Toolformer: Language Models Can Teach Themselves to Use Tools

Radek Bartyzal
MITON Times
9. 3. 2023

Goal

- LMs are good at generating text

- we have a lot of expert systems good at other things

- why not use external APIs as sources of data?
  - external memory
  - computation abilities
  - etc.

How to do it?

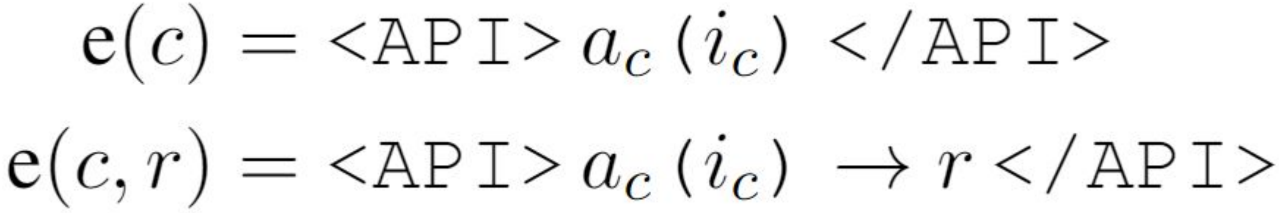
1. Train **LM** on standard **dataset** the standard way

2. Augment the dataset => **new dataset** with tool usage
   - = this is the new stuff

3. Fine-tune the **LM** on the **new dataset** the standard way

Augmenting the dataset

- API calls must be represented by string in a form:
  - a = API name
  - i = input
  - r = API call result



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

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

Augmenting the dataset

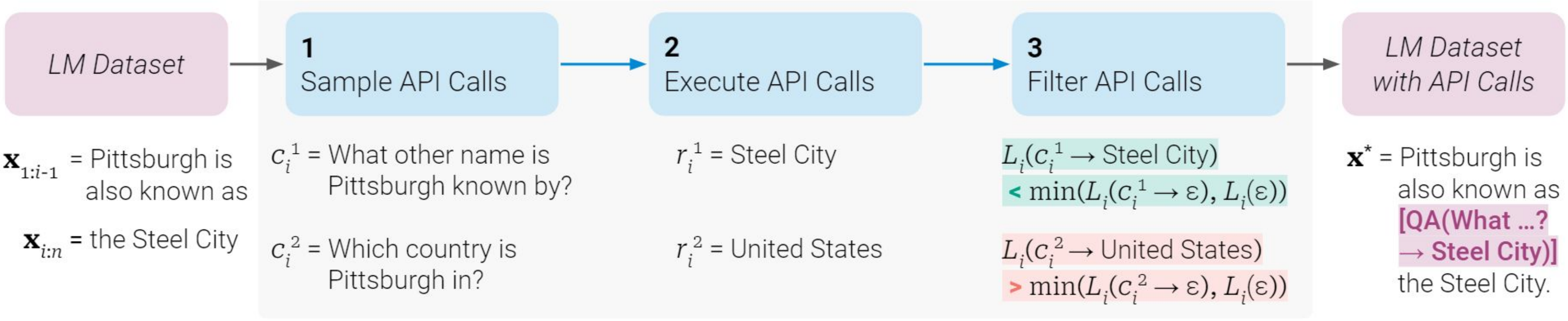
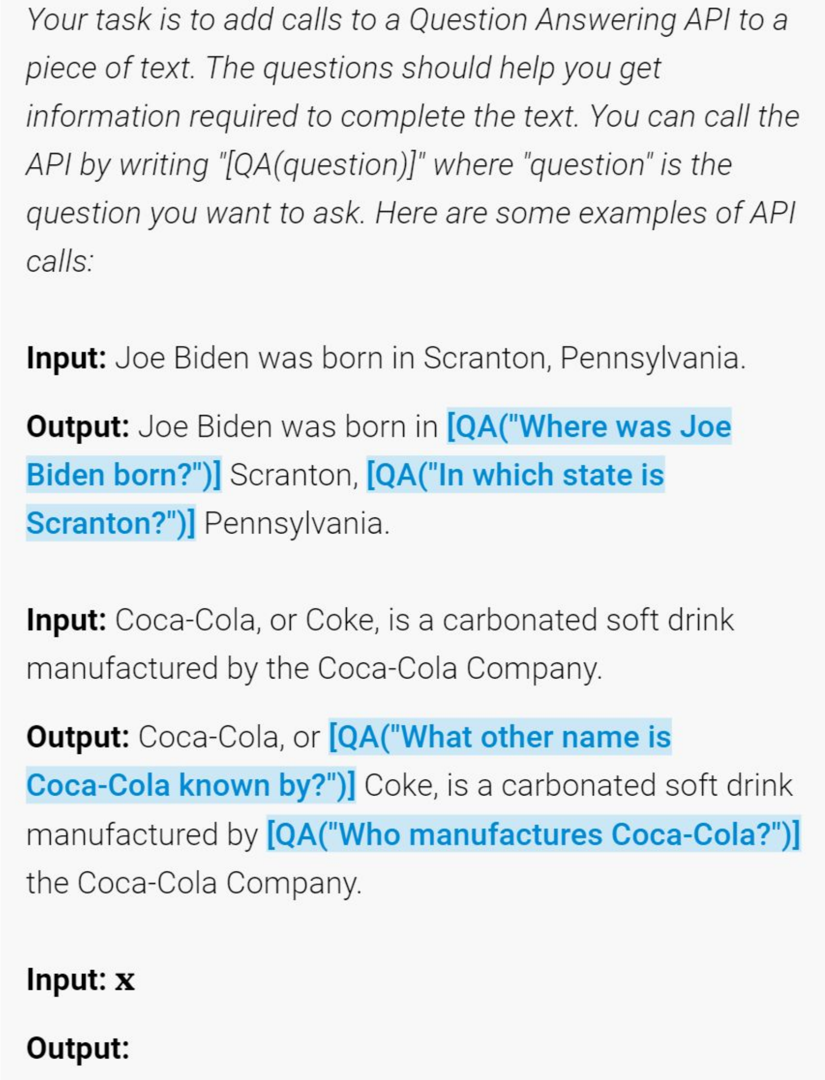

