

Gry, notatki, łańcuchy myśli, narzędzia i inne

Paweł Rychlikowski

Instytut Informatyki UWr

21 stycznia 2026

Początek

Zacznijmy od dwóch przypadków użycia treningu małych modeli językowych

- Przypadek 1: Connect 4
- Przypadek 2: Muzyka (irlandzka, korpus Irishman)

Connect Four



- Fajny wariant kółka i krzyżyka
- W większości sytuacji 7 możliwych ruchów
- Łatwo stworzyć agenta, który gra sensownie, a jednocześnie niedeterministycznie.
- Taki agent może nam wygenerować partie, jest też naturalnym przeciwnikiem.

Agent-0

- Minimax na poziomie 5, bez heurystyki (tylko zwycięstwa i przegrane, reszta stanów ma wartość 0)
- Remisy rozwiązuje prostymi symulacjami Monte Carlo
- Niepowtarzające się partie, sensowna jakość

Agent-0

- Minimax na poziomie 5, bez heurystyki (tylko zwycięstwa i przegrane, reszta stanów ma wartość 0)
- Remisy rozwiązuje prostymi symulacjami Monte Carlo
- Niepowtarzające się partie, sensowna jakość

Agent0 wygenerował 20K rozgrywek (nawiasem mówiąc użytych na przedmiocie sieci neuronowe)

Agent-0

- Minimax na poziomie 5, bez heurystyki (tylko zwycięstwa i przegrane, reszta stanów ma wartość 0)
- Remisy rozwiązuje prostymi symulacjami Monte Carlo
- Niepowtarzające się partie, sensowna jakość

Agent0 wygenerował 20K rozgrywek (nawiasem mówiąc użytych na przedmiocie sieci neuronowe)

Jakość rozgrywek

- Agent-0 vs Random: **(500-0-0)**

Korpus C4

Tokeny == znaki

S11326354624002233001B

S003001141011222434423321602345355546666655D

S1015114330400505542233641446331665560622B

S3430400600143364144122131130666622B

S323352465224555134411136200A

S514323233215121142245354435500104000B

Korpus C4

Tokeny == znaki

S11326354624002233001B

S003001141011222434423321602345355546666655D

S1015114330400505542233641446331665560622B

S3430400600143364144122131130666622B

S323352465224555134411136200A

S514323233215121142245354435500104000B

Zadanie dla modelu językowego

LM traktuje to jako ciąg symboli (11-elementowy alfabet). Podczas treningu musi

- nauczyć się reguł,
- nauczyć się warunków końcowych,
- nauczyć się rozsądnie grać.

Korpus C4

Tokeny == znaki

S11326354624002233001B

S003001141011222434423321602345355546666655D

S1015114330400505542233641446331665560622B

S3430400600143364144122131130666622B

S323352465224555134411136200A

S514323233215121142245354435500104000B

Zadanie dla modelu językowego

LM traktuje to jako ciąg symboli (11-elementowy alfabet). Podczas treningu musi

- nauczyć się reguł,
- nauczyć się warunków końcowych,
- nauczyć się rozsądnie grać.

Ustawienia nano-GPT z Szekspira



Wyniki dla Agenta-LM1

- Dokładność przewidywania stanów końcowych: około 50% (czyli skuteczność odpowiedzi na pytanie czy gra się skończyła i kto wygrał)
- Wyniki pojedynku Agenta0 and Agenta-LM1: **(887-90-23)** (około 10% zwycięstw)

Wyniki dla Agenta-LM1

- Dokładność przewidywania stanów końcowych: około 50% (czyli skuteczność odpowiedzi na pytanie czy gra się skończyła i kto wygrał)
- Wyniki pojedynku Agenta0 and Agenta-LM1: **(887-90-23)** (około 10% zwycięstw)

Komentarz

- Pokonanie swojego mistrza można uważać za sukces (nawet jeżeli tylko w 10% gier)
- Przewidywanie ruchów lepsze niż księga otwarć (46% vs 30%)
- Topline dla przewidywania ruchów jest 57.6% (bo rozgrywki mają sporo elementu losowości)

Korpus C4-2C

Poprzednia wersja

S11326354624002233001B

S323352465224555134411136200A

W C4-2C używamy innych symboli dla obu graczy

S11326354624002233001B

S323352465224555134411136200A

Wyniki Agenta-LM2

- Dokładność przewidywania stanów końcowych: około 75%
- Pojedynek Agent0 and Agent-LM2: **(844-134-22)**
- Losowy gracz kontra Agent-LM2: **(8-992-0)**

Wyniki Agenta-LM2

- Dokładność przewidywania stanów końcowych: około 75%
- Pojedynek Agent0 and Agent-LM2: **(844-134-22)**
- Losowy gracz kontra Agent-LM2: **(8-992-0)**

Komentarz

- Więcej informacji w tokenie == lepsza gra

Wyniki Agenta-LM2

- Dokładność przewidywania stanów końcowych: około 75%
- Pojedynek Agent0 and Agent-LM2: **(844-134-22)**
- Losowy gracz kontra Agent-LM2: **(8-992-0)**

Komentarz

- Więcej informacji w tokenie == lepsza gra
- Czy możemy upakować jeszcze więcej informacji w tokenie?

Mała wycieczka do świata szachów

Praca magisterska p. Kamila Michalaka proponowała taką tokenizację ruchów (która dała przyzwoitą jakość rozgrywki)

Type	Examples	Count
King moves	Ka1a2, Kf8g7	420
Queen moves	Qd1h6, Qd8e7	1456
Rook moves	Ra1b1, Re8e1	896
Bishop moves	Bf1g5, Bg7b2	560
Knight moves	Ng1f3, Nb7c6	336
Pawn moves	Pe2e4, Pb4c3	236
Castles	Ke1g1, Ke8c8	4
Pawn promotions	Pe7e8q, Pa2b1n	176
Special symbols	[EOS], [BOS], [PAD]	3
Total:		4087

Korpus C4-XY

Korpus C4-XY

- Ruch zawiera informację: player, x, y
- Rozmiar słownika: $42 \times 2 + 4$

Korpus C4-XY

- Ruch zawiera informację: player, x, y
- Rozmiar słownika: $42 \times 2 + 4$

Dodatkowa informacja

Testowane modele mają około 10.5M parametrów (drobne różnice wynikają z wielkości tablicy osadzeń)

Agent-LM3 results

- Wyniki meczów pomiędzy Agentem0 and Agentem-LM3:
(583-349-68)
- Wyniki meczów pomiędzy agentem losowym a Agentem-LM3:
(3-997-0)

Agent-LM3 results

- Wyniki meczów pomiędzy Agentem0 and Agentem-LM3:
(583-349-68)
- Wyniki meczów pomiędzy agentem losowym a Agentem-LM3:
(3-997-0)

Uwaga

Jak starczy czasu, to opowiemy sobie o śmiesznej idei **notatek** w kontekście gry C4.

Zimowa zagadka muzyczna (obrazek 1)

Uwaga

Do prawidłowego rozwiązania zagadki należy obejrzeć oba obrazki

Zimowa zagadka muzyczna (obrazek 1)

Uwaga

Do prawidłowego rozwiązania zagadki należy obejrzeć oba obrazki



Zimowa zagadka muzyczna (obrazek 2)

Zimowa zagadka muzyczna (obrazek 2)



Rozwiążanie zagadki

- Pierwszy obrazek jest wyciętym fragmentem oficjalnego teledysku piosenki Go West, zespołu Pet Shop Boys

Rozwiążanie zagadki

- Pierwszy obrazek jest wyciętym fragmentem oficjalnego teledysku piosenki Go West, zespołu Pet Shop Boys
- której melodia jest często śpiewana/grana na stadionach z tekstem:
Polskaaa, biało czeerwonii

Rozwiążanie zagadki

- Pierwszy obrazek jest wyciętym fragmentem oficjalnego teledysku piosenki Go West, zespołu Pet Shop Boys
- której melodia jest często śpiewana/grana na stadionach z tekstem:
Polskaaa, biało czeerwonii

Potraktujemy pierwsze nuty tej piosenki jako prefiks i zobaczymy, jaką kontynuację wymyśli (niewielki) model typu GPT.

Go West

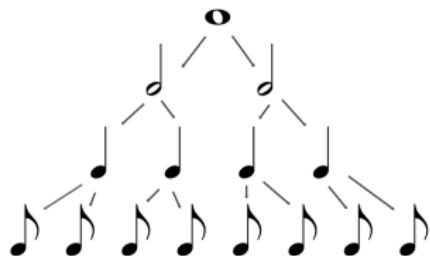
Pet Shop Boys



- Powyżej widzimy zapis (lejko zmodyfikowanej) linii melodycznej piosenki Go West, z którego utworzymy nasz prefiks
- (tak przy okazji, muzykę zapisuje się na tzw. pięciolinny, a te kóleczka z kreskami to nuty)

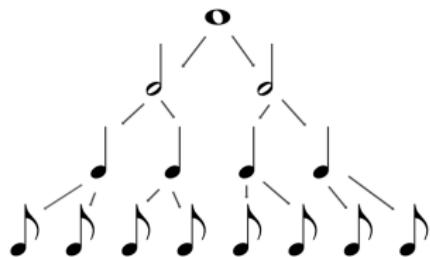
Muzyka: Lekcja 1

Nuty trwają jakiś czas (dość równy)



Muzyka: Lekcja 1

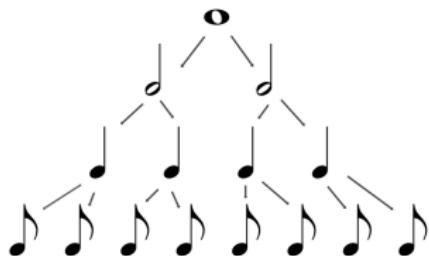
Nuty trwają jakiś czas (dość równy)



- ćwierćnuta to jedno podstawowe uderzenie perkusji

Muzyka: Lekcja 1

Nuty trwają jakiś czas (dość równy)



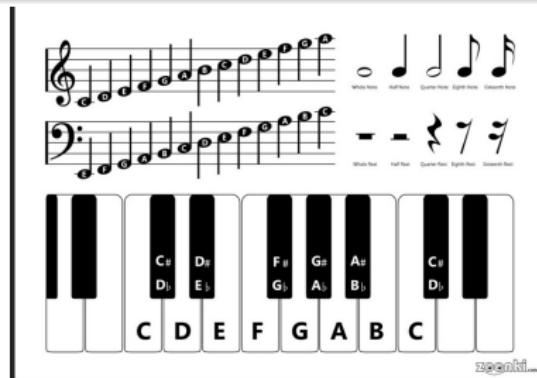
- ćwierćnuta to jedno podstawowe uderzenie perkusji
- Kropka przy nucie wydłuża jej czas o 50%

