shot prompting by converting tasks into HTML format.They make use of HTML prompting for generation and classification tasks. They utilize HTML templates, either manually chosen or generated by the model through auto-prompting, to define the HTML structure of the task. The template is subsequently populated with the task input and mask tokens acting as placeholders for the output. The instantiated template serves as the prompt for the model. Given that BART models reconstruct the entire input, straightforward heuristics are employed to correspond to the text surrounding the masks and extract the ultimate output.Their experiments demonstrate that HTLM surpasses previous outcomes in zero-shot prompting for summarization by generating prompts that effectively capture the underlying meaning of each summarization dataset. The experiments show that finetuning the model on structured data improved the performance of the model drastically as compared to other models that were pretrained using natural text only.

2.2 PAL: Program-aided Language Models

Recently, Large language models (LLMs) have shown the ability to perform mathematical calculations when provided with limited number of samples due to the use of "chain-of-thought" technique. This technique allows the model to understand the problem statement by dividing the problem statement into a number of steps and then evaluating each step. Even though LLM's do perform pretty well in this step-wise approach, they are prone to making some logical mistakes in calculations. In an attempt to solve this problem, the authors of the paper [2] introduced Program-aided Language (PAL) models. These models read the natural language problem statements, then generates programs as the intermediate step used for reasoning and then runs these programs to a run-time engine like the Python interpreter. In this setup, the LLM's primary task is to decompose the problem into executable steps, while the computational aspect is handled by the runtime engine, in this case, the Python interpreter. This paper specifically uses the Python interpreter for the computation task. They make use of CODEX (code-davinci-002) as their base LLM. For their experimental analysis they make use of following benchmark datasets: GSM8K, ASDIV, MAWPS, SVAMP and BIG-Bench Hard. Experiments conducted by the authors also showed that the technique also worked when the base Language model was changed to a text language model like text-davinci-002 and text-davinci-003. The only limitation of this approach is that it guarantees the correctness of the results under the assumption that the programmatic steps themselves are accurate.

2.3 Toolformer: Language Models Can Teach Themselves to Use Tools

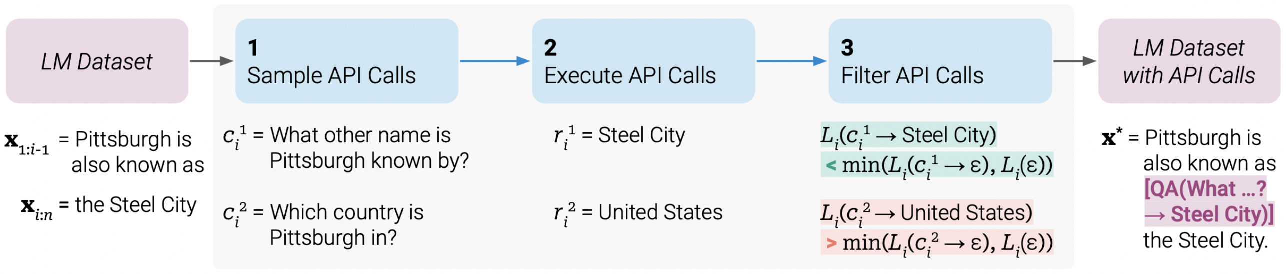
The paper [4] forms the base of our project. Language models though adept with various natural language tasks, usually do not perform well for calculations and do not have access to up-to-date information related to recent events. This paper attempts to solve this problem by providing the models the ability to access external tools like search engines, calculator and the calendar. Prior to the introduction of this paper, the existing systems required huge amounts of human annotated data. The authors in this paper introduce a new model - Toolformer that learns in a self-supervised way without requiring large amounts of human annotations and is able to determine which , when and how the external tool is to be used without losing generality of the LM. They aim at providing the model this ability to use the tools using the help of API calls. The only limitation is that it should be possible to represent the input and output of these API calls as text sequences. The training dataset is first augmented with API calls using the following steps- sampling API calls, filtering API calls and model finetuning, after which comes the inference, which is a regular decoding step until the response is identified. The paper mainly focuses on Question Answering, Calculator, Wikipedia search, Machine translation system and the Calendar as the external tools that would help address the limitations of the current Language models. They make use of a subset of the CCNet dataset for finetuning and GPT-J as their foundational model. The model is evaluated based on the SQuAD, Google-RE and T-REx subsets of the LAMA benchmark. Their experiments offered compelling evidence that Toolformer significantly enhances the zero-shot performance of a 6.7 billion parameter GPT-J model, enabling it to surpass the performance of a much larger GPT-3 model across various downstream tasks. Even though the model shows promising results, it still has a few limitations. One such limitation is that the the model is not able to use the external tools in a chain i.e use the output of the first tool as the input for the second tool. The second limitation is that the current implementation does not allow the tool to be used in an interactive way. Also, the models trained with Toolformer are sensitive to the exact wording of the input when deciding whether or not to call the tool APIs and are also sample efficient.

