Toolformer: Language Models Can Teach Themselves to Use Tools

Timo Schick 저술 · 2023

(Meta AI Research)

00 Summary

- Ppl을 낮추는 API call 기반의 corpus augmentation을 통해 모델 파인튜닝함
- 기본적인 성능도 개선되고, 디코딩 중간마다 API를 부를지 말지 결정해서 API를 통해 외부로 부터 결과값을 디코딩 중에 삽입함
- In context learning에 의존한 데이터 augmentation 기법이라 범용성은 있지만 모델에 의존성이 크고
- 여기서 사용한 API는 크게 5종류로 앞으로 어떤 API까지 적용이 가능할지는 상상해봐야함
- API로 augmentation한 문서를 merge하는 과정이 추가로 요구됨, 이 부분은 공수가 좀 더 들어갈듯
  - 1 input -> 1 API call

01 Abstract

- LLM이 잘하긴하지만 역설적이게도 간단한 연산은 못함
  - They also, paradoxically, struggle with basic functionality, such as arithmetic or factual lookup, where much simpler and smaller models excel.
- External tools via simple APIs을 사용해서 모델 성능 개선
- Toolformer는 언제 APIs를 호출할지, 어떤 종류를 호출할지, 어떤 arguments를 넣을지등을 결정해서 furue token prediction을 개선
  - 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.
- 논문에서 다루는 APIs는 5가지
  - 계산기, Q&A 시스템, 검색엔진, 번역시스템, 캘린더

02 Introduction

- LLM의 문제점
  - LLMs은 크기가 커야됨, 최신정보 업데이트가 어렵다
  - hallucinate 문제가 따라다닌다
    - Low-resource 언어 이해 잘못함
    - 수학 잘 못함
    - 시간에 대한 개념 부족
- 이러한 이슈를 극복하기 위해선 external tools을 사용해볼 수 있음
  - Toolformer 제안
    - GPT-J model (Wang and Komatsuzaki, 2021) with 6.7B parameters
    - use a self-supervised loss to determine which of these API calls actually help the model in predicting future tokens

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.

03 Approach

- 주어진 텍스트에 API Calls을 넣기 위해선 Special tokens 사용!
  - [] (여기선 편의상 <API>로 표기

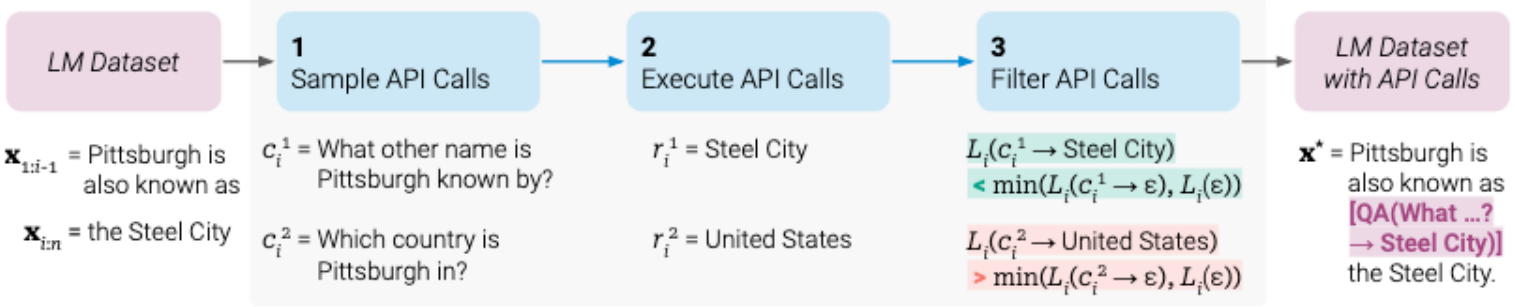

Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text $\mathbf{x}$, we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \ldots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i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

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:

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

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

[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.

03 Approach

- 3단계
  - In-context learning 을 사용해서 potential API calls을 샘플링
    - 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.
  - API calls을 실행하고 predicting future tokens에 도움이 되는 답변들인지 체크
  - 필터링 후에 서로 다른 툴에서 나온 API calls을 병합 후 augmented dataset C*으로 저장하고 여기에 대해서 finetuning 진행

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:
```

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:
```

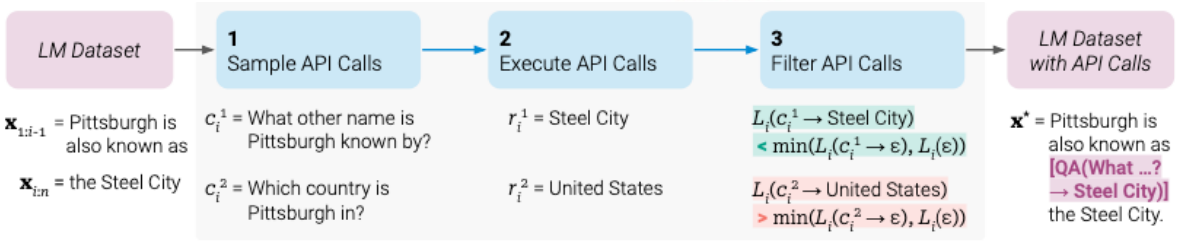

Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text $\mathbf{x}$, we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \ldots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i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