Muzyka: Lekcja 2

Nuty mają **wysokość** (zależną od częstotliwości)

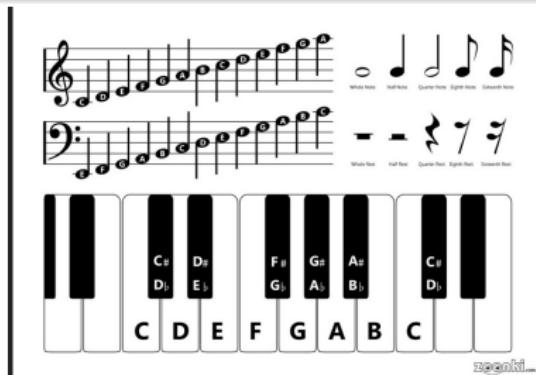
Muzyka: Lekcja 2

Nuty mają **wysokość** (zależną od częstotliwości)



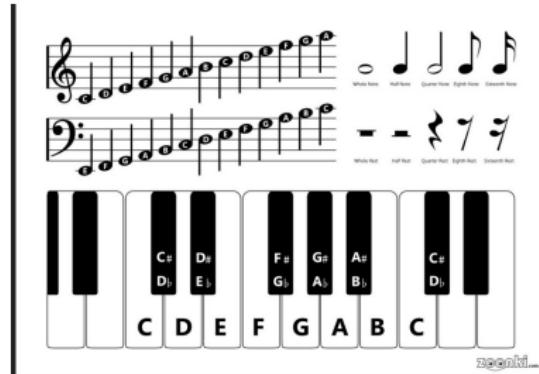
Muzyka: Lekcja 2

Nuty mają **wysokość** (zależną od częstotliwości)



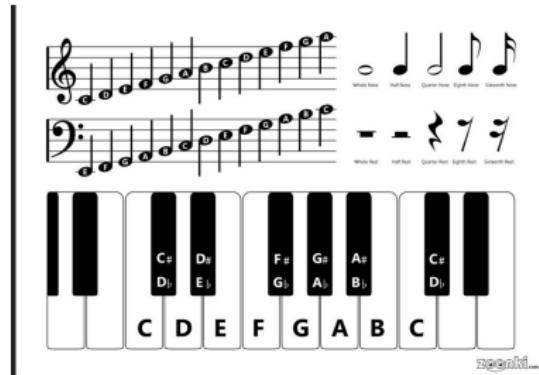
- Wysokość zaznaczamy wybierając odpowiednią pozycję na pięciolinie
- Można ją skonwertować na liczbę całkowitą (numer klawisza, od lewej do prawej, licząc czarne i białe)

Muzyka: Lekcja 2 (cd)



- Gdy zwiększamy wysokość o 1, to częstotliwość mnożymy przez $\sqrt[12]{n}$

Muzyka: Lekcja 2 (cd)



- Gdy zwiększamy wysokość o 1, to częstotliwość mnożymy przez $\sqrt[12]{n}$
- Kwestia tonacji również pominiemy (choć łatwo ją dodać)

Formaty muzyczne

- Istnieje dużo formatów muzycznych
- Prostym formatem zapisującym muzykę jest format **abc**
- Uniwersalnym formatem zapisującym muzykę jest format **midi** (który umożliwia zapis utworów wieloinstrumentowych)

Format ABC

Format abc jest na prosty, można wytrenować transformer na znakach i powinien generować poprawne składniowo utwory (pomysł na miniprojekt)

```
<score lang="ABC">
X:1
T:The Legacy Jig
M:6/8
L:1/8
R:jig
K:G
GFG BAB | gfg gab | GFG BAB | d2A AFD |
GFG BAB | gfg gab | age edB |1 dBA AFD :|2 dBA ABd |:
efe edB | dBA ABd | efe edB | gdB ABd |
efe edB | d2d def | gfe edB |1 dBA ABd :|2 dBA AFD |]
</score>
```

Nasz korpus (wersja korpusu IrishMan)

- **43K** irlandzkich piosenek folkowych
- **3.5M** nutek, **490** różnych tokenów
- Kreski taktowe dodane (może ułatwiają?)
- (btw. Szekspir napisał koło miliona znaków w swoich dramatach)

1 wiersz z korpusu

```
A4_0.5 F#4_0.5 F#4_1.0 A4_0.5 F#4_0.5 D5_0.5 B4_0.5 | A4_0.5 F#4_0.5 F#4_1.0 G4_1.5 B4_0.5 |
A4_0.5 F#4_0.5 F#4_1.0 A4_0.5 D5_0.5 D5_1.0 | E5_0.5 G5_0.5 F#5_0.5 E5_0.5 F#5_0.5 D5_0.5 D5_0.5 B4_0.5 |
D5_1.5 F#5_0.5 A5_1.0 A5_0.5 F#5_0.5 | E5_0.5 A4_0.5 C#5_1/3 B4_1/3 A4_1/3 E5_0.5 A4_0.5 C#5_1/3 B4_1/3 A4_1/
D5_1.5 F#5_0.5 A5_1.0 A5_0.5 G5_0.5 | F#5_0.5 A5_0.5 E5_0.5 G5_0.5 F#5_0.5 D5_0.5 D5_0.5 A4_0.5 |
D5_1.5 F#5_0.5 A5_1.0 A5_0.5 F#5_0.5 | E5_0.5 A4_0.5 C#5_1/3 B4_1/3 A4_1/3 E5_0.5 A4_0.5 C#5_1/3 B4_1/3 A4_1/
F#5_1.5 F#5_0.5 G5_1.0 F#5_0.5 G5_0.5 | A5_0.5 F#5_0.5 G5_0.5 E5_0.5 F#5_0.5 D5_0.5 D5_1.0 |
```

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

Toolformer: zestaw narzędzi

① Kalkulator – oblicza wyrażenia

Toolformer: zestaw narzędzi

- ① Kalkulator – oblicza wyrażenia
- ② System odpowiadania na pytania w języku naturalnym (wykorzystujący bazy wiedzy)

Toolformer: zestaw narzędzi

- ① Kalkulator – oblicza wyrażenia
- ② System odpowiadania na pytania w języku naturalnym (wykorzystujący bazy wiedzy)
- ③ Prosta wyszukiwarka do Wikipedii

Toolformer: zestaw narzędzi

- ① Kalkulator – oblicza wyrażenia
- ② System odpowiadania na pytania w języku naturalnym (wykorzystujący bazy wiedzy)
- ③ Prosta wyszukiwarka do Wikipedii
- ④ Kalendarz (chyba nie całkiem potrzebny)

Toolformer: zestaw narzędzi

- ① Kalkulator – oblicza wyrażenia
- ② System odpowiadania na pytania w języku naturalnym (wykorzystujący bazy wiedzy)
- ③ Prosta wyszukiwarka do Wikipedii
- ④ Kalendarz (chyba nie całkiem potrzebny)
- ⑤ Słownik wielojęzyczny

Toolformer: zestaw narzędzi

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.

- Te kolorowe wstawki zostają umieszczone automatycznie w korpusie.
- Składają się z wywołania API, strzałki i wyniku zwróconego prez API
- Wszystko w nawiasach kwadratowych, żeby łatwo usunąć przed prezentowaniem użytkownikowi

Toolformer: zestaw narzędzi

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.

- Te kolorowe wstawki zostają umieszczone automatycznie w korpusie.
- Składają się z wywołania API, strzałki i wyniku zwróconego prez API
- Wszystko w nawiasach kwadratowych, żeby łatwo usunąć przed prezentowaniem użytkownikowi

Uwaga

W **ostatecznym treningu** analizując tokeny po **that** model widzi cały ten tekst (tekst oryginalny i treść w nawiasie)

Toolformer: zestaw narzędzi

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.

Toolformer: zestaw narzędzi

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.

We wszystkich przypadkach odpowiedź pomaga generować kolejne symbole

Toolformer: jak generować wywołania

Jakieś pomysły?

Toolformer: jak generować wywołania

Jakieś pomysły?

- Tradycyjny schemat few-shot learning
- Dla każdego API trzeba przygotowujemy szablon

Toolformer: jak generować wywołania

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:

Toolformer: wzbogacanie korpusu

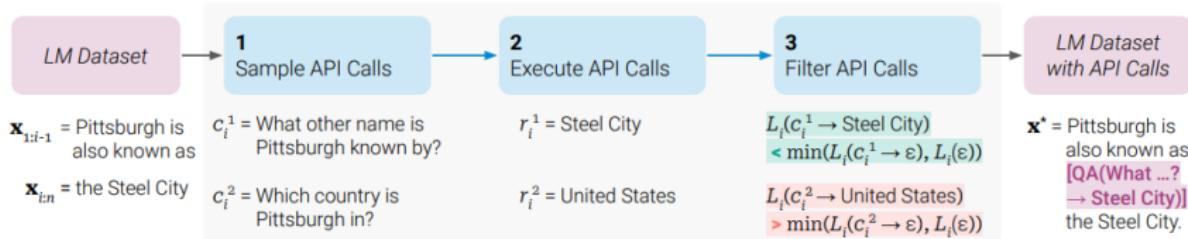


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text \mathbf{x} , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i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

- Zakładamy, że między *known as* a *the Steel* będzie API-call (skąd to wiemy – na początku losujemy, może z jakimiś heurystykami)
- Możemy też wziąć zdanie i wygenerować wywołania (jak na poprzednim slajdzie)

Toolformer: wzbogacanie korpusu

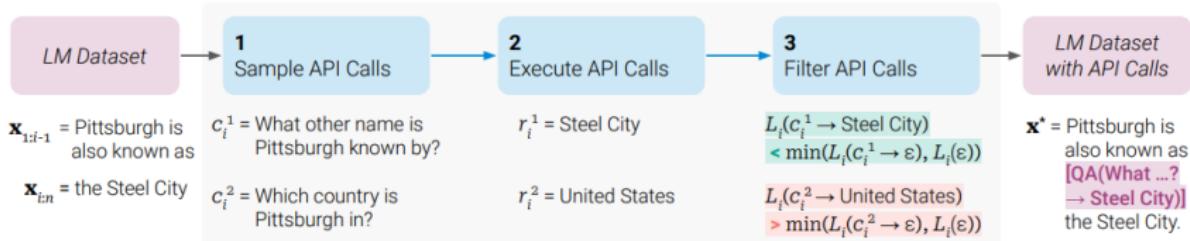


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text x , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the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

- W danym miejscu generujemy, za pomocą (innego?) LLM-a wywołania do API (punkt 1)