LM Dataset
1
Sample API Calls
2
Execute API Calls
3
Filter API Calls
LM Dataset with API Calls
$\mathbf{x}_{1:i-1}$ = Pittsburgh is also known as
$\mathbf{x}_{i:n}$ = the Steel City
$c_i^{\,1}$ = What other name is Pittsburgh known by?
$c_i^{\,2}$ = Which country is Pittsburgh in?
$r_i^{\,1}$ = Steel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

Figure 2: 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 $\mathbf{x}^*$.

Sampling API Calls

prompt LM to add API calls into each sample:

1. sample up to *k* candidate positions for doing API calls
    - = what's the probability of LM generating <API> token at position *i*
    - keep top *K* positions
2. generate *m* API calls at each position
    - for each type of API



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: $\mathbf{x}$

Output:

Filtering API calls

- Cross-entropy loss of a model prefixed with **z**: $L_i(\mathbf{z}) = -\sum_{j=i}^{n} w_{j-i} \cdot \log p_M(x_j \mid \mathbf{z}, x_{1:j-1})$

- L+ = API call + its result
- L- = MIN(no API call, API call but empty result)

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

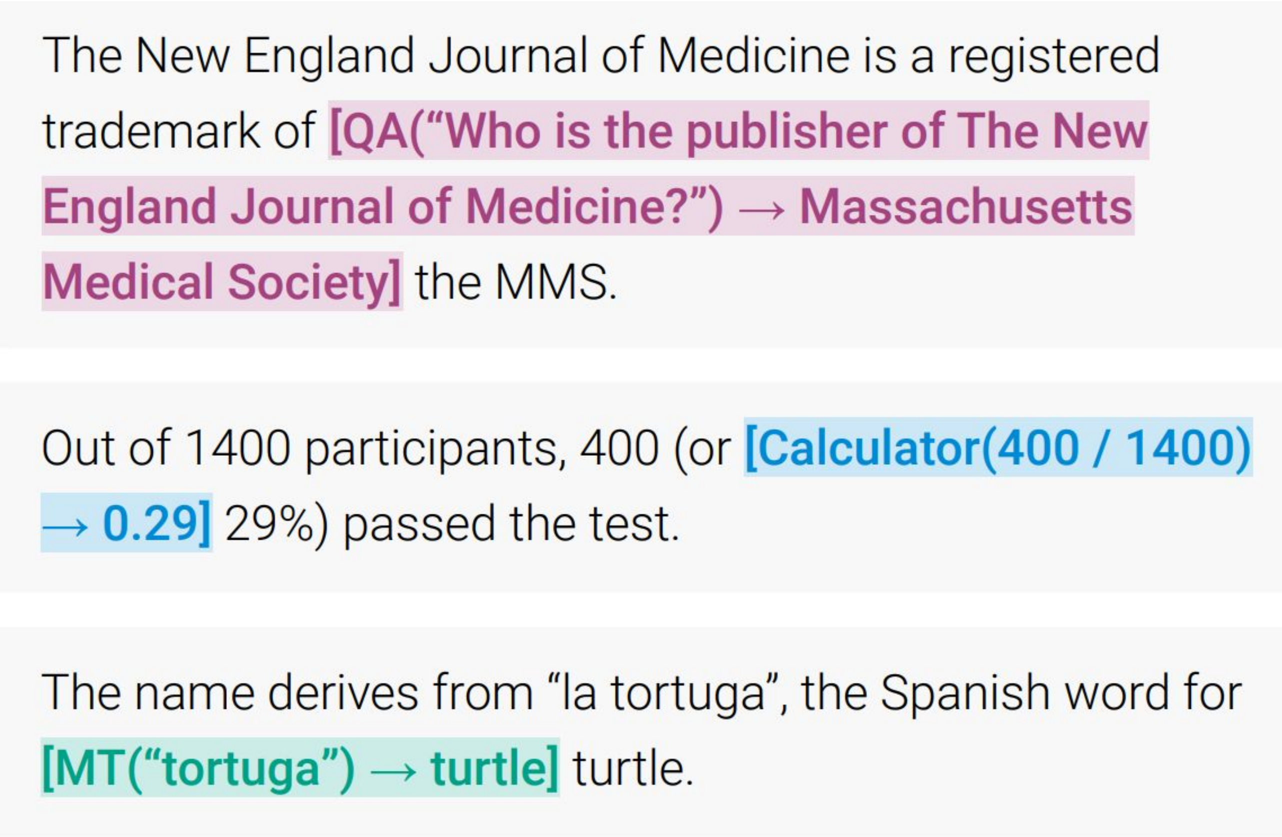
- Keep API calls where: $L_i^- - L_i^+ \geq \tau_f$
  -
  - **=> adding the API call and its result reduces the loss by at least τf, compared to not doing any API call or obtaining no result from it**

Inference

- LM generates tokens one by one

- when it generates special token “->” the external API is called and the result is appended to the generated text

- generation then resumes with these tokens given to LM



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.

