Toolformer: Language Models Can Teach Themselves to Use Tools

Timo Schick   Jane Dwivedi-Yu   Roberto Dessì[†]   Roberta Raileanu

Maria Lomeli   Luke Zettlemoyer   Nicola Cancedda   Thomas Scialom

Meta AI Research [†]Universitat Pompeu Fabra

Abstract

Language models (LMs) exhibit remarkable abilities to solve new tasks from just a few examples or textual instructions, especially at scale. They also, paradoxically, struggle with basic functionality, such as arithmetic or factual lookup, where much simpler and smaller models excel. In this paper, we show that LMs can teach themselves to *use external tools* via simple APIs and achieve the best of both worlds. We introduce *Toolformer*, a model trained to decide which APIs to call, when to call them, what arguments to pass, and how to best incorporate the results into future token prediction. This is done in a self-supervised way, requiring nothing more than a handful of demonstrations for each API. We incorporate a range of tools, including a calculator, a Q&A system, a search engine, a translation system, and a calendar. Toolformer achieves substantially improved zero-shot performance across a variety of downstream tasks, often competitive with much larger models, without sacrificing its core language modeling abilities.

1 Introduction

Large language models achieve impressive zero- and few-shot results on a variety of natural language processing tasks (Brown et al., 2020; Chowdhery et al., 2022, i.a.) and show several emergent capabilities (Wei et al., 2022). However, all of these models have several inherent limitations that can at best be partially addressed by further scaling. These limitations include an inability to access up-to-date information on recent events (Komeili et al., 2022) and the related tendency to hallucinate facts (Maynez et al., 2020; Ji et al., 2022), difficulties in understanding low-resource languages (Lin et al., 2021), a lack of mathematical skills to perform precise calculations (Patel et al., 2021) and an unawareness of the progression of time (Dhingra et al., 2022).

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Figure 1: Exemplary predictions of Toolformer. The model autonomously decides to call different APIs (from top to bottom: a question answering system, a calculator, a machine translation system, and a Wikipedia search engine) to obtain information that is useful for completing a piece of text.

A simple way to overcome these limitations of today's language models is to give them the ability to *use external tools* such as search engines, calculators, or calendars. However, existing approaches either rely on large amounts of human annotations (Komeili et al., 2022; Thoppilan et al., 2022) or limit tool use to task-specific settings only (e.g., Gao et al., 2022; Parisi et al., 2022), hindering a more widespread adoption of tool use in LMs. Therefore, we propose *Toolformer*, a model that learns to use tools in a novel way, which fulfills the following desiderata:

- The use of tools should be learned in a self-supervised way without requiring large amounts of *human annotations*. This is impor-
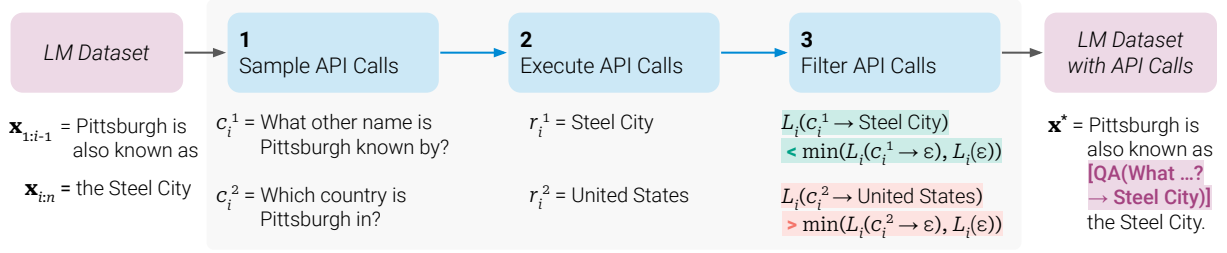
| | **1** | **2** | **3** | |
|LM Dataset | Sample API Calls | Execute API Calls | Filter API Calls | LM Dataset with API Calls |

$\mathbf{x}_{1:i-1}$ = 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

$L_i(c_i^1 \to \text{Steel City})$
$< \min(L_i(c_i^1 \to \varepsilon), L_i(\varepsilon))$

$L_i(c_i^2 \to \text{United States})$
$> \min(L_i(c_i^2 \to \varepsilon), L_i(\varepsilon))$

$\mathbf{x}^\star$ = Pittsburgh is also known as [QA(What ...? → Steel City)] the Steel City.

Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text $\mathbf{x}$, we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \ldots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the next tokens. All remaining API calls are interleaved with the original text, resulting in a new text $\mathbf{x}^*$.

tant not only because of the costs associated with such annotations, but also because what humans find useful may be different from what a model finds useful.

- The LM should not lose any of its *generality* and should be able to decide for itself *when* and *how* to use which tool. In contrast to existing approaches, this enables a much more comprehensive use of tools that is not tied to specific tasks.

Our approach for achieving these goals is based on the recent idea of using large LMs with *in-context learning* (Brown et al., 2020) to generate entire datasets from scratch (Schick and Schütze, 2021b; Honovich et al., 2022; Wang et al., 2022): Given just a handful of human-written examples of how an API can be used, we let a LM annotate a huge language modeling dataset with potential API calls. We then use a self-supervised loss to determine which of these API calls actually help the model in predicting future tokens. Finally, we finetune the LM itself on the API calls that it considers useful. As illustrated in Figure 1, through this simple approach, LMs can learn to control a variety of tools, and to choose for themselves which tool to use when and how.

As our approach is agnostic of the dataset being used, we can apply it to the exact same dataset that was used to pretrain a model in the first place. This ensures that the model does not lose any of its generality and language modeling abilities. We conduct experiments on a variety of different downstream tasks, demonstrating that after learning to use tools, Toolformer, which is based on a pretrained GPT-J model (Wang and Komatsuzaki, 2021) with 6.7B parameters, achieves much stronger zero-shot results, clearly outperforming a much larger GPT-3 model (Brown et al., 2020) and

several other baselines on various tasks.

2 Approach

Our aim is to equip a language model M with the ability to use different tools by means of API calls. We require that inputs and outputs for each API can be represented as text sequences. This allows seamless insertion of API calls into any given text, using special tokens to mark the start and end of each such call.

We represent each API call as a tuple $c = (a_c, i_c)$ where a_c is the name of the API and i_c is the corresponding input. Given an API call c with a corresponding result r, we denote the linearized sequences of the API call not including and including its result, respectively, as:

$$\mathbf{e}(c) = \texttt{<API>}\, a_c\,(i_c)\,\texttt{</API>}$$
$$\mathbf{e}(c,r) = \texttt{<API>}\, a_c\,(i_c)\,\to r\,\texttt{</API>}$$

where "<API>", "</API>" and "→" are special tokens.[1] Some examples of linearized API calls inserted into text sequences are shown in Figure 1.