Toolformer: wzbogacanie korpusu

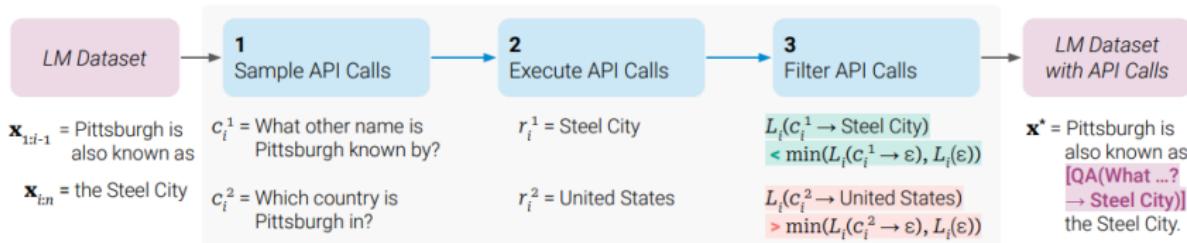


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text x , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the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

- W danym miejscu generujemy, za pomocą (innego?) LLM-a wywołania do API (punkt 1)
- ... które następnie wykonujemy (punkt 2)

Toolformer: wzbogacanie korpusu

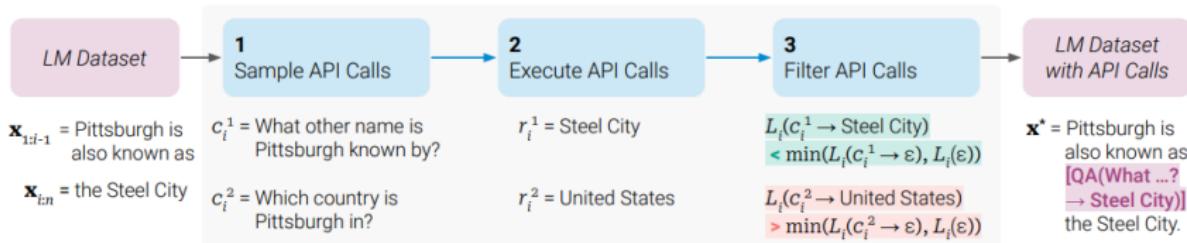


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text x , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the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

- W danym miejscu generujemy, za pomocą (innego?) LLM-a wywołania do API (punkt 1)
- ... które następnie wykonujemy (punkt 2)

Pozostała decyzja, który (i czy jakiś) wybrać.

Toolformer: wzbogacanie korpusu

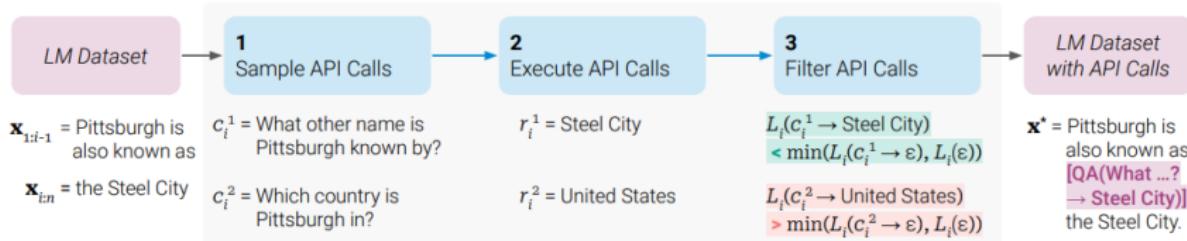


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text x , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the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

- Wklejamy (punkt 3) **przed** naszym zdaniem/akapitem:
 - ▶ albo **nic**
 - ▶ albo **samo pytanie**
 - ▶ albo **pytanie z odpowiedzią**

Toolformer: wzbogacanie korpusu

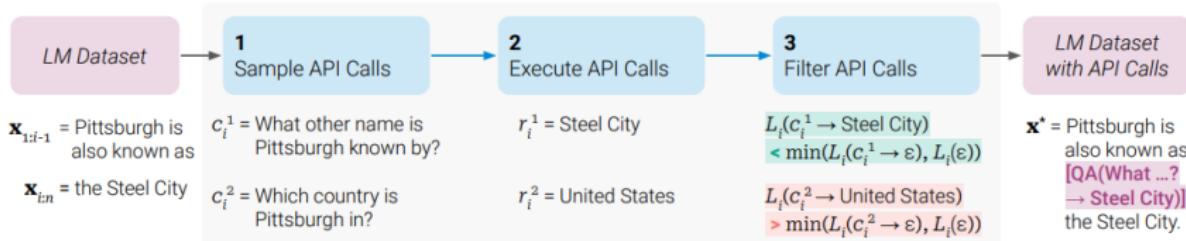


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text x , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the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

- Wklejamy (punkt 3) **przed** naszym zdaniem/akapitem:
 - ▶ albo **nic**
 - ▶ albo **samo pytanie**
 - ▶ albo **pytanie z odpowiedzią**
- Liczymy koszt wygenerowania kontynuacji w każdym wariantów

Toolformer: wzbogacanie korpusu

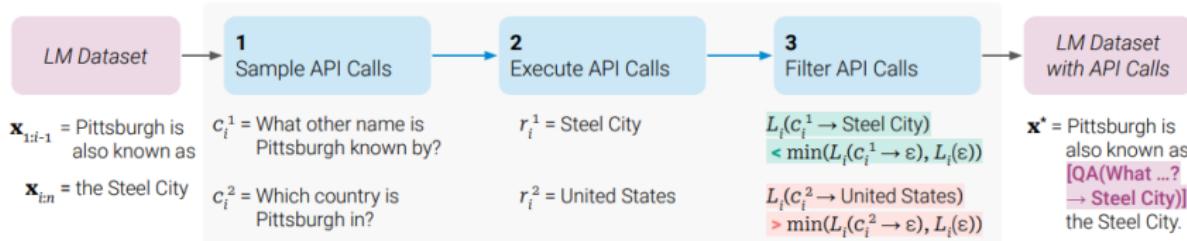


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text x , we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the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

- Wklejamy (punkt 3) **przed** naszym zdaniem/akapitem:
 - ▶ albo **nic**
 - ▶ albo **samo pytanie**
 - ▶ albo **pytanie z odpowiedzią**
- Liczymy koszt wygenerowania kontynuacji w każdym wariantów
- Jeżeli trzeci jest najniższy, to znaczy, że to dobre miejsce do wywołania API

Toolformer: inferencja



- Mamy model, który generuje teksty z „kolorowymi” zapytaniami.
- Jak wygeneruje znacznik API to generujemy dalej aż do wygenerowania strzałki i następnie wywołujemy API z odpowiednim argumentem.

Toolformer: inferencja



- Mamy model, który generuje teksty z „kolorowymi” zapytaniami.
- Jak wygeneruje znacznik API to generujemy dalej aż do wygenerowania strzałki i następnie wywołujemy API z odpowiednim argumentem.
- ... a wynik wklejamy do tekstu (nie musimy go generować, choć model by chciał :)

Toolformer: inferencja



- Mamy model, który generuje teksty z „kolorowymi” zapytaniami.
- Jak wygeneruje znacznik API to generujemy dalej aż do wygenerowania strzałki i następnie wywołujemy API z odpowiednim argumentem.
- ... a wynik wklejamy do tekstu (nie musimy go generować, choć model by chciał :)

Zwróćmy uwagę, że teraz generują się **mądrzejsze** wywołania API niż podczas treningu, bo

Toolformer: inferencja



- Mamy model, który generuje teksty z „kolorowymi” zapytaniami.
- Jak wygeneruje znacznik API to generujemy dalej aż do wygenerowania strzałki i następnie wywołujemy API z odpowiednim argumentem.
- ... a wynik wklejamy do tekstu (nie musimy go generować, choć model by chciał :)

Zwróćmy uwagę, że teraz generują się **mądrzejsze** wywołania API niż podczas treningu, bo w korpusie zostały tylko te użyteczne.

Transformery wnioskujące

Rozpoczynamy nowy wątek, związany z możliwościami wnioskowania przeprowadzonymi przez transformery

Transformery wnioskujące

Rozpoczynamy nowy wątek, związany z możliwościami wnioskowania przeprowadzonymi przez transformery

Quasi-rozumowanie na poziomie GPT-2

Suppose I have 4 apples, and I gave two of them to my sister,

Transformery wnioskujące

Rozpoczynamy nowy wątek, związany z możliwościami wnioskowania przeprowadzonymi przez transformery

Quasi-rozumowanie na poziomie GPT-2

Suppose I have 4 apples, and I gave two of them to my sister, and she gave one of them to her sister, and it started to grow. Would I be able to see how much time it had taken for the apple to grow?

We will first give you a number which you will later use to work out how much time it has taken for the apple to grow:

(Time it took to grow 2 apples, shown as a clock) \times (Size of apple) = "time taken"

What this tells us is that:

$$2 \times 3 \times 2 = 44 \text{ (seconds in 24 hours)}$$

So the question is, what does 44 seconds look like?

Let's do a quick experimen

Transformery wnioskujące

Oczywiście GPT-3 i dalsze modele są sporo lepsze, ale daleko do doskonałości!

Transformery wnioskujące

Oczywiście GPT-3 i dalsze modele są sporo lepsze, ale daleko do doskonałości!

Uwaga

Jeden źle wygenerowany token może rozwalić całe rozumowanie!

Transformery wnioskujące

Oczywiście GPT-3 i dalsze modele są sporo lepsze, ale daleko do doskonałości!

Uwaga

Jeden źle wygenerowany token może rozwalić całe rozumowanie!

Przyjrzymy się na kolejnych slajdach bardzo „płodnej” idei **łańcucha myśli**, która ułatwia/umożliwia rozumowanie w modelach autoregresywnych (takich jak GPT).

Chain of Thought (CoT)

(a) Few-shot

Examples

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. X

Chain of Thought (CoT)

Examples

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. ✗

CoT Examples

(b) Few-shot-CoT (Wei et al., 2022)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Step-by-step Answer

Chain of Thought (CoT)

Examples

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. ✗

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 ✗

b) Few-shot-CoT (Wei et al., 2022)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

CoT Examples

Step-by-step Answer

Chain of Thought (CoT)

Examples

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The answer is 8. ✗

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

b) Few-shot-CoT (Wei et al., 2022)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

CoT Examples

Step-by-step Answer

(d) Zero-shot-CoT (Kojima et al., 2022)

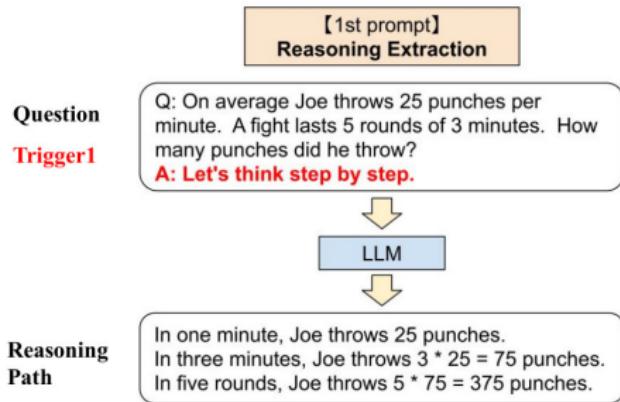
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: Let's think step by step.