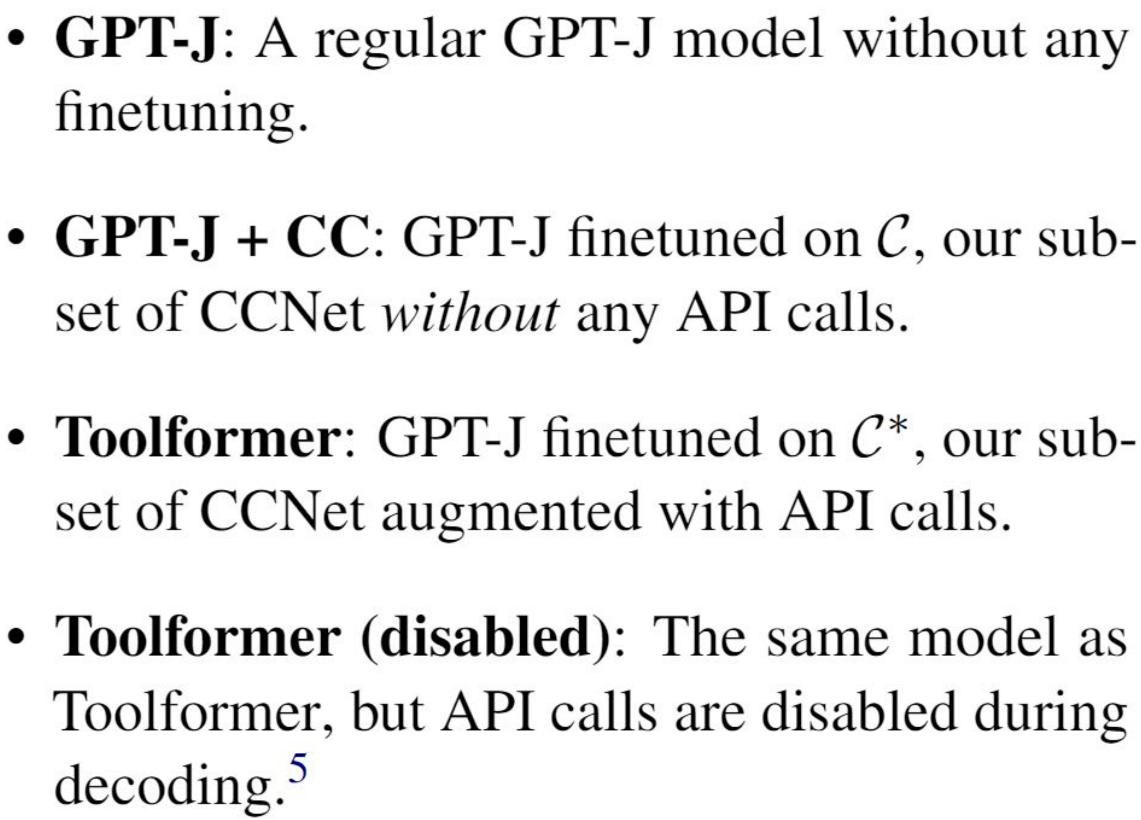
Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

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

Used tools in paper

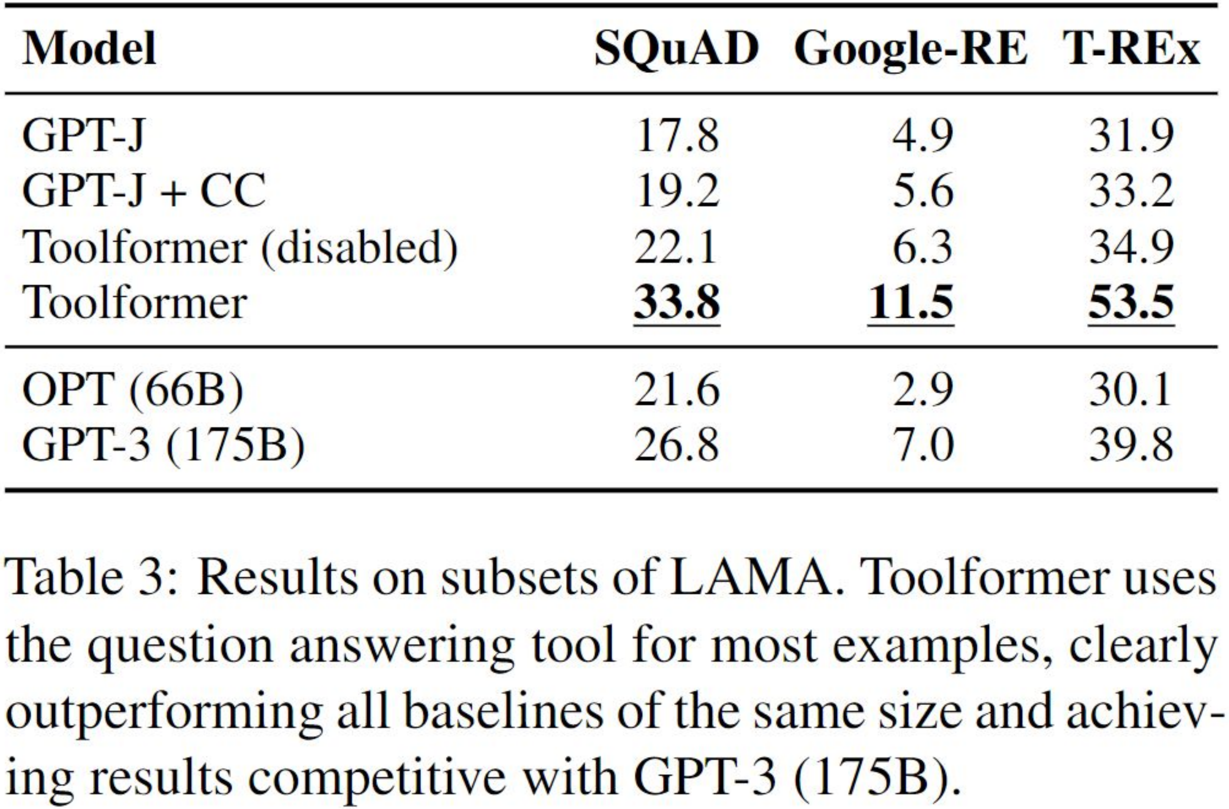
- Question Answering
  - another LM fine-tuned on QA
- Calculator:
  - only 4 simple ops, result rounded to 2 decimals
- Wikipedia search:
  - BM25 retrieval model, returns snippets
- Machine Translation:
  - another LM
- Calendar:
  - always returns current day