Given a dataset $\mathcal{C} = \{\mathbf{x}^1, \ldots, \mathbf{x}^{|\mathcal{C}|}\}$ of plain texts, we first convert this dataset into a dataset $\mathcal{C}^*$ augmented with API calls. This is done in three steps, illustrated in Figure 2: First, we exploit the in-context learning ability of M to sample a large number of potential API calls. We then execute these API calls and finally check whether the obtained responses are helpful for predicting future tokens; this is used as a filtering criterion. After filtering, we merge API calls for different tools, resulting in the augmented dataset $\mathcal{C}^*$, and finetune

---

[1]In practice, we use the token sequences " [", "]" and "->" to represent "<API>", "</API>" and "→", respectively. This enables our approach to work without modifying the existing LM's vocabulary. For reasons of readability, we still refer to them as "<API>", "</API>" and "→" throughout this section.

Figure 3: An exemplary prompt $P(\mathbf{x})$ used to generate API calls for the question answering tool.

M itself on this dataset. Each of these steps is described in more detail below.

Sampling API Calls For each API, we write a prompt $P(\mathbf{x})$ that encourages the LM to annotate an example $\mathbf{x} = x_1, \ldots, x_n$ with API calls. An example of such a prompt for a question answering tool is shown in Figure 3; all prompts used are shown in Appendix A.2. Let $p_M(z_{n+1} \mid z_1, \ldots, z_n)$ be the probability that M assigns to token z_{n+1} as a continuation for the sequence $z_1, \ldots, z_n$. We first sample up to k candidate *positions* for doing API calls by computing, for each $i \in \{1, \ldots, n\}$, the probability

$$p_i = p_M(\texttt{<API>} \mid P(\mathbf{x}), x_{1:i-1})$$

that M assigns to starting an API call at position i. Given a sampling threshold τ_s, we keep all positions $I = \{i \mid p_i > \tau_s\}$; if there are more than k such positions, we only keep the top k.

For each position $i \in I$, we then obtain up to m API calls $c_i^1, \ldots, c_i^m$ by sampling from M given the sequence $[P(\mathbf{x}), x_1, \ldots, x_{i-1}, \texttt{<API>}]$ as a prefix and $\texttt{</API>}$ as an end-of-sequence token.[2]

---

[2] We discard all examples where M does not generate the $\texttt{</API>}$ token.

Executing API Calls As a next step, we execute all API calls generated by M to obtain the corresponding results. How this is done depends entirely on the API itself – for example, it can involve calling another neural network, executing a Python script or using a retrieval system to perform search over a large corpus. The response for each API call c_i needs to be a single text sequence r_i.

Filtering API Calls Let i be the position of the API call c_i in the sequence $\mathbf{x} = x_1, \ldots, x_n$, and let r_i be the response from the API. Further, given a sequence $(w_i \mid i \in \mathbb{N})$ of *weights*, let

$$L_i(\mathbf{z}) = -\sum_{j=i}^{n} w_{j-i} \cdot \log p_M(x_j \mid \mathbf{z}, x_{1:j-1})$$

be the weighted cross entropy loss for M over the tokens $x_i, \ldots, x_n$ if the model is prefixed with $\mathbf{z}$. We compare two different instantiations of this loss:

$$L_i^+ = L_i(\mathrm{e}(c_i, r_i))$$
$$L_i^- = \min\left(L_i(\varepsilon), L_i(\mathrm{e}(c_i, \varepsilon))\right)$$

where ε denotes an empty sequence. The former is the weighted loss over all tokens $x_i, \ldots, x_n$ if the API call and its result are given to M as a prefix;[3] the latter is the minimum of the losses obtained from (i) doing no API call at all and (ii) doing an API call, but not providing the response. Intuitively, an API call is helpful to M if providing it with both the input *and* the output of this call makes it easier for the model to predict future tokens, compared to not receiving the API call at all, or receiving only its input. Given a filtering threshold τ_f, we thus only keep API calls for which

$$L_i^- - L_i^+ \geq \tau_f$$

holds, i.e., adding the API call and its result *reduces* the loss by at least τ_f, compared to not doing any API call or obtaining no result from it.

Model Finetuning After sampling and filtering calls for all APIs, we finally merge the remaining API calls and interleave them with the original inputs. That is, for an input text $\mathbf{x} = x_1, \ldots, x_n$ with a corresponding API call and result (c_i, r_i) at position i, we construct the new sequence $\mathbf{x}^* =$

---

[3] We provide $\mathrm{e}(c_i, r_i)$ as a prefix instead of inserting it at position i because M is not yet finetuned on any examples containing API calls, so inserting it in the middle of $\mathbf{x}$ would interrupt the flow and not align with patterns in the pretraining corpus, thus hurting perplexity.

$x_{1:i-1}, \mathrm{e}(c_i, r_i), x_{i:n}$; we proceed analogously for texts with multiple API calls. Doing this for all $\mathbf{x} \in \mathcal{C}$ results in the new dataset $\mathcal{C}^*$ augmented with API calls. We use this new dataset to finetune M, using a standard language modeling objective. Crucially, apart from inserted API calls the augmented dataset $\mathcal{C}^*$ contains the exact same texts as $\mathcal{C}$, the original dataset. As a consequence, finetuning M on $\mathcal{C}^*$ exposes it to the same content as finetuning on $\mathcal{C}$. Moreover, as API calls are inserted in exactly those positions and with exactly those inputs that help M predict future tokens, finetuning on $\mathcal{C}^*$ enables the language model to decide when and how to use which tool, based purely on its own feedback.

Inference When generating text with M after finetuning with our approach, we perform regular decoding until M produces the "→" token, indicating that it next expects the response for an API call. At this point, we interrupt the decoding process, call the appropriate API to get a response, and continue the decoding process after inserting both the response and the `</API>` token.

3 Tools

We explore a variety of tools to address different shortcomings of regular LMs. The only constraints we impose on these tools is that (i) both their inputs and outputs can be represented as text sequences, and (ii) we can obtain a few demonstrations of their intended use. Concretely, we explore the following five tools: a question answering system, a Wikipedia search engine, a calculator, a calendar, and a machine translation system. Some examples of potential calls and return strings for the APIs associated with each of these tools are shown in Table 1. We briefly discuss all tools below; further details can be found in Appendix A.

Question Answering Our first tool is a question answering system based on another LM that can answer simple factoid questions. Specifically, we use *Atlas* (Izacard et al., 2022), a retrieval-augmented LM finetuned on Natural Questions (Kwiatkowski et al., 2019).

Calculator As a second tool, we use a calculator that can perform simple numeric calculations; we only support the four basic arithmetic operations. Results are always rounded to two decimal places.

Wikipedia Search Our third tool is a search engine that, given a search term, returns short text snippets from Wikipedia. Compared to our question answering tool, this search enables a model to get more comprehensive information on a subject, but requires it to extract the relevant parts by itself. As our search engine, we use a BM25 retriever (Robertson et al., 1995; Baeza-Yates et al., 1999) that indexes the Wikipedia dump from KILT (Petroni et al., 2021).

