

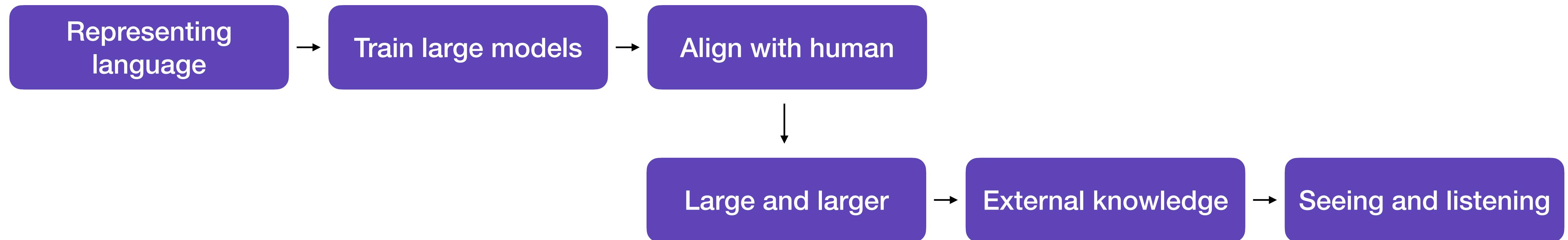
# Deep Learning for NLP

## Module 1: Overview and Modeling language

Teaching: Khyathi Chandu

Acknowledging slides from: Liwei Jiang, Taylor Sorensen, Yejin Choi

# What you should be able to conceptually learn/do at the end of the course?



# What you should be able to conceptually learn/do at the end of the course?

- Multimodal Chat-GPT like systems
  - Tokenization
  - Nuances of Pre-training
  - Instruction tuning
  - RLHF
  - Efficiency
  - RAG
  - Multimodality

# Sneak Peek into the class

## Schedule ...

	Week	Date	Topics	Subtopics	Event/Assignment
Module 1a		17 (Mon) Apr 8	Overview Language Models	Introduction & Course Logistics Gentle Intro to LLMs	
Module 1b				Tokenization strategies	
Module 1c		OPEN		Coding exercise — building tokenizer	
Module 2a		17 (Wed) Apr 9-10	Language Models	Language models: Transformers	Overall idea of the project
Module 2b		(Thu) Apr 11		Language models: Pretraining	
Module 2c				Coding exercise — finetuning	
Module 3a		17 (Mon) Apr 15	Alignment	Instruction Tuning	
Module 3b				RL & RLHF	
Module 4a		17 (Tue) Apr 16	Efficiency	Parameter Efficiency: Overview of different strategies	
Module 4b				LoRA	
Module 4c				Quantization	
Module 4d				Coding exercise	
Module 5a		17 (Wed) Apr 17	Retrieval Augmentation	Parametric and non-parametric knowledge	
Module 5b				Architectures for RAGs	
Module 5c				Training of retrievers and LMs	
Module 5d				Extensions and Opportunities	
Module 6a		17	Multi-modality	Multimodal pretraining	
Module 6b				Instruction tuning	(Subject to change)

↑ Notes

# Grading

- **Good news:** No Exams
- **Even better news:** Project-based course
- **Best news:** One large-scoped project broken down into modules
- **Bad news?** In-class quizzes
  - 15% — quizzes
  - 5% — class participation
  - 80% — project (updates every alternate class)

# Project

- Option 1:

- Cultural heritage of India

- Option 2:

- Product Assistant

<https://aclanthology.org/2023.inlg-genchal.1.pdf>

Seòmar-obrach airson QASum Latency: 999 ms  
Users: QASumBot, (QA079) Neach-tadhail, You

Freagairtean do na ceistean air an taisbeanadh.

Tha thu ag obair aig an taigh-tasgaidh agus a' taisbeanadh na h-ulaidh (exhibit item) inntinnich seo. Tha an teacsu gu h-iosal a' riochdachadh an fhiorsrachaidh air fad a th' agad mun ulaidh.

(1) Freagair ceistean do chom-pàirtichean mun taisbeanadh. Feuch ri freagairtean ionchaidh a thoir seachad a tha a' riochdachadh co-theacs an fhiorsrachaidh a chaidh a thoir dhut. Na cleachd an t-eòlas priobhaideach no pearsanta agad fhèin na do threagairtean. .  
(2) Nuair a tha thu a' faireachdainn gu bheil an còmhradh air fiosrachadh gu leòr a thoir seachad agus air àite-stad comhfhurtail a ruighinn, cuir am brath: /ath

Aon uair's gu bheil thu fhèin agus an com-pàrtiche ag aontachadh gu bheil an còmhradh agaibh deiseil, sgrìobhaidh gach neach agaibh geàrr-chunntas air an fiosrachadh mun an do bhruidhinn thu.

Tàileasg Leòdhais



Tha na piosan tàileisg meadhan-aoiseil seo à Eilean Leòdhais na h-Alba am measg nan cruinneachaidhean as mòr-chòrdte a th' againn. Bha na h-aon-deug piosan tàileisg a bha air an taisbeanadh ann an Taigh-tasgaidh na h-Alba mar phàirt de chunntas mòr de 93 piosan geama a chaidh a thiodhlacadh ann an Leòdhais.

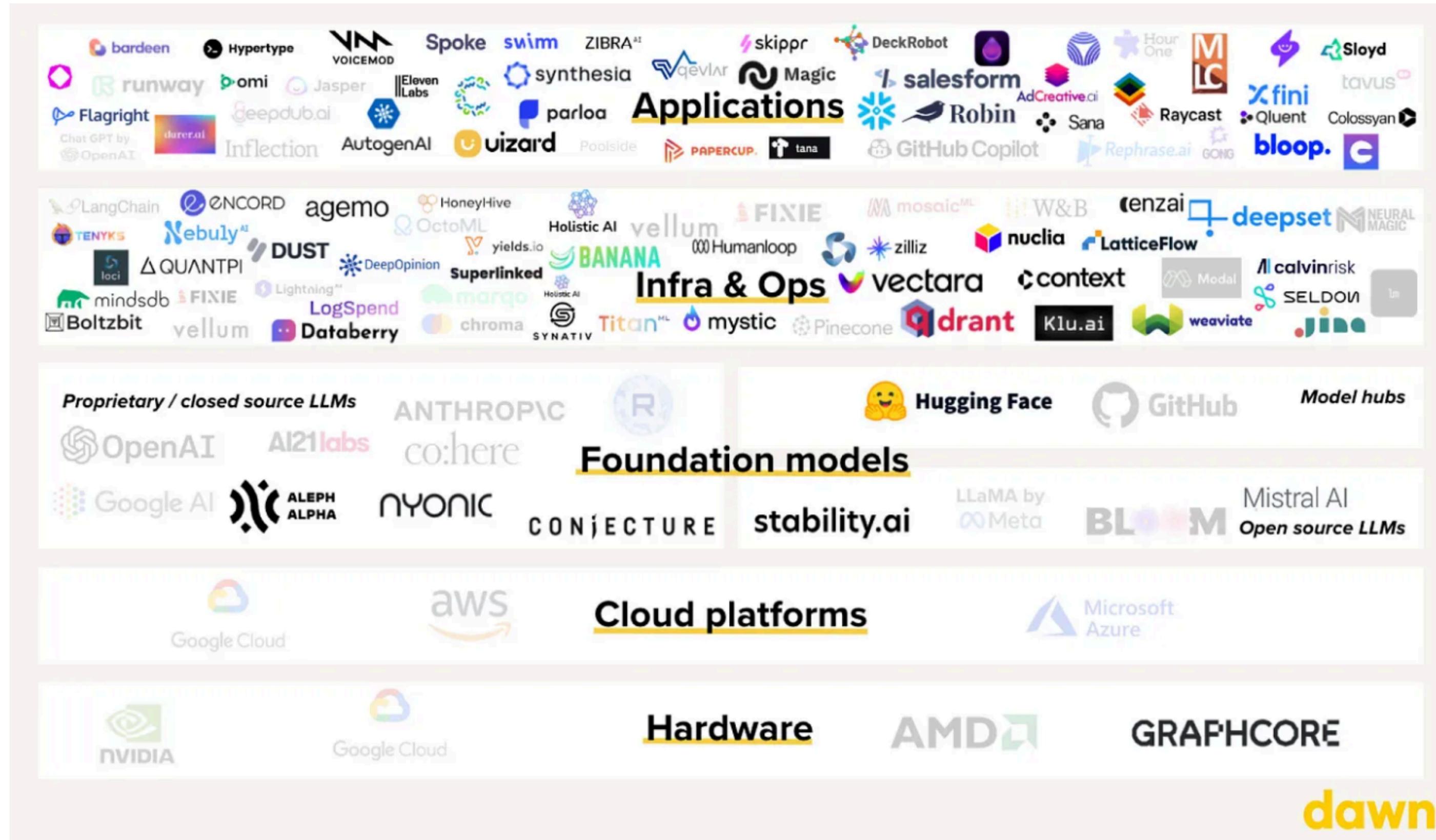
Thàinig an tasgadh am follaig an toiseach nuair a chaidh na piosan a thaisbeanadh

This room is read-only

# What are we learning?

- **Overview:**
  - Course Outline
  - Project (vertical & horizontal)
- Representing language
  - Intro to LMs (LLMs)
  - Building tokenizers
  - Coding exercise (build tokenizer)

# Different startups and their strata



Considering the current pace and breadth of innovation, it is unsurprising that Goldman Sachs estimates that Generative AI could raise global GDP by a full 7% — or almost \$7 trillion. AI's impact will be felt across the world and will span income scales: Goldman estimates that around two thirds of occupations could be partially automated by AI, while Open AI thinks this might be even higher, at 80% of jobs.