2.4 BLOOM: A 176B-Parameter Open-Access Multilingual Language Model

In the paper [3] the authors present the BLOOM model, which is a Transformer language model with a decoder-only architecture. BLOOM was trained on the ROOTS corpus, a dataset that encompasses hundreds of sources in a total of 59 languages, including 46 natural languages and 13 programming languages. BLOOM was developed by BigScience, a collaborative effort involving hundreds of researchers. It underwent training on the French government-funded Jean Zay supercomputer over a span of 3.5 months. The paper documents the entire process of creating BLOOM, starting from the construction of its training dataset ROOTS, all the way to its architecture and tokenizer design. Furthermore, the evaluation results of BLOOM and other large language models are discussed, revealing competitive performance that sees improvement through multitask finetuning. The authors anticipate that the release of this potent multilingual language model will unlock fresh possibilities and research avenues for large language models. Additionally, they hope that sharing their experiences will assist the machine learning research community in organizing new large-scale collaborative projects akin to BigScience. This collaborative framework not only enables outcomes that are unattainable for individual research groups but also encourages diverse individuals from various backgrounds to contribute their ideas and engage in the development of substantial advancements in the field.

2.5 TALM: Tool Augmented Language Models

In their work, the authors introduce Tool Augmented Language Models (TALM), a method that enhances language models by incorporating non-differentiable tools and utilizing an iterative "self-play" technique to improve performance, even with limited initial tool demonstrations. They make use of the pretrained T5 model for finetuning, inference and evaluation, and it is evaluated using knowledge-oriented Natural Questions (NQ) which is a diverse QA task and MathQA. Consistently, TALM demonstrates superior performance compared to a non-augmented language model on both a knowledge-oriented task (NQ) and a reasoning task



LM Dataset
1 Sample API Calls
2 Execute API Calls
3 Filter API Calls
LM Dataset with API Calls
$\mathbf{x}_{1:i-1}$ = Pittsburgh is also known as
$\mathbf{x}_{i:n}$ = the Steel City
c_i^1 = What other name is Pittsburgh known by?
c_i^2 = Which country is Pittsburgh in?
r_i^1 = Steel City
r_i^2 = United States
$L_i(c_i^1 \rightarrow$ Steel City) $< \min(L_i(c_i^1 \rightarrow \varepsilon), L_i(\varepsilon))$
$L_i(c_i^2 \rightarrow$ United States) $> \min(L_i(c_i^2 \rightarrow \varepsilon), L_i(\varepsilon))$
$\mathbf{x}^*$ = Pittsburgh is also known as [QA(What ...? → Steel City)] the Steel City.

Figure 2: Overview of the steps of the toolformer [4]

(MathQA).The limitation of this approach is that it is only useful where a language model is finetuned for downstream tasks.

3 IMPLEMENTATION

3.1 Architecture

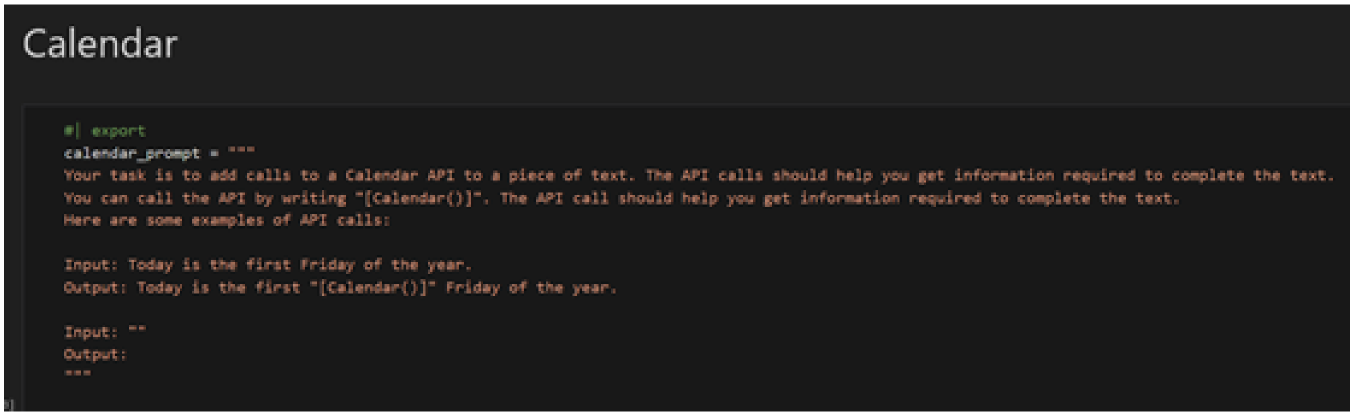
Our implementation is based on an existing work on the Toolformer [4]. In our case, to enable language models to scale its performance we use pre-trained causal language models on the HuggingFace platform [6]. We have incorporated a multilingual language model BLOOM [3] (bigscience/bloom-560m) . It is an autoregressive Large Language Model (LLM), that has been trained on massive volumes of text data using extensive computational resources to generate text continuations from given prompts. BLOOM is particularly used since it is an open-source dataset and has no usage limitation. We retained the hyperparameters as configured in the BLOOM model with *3B parameters, 70 layers and 112 additional heads, sequence length of 2048 tokens used.* We also use a pre-trained tokenizer from BLOOM that has a vocabulary size of 250k and which uses the byte-level Byte Pair Encoding algorithm (BPE). We are currently augmenting the dataset and inferencing with a batch size of 1.