Machine Translation System Our fourth tool is a machine translation system based on a LM that can translate a phrase from any language into English. More concretely, we use the 600M parameter NLLB (Costa-jussà et al., 2022) as our multilingual machine translation model that works for 200 languages (including low-resource ones). The source language is automatically detected using the *fastText* classifier (Joulin et al., 2016), while the target language is always set to English.

Calendar Our final tool is a calendar API that, when queried, returns the current date without taking any input. This provides temporal context for predictions that require some awareness of time.

4 Experiments

We investigate whether our approach enables a model to use tools without any further supervision and to decide for itself when and how to call which of the available tools. To test this, we select a variety of downstream tasks where we assume at least one of the considered tools to be useful, and evaluate performance in zero-shot settings (Section 4.2). Beyond that, we also ensure that our approach does not hurt the model's core language modeling abilities; we verify this by looking at perplexity on two language modeling datasets (Section 4.3). Finally, we investigate how the ability to learn using tools is affected by model size (Section 4.4).

4.1 Experimental Setup

Dataset Generation Throughout all of our experiments, we use a subset of CCNet (Wenzek et al., 2020) as our language modeling dataset $\mathcal{C}$ and GPT-J (Wang and Komatsuzaki, 2021) as our language model M. To reduce the computational cost of annotating $\mathcal{C}$ with API calls, we define heuristics for some APIs to get a subset of $\mathcal{C}$ for which API calls are more likely to be helpful than for an average text. For example, we only consider texts for the calculator tool if they contain at least three numbers. Details of the heuristics used are given in

API Name	Example Input	Example Output
Question Answering	Where was the Knights of Columbus founded?	New Haven, Connecticut
Wikipedia Search	Fishing Reel Types	Spin fishing > Spin fishing is distinguished between fly fishing and bait cast fishing by the type of rod and reel used. There are two types of reels used when spin fishing, the open faced reel and the closed faced reel.
Calculator	27 + 4 * 2	35
Calendar	ε	Today is Monday, January 30, 2023.
Machine Translation	sûreté nucléaire	nuclear safety

Table 1: Examples of inputs and outputs for all APIs used.

	Number of Examples		
API	$\tau_f = 0.5$	$\tau_f = 1.0$	$\tau_f = 2.0$
Question Answering	51,987	18,526	5,135
Wikipedia Search	207,241	60,974	13,944
Calculator	3,680	994	138
Calendar	61,811	20,587	3,007
Machine Translation	3,156	1,034	229

Table 2: Number of examples with API calls in $\mathcal{C}^*$ for different values of our filtering threshold τ_f.

Appendix A. For obtaining $\mathcal{C}^*$ from $\mathcal{C}$, we perform all steps described in Section 2 and additionally filter out all examples for which all API calls were eliminated in the filtering step.[4] For the weighting function, we use

$$w_t = \frac{\tilde{w}_t}{\sum_{s \in \mathbb{N}} \tilde{w}_s} \text{ with } \tilde{w}_t = \max(0, 1 - 0.2 \cdot t)$$

to make sure that API calls happen close to where the information provided by the API is actually helpful for the model. The thresholds τ_s and τ_f are chosen individually for each tool to ensure a sufficiently larger number of examples; see Appendix A for details. Table 2 shows relevant statistics of our final dataset augmented with API calls.

Model Finetuning We finetune M on $\mathcal{C}^*$ using a batch size of 128 and a learning rate of $1 \cdot 10^{-5}$ with linear warmup for the first 10% of training. Details of our finetuning procedure are given in Appendix B.

Baseline Models Throughout the remainder of this section, we mainly compare the following models:

- **GPT-J**: A regular GPT-J model without any finetuning.

- **GPT-J + CC**: GPT-J finetuned on $\mathcal{C}$, our subset of CCNet *without* any API calls.

- **Toolformer**: GPT-J finetuned on $\mathcal{C}^*$, our subset of CCNet augmented with API calls.

- **Toolformer (disabled)**: The same model as Toolformer, but API calls are disabled during decoding.[5]

For most tasks, we additionally compare to OPT (66B) (Zhang et al., 2022) and GPT-3[6] (175B) (Brown et al., 2020), two models that are about 10 and 25 times larger than our other baseline models, respectively.

4.2   Downstream Tasks

We evaluate all models on a variety of downstream tasks. In all cases, we consider a prompted zero-shot setup – i.e., models are instructed to solve each task in natural language, but we do not provide any in-context examples. This is in contrast to prior work on tool use (e.g., Gao et al., 2022; Parisi et al., 2022), where models are provided with dataset-specific examples of how a tool can be used to solve a concrete task. We choose the more challenging zero-shot setup as we are interested in seeing whether Toolformer works in precisely those cases where a user does not specify in advance which tools should be used in which way for solving a specific problem.

We use standard greedy decoding, but with one modification for Toolformer: We let the model start an API call not just when <API> is the most likely

---

token, but whenever it is one of the k most likely tokens. For $k = 1$, this corresponds to regular greedy decoding; we instead use $k = 10$ to increase the disposition of our model to make use of the APIs that it has access to. At the same time, we only at most one API call per input to make sure the model does not get stuck in a loop where it constantly calls APIs without producing any actual output. The effect of these modifications is explored in Section 5.

4.2.1 LAMA

We evaluate our models on the SQuAD, Google-RE and T-REx subsets of the LAMA benchmark (Petroni et al., 2019). For each of these subsets, the task is to complete a short statement with a missing fact (e.g., a date or a place). As LAMA was originally designed to evaluate *masked* language models (e.g., Devlin et al., 2019), we filter out examples where the mask token is not the final token, so that the remaining examples can be processed in a left-to-right fashion. To account for different tokenizations and added complexity from not informing the model that a single word is required, we use a slightly more lenient evaluation criterion than exact match and simply check whether the correct word is within the first five words predicted by the model. As LAMA is based on statements obtained directly from Wikipedia, we prevent Toolformer from using the Wikipedia Search API to avoid giving it an unfair advantage.

Results for all models can be seen in Table 3. All GPT-J models without tool use achieve similar performance. Crucially, Toolformer clearly outperforms these baseline models, improving upon the best baseline by 11.7, 5.2 and 18.6 points, respectively. It also clearly outperforms OPT (66B) and GPT-3 (175B), despite both models being much larger. This is achieved because the model independently decides to ask the question answering tool for the required information in almost all cases (98.1%); for only very few examples, it uses a different tool (0.7%) or no tool at all (1.2%).