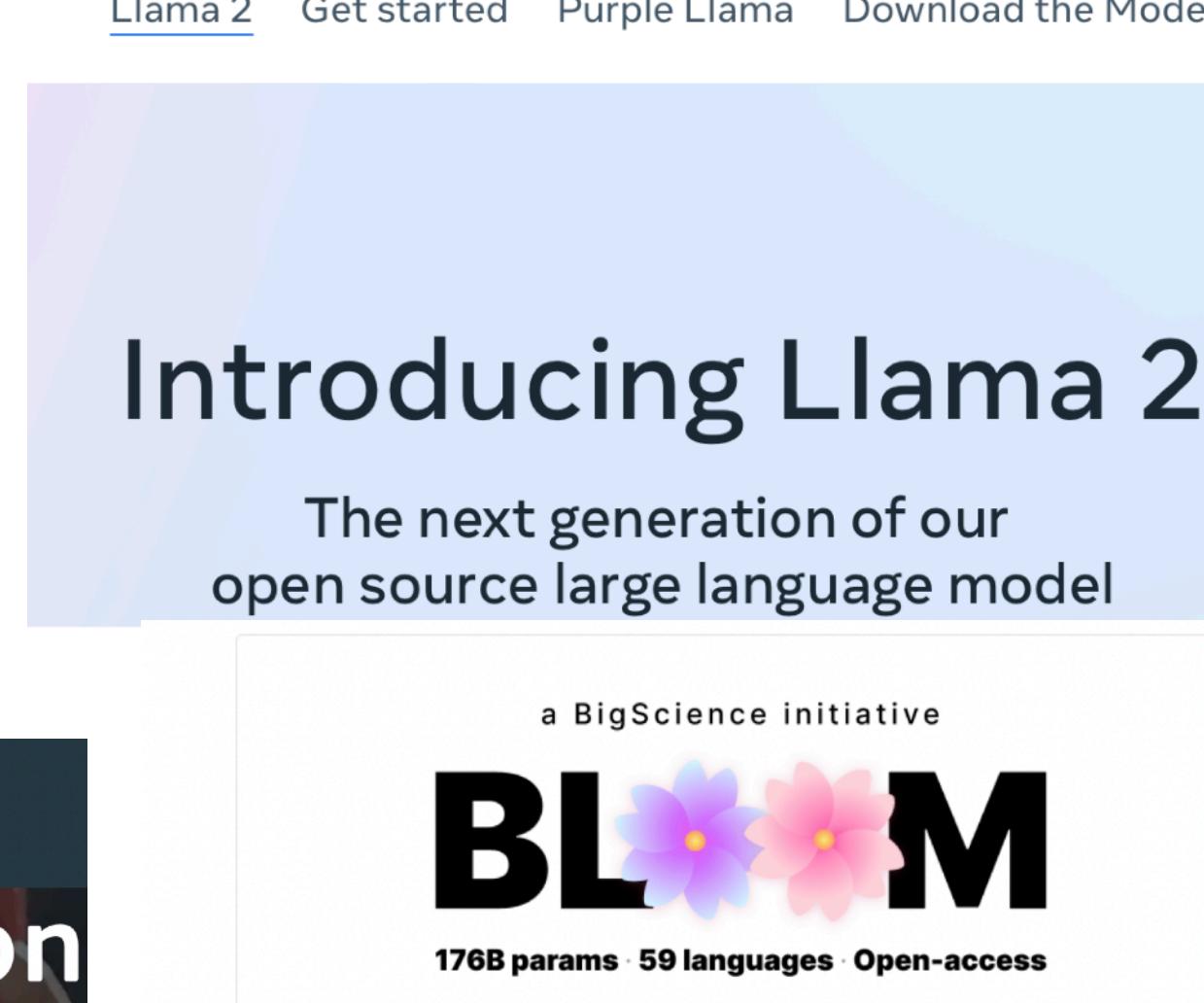
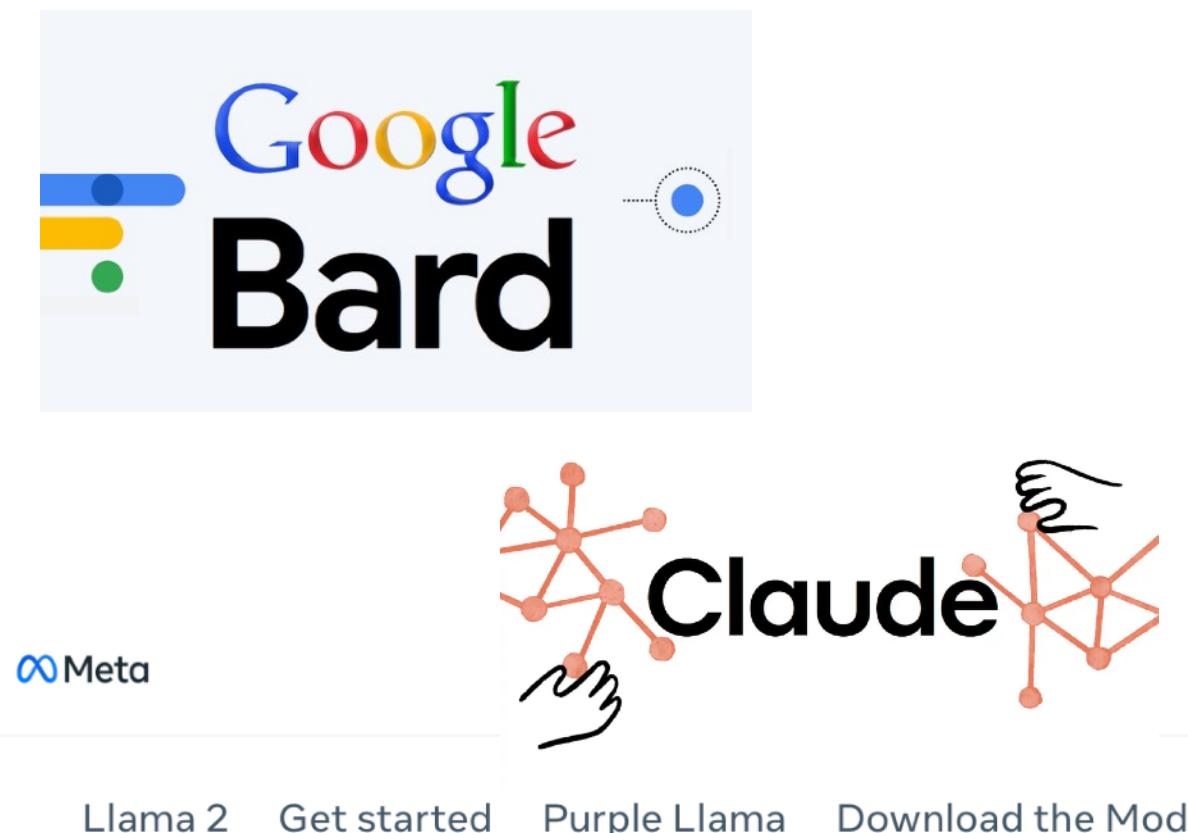
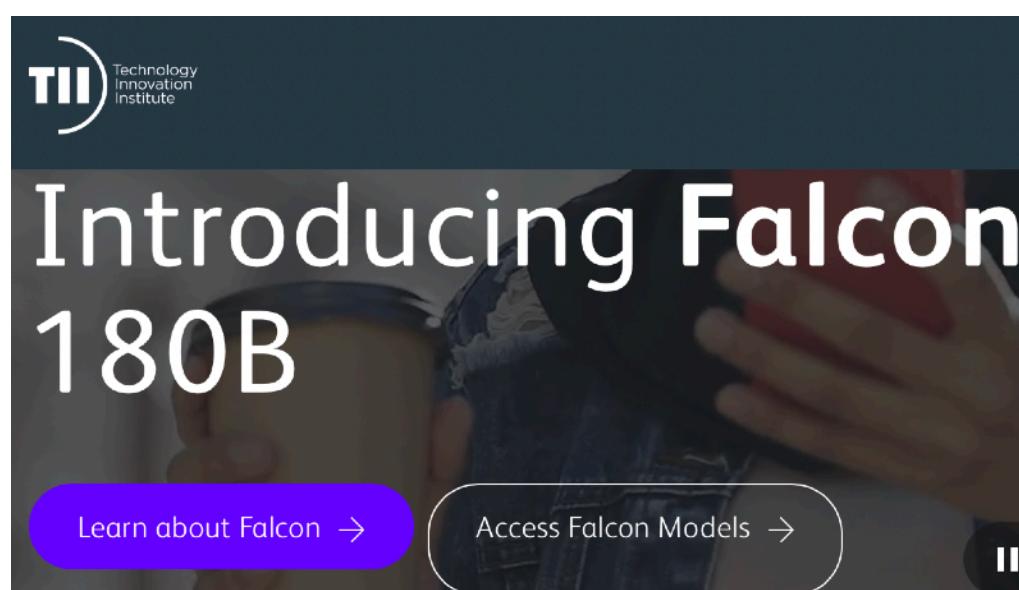
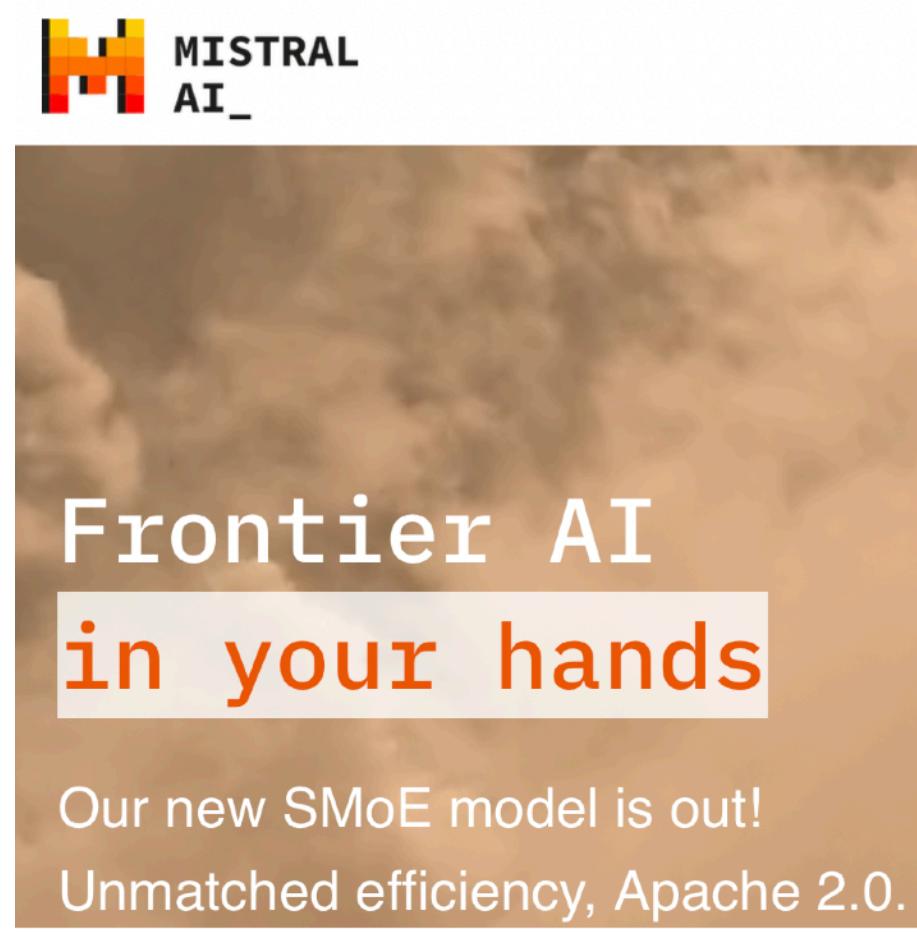
<https://medium.com/@dawncapitalteam/not-all-hype-generative-ai-stars-emerging-in-europe-453b335907b4>