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Two-stage Prompting Step-by-step Answer

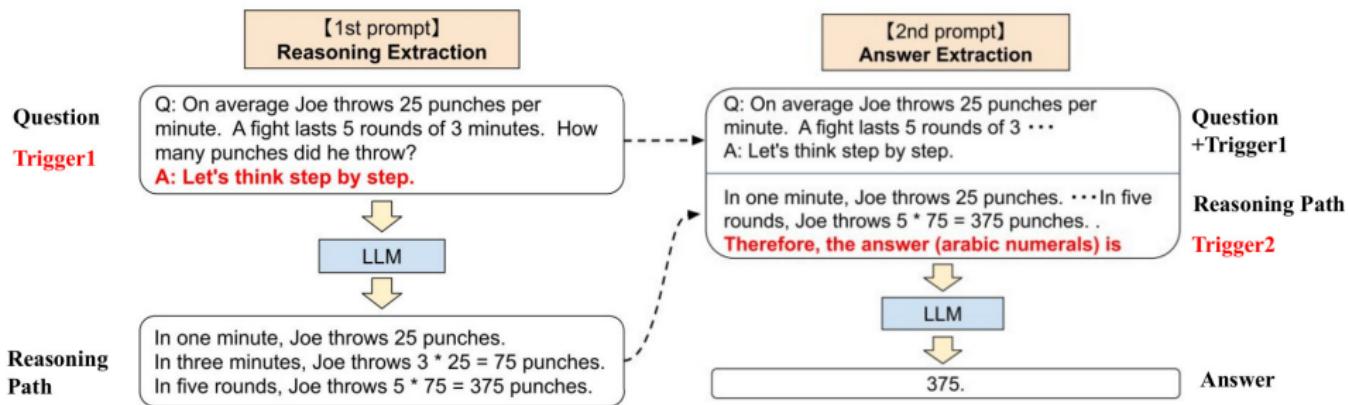
Zero-Shot Chain of Thought (CoT)

For **zero-shot CoT**, a **two-stage prompting** is applied:



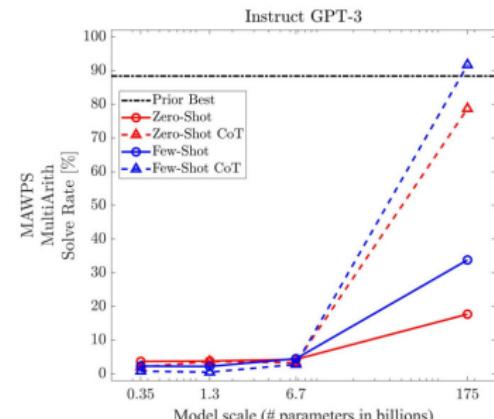
Zero-Shot Chain of Thought (CoT)

For **zero-shot CoT**, a **two-stage prompting** is applied:



Arithmetic Reasoning - Observations

- Both zero-shot and few-shot chain of thought promptings are emergent **abilities of model scale**.
- They do not positively impact performance for small models, but start to yield performance gains when used with models with more than ~100B parameters.
- Few-shot CoT achieves **better** performance on LLM than zero-shot CoT.
- Instruct GPT-3: text-davinci-002 achieves similar performance as PaLM 540B model



Ablation Study - Variations of Few-Shot CoT

Change the types of CoT:

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Equation only



5+6=11. The answer is 11.

Ablation Study - Variations of Few-Shot CoT

Change the types of CoT:

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

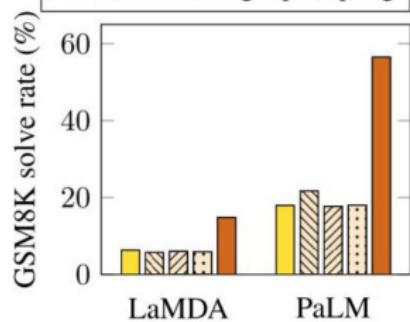
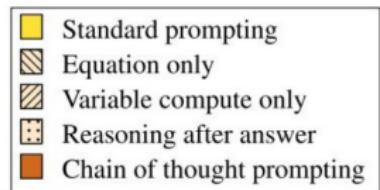
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Equation only



$5+6=11$. The answer is 11.



Natural language in reasoning matters.

(Wei et al., 2022)

71

Ablation Study - Variations of Few-Shot CoT

Change the types of CoT:

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Variable compute only



.....
..... *The
answer is 11.*

Ablation Study - Variations of Few-Shot CoT

Change the types of CoT:

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

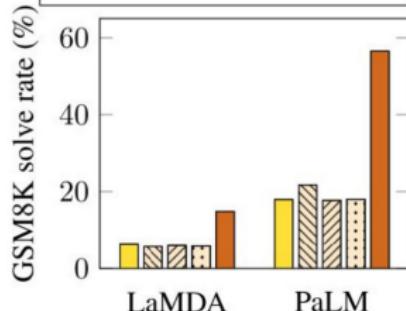
(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Variable compute only



.....
..... *The
answer is 11.*

- Standard prompting
- Equation only
- Variable compute only
- Reasoning after answer
- Chain of thought prompting



More intermediate computation does not help with the final answer.

(Wei et al., 2022)

Ablation Study - Variations of Few-Shot CoT

Change the types of CoT:

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Reasoning after answer



*The answer is 11.
Roger started with
5 balls. 2 cans of 3
tennis balls each is
6 tennis balls. $5 +$
 $6 = 11$.*

Ablation Study - Variations of Few-Shot CoT

Change the types of CoT:

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

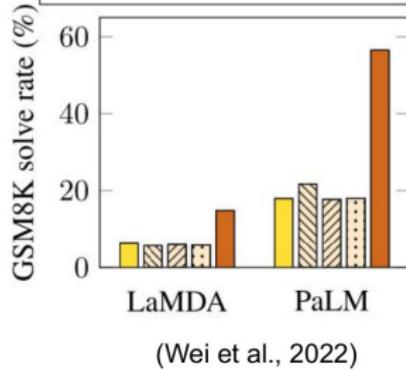
(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

Reasoning after answer



The answer is 11.
Roger started with
5 balls. 2 cans of 3
tennis balls each is
6 tennis balls. $5 +$
 $6 = 11$.

- █ Standard prompting
- █ Equation only
- █ Variable compute only
- █ Reasoning after answer
- █ Chain of thought prompting



CoT is not just activating knowledge seen in pre-training.

(Wei et al., 2022)

Jak się uda, to posłuchamy

Transformery wnioskujące. Programiści

- W koncepcji brudnopisów LM de facto wykonywał pewien program.

Transformery wnioskujące. Programiści

- W koncepcji brudnopisów LM de facto wykonywał pewien program.
- Czy znamy lepsze narzędzie do wykonywania programów?

Transformery wnioskujące. Programiści

- W koncepcji brudnopisów LM de facto wykonywał pewien program.
- Czy znamy lepsze narzędzie do wykonywania programów?

Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks

^{§, *}Wenhu Chen,[§]Xueguang Ma*, [†]Xinyi Wang, [◦]William W. Cohen

[§]University of Waterloo

^{*}Vector Institute, Toronto

[†]University of California, Santa Barbara

[◦]Google Research

{wenhuchen,x93ma}@uwaterloo.ca, xinyi_wang@ucsb.edu, wcohen@google.com

Reviewed on OpenReview: <https://openreview.net/forum?id=YfZ4ZPt8zd>

Abstract

Recently, there has been significant progress in teaching language models to perform step-by-step reasoning to solve complex numerical reasoning tasks. Chain-of-thoughts prompting (CoT) is the state-of-art method for many of these tasks. CoT uses language models to produce text describing reasoning, and computation, and finally the answer to a question. Here we propose ‘Program of Thoughts’ (PoT), which uses language models (mainly Codex) to generate text and programming language statements, and finally an answer. In PoT, the computation can be delegated to a program interpreter, which is used to execute the generated program, thus decoupling complex computation from reasoning and language understanding. We evaluate PoT on five math word problem datasets and three financial-QA datasets in both few-shot and zero-shot settings. We find that PoT has an average performance gain over CoT of around 12% across all datasets. By combining PoT with self-consistency decoding, we can achieve extremely strong performance on all the math datasets and financial datasets. All of our data and code will be released.

Program-of-thoughts. Ogólna idea

- Model językowy pisze program rozwiązujejący dane zagadnienie

Program-of-thoughts. Ogólna idea

- Model językowy pisze program rozwiązujący dane zagadnienie
- Program jest wykonywany przez interpreter Pythona

Program-of-thoughts. Ogólna idea

- Model językowy pisze program rozwiązujący dane zagadnienie
- Program jest wykonywany przez interpreter Pythona
- Wynik programu (być może zredagowany przez inaczej promptowany model) jest odpowiedzią

Program-of-thoughts. Ogólne idea

Question: In Fibonacci sequence, it follows the rule that each number is equal to the sum of the preceding two numbers. Assuming the first two numbers are 0 and 1, what is the 50th number in Fibonacci sequence?

The first number is 0, the second number is 1, therefore, the third number is $0+1=1$. The fourth number is $1+1=2$. The fifth number is $1+2=3$. The sixth number is $2+3=5$. The seventh number is $3+5=8$. The eighth number is $5+8=13$.
..... (Skip 1000 tokens)

The 50th number is 32,432,268,459.

CoT

32,432,268,459



```
length_of_fibonacci_sequence = 50  
fibonacci_sequence = np.zeros(length_of_ )  
fibonacci_sequence[0] = 0  
fibonacci_sequence[1] = 1  
For i in range(3, length_of_fibonacci_sequence):  
    fibonacci_sequence[i] = fibonacci_sequence[i-1] +  
        fibonacci_sequence[i-2]  
ans = fibonacci_sequence[-1]
```

PoT



12,586,269,025



Question: Ketty saves 20000 dollars to the bank. After three years, the sum with compound interest rate is 1000 dollars more than the sum with simple interest rate. What is the interest rate of the bank?

Assuming the interest rate is x . The sum after two years with simple interest rate is $20000 + x * 20000 * 3 = 20000 + 60000x$. The sum after two years with compound interest rate is $20000 * (1 + x)^3 = 200000 + 60000 * x + 60000x^2 + 20000x^3$. The difference can be written as $60000x^2 + 20000x^3 = 1000$. In order to solve x , we can use the quadratic formula. $x = (-b \pm \sqrt{b^2 - 4ac}) / 2a$, ..., $x = (-20000 \pm 6160) / 120000$, $x = -0.051333$.