4.2.2 Math Datasets

We test mathematical reasoning abilities on ASDiv (Miao et al., 2020), SVAMP (Patel et al., 2021) and the MAWPS benchmark (Koncel-Kedziorski et al., 2016). We again account for the fact that we test all models in a zero-shot setup by using a more lenient evaluation criterion: As the required output is always a number, we simply check for the first

Model	SQuAD	Google-RE	T-REx
GPT-J	17.8	4.9	31.9
GPT-J + CC	19.2	5.6	33.2
Toolformer (disabled)	22.1	6.3	34.9
Toolformer	**33.8**	**11.5**	**53.5**
OPT (66B)	21.6	2.9	30.1
GPT-3 (175B)	26.8	7.0	39.8

Table 3: Results on subsets of LAMA. Toolformer uses the question answering tool for most examples, clearly outperforming all baselines of the same size and achieving results competitive with GPT-3 (175B).

Model	ASDiv	SVAMP	MAWPS
GPT-J	7.5	5.2	9.9
GPT-J + CC	9.6	5.0	9.3
Toolformer (disabled)	14.8	6.3	15.0
Toolformer	**40.4**	**29.4**	**44.0**
OPT (66B)	6.0	4.9	7.9
GPT-3 (175B)	14.0	10.0	19.8

Table 4: Results for various benchmarks requiring mathematical reasoning. Toolformer makes use of the calculator tool for most examples, clearly outperforming even OPT (66B) and GPT-3 (175B).

number predicted by the model.[7]

Table 4 shows results for all benchmarks. While GPT-J and GPT-J + CC perform about the same, Toolformer achieves stronger results even when API calls are disabled. We surmise that this is because the model is finetuned on many examples of API calls and their results, improving its own mathematical capabilities. Nonetheless, allowing the model to make API calls more than doubles performance for all tasks, and also clearly outperforms the much larger OPT and GPT-3 models. This is because across all benchmarks, for 97.9% of all examples the model decides to ask the calculator tool for help.

4.2.3 Question Answering

We look at Web Questions (Berant et al., 2013), Natural Questions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017), the three question answering datasets considered by Brown et al. (2020). For evaluation, we check whether the first 20 words predicted by a model contain the correct answer instead of requiring an exact match. For Toolformer, we disable the question answering tool as

---

[7]An exception to this is if the model's prediction contains an equation (e.g., "The correct answer is 5+3=8"), in which case we consider the first number after the "=" sign to be its prediction.

Model	WebQS	NQ	TriviaQA
GPT-J	18.5	12.8	43.9
GPT-J + CC	18.4	12.2	45.6
Toolformer (disabled)	18.9	12.6	46.7
Toolformer	**26.3**	**17.7**	**48.8**
OPT (66B)	18.6	11.4	45.7
GPT-3 (175B)	<u>29.0</u>	<u>22.6</u>	<u>65.9</u>

Table 5: Results for various question answering dataset. Using the Wikipedia search tool for most examples, Toolformer clearly outperforms baselines of the same size, but falls short of GPT-3 (175B).

Model	Es	De	Hi	Vi	Zh	Ar
GPT-J	15.2	**16.5**	1.3	8.2	**18.2**	**8.2**
GPT-J + CC	15.7	14.9	0.5	8.3	13.7	4.6
Toolformer (disabled)	19.8	11.9	1.2	10.1	15.0	3.1
Toolformer	**20.6**	13.5	**1.4**	**10.6**	16.8	3.7
OPT (66B)	0.3	0.1	1.1	0.2	0.7	0.1
GPT-3 (175B)	3.4	1.1	0.1	1.7	17.7	0.1
GPT-J (All En)	24.3	27.0	23.9	23.3	23.1	23.6
GPT-3 (All En)	24.7	27.2	26.1	24.9	23.6	24.0

Table 6: Results on MLQA for Spanish (Es), German (De), Hindi (Hi), Vietnamese (Vi), Chinese (Zh) and Arabic (Ar). While using the machine translation tool to translate questions is helpful across all languages, further pretraining on CCNet deteriorates performance; consequently, Toolformer does not consistently outperform GPT-J. The final two rows correspond to models that are given contexts and questions in English.

this would make solving the tasks trivial, especially given that the underlying QA system was finetuned on Natural Questions.

Results are shown in Table 5. Once again, Toolformer clearly outperforms all other models based on GPT-J, this time mostly relying on the Wikipedia search API (99.3%) to find relevant information. However, Toolformer still lags behind the much larger GPT-3 (175B) model. This is likely due to both the simplicity of our search engine (in many cases, it returns results that are clearly not a good match for a given query) and the inability of Toolformer to *interact* with it, e.g., by reformulating its query if results are not helpful or by browsing through multiple of the top results. We believe that adding this functionality is an exciting direction for future work.

4.2.4 Multilingual Question Answering

We evaluate Toolformer and all baseline models on MLQA (Lewis et al., 2019), a multilingual question-answering benchmark. A context paragraph for each question is provided in English, while the question can be in Arabic, German, Spanish, Hindi, Vietnamese, or Simplified Chinese. In order to solve the task, the model needs to be able to understand both the paragraph and the question, so it may benefit from translating the question into English. Our evaluation metric is the percentage of times the model's generation, capped at 10 words, contains the correct answer.

Results are shown in Table 6. Using API calls consistently improves Toolformer's performance for all languages, suggesting that it has learned to make use of the machine translation tool. Depending on the language, this tool is used for 63.8% to 94.9% of all examples; the only exception to this is Hindi, for which the machine translation tool is used in only 7.3% of cases. However, Tool-former does not consistently outperform vanilla GPT-J. This is mainly because for some languages, finetuning on CCNet deteriorates performance; this might be due to a distribution shift compared to GPT-J's original pretraining data.

OPT and GPT-3 perform surprisingly weak across all languages, mostly because they fail to provide an answer in English despite being instructed to do so. A potential reason for GPT-J not suffering from this problem is that it was trained on more multilingual data than both OPT and GPT-3, including the EuroParl corpus (Koehn, 2005; Gao et al., 2020). As an upper bound, we also evaluate GPT-J and GPT-3 on a variant of MLQA where both the context and the question are provided in English. In this setup, GPT-3 performs better than all other models, supporting our hypothesis that its subpar performance on MLQA is due to the multilingual aspect of the task.

4.2.5 Temporal Datasets

To investigate the calendar API's utility, we evaluate all models on TEMPLAMA (Dhingra et al., 2022) and a new dataset that we call DATESET. TEMPLAMA is a dataset built from Wikidata that contains cloze queries about facts that change with time (e.g., "Cristiano Ronaldo plays for ___") as well as the correct answer for the years between 2010 and 2020. DATESET, described in Appendix D, is also generated through a series of templates, but populated using a combination of random dates/durations (e.g., "What day of the week was it 30 days ago?"). Critically, knowing the current date is required to answer these questions.

Model	TEMPLAMA	DATESET
GPT-J	13.7	3.9
GPT-J + CC	12.9	2.9
Toolformer (disabled)	12.7	5.9
Toolformer	**16.3**	**27.3**
OPT (66B)	14.5	1.3
GPT-3 (175B)	15.5	0.8

Table 7: Results for the temporal datasets. Toolformer outperforms all baselines, but does not make use of the calendar tool for TEMPLAMA.