Experiments: used 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]

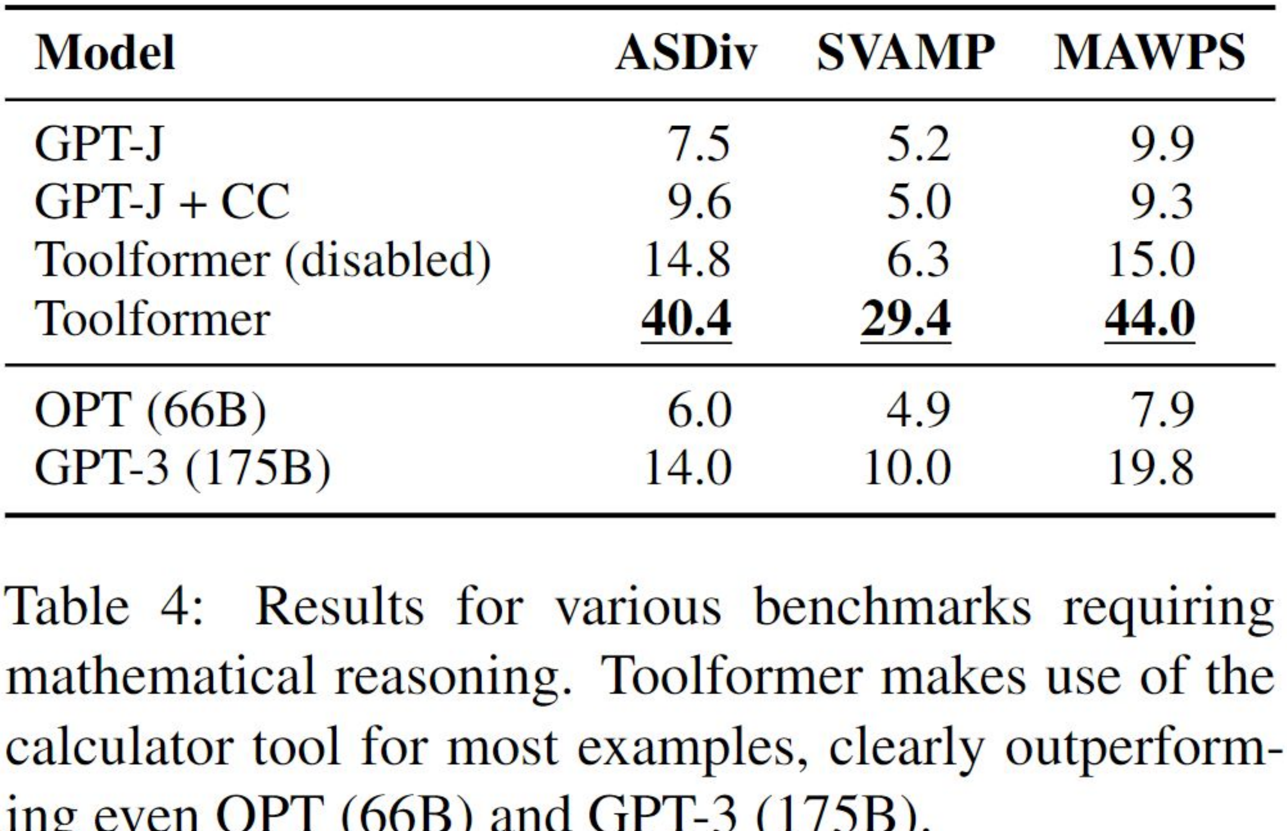
Experiments: complete a short statement with a missing fact



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).