CoT

-0.051333



```
interest_rate = Symbol('x')  
sum_in_two_years_with_simple_interest= 20000 +  
interest_rate * 20000 * 3  
sum_in_two_years_with_compound_interest = 20000 * (1 +  
interest_rate)**3  
# Since compound interest is 1000 more than simple interest.  
ans = solve(sum_after_in_yeras_with_compound_interest -  
sum_after_two_years_in_compound_interest - 1000,  
interest_rate)
```

PoT



x = 0.24814



Program-of-thoughts. Dodatkowe szczegóły

- Jako model bazowy użyto modelu **Codex** (o modelach dla kodu na ostatnim wykładzie)
- Zauważono problem: model miał tendencję do opisywania zadania w komentarzu, a nie do kodowania (leniuszek!)

Program-of-thoughts. Dodatkowe szczegóły

- Jako model *bazowy* użyto modelu **Codex** (o modelach dla kodu na ostatnim wykładzie)
- Zauważono problem: model miał tendencję do opisywania zadania w komentarzu, a nie do kodowania (leniuszek!)
- Rozwiązanie: zmniejszyć prawdopodobieństwo znaku **#**

Nowe zadanie. Game of 24



$$(5+7)\times(9-7)$$

- Cel: Stworzyć wyrażenie dokładnie z tych cyfr (małych liczb), w dowolnej kolejności, równe 24
- Wymaga kombinatorycznego sprawdzania możliwości, intuicji ogólnoarytmetycznych i związanych z tym zadaniem

CoT is not enough

- Error propagation: one incorrect step leads to cumulative errors
- Chain structure limitation: the scope of exploration is limited
- Uncertainty: greedy decoding may not lead to a great reasoning path

Q: Calculate $(2+3)*5$

A:

Calculate $2+3$, we get 6

$$6*5 = 30$$

The final answer is 30

Cumulative error

Q: Can 1, 2, 3, 4 get 24 in game 24?

A:

$$1+2 = 3$$

$$3*3 = 9$$

$$9+4 = 13$$

$$13 \neq 24$$

So 1,2,3,4 cannot get 24 in game 24.

Limited exploration

Q: What is $1+2+3+\dots+6$?

A:

$$1+2 = 3$$

$$3+3 = 6$$

$$6+4 = 10$$

$$10+5 = 15$$

$$15+6 = 21$$

So $1+2+3+4+5+6=21$.

Correct yet not good

CoT is not enough

- Error propagation: one incorrect step leads to cumulative errors
- Chain structure limitation: the scope of exploration is limited
- Uncertainty: greedy decoding may not lead to a great reasoning path

Q: Calculate $(2+3)*5$

A:

Calculate $2+3$, we get 6

$$6*5 = 30$$

The final answer is 30

Cumulative error

Q: Can 1, 2, 3, 4 get 24 in game 24?

A:

$$1+2 = 3$$

$$3*3 = 9$$

$$9+4 = 13$$

$$13 \neq 24$$

So 1,2,3,4 cannot get 24 in game 24.

Limited exploration

Q: What is $1+2+3+\dots+6$?

A:

TODO: znajdź lepszy przykład

$$1+2 = 3$$

$$3+3 = 6$$

$$6+4 = 10$$

$$10+5 = 15$$

$$15+6 = 21$$

So $1+2+3+4+5+6=21$.



Correct yet not good

Elementy przeszukiwania

- Wydaje się, że jakieś formy przeszukiwania (czyli klasycznej AI) nie unikniemy.

Elementy przeszukiwania

- Wydaje się, że jakieś formy przeszukiwania (czyli klasycznej AI) nie unikniemy.
- Pojedyncze kroki tego przeszukiwania mogą być koszmarnie drogie – mogą zakładać generację wielu tokenów przez (L)LM.

Elementy przeszukiwania

- Wydaje się, że jakieś formy przeszukiwania (czyli klasycznej AI) nie unikniemy.
- Pojedyncze kroki tego przeszukiwania mogą być koszmarnie drogie – mogą zakładać generację wielu tokenów przez (L)LM.
- Ogranicza nas to do kilku-kilkudziesięciu sprawdzonych wariantów (w porównaniu do dziesiątek tysięcy (AlphaZero), czy milionów (Stockfish)).

Elementy przeszukiwania

- Wydaje się, że jakieś formy przeszukiwania (czyli klasycznej AI) nie unikniemy.
- Pojedyncze kroki tego przeszukiwania mogą być koszmarnie drogie – mogą zakładać generację wielu tokenów przez (L)LM.
- Ogranicza nas to do kilku-kilkudziesięciu sprawdzonych wariantów (w porównaniu do dziesiątek tysięcy (AlphaZero), czy milionów (Stockfish)).
- Ale z drugiej strony jest to bliżej ludzkiej wersji przeszukiwania

A trick for COT: Self-consistency

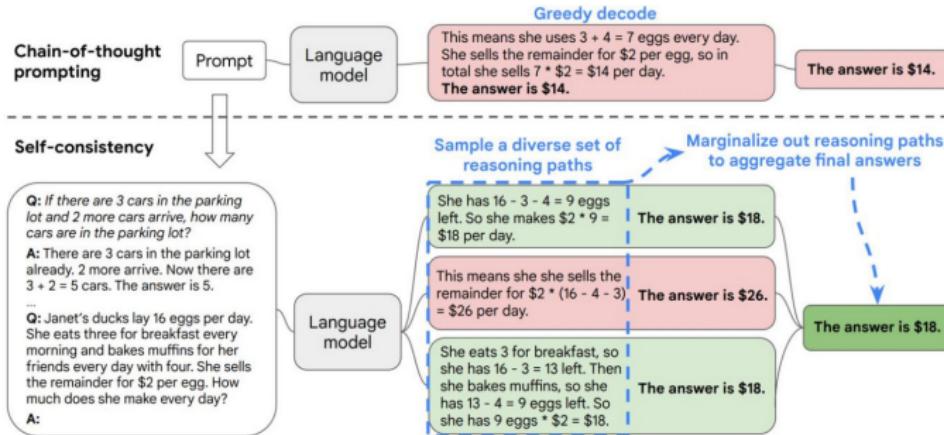


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the “greedy decode” in CoT prompting by sampling from the language model’s decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set.

Wang, Xuezhi, et al. "Self-consistency improves chain of thought reasoning in language models." arXiv preprint arXiv:2203.11171 (2022).

Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Shunyu Yao

Princeton University

Dian Yu

Google DeepMind

Jeffrey Zhao

Google DeepMind

Izhak Shafran

Google DeepMind

Thomas L. Griffiths

Princeton University

Yuan Cao

Google DeepMind

Karthik Narasimhan

Princeton University

Abstract

Language models are increasingly being deployed for general problem solving across a wide range of tasks, but are still confined to token-level, left-to-right decision-making processes during inference. This means they can fall short in tasks that require exploration, strategic lookahead, or where initial decisions play a pivotal role. To surmount these challenges, we introduce a new framework for language model inference, “Tree of Thoughts” (ToT), which generalizes over the popular “Chain of Thought” approach to prompting language models, and enables exploration over coherent units of text (“thoughts”) that serve as intermediate steps toward problem solving. ToT allows LMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices. Our experiments show that ToT significantly enhances language models’ problem-solving abilities on three novel tasks requiring non-trivial planning or search: Game of 24, Creative Writing, and Mini Crosswords. For instance, in Game of 24, while GPT-4 with chain-of-thought prompting only



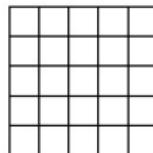
ToT. Zadania

- ① **Game of 24:** Nieoczywista manipulacja prostymi wyrażeniami arytmetycznymi (była parę slajdów wcześniej)
- ② **5x5 Crosswords:** Rozwiązywanie krzyżówek
- ③ **Creative writing:** Napisz „opowiadanie” na 4 akapity, które kończą się określonymi zdaniami.

Zadania dla ToT. Crosswords



Crosswords 5x5 - Puzzle number 30



In order to indicate an answer, you must click on the number associated with its definition.

Horizontal

- 1 To hold onto
- 2 One who snares birds with birdlime; a hanger-on
- 3 Shaded or dyed; a 3 person card game using 40 cards
- 4 To subdue; to terrify; a Central American timber tree
- 5 A limestone region marked by sinks, abrupt ridges, protuberant rocks

Vertical

- 1 A loose outer coat; a cover or pretext
- 2 A semitone
- 3 To exclude
- 4 An interjection expressing exasperation; nonsense
- 5 To salute

- Odpowiedzią jest 10 słów pięcioliterowych
- Definicje w normalnym angielskiem
- Konieczność rozwiązywania więzów związanych ze współdzieleniem literek (CSP)

Zadania dla ToT. Creative writing

Napisz spójny tekst zawierający 4 akapity, kończące się następującymi zdaniami:

Zadania dla ToT. Creative writing

Napisz spójny tekst zawierający 4 akapity, kończące się następującymi zdaniami:

Przykładowe zdania

- ① Wykonywanie stania na rękach nie jest trudne – wystarczy stanąć na rękach.
- ② Zaskoczyła go wszechogarniająca woń smażonych kotletów.
- ③ Kiedy nie spodobał jej się facet, który próbował ją poderwać, zaczęła używać języka migowego.
- ④ Każdy twój znajomy ma w głowie inny obraz twojej osoby.

Zadania dla ToT. Creative writing

Napisz spójny tekst zawierający 4 akapity, kończące się następującymi zdaniami:

Przykładowe zdania

- ① Wykonywanie stania na rękach nie jest trudne – wystarczy stanąć na rękach.
- ② Zaskoczyła go wszechogarniająca woń smażonych kotletów.
- ③ Kiedy nie spodobał jej się facet, który próbował ją poderwać, zaczęła używać języka migowego.
- ④ Każdy twój znajomy ma w głowie inny obraz twojej osoby.

Ciężko to zadanie rozwiązać bez **planu**, który jakoś łączy oderwane zdania w spójną opowieść

ToT: Ogólna idea

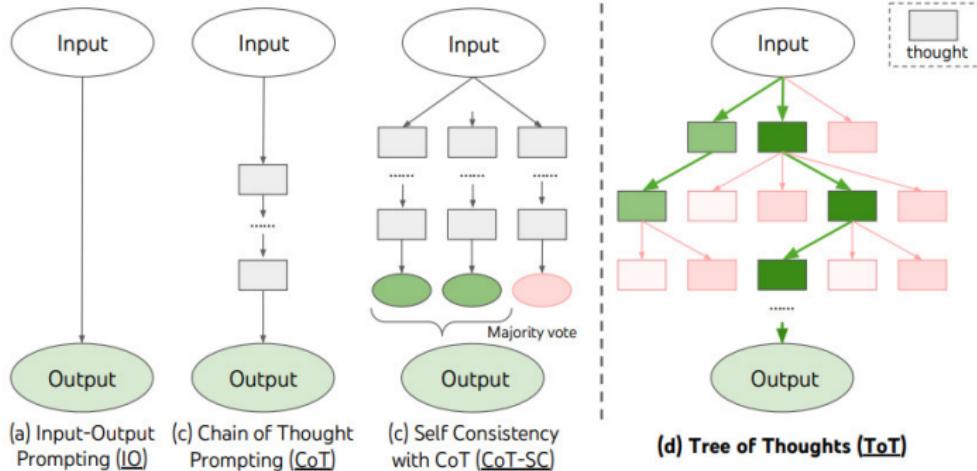


Figure 1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a *thought*, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 2,4,6.