For both tasks, we use the same evaluation as for the original LAMA dataset.

Results shown in Table 7 illustrate that Toolformer outperforms all baselines for both TEMPLAMA and DATESET. However, closer inspection shows that improvements on TEMPLAMA can not be attributed to the calendar tool, which is only used for 0.2% of all examples, but mostly to the Wikipedia search and question answering tools, which Toolformer calls the most. This makes sense given that named entities in TEMPLAMA are often so specific and rare that even knowing the exact date alone would be of little help. The best course of action for this dataset – first querying the calendar API to get the current date, and then querying the question answering system with this date – is not only prohibited by our restriction of using at most one API call per example, but also hard to learn for Toolformer given that all API calls in its training data are sampled independently.

For DATESET, on the other hand, the considerable improvement of Toolformer compared to other models can be fully accredited to the calendar tool, which it makes use of for 54.8% of all examples.

4.3 Language Modeling

In addition to verifying improved performance on various downstream tasks, we also want to ensure that language modeling performance of Toolformer does not degrade through our finetuning with API calls. To this end, we evaluate our models on two language modeling datasets: WikiText (Merity et al., 2017) and a subset of 10,000 randomly selected documents from CCNet (Wenzek et al., 2020) that were not used during training. Perplexities of various models are shown in Table 8. As one would expect, finetuning on CCNet leads to slightly improved performance on a different CCNet subset, but it slightly deteriorates performance on WikiText, presumably because the original pre-

Model	WikiText	CCNet
GPT-J	**9.9**	10.6
GPT-J + CC	10.3	**10.5**
Toolformer (disabled)	10.3	**10.5**

Table 8: Perplexities of different models on WikiText and our validation subset of CCNet. Adding API calls comes without a cost in terms of perplexity for language modeling without any API calls.

training data for GPT-J is more similar to WikiText than our randomly selected subset of CCNet. Most importantly, however, training on $\mathcal{C}^*$ (our dataset annotated with API calls) does not lead to an increase in perplexity compared to training on $\mathcal{C}$ when API calls are disabled at inference time.[8]

4.4 Scaling Laws

We investigate how the ability to ask external tools for help affects performance as we vary the size of our LM. To this end, we apply our approach not just to GPT-J, but also to four smaller models from the GPT-2 family (Radford et al., 2019), with 124M, 355M, 775M and 1.6B parameters, respectively. We do so using only a subset of three tools: the question answering system, the calculator, and the Wikipedia search engine. Apart from this, we follow the experimental setup described in Section 4.1.

Figure 4 shows that the ability to leverage the provided tools only emerges at around 775M parameters: smaller models achieve similar performance both with and without tools. An exception to this is the Wikipedia search engine used mostly for QA benchmarks; we hypothesize that this is because the API is comparably easy to use. While models become better at solving tasks *without* API calls as they grow in size, their ability to make good use of the provided API improves at the same time. As a consequence, there remains a large gap between predictions with and without API calls even for our biggest model.

5 Analysis

Decoding Strategy We investigate the effect of our modified decoding strategy introduced in Section 4.2, where instead of always generating the

---

[8]We do not evaluate the perplexity of Toolformer with API calls enabled as computing the probability $p_M(x_t \mid x_1, \ldots, x_{t-1})$ of token x_t given $x_1, \ldots, x_{t-1}$ would require marginalizing over all potential API calls that the model could make at position t, which is intractable.
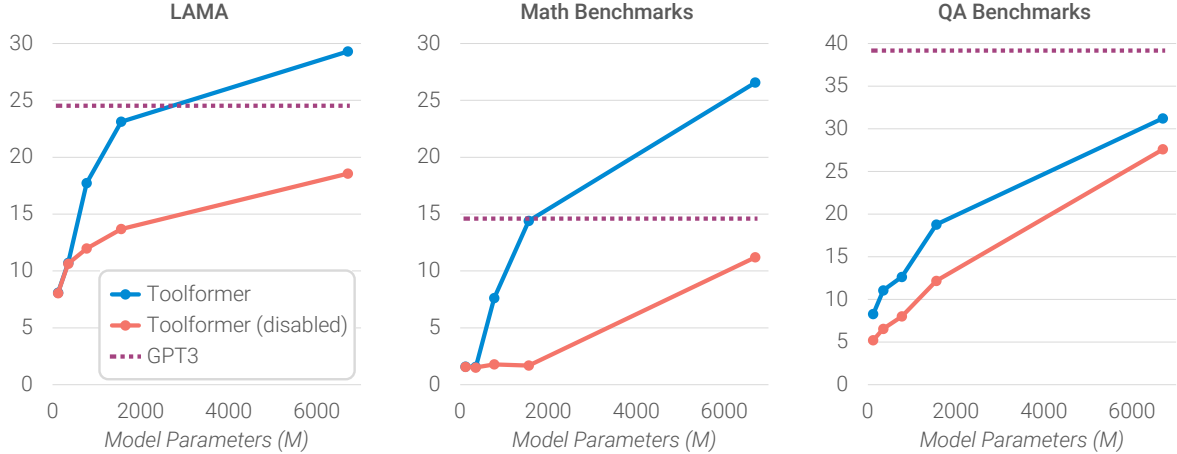
Figure 4: Average performance on LAMA, our math benchmarks and our QA benchmarks for GPT-2 models of different sizes and GPT-J finetuned with our approach, both with and without API calls. While API calls are not helpful to the smallest models, larger models learn how to make good use of them. Even for bigger models, the gap between model predictions with and without API calls remains high.

most likely token, we generate the `<API>` token if it is one of the k most likely tokens. Table 9 shows performance on the T-REx subset of LAMA and on WebQS for different values of k. As expected, increasing k leads to the model doing API calls for more examples – from 40.3% and 8.5% with $k = 1$ (i.e., regular greedy decoding) to 98.1% and 100% for $k = 10$. While for T-REx, there is already a clear improvement in performance with greedy decoding, on WebQS our model only starts to make a substantial number of API calls as we slightly increase k. Interestingly, for $k = 1$ the model is calibrated to some extent: It decides to call APIs for examples that it would perform particularly badly on without making API calls. This can be seen from the fact that performance on examples where it decides *not* to make an API call (44.3 and 19.9) is higher than average performance if no API calls are made at all (34.9 and 18.9). However, this calibration is lost for higher values of k.