03 Approach

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
```

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)]"
```

Prompt

당신의 임무는 질문에 전화를 추가하는 것입니다. 텍스트 조각에 API를 응답합니다.
질문은 당신이 얻는 데 도움이 될 것입니다 완료하는 데 필요한 정보 텍스트. 작성하여 API를 호출할 수 있습니다.
"[QA(질문)]" 여기서 "질문"은 묻고 싶은 질문. 여기 몇 가지가 있습니다
API 호출의 예:

입력: Joe Biden은 Scranton에서 태어났습니다.
펜실베니아.
출력: Joe Biden은 [QA("어디서
Joe Biden이 태어났습니까?")] Scranton,
[QA("스크랜턴은 어느 주에 있습니까?")]
펜실베니아.

입력: Coca-Cola 또는 Coke는
코카콜라 컴퍼니에서 제조한
탄산 청량 음료
출력: Coca-Cola 또는 [QA("기타 이름은 코카콜라로 알려져 있습니까?")] 콜라는
코카콜라 컴퍼니에서 제조한
[QA("누가 코카콜라를 제조합니까?")]
탄산 청량 음료

입력: 윤석열 대통령은

Output

"[QA(질문)]" 여기서 "질문"은 묻고 싶은 질문. 여기 몇 가지가 있습니다
API 호출의 예:

입력: Joe Biden은 Scranton에서 태어났습니다.
펜실베니아.
출력: Joe Biden은 [QA("어디서
Joe Biden이 태어났습니까?")] Scranton,
[QA("스크랜턴은 어느 주에 있습니까?")]
펜실베니아.

입력: Coca-Cola 또는 Coke는
코카콜라 컴퍼니에서 제조한
탄산 청량 음료
출력: Coca-Cola 또는 [QA("기타 이름은 코카콜라로 알려져 있습니까?")] 콜라는
코카콜라 컴퍼니에서 제조한
[QA("누가 코카콜라를 제조합니까?")]
탄산 청량 음료

입력: 윤석열 대통령은 믿을 좋아한다.
출력: 윤석열은 [QA("윤석열 대통령의 별명은 무엇입니까?")] 대통령은
[QA("윤석열의 다른 별명이 있나요?")] 믿을
좋아한다

Model

GLM39B-230110

Generate

finetuning 진행

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 $\mathbf{x}^*$.

03 Approach

- Sampling API Calls
  - 그림과 같이 정답이 있는 지문에 대해서 prompt를 사람이 생성하고, 이걸 학습시켜서 api 맵핑 및 호출시키는 것
  - 확률기반으로 api 콜링을 할만한 k개의 후보 포지션을 샘플링하고(API 스페셜토큰 생성확률이 높은!) -> 각 포지션마다 api 종류 중에 확률이 높은 m개를 또 샘플링
  - </API>를 eos 토큰으로 썼기 때문에 이거 없는 샘플은 버림

Sampling API Calls

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.

Figure 3

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:

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

03 Approach

- Executing API Calls
  - API 후보들을 실행
  - API 실행은 API 종류에 따라 다양
    - 파이썬 스크립트
    - 검색엔진
    - 번역기 등등

- Filtering API Calls
  - API 호출시 loss를 threshold이상으로 loss를 줄여주는 API call만 살아남음
  - Q) Prefix로 쓴다는게 무슨말일까?

[3] We provide $e(c_i, r_i)$ 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.

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.

03 Approach

- Model Finetuning
  - 필터링 후 결과 병합해서 original input에다가 끼워넣음
    - In context learning으로 각각 만들었기 때문에 API call도 각각 부르게 되어있음
  - Augmented Corpus로 파인튜닝
    - Standard LM objective 활용

Model Finetuning_1	Model Finetuning_2
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 $\mathbf{x}^* =$	$x_{1:i-1}, \mathbf{e}(c_i, r_i), x_{i:n}$; 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.

[3] We provide $\mathbf{e}(c_i, r_i)$ 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.

03 Approach

- Inference
- Regular decoding 을 진행하되 모델이 -> token을 생성하면 디코딩 멈췄다가 API 호출!
- 결과 나오면 다시 디코딩 진행

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.

04 Tools

- 5가지 툴
- QA system: LM, retrieval-augmented LM, Atlas
- Wikipedia search engine
- Caculator
- Calendar
- Machine translation system: LM

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.

05 Experiments

weighting func

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.