ToT: Ogólna idea

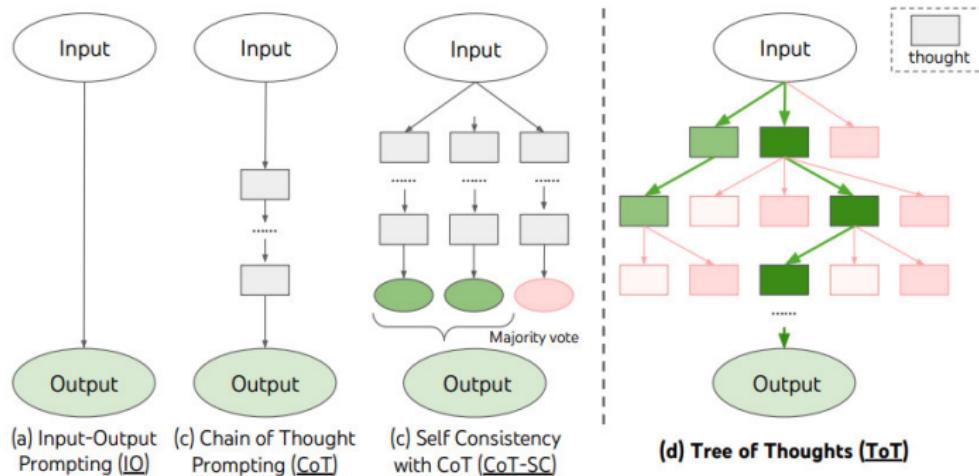


Figure 1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a *thought*, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 2,4,6.

Model sam będzie decydował, jak ma rozwijać się drzewo przeszukiwań (generował następni, decydował o zamknięciu gałęzi) – oczywiście stosując się do limitów na głębokość, szerokość i liczbę węzłów w drzewie.

ToT: Konkrety

	Game of 24	Creative Writing	5x5 Crosswords
Input	4 numbers (4 9 10 13)	4 random sentences	10 clues (h1. presented;..)
Output	An equation to reach 24 $(13-9)*(10-4)=24$	A passage of 4 paragraphs ending in the 4 sentences	5x5 letters: SHOWN; WIRRA; AVAIL; ...
Thoughts	3 intermediate equations $(13-9=4 \text{ (left } 4,4,10\text{); } 10-4=6 \text{ (left } 4,6\text{); } 4*6=24)$	A short writing plan (1. Introduce a book that connects...)	Words to fill in for clues: (h1. shown; v5. naled; ...)
#ToT steps	3	1	5-10 (variable)

Table 1: Task overview. Input, output, thought examples are in blue.

- Czym są myśli? (decyzja projektanta, przekazywana w promptcie)
- Jak generować kolejne myśli? (samplować z LM-u)

ToT: Konkrety

4.1 Game of 24

Game of 24 is a mathematical reasoning challenge, where the goal is to use 4 numbers and basic arithmetic operations (+-*-) to obtain 24. For example, given input “4 9 10 13”, a solution output could be “ $(10 - 4) * (13 - 9) = 24$ ”.

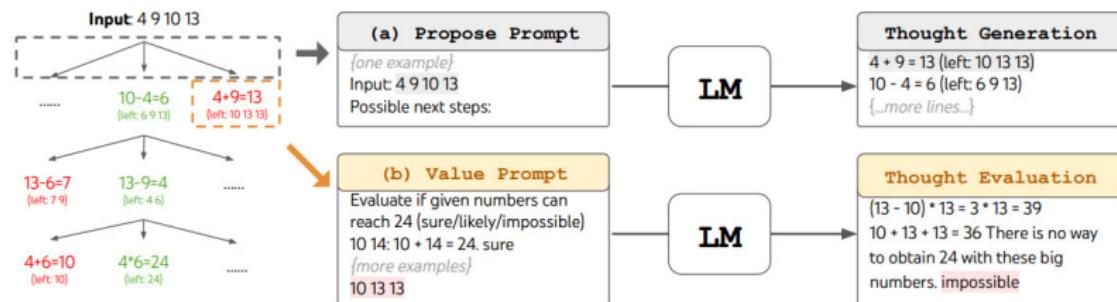


Figure 2: ToT in a game of 24. The LM is prompted for (a) thought generation and (b) valuation.

- Schemat few-shot learning
- **Propose prompt** do generacji myśli.
- **Value prompt** do generacji oceny danej myśli.

ToT: wyniki dla Game of 24

Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC ($k=100$)	9.0%
ToT (ours) ($b=1$)	45%
ToT (ours) ($b=5$)	74%
IO + Refine ($k=10$)	27%
IO (best of 100)	33%
CoT (best of 100)	49%

Table 2: Game of 24 Results.

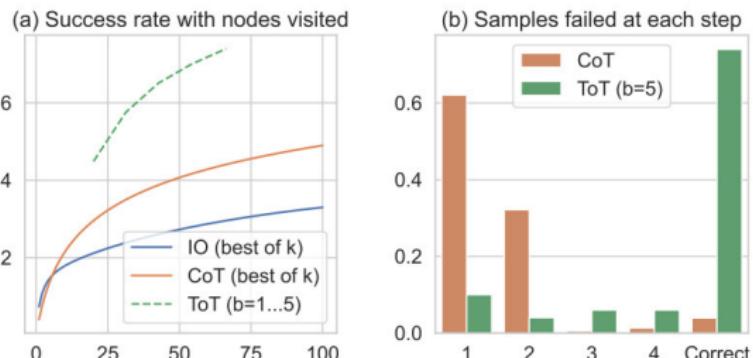
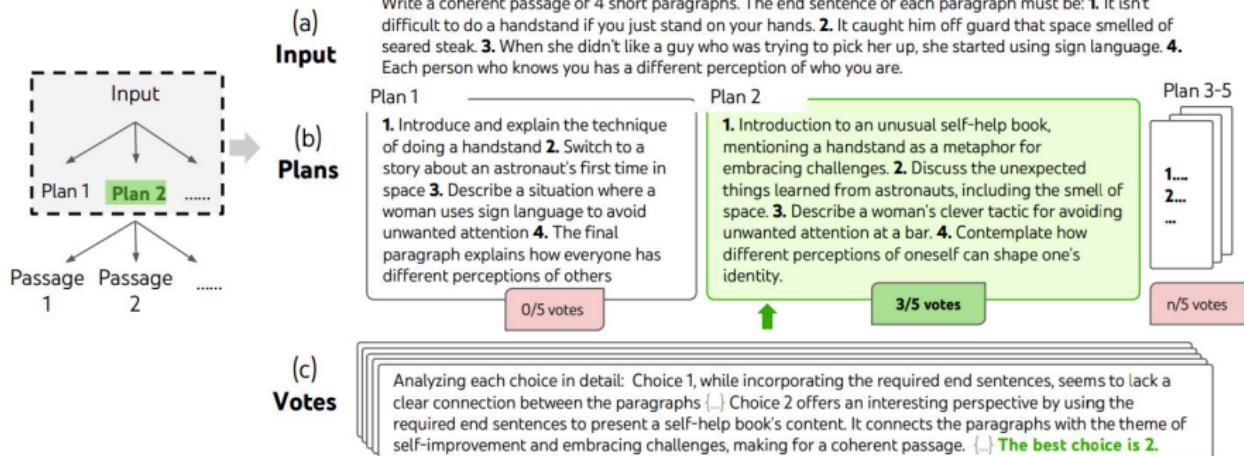


Figure 3: Game of 24 (a) scale analysis & (b) error analysis.

- k – liczba próbek przy generacji wariantów CoT
- b – liczba najlepszych stanów, którymi się zajmujemy

ToT: ewaluacja historyjek



- Model językowy (LLM) ocenia spójność tekstu!

ToT: Krzyżówki

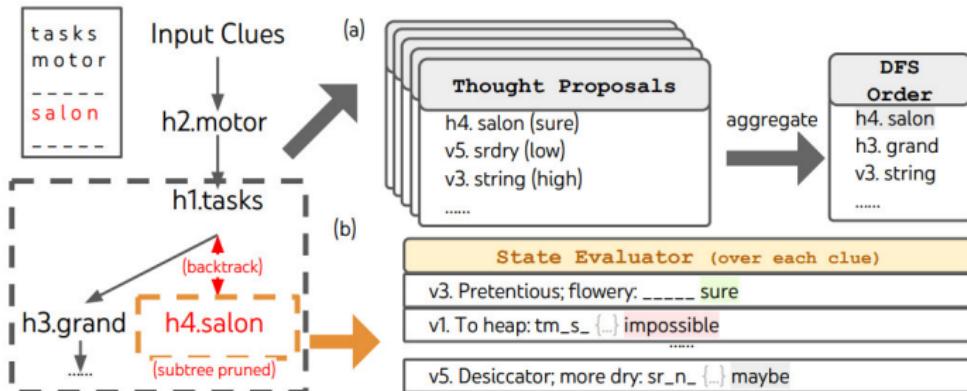


Figure 6: In Mini Crosswords, (a) how thoughts are proposed and aggregated in a priority queue for depth-first search (DFS), and (b) how a state is evaluated based on the possibility of filling in each remaining word clue, and pruned if any remaining clue is deemed not possible to fill by the LM. Then DFS backtracks to the parent state and explore the next promising thought for clue.

W zasadzie można nie omawiać dokładnie, wkleięm dla kompletności.

Learning to Reason and Memorize with Self-Notes

Jack Lanchantin*
Meta AI

Shubham Toshniwal*
NVIDIA

Jason Weston
Meta AI

Arthur Szlam
Meta AI

Sainbayar Sukhbaatar
Meta AI

Abstract

Large language models have been shown to struggle with multi-step reasoning, and do not retain previous reasoning steps for future use. We propose a simple method for solving both of these problems by allowing the model to take *Self-Notes*. Unlike recent chain-of-thought or scratchpad approaches, the model can deviate from the input context at any time to explicitly think and write down its thoughts. This allows the model to perform reasoning on the fly as it reads the context and even integrate previous reasoning steps, thus enhancing its memory with useful information and enabling multi-step reasoning. Experiments across a wide variety of tasks demonstrate that our method can outperform chain-of-thought and scratchpad methods by taking Self-Notes that interleave the input text.