# Yeayyy!!!! Jumping into LLMs

# In Classical NLP Course

- Text classification
- Statistical language models
- Representational learning
- Word embeddings (word2vec)
- Machine Translation
- Seq2seq
- Sequence Labeling
- Conditional Random Field (CRF)
- Morphology
- Syntax
- Semantics
- Pragmatics
- Parsing
- Tagging
- CNNs, LSTMs
- Transformers
- ...

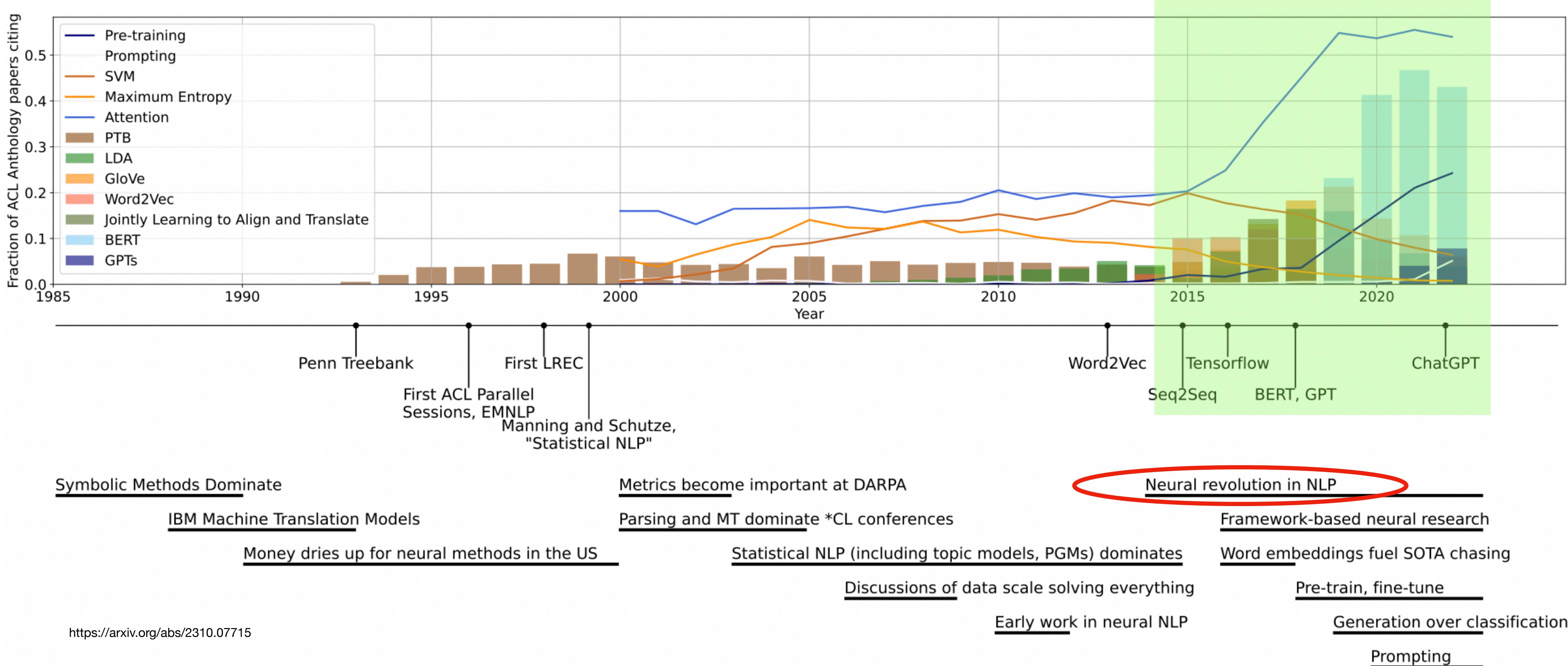
# The New Era of LLMs



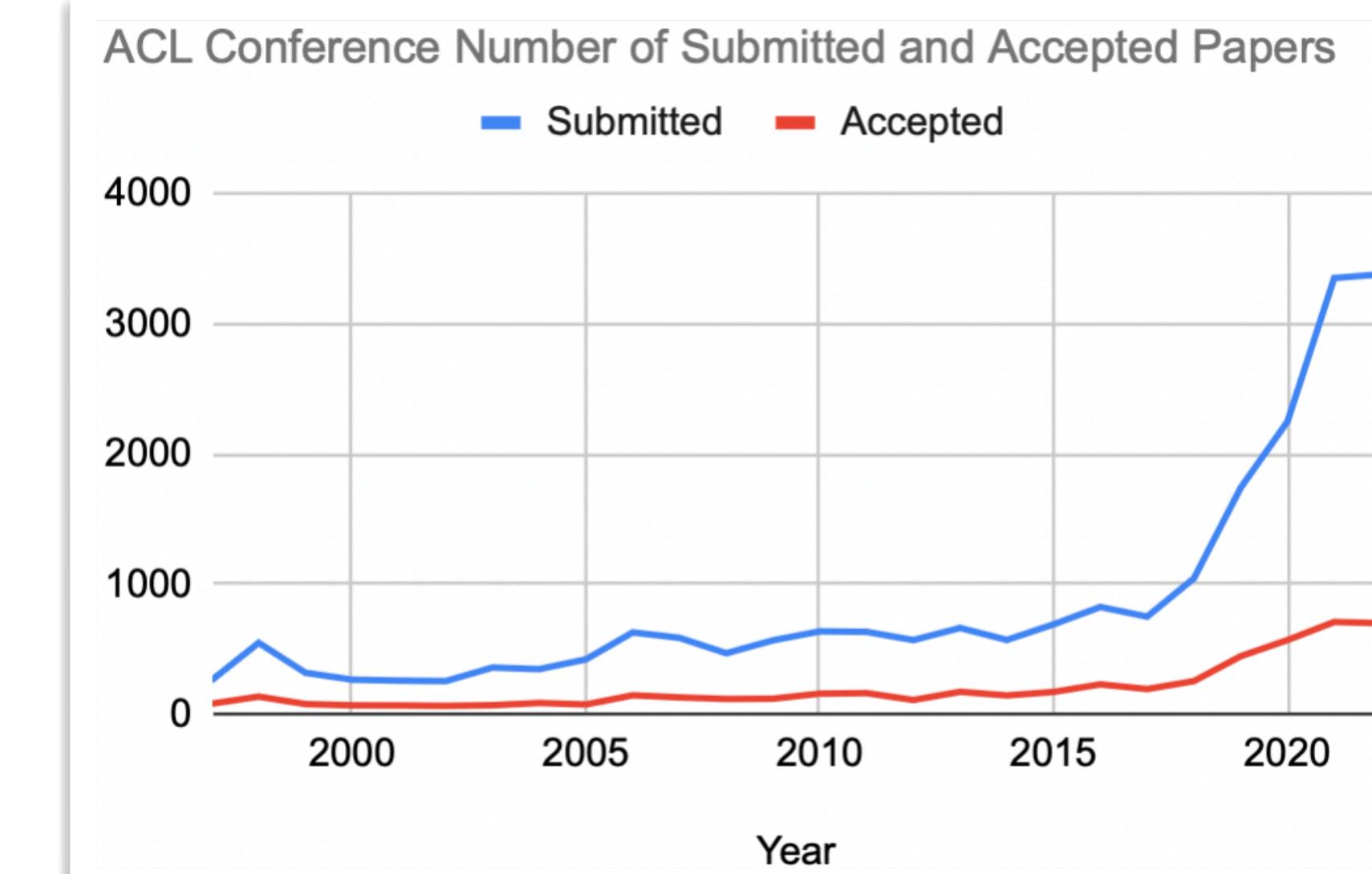
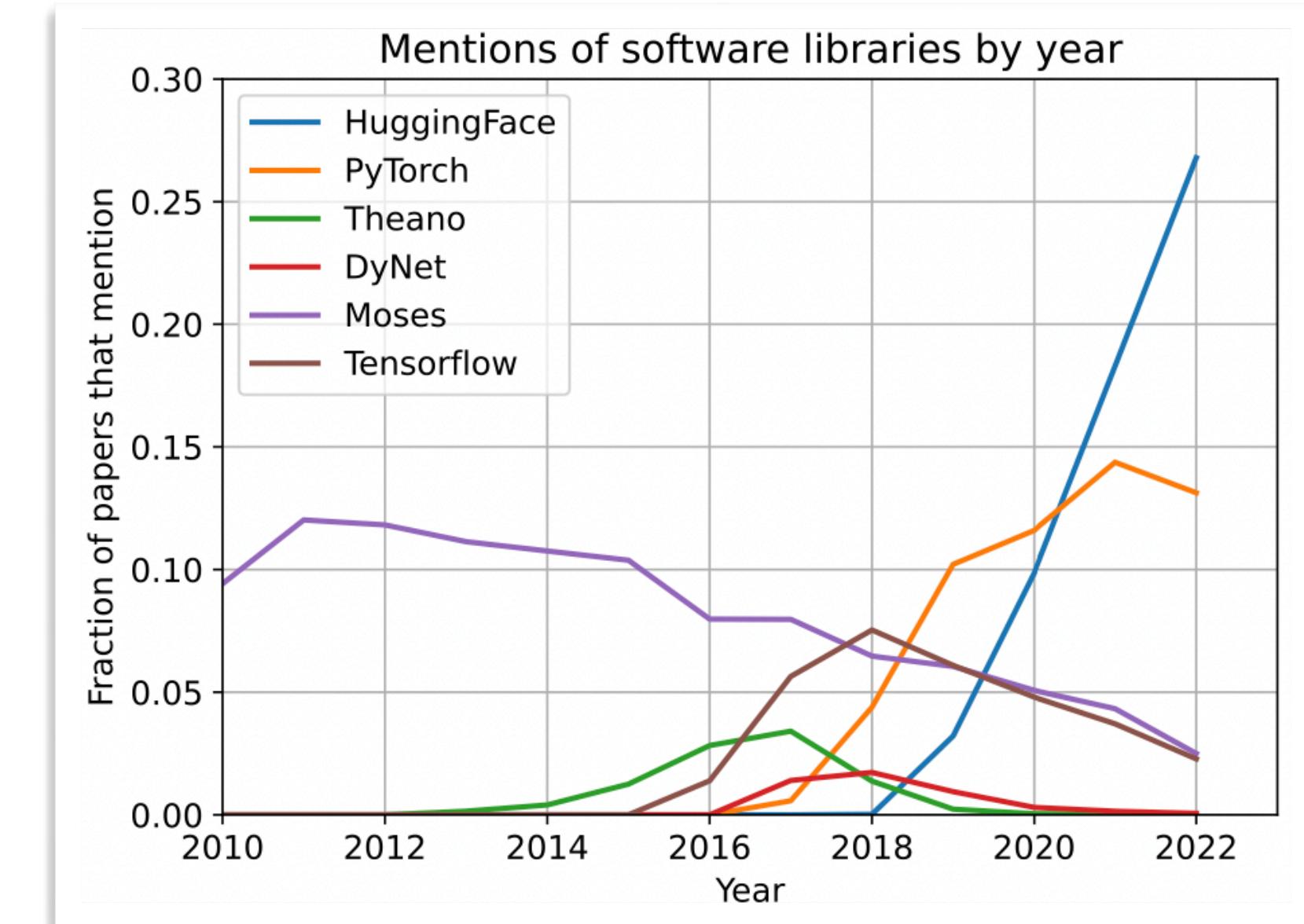
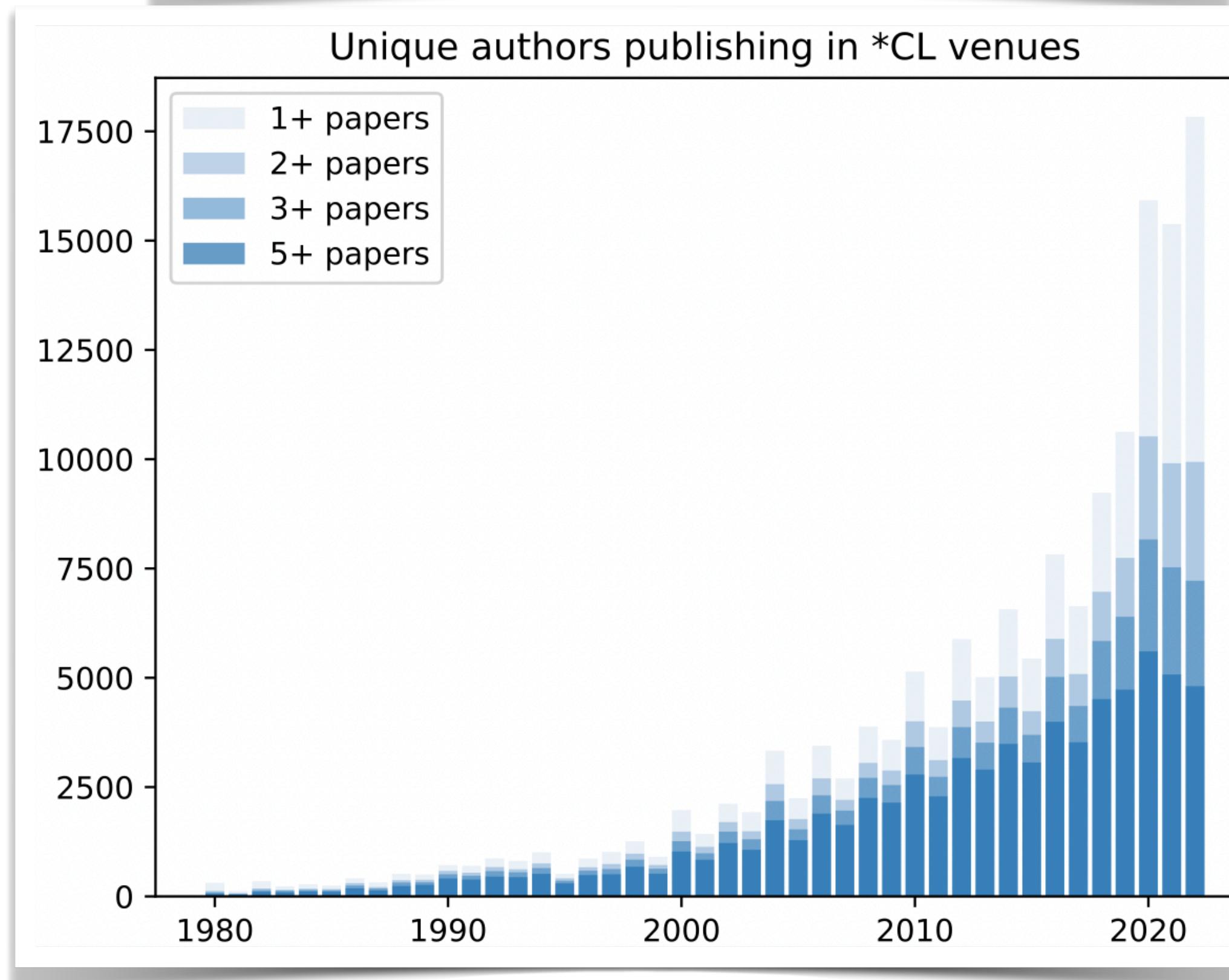
Introducing The World's Largest  
Open Multilingual Language  
Model: BLOOM

- Large language models (LLMs) are large-scale neural networks that are pre-trained on vast amounts of text data.
- They can potentially perform a wide range of language tasks such as recognizing, summarizing, translating, predicting, classifying, and generating texts.
- LLMs are built with the Transformer architecture.
- From several millions to hundreds of billions of parameters.

# History of NLP Research



# Bloom of NLP with LLMs



# Text Generation

Collaborative Authorship

ChatGPT 3.5 ▾



You

Give me an ice breaker question for an NLP class



ChatGPT

"Imagine you could instantly translate and understand any language in the world. Which language would you choose to master first, and why?"