Table2

API	Number of Examples $\tau_f = 0.5$	$\tau_f = 1.0$	$\tau_f = 2.0$
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.

- Dataset Generation
  - use a subset of CCNet as our language modeling dataset C
  - GPT- J (Wang and Komatsuzaki, 2021) as our language model M
  - To reduce the computational cost of annotating C with API calls, we define heuristics
- Dataset Generation
  - finetune M on C∗ using a batch size of 128 and a learning rate of 1 · 10−5 with linear warmup for the first 10% of training.

05 Experiments

Downstream Tasks

- 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. (top k 안에 가 생성되도 호출?!)
- 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
  - API 결과들 때문에 디코딩 느려지지 않게 하나의 API call만 수행

LAMA

- We evaluate our models on the SQuAD, Google- RE and T-REx sul

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).

Math Datasets

- We test mathematical reasoning abilities on ASDiv (Miao et al., 20 benchmark

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).

Question Answering

- We look at Web Questions (Berant et al., 2013), Natural Questions 2017), the three question answering datasets considered by Brow

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).

Multilingual Question Answering

- We evaluate Toolformer and all baseline models on MLQA (Lewis e benchmark
- does not consistently outperform vanilla GPT-J. This is mainly bec deteriorates performance; this might be due to a distribution shift
  - CCNet에 학습한거 때문이다?!

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.

05 Experiments

Downstream Tasks

- 모델 크키가 커야 Toolformer도 잘된다

Scaling Laws

- We investigate how the ability to ask external tools for help affects performance as we vary the size of our LM
- GPT-2 family (Radford et al., 2019), with 124M, 355M, 775M and 1.6B parameters, respectively.

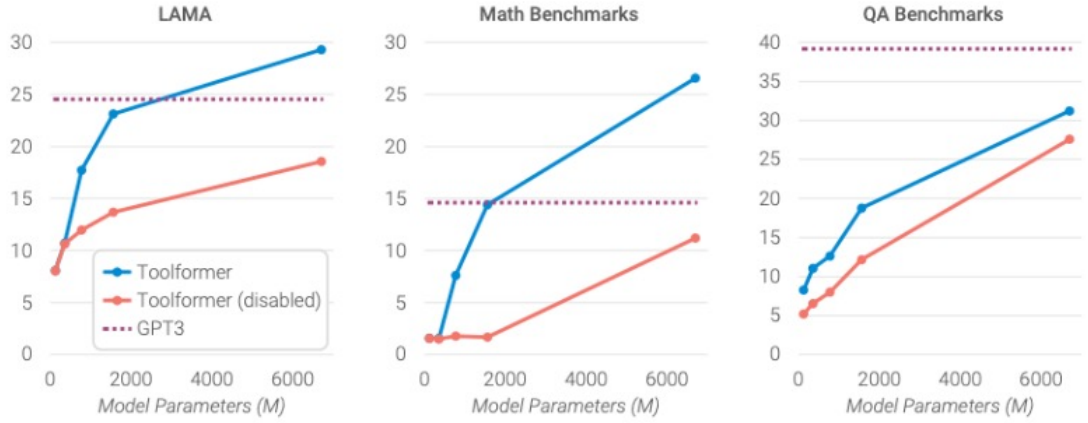

Figure 4: Average performance on LAMA, our math benchmarks and our QA benchmarks for GPT-2 models of different sizes and GPT-J finetuned with our approach, both with and without API calls. While API calls are not helpful to the smallest models, larger models learn how to make good use of them. Even for bigger models, the gap between model predictions with and without API calls remains high.

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 ___")`

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.

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.
  - 성능 저하 없다구!
- Most importantly, however, training on C∗ (our dataset annotated with API calls) does not lead to an increase in perplexity compared to training on C when API calls are disabled at inference time.8

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.

05 Experiments

Analysis

- K가 높을 수록 퍼포먼스가 높다
- LM loss차이가 클수록 useful한편이다

Deocidng Strategy

- we generate the 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

	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 (%).

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.
- However, some amount of noise in the API calls that are not filtered can actually be useful as it forces the model finetuned on C∗ to not always blindly follow the results of each call it makes.

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.

06 Limitations & Conclusion

Limitations

- Chain으로 동작하진 않음 (A api input -> B api input (X))
- Sample-inefficient
  - 수백만개의 문서를 처리해도 계산기API에 적절한 샘플은 천개만 생성됨
- Toolformer가 API 부를지 말지 결정할때 tool의 computational cost등을 고려하지않음

Conclusion

- Future token의 ppl을 감소시키는 api calls로 만든 샘플로 모델 파인튜닝함
- 6.7B GPT-J 기준 제로샷 퍼포먼스 크게 개선됨

Appendix

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

- 제로샷이지만 프롬프트줌
  - we use the following prompt: `Please complete the following text so that it is factually correct: x.`

Thank You!

Do you have any questions?