3.2 Toolformer: Approach and Analysis

In order to implement the Toolformer [4], we need to transform the existing dataset C to C* by augmenting API calls. The steps to transform the dataset is shown in Figure 2. These API calls support different tools. The inputs and outputs to these API calls are represented as sequences of text. The existing implementation uses special tokens to indicate the start and end of each API call. The start token is indicated by the '[', the end token by ']' and '->' is the output token. To convert the dataset into C* we follow 3 steps as shown in Figure 2.

- **Sampling API calls:** For each API call we store the prompt P as shown in Figure 3. This encourages the LM to annotate the dataset with API calls. During sampling, we calculate the probability of the position of API calls for all positions $i \in \{1.., n\}$. From this, we sample up to k positions. We also store a sampling threshold value τ_s and keep the top k values that are greater than the sampling threshold i.e $I = \{i | p_i > \tau_s\}$.
- **Obtain API response:** From each of the k positions obtained in the sampling, we obtain upto m API calls for each position $i \in I$ given the sequence $[P(x), x_1, ..x_{i-1}, [APIcall], x_i, x_n]$ where dataset $C = \{x_1, .., x_n\}$ and n is the length of the dataset. We then execute all the API calls that are invoked using internal utilities. Currently we have implemented the Calculator, Wolfram API and the Calendar API that are python scripts. The response of these API queries are a single text response.
- **Filtering API response:** In order to filter API calls, we consider a sequence of weights $(w_i | i \in N)$. The weighted cross-entropy loss is calculated by: $L_i(z) = \sum_{j=1}^{n} w_{j-1} * log p_M(x_j | z_j, x_{i:j-1})$ for M when prefixed by z. We consider two instances of the loss:
  - Positive loss: This is the weighted loss taken over all tokens $x_i, ..x_n$ if the API call and the result are included in the prefix of M. It is represented by $L_i^+ : L(e(c_i, r_i))$.
  - Negative loss: This is calculated as the minimum of the weighted loss when (i) doing no API call

[0]https://huggingface.co/bigscience/bloom


Calendar

```
#| export
calendar_prompt = """
Your task is to add calls to a Calendar API to a piece of text. The API calls should help you get information required to complete the text.
You can call the API by writing "[Calendar()]". The API call should help you get information required to complete the text.
Here are some examples of API calls:

Input: Today is the first Friday of the year.
Output: Today is the first "[Calendar()]" Friday of the year.

Input: ""
Output:
"""
```

Figure 3: Example prompt for the Calendar API

and (ii) doing an API call but getting no response. This is represented by $L_i^- : min(L_i(\epsilon), L(e(c_i, \epsilon)))$ We consider a filtering threshold τ_f and filter out all API calls that have $L_i^- - L_i^+ < \tau_f$.

Model Finetuning and Inference: After sampling and filtering API calls, we plan to augment all the APIs and interleave them with current input sequences that will give us the new dataset C*. In the current implementation of the paper, the model is yet to be finetuned on the augmented prefix inputs. Finetuning on C* would further help the LM determine which tool to be used when and how. In order to inference the generated output, we will decode the response until we find the output token "->". The appropriate API call is made and decoding is continued.