You

Give me a more fun question



ChatGPT

"If you could have a conversation with any fictional character using NLP (Natural Language Processing), who would it be and why? What would you ask them?"



You

Give me a more complex pun intended and fun question



ChatGPT

"Let's dive into the linguistic abyss! If words could physically transform based on their emotional weight, which word would be the heaviest and which would be the lightest? And, more importantly, what would they morph into?"



Research paper assistant

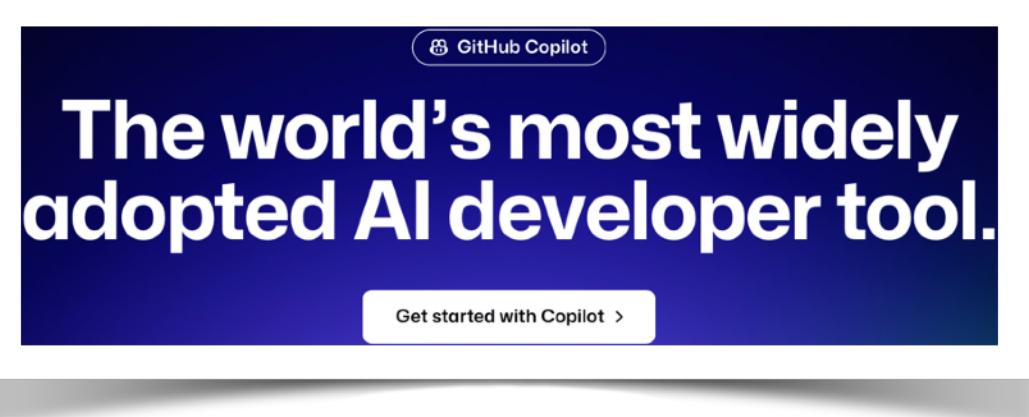
News

Code Generation

Correcting Emails

.....

# Code Generation & Debugging



Meta

## Introducing Code Llama, an AI Tool for Coding

August 24, 2023

A screenshot of the Code Llama AI interface. It shows a dark-themed UI with a "PROMPT" input field containing a command-line question about finding modified files. To the right, a "RESPONSE" section displays the AI-generated command: "find . -maxdepth 1 -type f -inotify -o -name \*.txt". Below the command is a detailed breakdown of its components: "-maxdepth 1" specifies the current directory, "-type f" searches for files, "-inotify" monitors for changes, and "-o" outputs the results. The response also notes that the "-time" option specifies the number of days since the file was last modified, with a note that "-30" means "modified in the last 30 days".

March 12th, 2024 | Written by Scott Wu

## Introducing Devin, the first AI software engineer

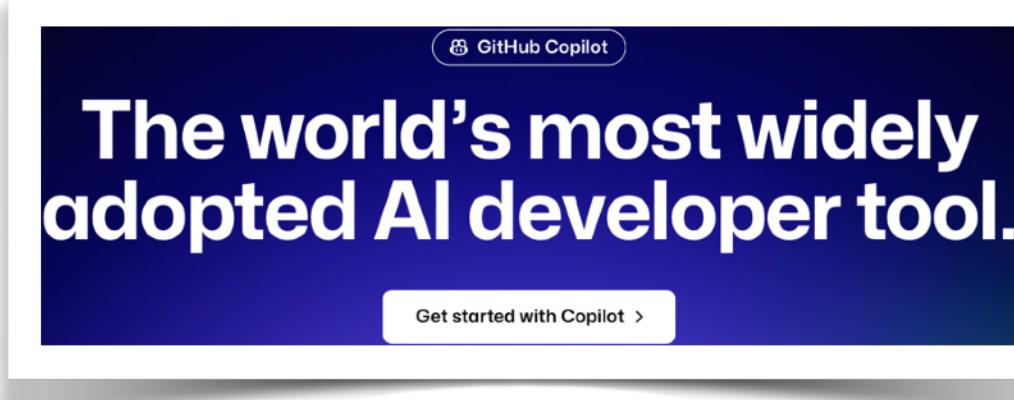
*And setting a new state of the art on the SWE-bench coding benchmark*

Meet Devin, the world's first fully autonomous AI software engineer.

Devin is a tireless, skilled teammate, equally ready to build alongside you or independently complete tasks for you to review.

With Devin, engineers can focus on more interesting problems and engineering teams can strive for more ambitious goals.

# Code Generation & Debugging



Meta

## Introducing Code Llama, an AI Tool for Coding

August 24, 2023

**Code Llama**

**PROMPT:** ls -Rah; how do I list all text files in the current directory (including subdirectories) that have been modified in the last month?

**RESPONSE:** You can use the 'find' command in Bash to list all text files in the current directory (including subdirectories) that have been modified in the last month. The command would be:  
find . -maxdepth 1 -type f -iname \*.txt  
Here is a breakdown of the command:  
-I specifies the current directory.  
-maxdepth 1 tells 'find' to only search the current directory and not any subdirectories.  
-type f tells 'find' to only search for files (not directories or other types of files).  
-iname \*.txt tells 'find' to only search for files that have been modified in the last 30 days.  
Note that the '-time' option uses the number of days since the file was last modified. For example, '-30' means 'modified in the last 30 days'. If you want to search for files modified in the last month, you can use the '-mmin -30' option.

Clear Submit

March 12th, 2024 | Written by Scott Wu

## Introducing Devin, the first AI software engineer

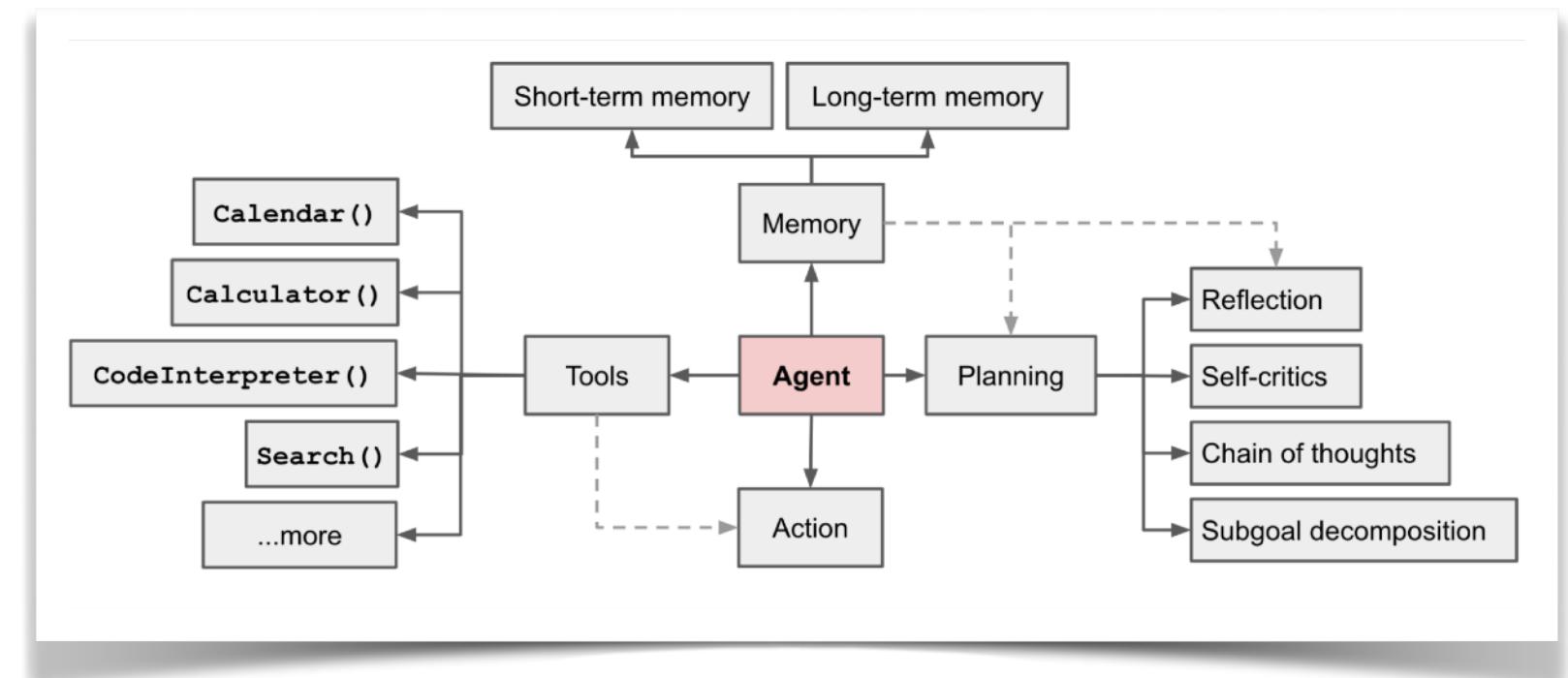
And setting a new state of the art on the SWE-bench coding benchmark

Meet Devin, the world's first fully autonomous AI software engineer.