Data Quality We qualitatively analyze some API calls generated with our approach for different APIs. Table 10 shows some examples of texts from CCNet augmented with API calls, as well as the corresponding score $L_i^- - L_i^+$ that is used as a filtering criterion, and whether the API calls made by the model are intuitively useful in the given context. As can be seen, high values of $L_i^- - L_i^+$ typically correspond to useful API calls, whereas low values correspond to API calls that do not provide any information that is useful for predicting future tokens. There are some exceptions, e.g., an API call for

	T-REx				**WebQS**			
k	**All**	**AC**	**NC**	**%**	**All**	**AC**	**NC**	**%**
0	34.9	–	34.9	0.0	18.9	–	18.9	0.0
1	47.8	53.0	44.3	40.3	19.3	17.1	19.9	8.5
3	52.9	58.0	29.0	82.8	**26.3**	26.5	6.6	99.3
10	**53.5**	54.0	22.5	98.1	**26.3**	26.4	–	100.0

Table 9: Toolformer results on the T-REx subset of LAMA and on WebQS for different values of k used during decoding. Numbers shown are overall performance (All), performance on the subset where the model decides to make an API call (AC) and all remaining examples (NC), as well as the percentage of examples for which the model decides to call an API (%).

"Fast train success" in the fourth example that does not give any relevant information but still reduces perplexity. However, some amount of noise in the API calls that are not filtered can actually be useful as it forces the model finetuned on $\mathcal{C}^*$ to not always blindly follow the results of each call it makes.

6 Related Work

Language Model Pretraining There are various approaches that augment language models with some form of additional textual information during pretraining, including various forms of metadata (Keskar et al., 2019), HTML tags (Aghajanyan et al., 2021), Wikipedia markup (Schick et al., 2022), or related texts obtained from an information retrieval system (Guu et al., 2020; Borgeaud et al., 2021; Izacard et al., 2022). For all of these

Example	$L_i^- - L_i^+$	Useful
The Flodden Window (a war memorial dedicated to The Middleton Archers), in the Grade I-listed Church of St Leonard in Middleton is said to be the oldest war memorial in the United Kingdom. `<API>` **WikiSearch(War memorial Flodden) → Battle of Flodden > Commemoration > The stained-glass Flodden Window in Middleton Parish Church [. . .] was constructed by Sir Richard Assheton in memory of the Battle of Flodden and the archers from Middleton who fought in it.** `</API>` Sir Richard Assheton of Middleton (who built St Leonard) was granted knighthood [. . .]	5.49	✓
Note: The WL will be open on Friday, `<API>` **Calendar() → Today is Thursday, March 9, 2017.** `</API>` March 10, and Sunday, March 19 for regular hours.	2.11	✓
The Nile has an approximate length of `<API>` **QA(What is the approximate length of the Nile?) → 6,853 km** `</API>` 6,853 kilometers, the White Nile being its main source.	2.08	✓
If Venus had an atmosphere similar to Earth's then you would expect Venus' mean temperature to be 499 K (1.74 x 287) rather than 735 K which is `<API>` **Calculator(735 / 499) → 1.47** `</API>` 1.47 (735 / 499) times hotter than it should be.	1.59	✓
You are here: Home / Featured / Catch this fast train to success! `<API>` **WikiSearch(Fast train success) → Fast Train > It also peaked at #23 on the Canadian CHUM singles chart, on June 26, 1971. The success of this single established Myles Goodwyn as the band's main songwriter, and made it possible for April Wine to record a second album.** `</API>` Don't wait weeks and [. . .]	0.92	✗
Os Melhores Escolas em Jersey 2020 `<API>` **MT(Os Melhores Escolas em Jersey) → The Best Schools in Jersey** `</API>` On this page you can search for Universities, Colleges and Business schools in Jersey	0.70	✓
Enjoy these pictures from the `<API>` **Calendar() → Today is Friday, April 19, 2013.** `</API>` Easter Egg Hunt.	0.33	✓
85 patients (23%) were hospitalised alive and admitted to a hospital ward. Of them, `<API>` **Calculator(85 / 23) → 3.70** `</API>` 65% had a cardiac aetiology [. . .]	−0.02	✗
But hey, after the `<API>` **Calendar() → Today is Saturday, June 25, 2011.** `</API>` Disneyland fiasco with the fire drill, I think it's safe to say Chewey won't let anyone die in a fire.	−0.41	✗
The last time I was with `<API>` **QA(Who was last time I was with?) → The Last Time** `</API>` him I asked what he likes about me and he said he would tell me one day.	−1.23	✗

Table 10: Examples of API calls for different tools, sorted by the value of $L_i^- - L_i^+$ that is used as a filtering criterion. High values typically correspond to API calls that are intuitively useful for predicting future tokens.

approaches, additional information is *always* provided, regardless of whether it is helpful or not. In contrast, Toolformer learns for itself to explicitly asks for the right information.

Tool Use Several approaches aim to equip LMs with the ability to use external tools such as search engines (Komeili et al., 2022; Thoppilan et al., 2022; Lazaridou et al., 2022; Shuster et al., 2022; Yao et al., 2022), web browsers (Nakano et al., 2021), calculators (Cobbe et al., 2021; Thoppilan et al., 2022), translation systems (Thoppilan et al., 2022) and Python interpreters (Gao et al., 2022). The way these models learn to use tools can roughly be divided into two approaches: Either they rely on large amounts of human supervision (Komeili et al., 2022; Nakano et al., 2021; Thoppilan et al., 2022) or they work by prompting the language model in a few-shot setup tailored towards a specific task where it is known a priori which tools needs to be

used (Gao et al., 2022; Lazaridou et al., 2022; Yao et al., 2022). In contrast, the self-supervised nature of Toolformer enables it to learn how and when to use tools without requiring a specific prompt that shows task-specific examples of how a tool could be used. Perhaps most closely related to our work is TALM (Parisi et al., 2022), an approach that uses a similar self-supervised objective for teaching a model to use a calculator and a search engine, but explores this only in settings where a model is finetuned for downstream tasks.

Bootstrapping The idea of using self-training and bootstrapping techniques to improve models has been investigated in various contexts, ranging from word sense disambiguation (Yarowsky, 1995), relation extraction (Brin, 1999; Agichtein and Gravano, 2000), parsing (McClosky et al., 2006; Reichart and Rappoport, 2007), sequence generation (He et al., 2020), few-shot text classi-

fication ([Schick and Schütze, 2021a](#)) and retrieval ([Izacard and Grave, 2021](#)) to reasoning ([Zelikman et al., 2022](#)). In a similar spirit to these approaches, Toolformer is trained on its own predictions after applying a perplexity-based filtering step.

7 Limitations

While our approach enables LMs to learn how to use a variety of tools in a self-supervised way, there are some clear limitations to what can be achieved with our method in its current form. One such limitation is the inability of Toolformer to use tools in a *chain* (i.e., using the output of one tool as an input for another tool). This is due to the fact that API calls for each tool are generated independently; as a consequence, there are no examples of chained tool use in the finetuning dataset. Our current approach also does not allow the LM to use a tool in an *interactive* way – especially for tools such as search engines, that could potentially return hundreds of different results, enabling a LM to browse through these results or to refine its search query in a similar spirit to [Nakano et al. (2021)](#) can be crucial for certain applications. Beyond this, we found models trained with Toolformer to often be sensitive to the exact wording of their input when deciding whether or not to call an API; this is perhaps unsurprising given that LMs are known to be very sensitive to the prompt they are provided with in both zero- and few-shot settings ([Jiang et al., 2020](#); [Schick and Schütze, 2021a](#)). Depending on the tool, our method is also very sample-inefficient; for example, processing more than a million documents results in only a few thousand examples of useful calls to the calculator API. A potential solution to this problem might be to iteratively apply our approach, similar to how this is done in related bootstrapping approaches ([Schick and Schütze, 2021a](#); [Izacard and Grave, 2021](#); [Parisi et al., 2022](#)). Finally, when deciding whether or not to make an API call, Toolformer currently does not take into account the tool-dependent, computational cost incurred from making an API call.