Notatki – wzbogacanie wejścia

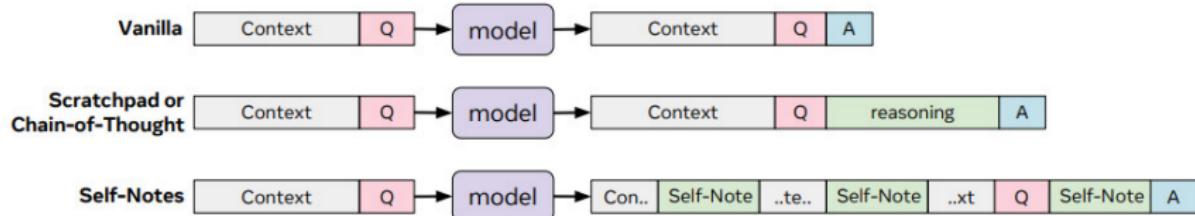


Figure 1: [top] **Vanilla** language models directly generate the answer (A) given the context and the question (Q). [middle] **Scratchpad and Chain-of-Thought** methods encourage the model to generate reasoning tokens before answering the question, but only after it has read the entire context. [bottom] **Self-Notes (ours)** allows the model to generate multiple internal reasoning notes that interleave the input context and question.

Notatki wrzucamy przetwarzając wejście token po tokenie. Możemy też je wrzucać do korpusu i robić dostrajanie.

Notatki – wzbogacanie wejścia

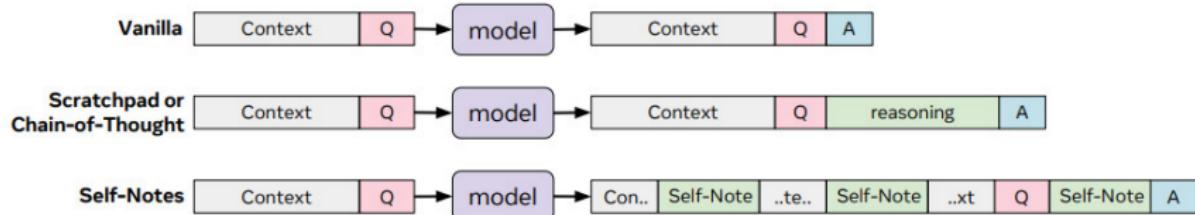


Figure 1: [top] **Vanilla** language models directly generate the answer (A) given the context and the question (Q). [middle] **Scratchpad and Chain-of-Thought** methods encourage the model to generate reasoning tokens before answering the question, but only after it has read the entire context. [bottom] **Self-Notes (ours)** allows the model to generate multiple internal reasoning notes that interleave the input context and question.

Notatki wrzucamy przetwarzając wejście token po tokenie. Możemy też je wrzucać do korpusu i robić dostrajanie.

Inna możliwa nazwa – **lematy**.

Notatki – wz bogacanie wejścia

Table 1: Input-Output pairs for the Vanilla, Scratchpad, and Self-Notes method on Toy-Story and Algorithmic tasks. The input consists of the `input context` and the `question`, the `answer` is to be generated by the model. The `highlighted` reasoning steps are not given at test time and is to be generated by the model.

Task	Vanilla	Scratchpad	Self-Notes
Toy-Story	<p>Mary has the ball. The ball is inside the box. The key is inside the box. <code>Q: Who has the key?</code> <code>Mary has the key.</code></p>	<p>Mary has the ball. The ball is inside the box. The key is inside the box. <code>Q: Who has the key?</code> <code>[Q: Who has the box?</code> <code>Mary has the box.</code> <code>Q: Who has the key?</code> <code>Mary has the key.]</code> <code>Mary has the key.</code></p>	<p>Mary has the ball. The ball is inside the box. <code>SQ: Who has the box?</code> Mary has the box. The key is inside the box. <code>SQ: Who has the key?</code> Mary has the key. <code>Q: Who has the key?</code> Mary has the key.</p>
Algorithmic	<pre>e = 3 ; e ++ ; i = 3 ; if i < e : e ++ ; print e e = 5 ;</pre>	<pre>e = 3 ; e ++ ; i = 3 ; if i < e : e ++ ; print e [e = 3 ; e ++ ; print e e = 4 ; i = 3 ; if i < e : e ++ ; print e e = 5 ;]</pre>	<pre>e = 3 ; e ++ ; print e e = 4 ; i = 3 ; if i < e : e ++ ; print e e = 5 ; print e e = 5 ;</pre>

Notatki – wzbogacanie wejścia

Table 2: Test Accuracy (in %) for the reasoning and state-tracking tasks in the supervised setup. ‘ indicates out-of-distribution harder test settings.

Task	Test Set	Vanilla	Scratchpad	Self-Notes
Toy-Story	1/2-hop	92.4 \pm 0.7	99.6 \pm 0.1	99.8 \pm 0.1
	3-hop*	57.0 \pm 0.3	96.4 \pm 0.9	98.5 \pm 0.3
	4-hop*	37.4 \pm 0.8	94.2 \pm 2.0	97.8 \pm 0.4
Algorithmic	2-100	44.6 \pm 1.0	72.2 \pm 5.7	95.5 \pm 0.2
	101-200*	24.4 \pm 2.1	11.6 \pm 2.0	85.0 \pm 0.6
Boolean Variable	3-8	99.7 \pm 0.1	100.0 \pm 0.0	100.0 \pm 0.0
	9-19*	71.3 \pm 0.8	73.7 \pm 2.4	75.2 \pm 2.1
Chess Piecetype	\leq 80	98.5 \pm 0.4	98.5 \pm 0.3	98.8 \pm 0.2
	\geq 81*	82.9 \pm 2.3	94.7 \pm 0.7	94.8 \pm 0.7
Chess Move	\leq 80	49.0 \pm 0.4	37.0 \pm 0.8	50.8 \pm 1.1
	\geq 81*	39.8 \pm 0.2	29.9 \pm 0.8	41.8 \pm 0.9

Notatki – wariant z nadzorem

	≤ 80	98.5 ± 0.4	98.5 ± 0.3	98.8 ± 0.2
Chess Piecetype	$\geq 81^*$	82.9 ± 2.3	94.7 ± 0.7	94.8 ± 0.7
	≤ 80	49.0 ± 0.4	37.0 ± 0.8	50.8 ± 1.1
Chess Move	$\geq 81^*$	39.8 ± 0.2	29.9 ± 0.8	41.8 ± 0.9

Idea

Modelowy przykład użycia to:

- Modelujemy **ciąg akcji** (na przykład ruchów w szachach)
- Notatką jest coś, co zależy od **stanu świata po akcji**

Zadanie 1: Chess Piecetype

the piecetypes can be implicitly tracked given the moves. Given a long sequence of moves, e.g. “c2 c4 e7 e5 g2 g3 b8 c6 f1 g2 g8 f6 b1 c3 f8 b4 c6”, the objective is to predict the piece at the last position mentioned (“c6”).

Notatki dla szachów

Zadanie 1: Chess Piecetype

the piecetypes can be implicitly tracked given the moves. Given a long sequence of moves, e.g. “c2 c4 e7 e5 g2 g3 b8 c6 f1 g2 g8 f6 b1 c3 f8 b4 c6”, the objective is to predict the piece at the last position mentioned (“c6”).

Notatka:

Notatki dla szachów

Zadanie 1: Chess Piecetype

the piecetypes can be implicitly tracked given the moves. Given a long sequence of moves, e.g. “c2 c4 e7 e5 g2 g3 b8 c6 f1 g2 g8 f6 b1 c3 f8 b4 c6”, the objective is to predict the piece at the last position mentioned (“c6”).

Notatka:

So the previous example would look like: “c2 P c4 e7 P e5 g2 P g3 b8 N c6 f1 B g2 g8 N f6 b1 N c3 f8 B b4 c6”, and it is therefore much easier with Self-Notes to predict the piecetype at “c6”, since we know that the last piece moved to “c6” was a knight during the move “b8 N c6”.

Zadanie 2: Chess Move

Chess Move. This task is to predict the end position of the current move given the start position [17]. For example, given the sequence of moves “c2 c4 e7 e5 g2 g3 b8 c6 f1 g2 g8 f6 b1 c3 f8 b4 c6”, the answer is the ground truth final position made in the game: “e5”. This task is harder than the Chess Piecetype task as the model needs to learn, state tracking, chess rules, and chess strategy in

Notatka: Jak przy Chess Piecetype

Notatki – Connect 4



Korpus:

S11326354624002233001B

S323352465224555134411136200A

Co mogłyby być notatką?

Notatki – Connect 4



Korpus:

S11326354624002233001B

S323352465224555134411136200A

Co mogłoby być notatką?

Odpowiedź

W zasadzie **cokolwiek sensownego** (czyli związanego ze stanem, najlepiej tym jego aspektem, który wiąże się z ostatnim ruchem)

Notatki – Connect 4

S101130206031...

Notatką, która prawie podwaja liczbę wygranych partii jest **bit parzystości** właśnie powstałej kolumny pionków

Ostatnia z tej serii prac

- Pamiętamy przykład PW z tekstem [what is funny in this tweet?](#)

Ostatnia z tej serii prac

- Pamiętamy przykład PW z tekstem [what is funny in this tweet?](#)
- Okazuje się, że jest to jak najbardziej badane zjawisko (z trochę innymi tekstami)

Ostatnia z tej serii prac

- Pamiętamy przykład PW z tekstem [what is funny in this tweet?](#)
- Okazuje się, że jest to jak najbardziej badane zjawisko (z trochę innymi tekstami)

Idea

- Pozwólmy modelowi wygenerować odpowiedź

Ostatnia z tej serii prac

- Pamiętamy przykład PW z tekstem [what is funny in this tweet?](#)
- Okazuje się, że jest to jak najbardziej badane zjawisko (z trochę innymi tekstami)

Idea

- Pozwólmy modelowi wygenerować odpowiedź
- Poprośmy go o zanalizowanie tej odpowiedzi

Ostatnia z tej serii prac

- Pamiętamy przykład PW z tekstem [what is funny in this tweet?](#)
- Okazuje się, że jest to jak najbardziej badane zjawisko (z trochę innymi tekstami)

Idea

- Pozwólmy modelowi wygenerować odpowiedź
- Poprośmy go o zanalizowanie tej odpowiedzi
- I wygenerujmy nową odpowiedź (w razie czego, czynności można powtórzyć)

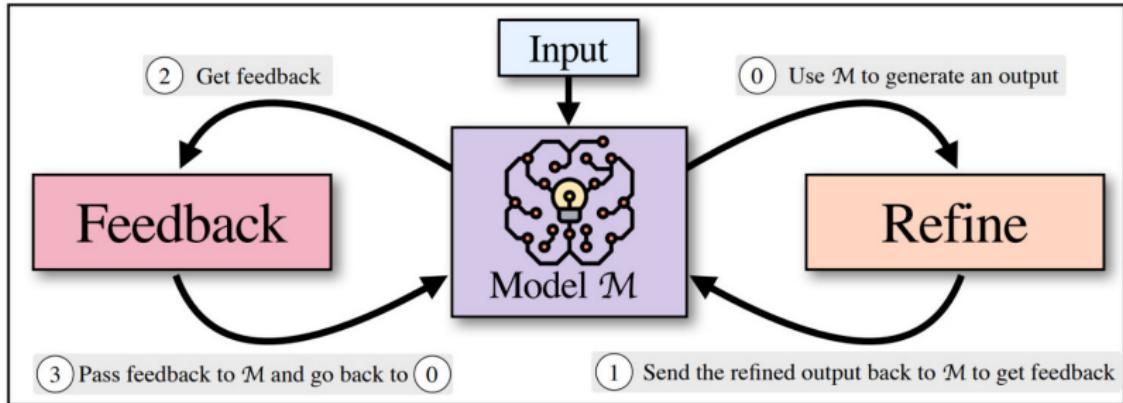


Figure 1: SELF-REFINE starts by taking an initially generated output (①), and passing it back to the same model \mathcal{M} (②) to get feedback (③); feedback on the initial output is passed back to the model (④), to iteratively refine (⑤) the previously generated output. SELF-REFINE is instantiated with a powerful language model such as GPT-3.5 and does not involve human assistance.