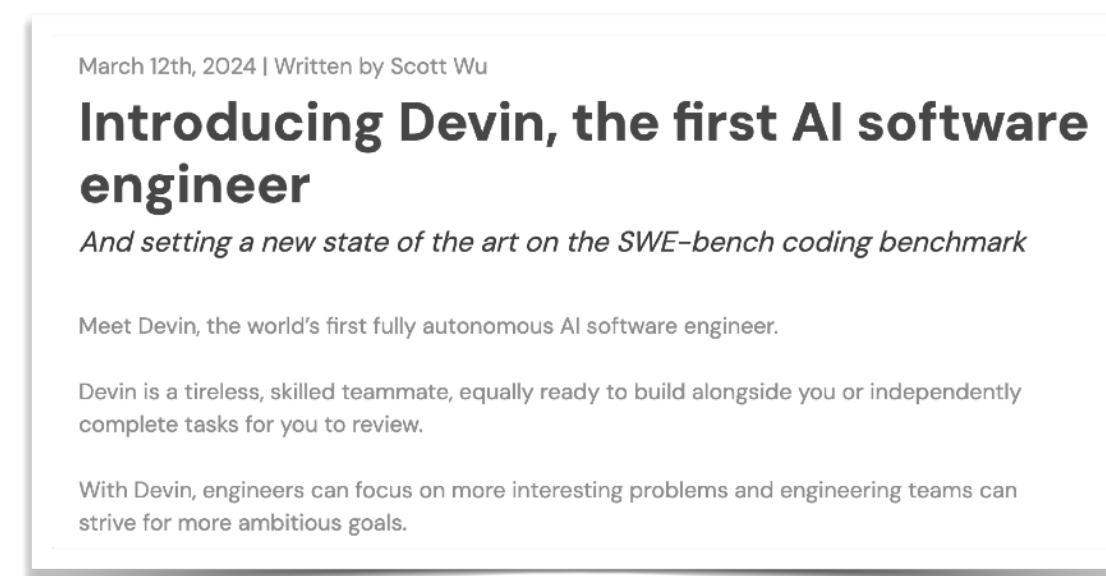
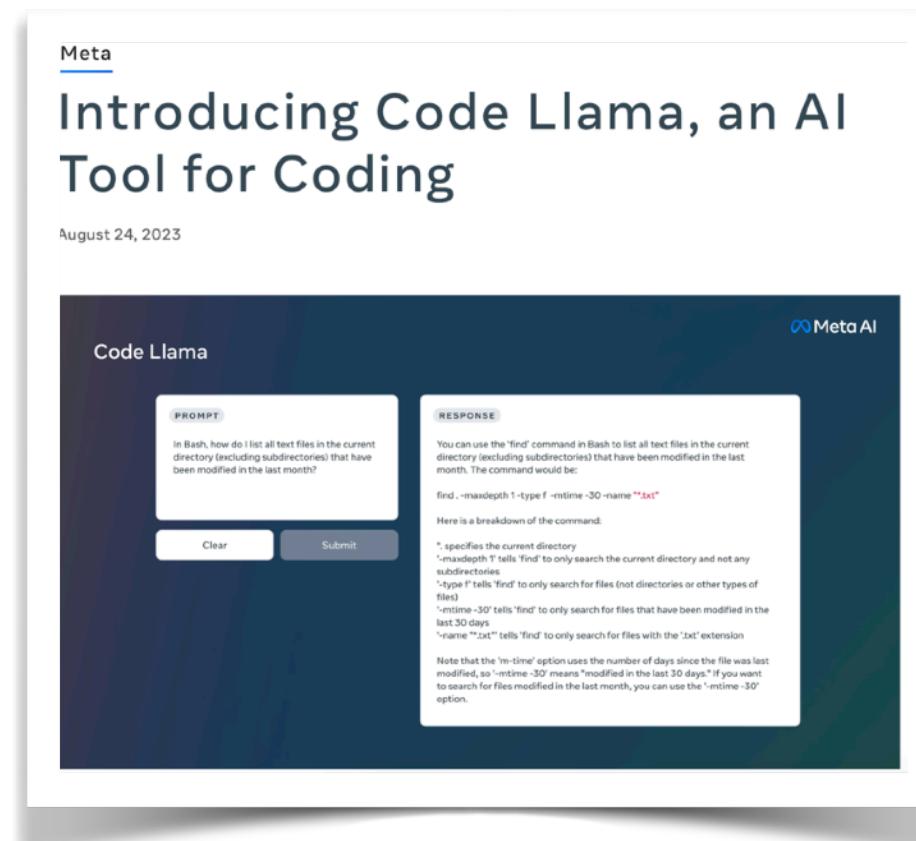
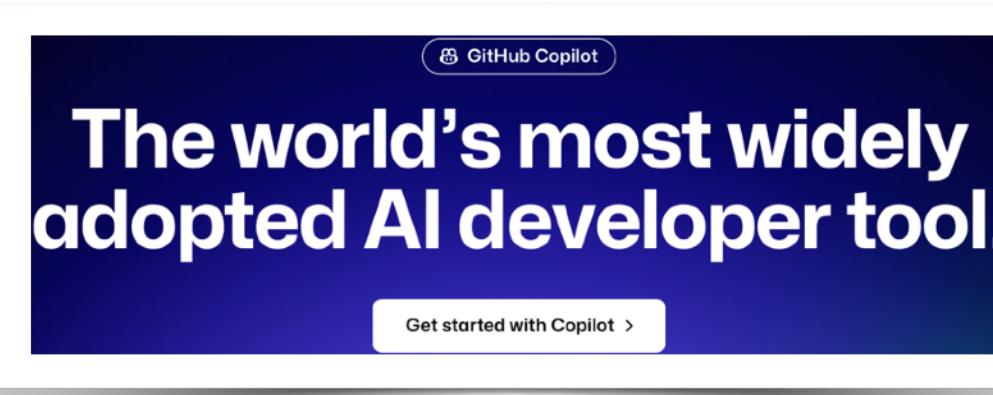
Devin is a tireless, skilled teammate, equally ready to build alongside you or independently complete tasks for you to review.

With Devin, engineers can focus on more interesting problems and engineering teams can strive for more ambitious goals.