8 Conclusion

We have introduced Toolformer, a language model that learns in a self-supervised way how to use different tools such as search engines, calculators, and translation systems via simple API calls. This is done by finetuning on a large number of sampled API calls that are filtered based on whether they reduce perplexity on future tokens. Toolformer considerably improves zero-shot performance of a 6.7B parameter GPT-J model, enabling it to even outperform a much larger GPT-3 model on a range of different downstream tasks.

References

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.](#)

Eugene Agichtein and Luis Gravano. 2000. [Snowball: Extracting relations from large plain-text collections.](#) In *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, page 85–94, New York, NY, USA. Association for Computing Machinery.

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*, volume 463. ACM press New York.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. [Semantic parsing on Freebase from question-answer pairs.](#) In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2021. [Improving language models by retrieving from trillions of tokens.](#)

Sergey Brin. 1999. [Extracting patterns and relations from the world wide web.](#) In *The World Wide Web and Databases*, pages 172–183, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. [Language models are few-shot learners.](#) In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, et al. 2022. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. 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 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. 2022. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2020. Revisiting self-training for neural sequence generation. In *International Conference on Learning Representations*.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor.

Gautier Izacard and Edouard Grave. 2021. Distilling knowledge from reader to retriever for question answering. In *International Conference on Learning Representations*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Atlas: Few-shot learning with retrieval augmented language models.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2022. Survey of hallucination in natural language generation. *ACM Computing Surveys*.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of machine translation summit x: papers*, pages 79–86.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2022. Internet-augmented dialogue generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8460–8478, Dublin, Ireland. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. 2022. Internet-augmented language models through few-shot prompting for open-domain question answering. *arXiv preprint arXiv:2203.05115*.

Patrick Lewis, Barlas Oğuz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2019. Mlqa: Evaluating cross-lingual extractive question answering. *arXiv preprint arXiv:1910.07475*.

Xi Victoria Lin, Todor Mihaylov, Mikel Artetxe, Tianlu Wang, Shuohui Chen, Daniel Simig, Myle Ott, Naman Goyal, Shruti Bhosale, Jingfei Du, Ramakanth Pasunuru, Sam Shleifer, Punit Singh Koura, Vishrav Chaudhary, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Zornitsa Kozareva, Mona Diab, Veselin Stoyanov, and Xian Li. 2021. Few-shot learning with multilingual language models.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders,

Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, Online. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

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

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 616–623, Prague, Czech Republic. Association for Computational Linguistics.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Timo Schick, Jane Dwivedi-Yu, Zhengbao Jiang, Fabio Petroni, Patrick Lewis, Gautier Izacard, Qingfei You, Christoforos Nalmpantis, Edouard Grave, and Sebastian Riedel. 2022. Peer: A collaborative language model.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the*

16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 255–269, Online. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021b. Generating datasets with pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6943–6951, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, Morteza Behrooz, William Ngan, Spencer Poff, Naman Goyal, Arthur Szlam, Y-Lan Boureau, Melanie Kambadur, and Jason Weston. 2022. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. 2022. Lamda: Language models for dialog applications.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France. European Language Resources Association.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pretrained transformer language models.

A API Details

When sampling and filtering API calls, by default we use values of $\tau_s = 0.05$ and $\tau_f = 1.0$ – i.e., we only make API calls at positions where the probability of the <API> token is at least 5%, and we keep API calls if they reduce the loss by at least 1.0. We only keep the top $k = 5$ such positions and sample up to $m = 5$ API calls for each position identified in a piece of text. Due to the heuristic filtering described below, we generate API calls for the calculator and machine translation system on only a small subset of $\mathcal{C}$; to compensate for this, we set $\tau_s = 0.0$, $k = 20$ and $m = 10$ for these tools. As the resulting sets of API calls are still comparably small, we additionally set $\tau_f = 0.5$.

A.1 Implementation

Question Answering We use the Atlas model of Izacard et al. (2022) finetuned on Natural Questions (Kwiatkowski et al., 2019) as our question answering system. For creating $\mathcal{C}^*$ we use Atlas-large, enabling us to efficiently process millions of API calls; during inference, we use the larger Atlas-xxl model.

Calculator Our calculator is based on a simple Python script and only supports the operators "+", "−", "∗", and "/". It does not return any result for syntactically invalid equations. For sampling API calls, we apply heuristic filters to our subset of CCNet and only process documents that either (i) contain at least three numbers within a window of 100 tokens, where one of these numbers is the result of applying a mathematical operation to the other two, (ii) contain one of the sequences "=", "equals", "equal to", "total of", "average of" followed by a number, or (iii) contain at least three numbers; for texts that only match the last criterion, we only keep a random subset of 1%.

Calendar For creating our dataset $\mathcal{C}^*$, we operate under the assumption that the calendar date in such cases should be the date that the document was created. We approximate this by extracting the date from the URL, if it is present. We filter out texts for which a date cannot be extracted, leaving around 18% of the documents.

Machine Translation For both training and inference, we use the 600M parameter NLLB (Costa-jussà et al., 2022) as our machine translation (MT) model. The source language is automatically detected using the fastText classifier (Joulin et al.,

2016), while the target language is always set to English. Since most of the CCNet dataset is in English, we filter out the parts that contain only English text before generating API calls. More specifically, we only keep those paragraphs which contain text chunks in a language other than English preceded and followed by English text. We use text chunks of size 10 tokens. To determine whether the middle text chunk is in a language different than English we again use the fastText classifier with a confidence greater than 0.8. We also filter out any text chunks that contain only numbers or special symbols. This filtering mechanism allows us to generate data more efficiently by focusing our API call generations in places where the MT tool is likely to be helpful. After generating the MT API calls, we additionally remove from our training set those where the input to the MT tool appears after the API call but not before it. While during data generation the model can look ahead to generate API calls, this is not possible at inference time, so we want to dissuade the model from calling the API in such cases.

A.2 Prompts

Below, we list the prompts used to sample API calls for each tool considered.