Experiments: Mathematical reasoning

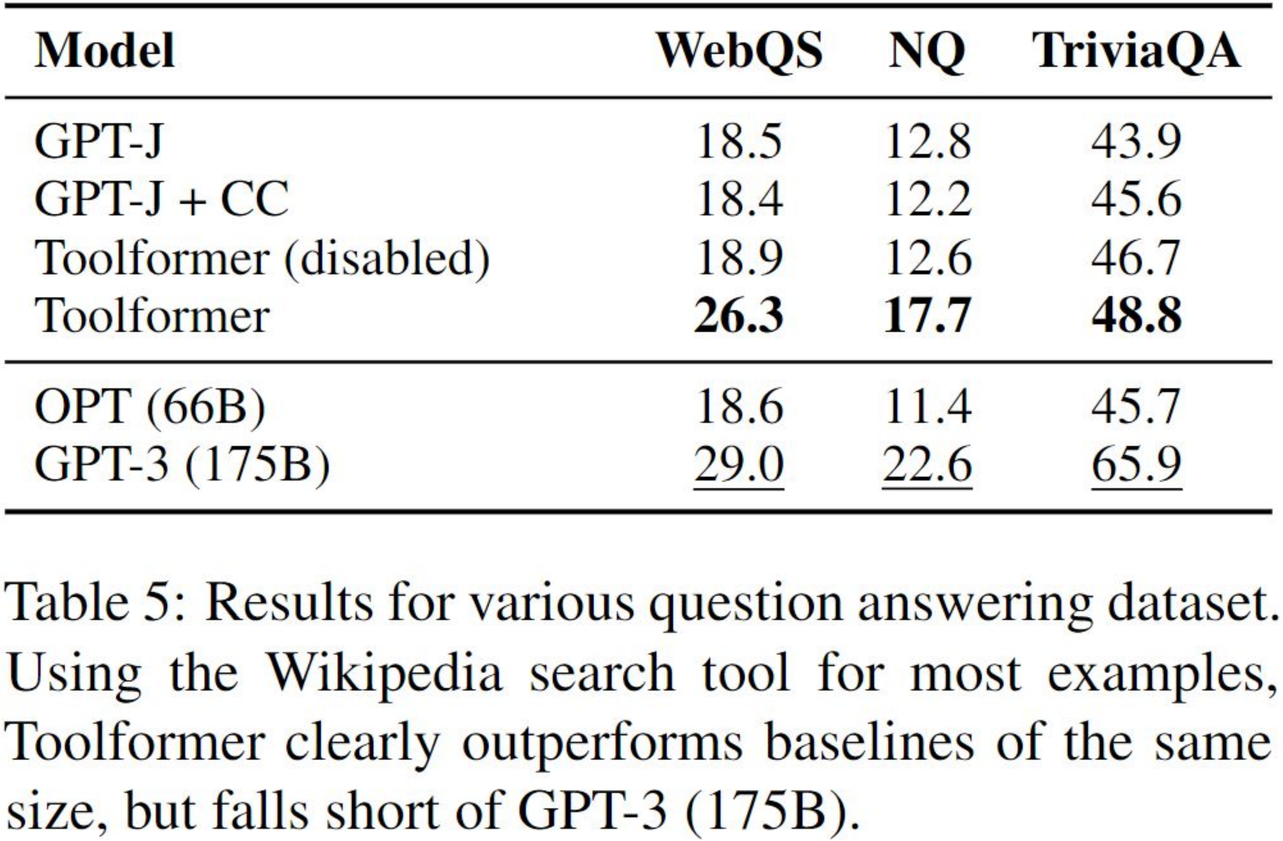


Model	ASDiv	SVAMP	MAWPS
GPT-J	7.5	5.2	9.9
GPT-J + CC	9.6	5.0	9.3
Toolformer (disabled)	14.8	6.3	15.0
Toolformer	**40.4**	**29.4**	**44.0**
OPT (66B)	6.0	4.9	7.9
GPT-3 (175B)	14.0	10.0	19.8

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).