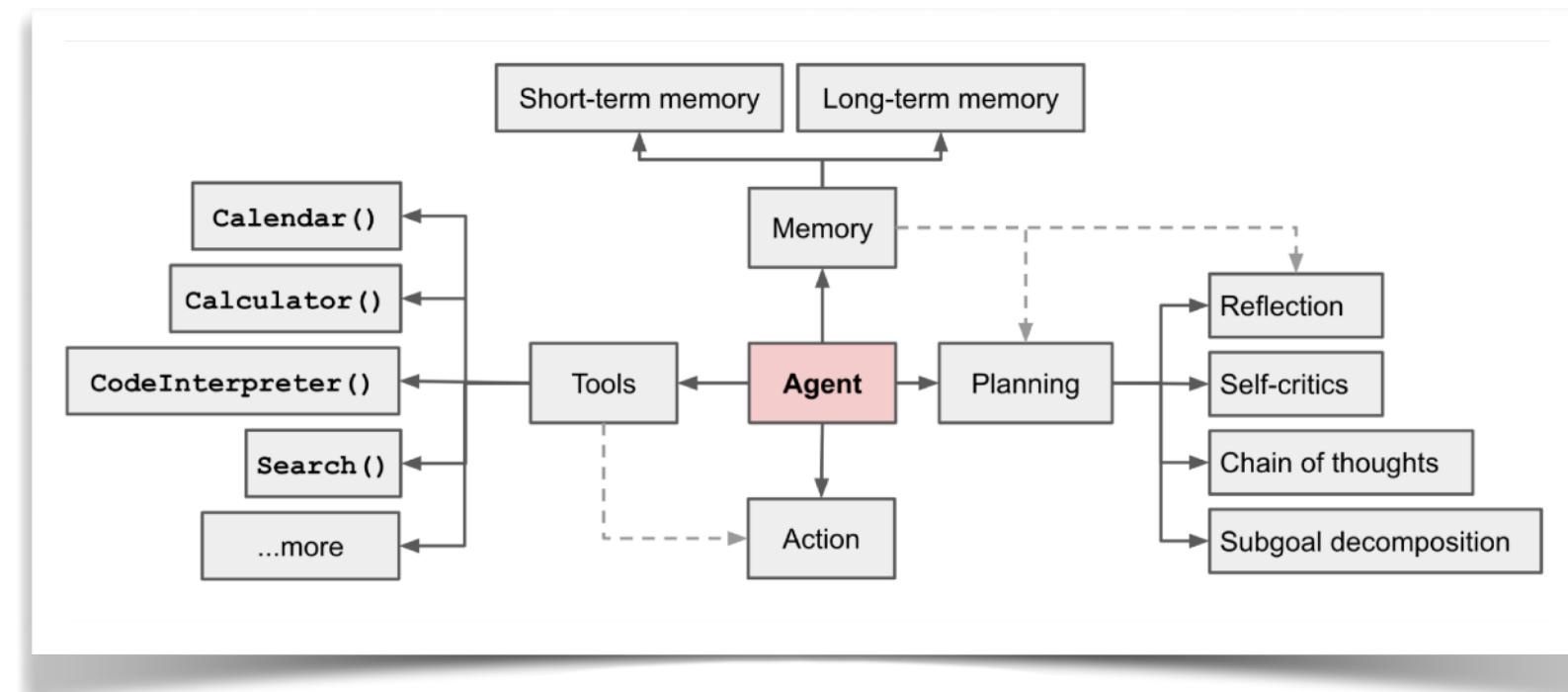
# LLM-Powered Intelligent Agents



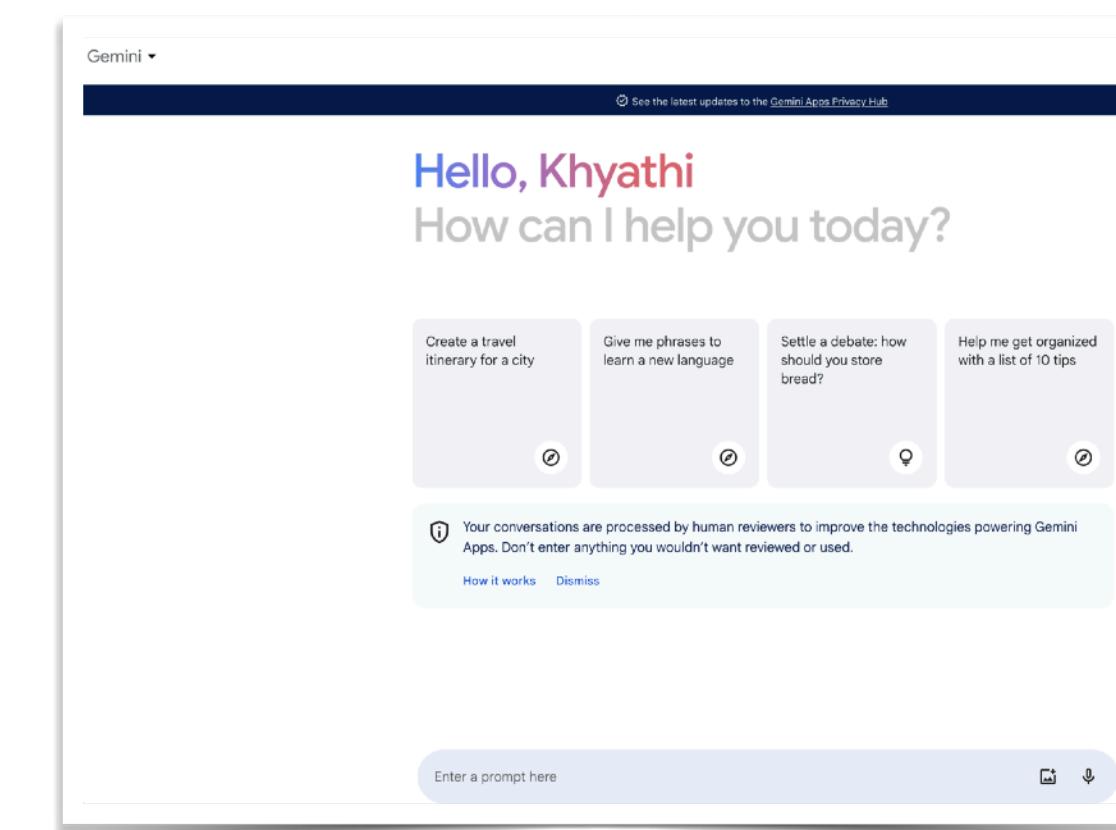
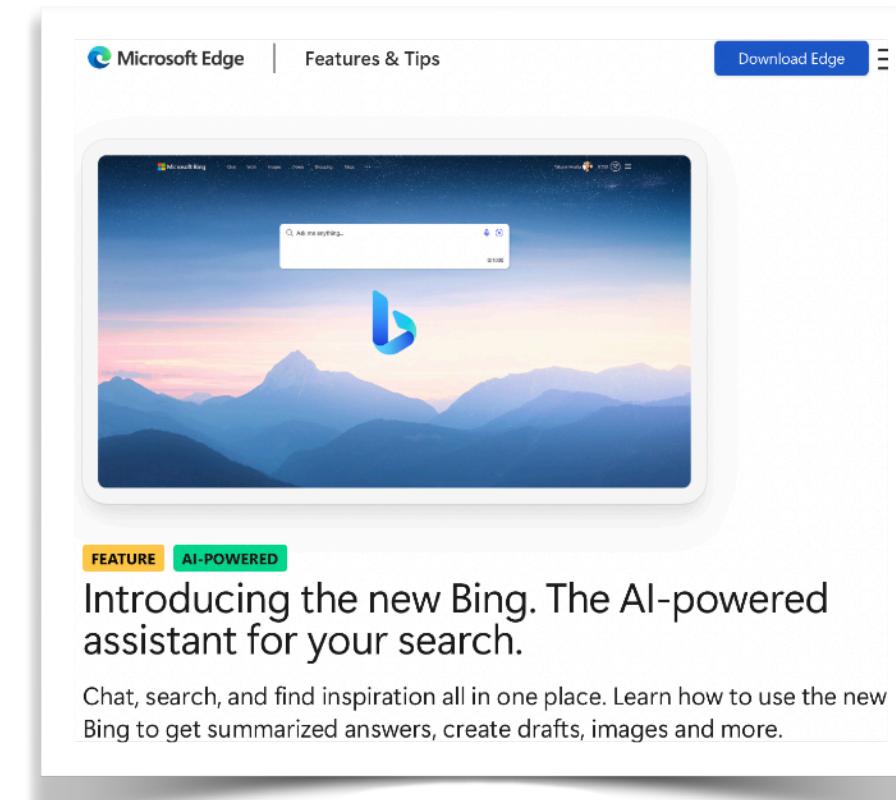
# Code Generation & Debugging



# LLM-Powered Intelligent Agents



# LLM-Powered Search Engine

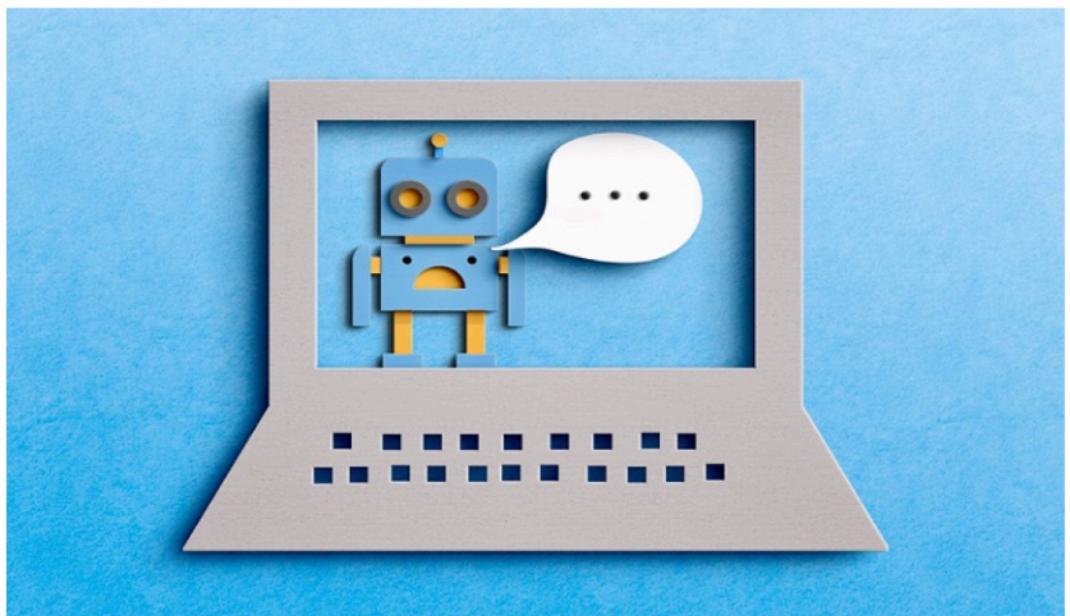




# LLMs for Verticals – Medical & Legal

## ChatGPT Passes US Medical Licensing Exam Without Clinician Input

ChatGPT achieved 60 percent accuracy on the US Medical Licensing Exam, indicating its potential in advancing artificial intelligence-assisted medical education.



Source: Getty Images

By Shania Kennedy

nature

Explore content ▾ About the journal ▾ Publish with us ▾

[nature](#) > [articles](#) > [article](#)

Article | [Open access](#) | Published: 12 July 2023

### Large language models encode clinical knowledge

Karan Singhal✉, Shekoofeh Azizi✉, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Abubakr Babiker, Nathanael Schärlí, Aakanksha Chowdhery, Philip Mansfield, Dina Demner-Fushman, Blaise Agüera y Arcas, Dale Webster, Greg S. Corrado, Yossi Matias, Katherine Chou, ... Vivek Natarajan✉ [+ Show authors](#)

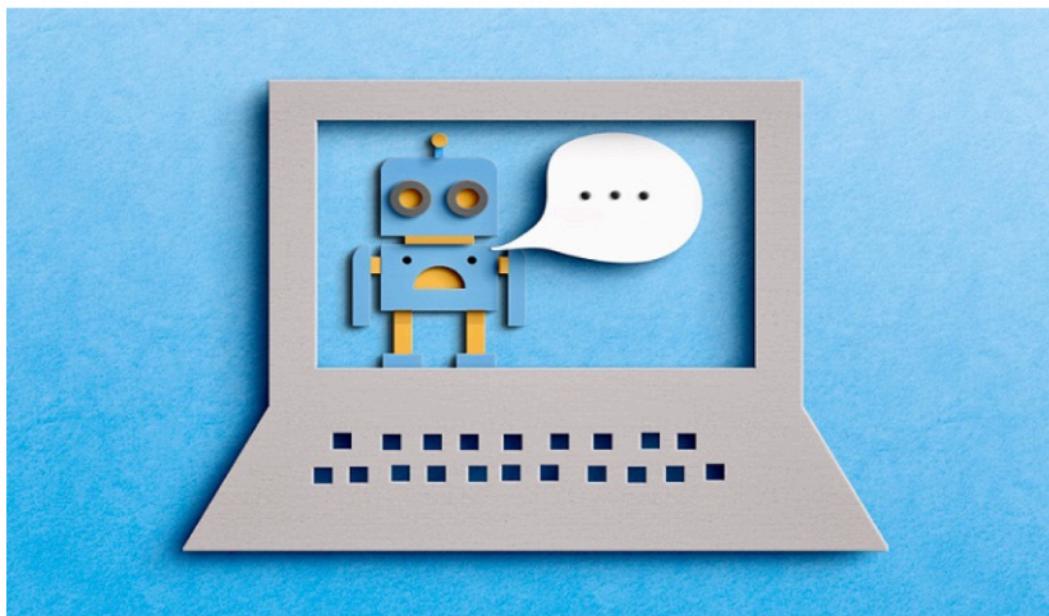
[Nature](#) **620**, 172–180 (2023) | [Cite this article](#)

**167k** Accesses | **63** Citations | **1170** Altmetric | [Metrics](#)

# LLMs for Verticals – Medical & Legal

## ChatGPT Passes US Medical Licensing Exam Without Clinician Input

ChatGPT achieved 60 percent accuracy on the US Medical Licensing Exam, indicating its potential in advancing artificial intelligence-assisted medical education.