Iteracyjna praca nad odpowiedzią

(a) Dialogue: x, y_t

User: I am interested in playing Table tennis.

Response: I'm sure it's a great way to socialize, stay active

(b) FEEDBACK fb

Engaging: Provides no information about table tennis or how to play it.

User understanding: Lacks understanding of user's needs and state of mind.

(c) REFINE y_{t+1}

Response (refined): That's great to hear (...) ! It's a fun sport requiring quick reflexes and good hand-eye coordination. Have you played before, or are you looking to learn?

(d) Code optimization: x, y_t

```
Generate sum of 1, ..., N
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

(e) FEEDBACK fb

This code is slow as it uses brute force. A better approach is to use the formula ... $(n(n+1))/2$.

(f) REFINE y_{t+1}

Code (refined)

```
def sum_faster(n):
    return (n*(n+1))//2
```

Figure 2: Examples of SELF-REFINE: an initial output generated by the base LLM and then passed back to the *same* LLM to receive feedback to the *same* LLM to refine the output . The top row illustrates this for dialog generation where an initial dialogue response can be transformed into a more engaging one that also understands the user by applying feedback. The bottom row illustrates this for code optimization where the code is made more efficient by applying feedback.

Algorytm

Algorithm 1 SELF-REFINE algorithm

Require: input x , model \mathcal{M} , prompts $\{p_{\text{gen}}, p_{\text{fb}}, p_{\text{refine}}\}$, stop condition $\text{stop}(\cdot)$

```
1:  $y_0 = \mathcal{M}(p_{\text{gen}} \| x)$                                 ▷ Initial generation (Eqn. 1)
2: for iteration  $t \in 0, 1, \dots$  do
3:    $fb_t = \mathcal{M}(p_{\text{fb}} \| x \| y_t)$                             ▷ Feedback (Eqn. 2)
4:   if  $\text{stop}(fb_t, t)$  then                                         ▷ Stop condition
5:     break
6:   else
7:      $y_{t+1} = \mathcal{M}(p_{\text{refine}} \| x \| y_0 \| fb_0 \| \dots \| y_t \| fb_t)$  ▷ Refine (Eqn. 4)
8:   end if
9: end for
10: return  $y_t$ 
```

Figure 3: The SELF-REFINE algorithm. See (§2) for a discussion of each component.

Retrieval-based Language Models

- Knowledge is encoded explicitly
- The model **learns to search** for relevant passages, then **use the retrieved information** for crafting knowledgeable response

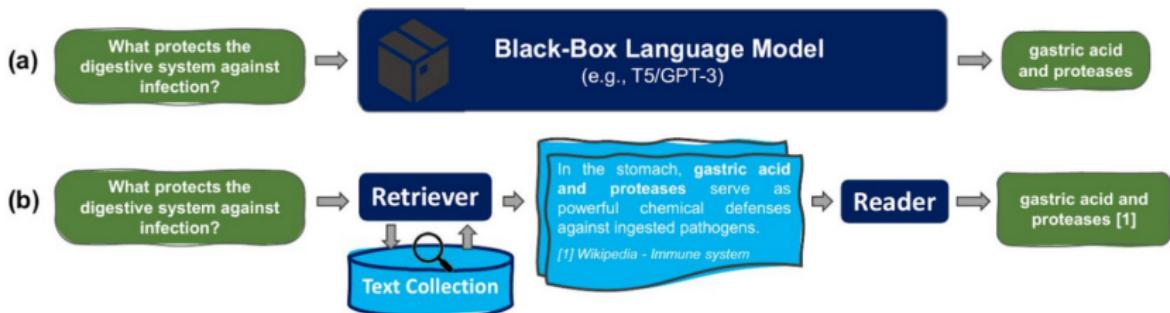


Figure from [stanford blog post](#) 8

Type 1: Token-level retrieval for LM

- Augment LM model with k NN-based model.
- Target is the next token.

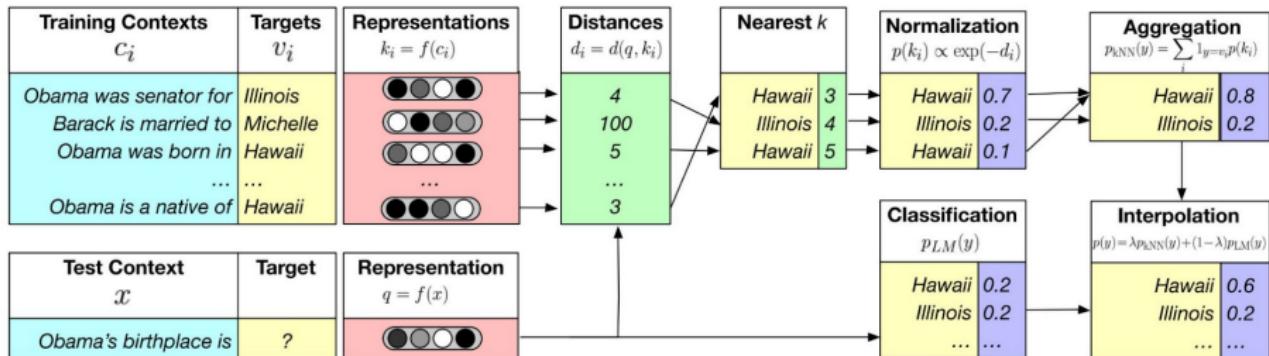


Figure from kNN-LM paper (Khandelwal et al. 2019) 13

Type 1: *Token*-level retrieval for LM

$$p_{\text{kNN}}(y|x) \propto \sum_{(k_i, v_i) \in \mathcal{N}} \mathbb{1}_{y=v_i} \exp(-d(k_i, f(x)))$$

$$p(y|x) = \lambda p_{\text{kNN}}(y|x) + (1 - \lambda) p_{\text{LM}}(y|x)$$

- No interaction between context encoder (for retrieval) and LM during training.
- What's the relationship between lambda and the size of database?

Type 2: Passage-level Retrieval for QA

- Contrastively train the retriever.
- Can be plugged into a QA system for retrieving context.

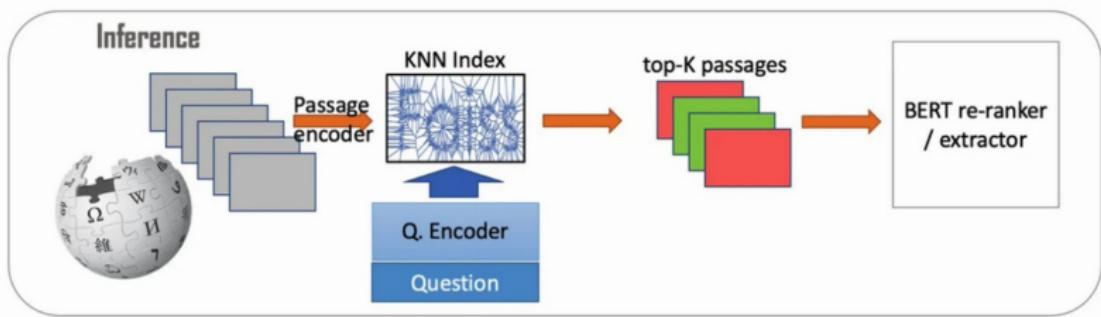


Figure from the [talk](#) of DRP paper (Karpukhin et al. 2020)

18

Type 2: Passage-level Retrieval for QA

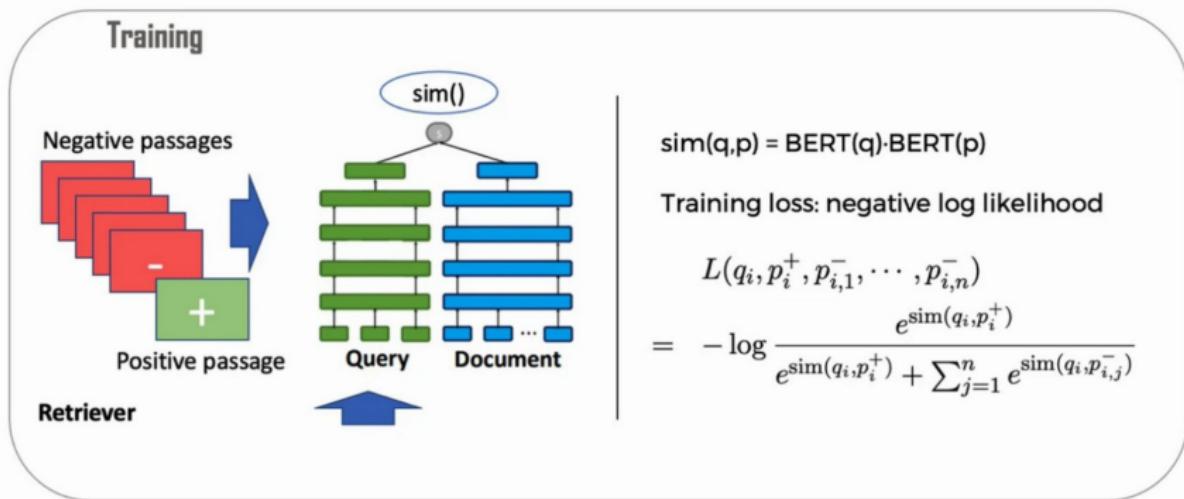


Figure from the [talk](#) of DRP paper (Karpukhin et al. 2020)

19