Experiments: QA

- Toolformer results are with disabled QA API

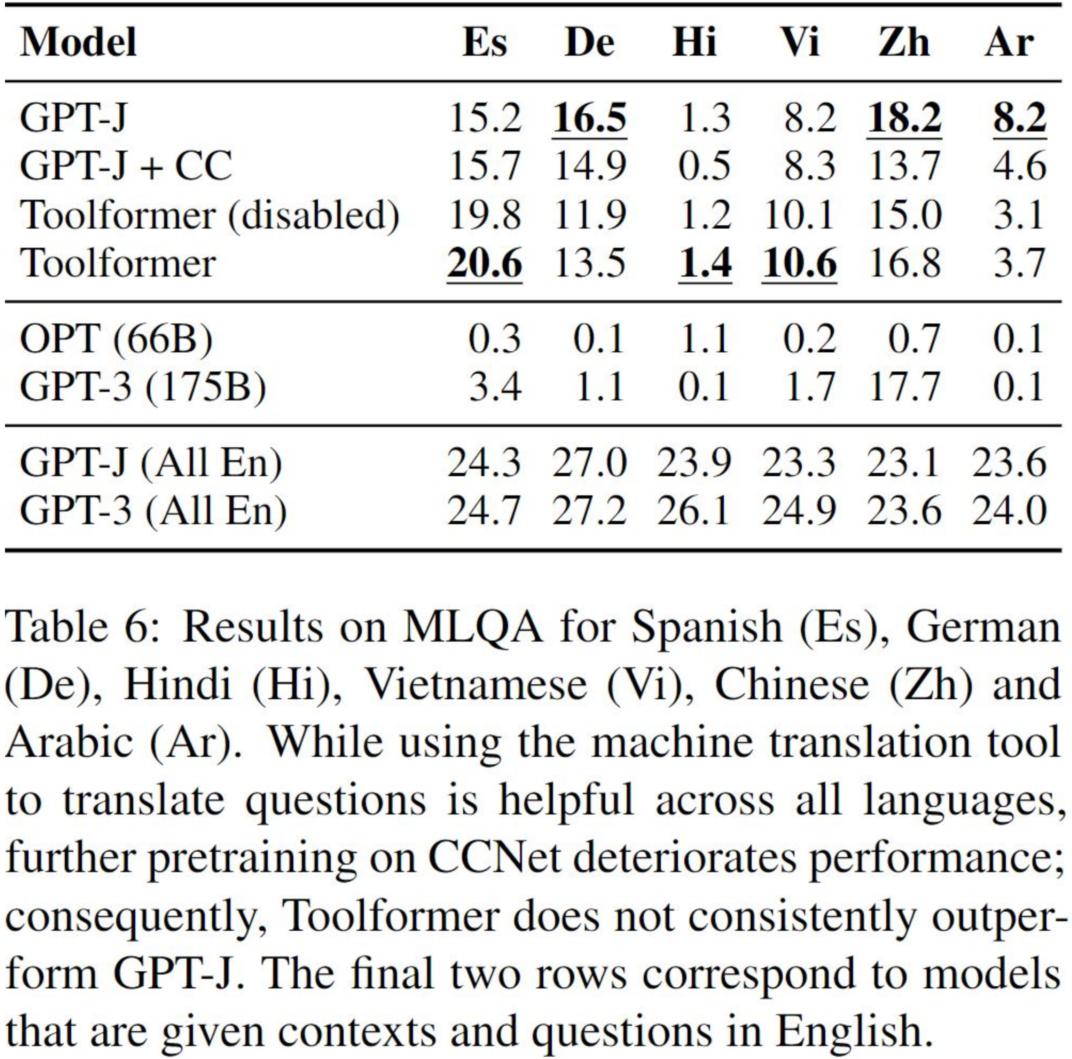


Model	WebQS	NQ	TriviaQA
GPT-J	18.5	12.8	43.9
GPT-J + CC	18.4	12.2	45.6
Toolformer (disabled)	18.9	12.6	46.7
Toolformer	**26.3**	**17.7**	**48.8**
OPT (66B)	18.6	11.4	45.7
GPT-3 (175B)	29.0	22.6	65.9

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).