Question Answering We use the following prompt for the question answering tool:

```
Your task is to add calls to a Question
Answering API to a piece of text.
The questions should help you get
information required to complete the
text. You can call the API by writing
"[QA(question)]" where "question" is the
question you want to ask. Here are some
examples of API calls:
Input: Joe Biden was born in Scranton,
Pennsylvania.
Output: Joe Biden was born in [QA("Where
was Joe Biden born?")] Scranton,
[QA("In which state is Scranton?")]
Pennsylvania.

Input: Coca-Cola, or Coke, is a
carbonated soft drink manufactured by
the Coca-Cola Company.
Output: Coca-Cola, or [QA("What other
name is Coca-Cola known by?")] Coke, is
a carbonated soft drink manufactured by
[QA("Who manufactures Coca-Cola?")] the
Coca-Cola Company.

Input: x
Output:
```

Calculator We use the following prompt for the calculator:

```
Your task is to add calls to a
Calculator API to a piece of text.
```

The calls should help you get information required to complete the text. You can call the API by writing "[Calculator(expression)]" where "expression" is the expression to be computed. Here are some examples of API calls:
Input: The number in the next term is 18 + 12 x 3 = 54.
Output: The number in the next term is 18 + 12 x 3 = [Calculator(18 + 12 * 3)] 54.

Input: The population is 658,893 people. This is 11.4% of the national average of 5,763,868 people.
Output: The population is 658,893 people. This is 11.4% of the national average of [Calculator(658,893 / 11.4%)] 5,763,868 people.

Input: A total of 252 qualifying matches were played, and 723 goals were scored (an average of 2.87 per match). This is three times less than the 2169 goals last year.
Output: A total of 252 qualifying matches were played, and 723 goals were scored (an average of [Calculator(723 / 252)] 2.87 per match). This is twenty goals more than the [Calculator(723 – 20)] 703 goals last year.

Input: I went to Paris in 1994 and stayed there until 2011, so in total, it was 17 years.
Output: I went to Paris in 1994 and stayed there until 2011, so in total, it was [Calculator(2011 – 1994)] 17 years.

Input: From this, we have 4 * 30 minutes = 120 minutes.
Output: From this, we have 4 * 30 minutes = [Calculator(4 * 30)] 120 minutes.

Input: x
Output:

Wikipedia Search

We use the following prompt for the Wikipedia search tool:

Your task is to complete a given piece of text. You can use a Wikipedia Search API to look up information. You can do so by writing "[WikiSearch(term)]" where "term" is the search term you want to look up. Here are some examples of API calls:
Input: The colors on the flag of Ghana have the following meanings: red is for the blood of martyrs, green for forests, and gold for mineral wealth.
Output: The colors on the flag of Ghana have the following meanings: red is for [WikiSearch("Ghana flag red meaning")] the blood of martyrs, green for forests, and gold for mineral wealth.

Input: But what are the risks during production of nanomaterials? Some

nanomaterials may give rise to various kinds of lung damage.
Output: But what are the risks during production of nanomaterials? [WikiSearch("nanomaterial production risks")] Some nanomaterials may give rise to various kinds of lung damage.

Input: Metformin is the first-line drug for patients with type 2 diabetes and obesity.
Output: Metformin is the first-line drug for [WikiSearch("Metformin first-line drug")] patients with type 2 diabetes and obesity.

Input: x
Output:

Machine Translation

We use the following prompt for the machine translation tool:

Your task is to complete a given piece of text by using a Machine Translation API.
You can do so by writing "[MT(text)]" where text is the text to be translated into English.
Here are some examples:

Input: He has published one book: O homem suprimido ("The Supressed Man")
Output: He has published one book: O homem suprimido [MT(O homem suprimido)] ("The Supressed Man")

Input: In Morris de Jonge's Jeschuah, der klassische jüdische Mann, there is a description of a Jewish writer
Output: In Morris de Jonge's Jeschuah, der klassische jüdische Mann [MT(der klassische jüdische Mann)], there is a description of a Jewish writer

Input: 南京高淳县住房和城乡建设局 城市新区 设 计 a plane of reference Gaochun is one of seven districts of the provincial capital Nanjing
Output: [MT(南京高淳县住房和城乡建设局 城市新区 设 计)] a plane of reference Gaochun is one of seven districts of the provincial capital Nanjing

Input: x
Output:

Calendar

We use the following prompt for the calendar tool:

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()]" 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: The president of the United
States is Joe Biden.
Output: The president of the United
States is [Calendar()] Joe Biden.

Input: The current day of the week is
Wednesday.
Output: The current day of the week is
[Calendar()] Wednesday.

Input: The number of days from now until
Christmas is 30.
Output: The number of days from now
until Christmas is [Calendar()] 30.

Input: The store is never open on the
weekend, so today it is closed.
Output: The store is never open on the
weekend, so today [Calendar()] it is
closed.

Input: x
Output:
```

B   Toolformer Training

We use up to 25k examples per API. Max sequence length 1,024. Effective batch size of 128. All models are trained using DeepSpeed's ZeRO-3 (Rasley et al., 2020). We used 8 NVIDIA A100 40GB GPUs with BF16. Training up to 2k steps, where we evaluate PPL on a small development set from CCNet containing 1,000 examples every 500 steps. We pick the checkpoint that performs best.

C   Zero-Shot Prompts

C.1   LAMA and TEMPLAMA

For both LAMA and TEMPLAMA, given an input text **x**, we use the following prompt: `Please complete the following text so that it is factually correct:` **x**.

C.2   Math Benchmarks

For all math benchmarks, given a context **x** and a question **q**, our prompt is: **x q** `The answer is.`

C.3   Question Answering

For all question answering datasets, including DATESET, we simply prefix the question with `Answer the following question:`. We append a question mark if the question does not already end with one.

C.4   Multilingual Question Answering

For MLQA, given a context **x** and a question **q**, our prompt is: `Your task is`

Template	Size
How many days {ago was, are there until} {*past_date*, *future_date*}?	400
What {day of the week, day of the month, month, year} was it (*current_date – past_date*) {days, weeks, months, years} ago?	800
What {day of the week, day of the month, month, year} will it be in (*future_date – current_date*) days?	800
What day of the week {is, was} it on {*past_date*, *future_date*}?	400
What {day of the week, day of the month, month, year} {is, was} it {the day before yesterday, yesterday, today, tomorrow, the day after tomorrow}?	4,000
What {day of the week, day of the month, month} {is, was} *holiday* this year?	1,800
How many {days, weeks, months, years} {ago was, are there until} *holiday* this year?	1,200
Total	9,400

Table 11: Templates used to create DATESET where a *current_date* is randomly selected. For each *current_date*, a random *past_date* and *future_date* is generated and used to fill each template, if relevant. The federal holidays in the United States (e.g., Thanksgiving) were used in the templates involving holidays.

```
to answer a question based on
the following paragraph:  x Now
answer the following question in
English:  q.
```

D   DATESET

DATESET is created by first randomly selecting 500 "current dates". For each current date, another relatively past/future date is randomly selected within a four-year range, and the two dates are used to fill the query templates in Table 11. An example of one such query using the first template would be, "How many days ago was August 14, 2020?" If called, the Calendar tool would return the presumed current date (e.g., "Today is Sunday, November 20, 2020").