3.3 Tools implemented

We have currently implemented 3 APIs and

- **Calculator:** We perform simple arithmetic functions and support basic 4 operations of '+', '-', '*' and '/'.
- **Calendar:** The second tool implemented is the Calendar that supports by appending only the current date to the prompt. It takes in no input.
- **WolframAPI:** We implement the WolframAPI to perform simple mathematical calculations given the worded problems as API input to perform mathematical operations.

3.4 Experimental Analysis

In Table 1 we have shown some of the examples run with the existing code base. The filtering threshold value, $\tau_f = 0.05$ and sampling threshold value, $\tau_s = 0.1$. The new implementation from our team includes the WolframAPI and the Calendar API in the file *nbs/prompt.ipynb* and we have also written a new test script to test the model and data generation called *tests/test_sample.py.*

4 FURTHER IMPLEMENTATION

Currently, through our initial research we were unable to find the original implementation of the Toolformer but found a few codes with their own limitations. We have decided to use one implementation as a reference to code further and make our Toolformer work for other APIs. Keeping the current code as a reference [5] we have extend the model to add APIs for Calendar and Wolframalpha. The code we have chosen does not generate the data using a GPT-J model and hence is sensitive to prompts. We divide our further implementation into three stages.

- **Data generation**
  We are currently able to generate data for Calendar and Wolframalpha API using example prompts. However, we observe the existing code limitation where the data generated is sensitive to the exact wording of their input when deciding whether or not to call an API. We plan to reduce this limitation so that we can use a reasonably large dataset to augment with our APIs. We believe this sensitivity is due to the limitation of the pretrained Bloom model that is used to generate the prompts.

- **Data handling**
  Once we are done with the data generation for multiple APIs which include original and new

Table 1: Example API calls for different tools indicating the $L_i^- - L_i^+$ used for filtering API calls

Examples	$L_i^- - L_i^+$	Useful
John has 20 apples and his friend gave him 7 more. John now has [Calculator(20 / 7)] 27 apples.	0.4047	No
From this, we have 3 * 30 minutes = [Calculator(3 * 30)] 90 minutes.	0.0682	Yes
Complex conjugate of 4 + 7i is [Wolframe("What is the complex conjugate of 4 + 7i")] 4 - 7i	0.0906	Yes

APIs we plan to combine them and fine tune our augmented dataset on a larger model like GPT-J. As mentioned in the original paper we will similarly finetune our data set which is not augmented (C) followed by finetuning the pretrained model on C*. The current code uses bloom as a pretrained model for data generation but the code for finetuning is not complete. We also face the limit of batch size of one that the reference code is limited to and will extend it to accept a large batch size. To do this we will use GPU's where we will evenly distributed the dataset on each GPU core to finetune.

- **Benchmarking for existing APIs and newly added APIs**
  Given the project timeline and feasibility We further plan to use Math specific data to benchmark the Wolframalpha API and Calculator API and also compare the results of finetuning with and without our newly added APIs to gauge our model. We however see the limitation where Wolframapha API has a cap of 20,000 free API calls after which a premium subscription is required.

5 PROPOSED TIMELINE

The experiments that need to be performed have been explained in Section 4. The entirety of our project is divided into mainly three parts- Data Generation and Data Handling, Benchmarking for existing and new APIs, Evaluation and Validation. The following subsection describes about the estimated timeline of the project along with the division of the tasks among the group members.

5.1 Division of Work

Time	Task
Before Homework 3	Literature survey, understanding the existing codebase and try executing the code and getting results. Creating dataset for additional APIs. Prepare homework 3 report
Milestone 1 (till 11/17)	Data Generation, Data Handling [Tanisha , Sumedha]
Milestone 2 (till 11/30)	Fine Tuning, Training and Benchmarking for the existing and new APIs [Sumedha, Madhushree]
Milestone 3 (till end of semester)	Evaluating on the dataset and validation [Tanisha, Madhushree] Wrap up the project and write the final report [Tanisha, Sumedha and Madhushree]

Table 2: Tentative Plan and Division of Work

REFERENCES

[1] Armen Aghajanyan, Dmytro Okhonko, Mike Lewis, Mandar Joshi, Hu Xu, Gargi Ghosh, and Luke Zettlemoyer. 2021. HTLM: Hyper-Text Pre-Training and Prompting of Language Models. (2021). arXiv:cs.CL/2107.06955

[2] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. PAL: Program-aided Language Models. *ArXiv* abs/2211.10435 (2022).

[3] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100* (2022).

[4] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda,

and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools.

[5] Phil Wang. 2023. toolformer-pytorch. https://github.com/lucidrains/toolformer-pytorch/tree/main. (2023).

[6] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6