Experiments: ML QA

- multi-lingual QA
- OPT, GPT3 struggle with answering in english even when asked to do so

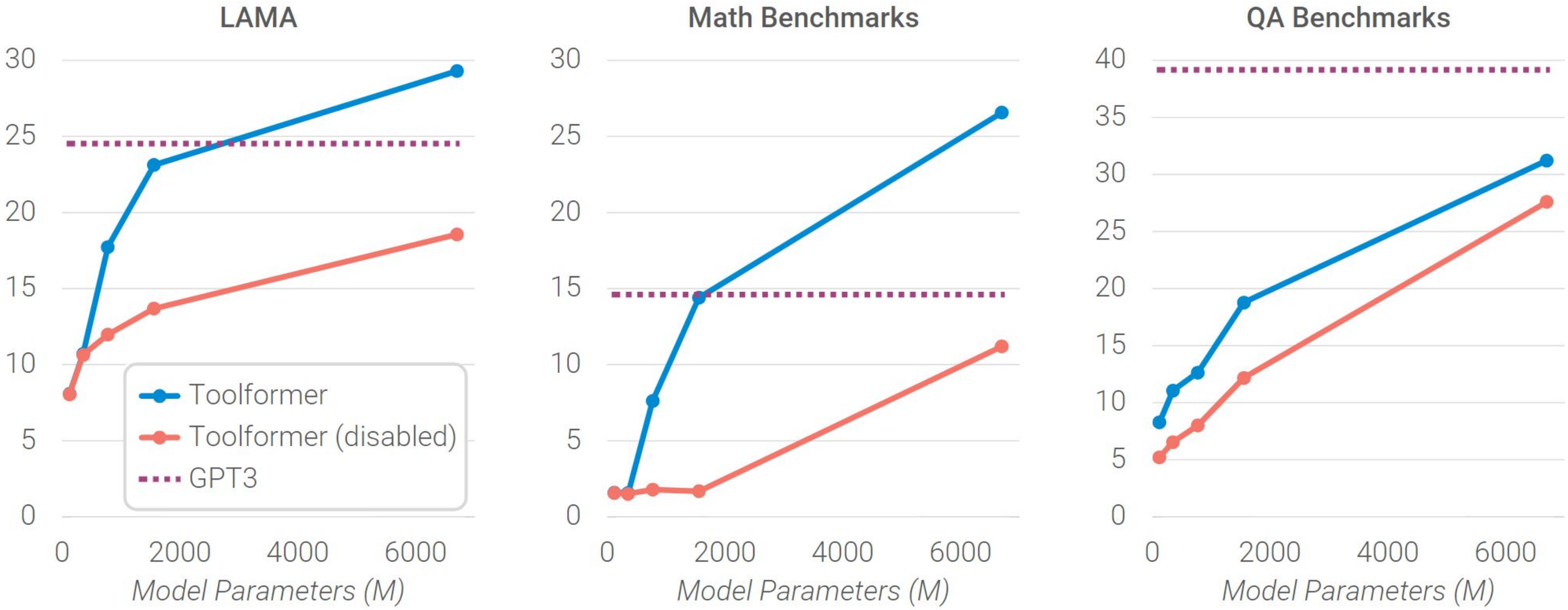


Model	Es	De	Hi	Vi	Zh	Ar
GPT-J	15.2	**16.5**	1.3	8.2	**18.2**	**8.2**
GPT-J + CC	15.7	14.9	0.5	8.3	13.7	4.6
Toolformer (disabled)	19.8	11.9	1.2	10.1	15.0	3.1
Toolformer	**20.6**	13.5	**1.4**	**10.6**	16.8	3.7
OPT (66B)	0.3	0.1	1.1	0.2	0.7	0.1
GPT-3 (175B)	3.4	1.1	0.1	1.7	17.7	0.1
GPT-J (All En)	24.3	27.0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

Experiments: Scaling Laws

- 600M params required to be able to use APIs:



Sources

- paper: https://arxiv.org/abs/2302.04761
-