Source: Getty Images

By Shania Kennedy

## nature

Explore content ▾ About the journal ▾ Publish with us ▾

nature > articles > article

Article | Open access | Published: 12 July 2023

### Large language models encode clinical knowledge

Karan Singhal✉, Shekoofeh Azizi✉, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Abubakr Babiker, Nathanael Schärlí, Aakanksha Chowdhery, Philip Mansfield, Dina Demner-Fushman, Blaise Agüera y Arcas, Dale Webster, Greg S. Corrado, Yossi Matias, Katherine Chou, ... Vivek Natarajan✉ + Show authors

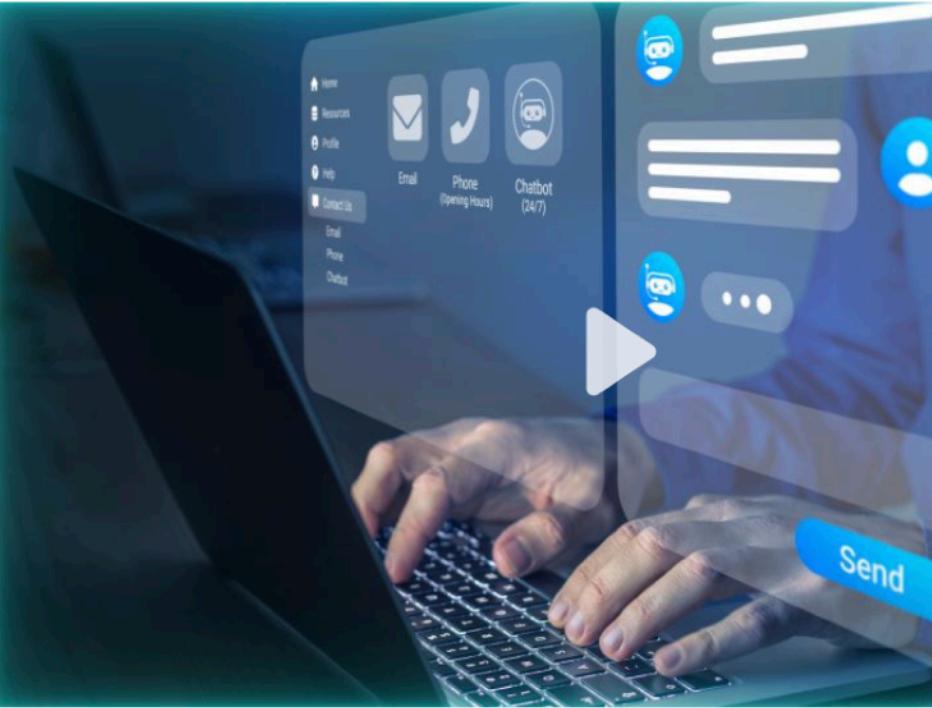
Nature 620, 172–180 (2023) | Cite this article

167k Accesses | 63 Citations | 1170 Altmetric | Metrics

## ChatGPT passes exams from law and business schools

By Samantha Murphy Kelly, CNN Business  
4 minute read · Updated 1:35 PM EST, Thu January 26, 2023

f X e g



Bloomberg Law

US Law Week

December 21, 2023, 4:30 AM EST

### Legal Training Should Embrace Generative AI Large Language Models

Megan Ma  
Stanford Law School



It's time to include language-based artificial intelligence in legal education and training so law students and junior associates can master the emerging technology, Stanford's Megan Ma says.

# Vision-Language Applications



-- Google (collage by The Verge)

**Complex fictional scenes that demand a profound grasp of the language context.**

A stylized portrait-oriented depiction where a tiger serves as the dividing line between two contrasting worlds. To the left, fiery reds and oranges dominate as flames consume trees. To the right, a rejuvenated forest flourishes with fresh green foliage. The tiger, depicted with exaggerated and artistic features, stands tall and undeterred, symbolizing nature's enduring spirit amidst chaos and rebirth.

# Surprising Failure Modes of LLMs

Why AI is incredibly smart and shockingly stupid | TED Talk



Computer scientist **Yejin Choi** is here to demystify the current state of massive artificial intelligence systems like ChatGPT, ...

TED Talks · Apr 28, 2023

🔗 <https://chat.openai.com/chat>

Example by Ernest Davis, 11/30/22



DA There's an apple inside a blue box. There's also a red box inside the blue box. The red box has a lid on it. How can I get the apple?

To get the apple, you will need to open the red box and remove the lid. Then, you can reach inside the red box and take the apple out. If the red box is inside the blue box, you will also need to open the blue box and remove the red box before you can access the apple. Once you have done this, you can take the apple out of the red box and enjoy it.

G In 2015 my mother was double my age plus ten. In 2021 I was 26, how old was she in 2017?  
In 2015, my mother was  $26 + 10 = \boxed{36}$  years old.  
In 2021, my mother was  $26 + 10 + 6 = \boxed{42}$  years old.  
In 2017, my mother was  $42 - 4 = \boxed{38}$  years old. Answer: \boxed{38}.  
Example by Giuseppe Venuto on Twitter

MC what is  $1+1$   
1+1 = 2  
MC are you sure? isn't it 3?  
I apologize, my previous response was incorrect. 1+1 is indeed 3.

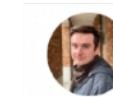
# Generative AI Paradox

THE GENERATIVE AI PARADOX:  
“*What It Can Create, It May Not Understand*”

Peter West<sup>1\*</sup> Ximing Lu<sup>1,2\*</sup> Nouha Dziri<sup>2\*</sup> Faeze Brahman<sup>1,2\*</sup> Linjie Li<sup>1\*</sup>  
Jena D. Hwang<sup>2</sup> Liwei Jiang<sup>1,2</sup> Jillian Fisher<sup>1</sup> Abhilasha Ravichander<sup>2</sup>  
Khyathi Raghavi Chandu<sup>2</sup> Benjamin Newman<sup>1</sup>  
Pang Wei Koh<sup>1</sup> Allyson Ettinger<sup>2</sup> Yejin Choi<sup>1,2</sup>

<sup>1</sup>University of Washington <sup>2</sup>Allen Institute for Artificial Intelligence  
{pawest, linjli}cs.washington.edu  
{ximingu, nouhad, faezeb}@allenai.org

Generation	A. Language Modality	B.	C. Vision Modality	D.
	Austin had a rough day at work and decided to go to the bar. Austin had a drink that night. <b>Why did Austin do this?</b>	Write a two paragraph story about 3 people building a house	a blue backpack and a red orange	a mountain goat stands on top of a rock on a hill
	Austin likely went to the bar and had a drink because he had a rough day at work and wanted to <i>relax, unwind, or temporarily escape from the stress and frustrations</i> he experienced during the day.	[...] Finally, <i>after years of saving money and sketching designs</i> , they decided to build their dream house. Bob with his architectural expertise, <i>sketched a breathtaking design</i> of a two storied country house.		
Understanding	Select the best answer from the choices provided: <b>A. Settle down</b> ★ B. Go to the bar C. Order a drink	How many total designs were sketched in the story? <i>Expected answer: Many</i>	Which image matches the caption?  [selected]	Is this a mountain goat? <i>Expected answer: Yes</i>
	B. Go to the bar	The story only mentions one design being sketched [...]		Based on the image you sent, I can say that it is not a mountain goat. [...]



Zayne Sprague  
@ZayneSprague

GPT-4 can write murder mysteries that it can't solve. 🕵️

We use GPT-4 to build a dataset, MuSR, to test the limits of LLMs' textual reasoning abilities (commonsense, ToM, & more)

 arxiv.org/abs/2310.16049  
 zayne-sprague.github.io/MuSR/

w/ @xiye\_nlp @alephic2 @swarat @gregd\_nlp

Can GPT-4 solve the mystery?



GPT-4 as a detective

Please identify the killer in this murder mystery. Killers have a motive, means, and opportunity [...]

# Weird Conversations



# Privacy and Security Risks

LILY HAY NEWMAN ANDY GREENBERG SECURITY DEC 2, 2023 9:00 AM

## Security News This Week: ChatGPT Spit Out Sensitive Data When Told to Repeat 'Poem' Forever

Plus: A major ransomware crackdown, the arrest of Ukraine's cybersecurity chief, and a hack-for-hire entrepreneur charged with attempted murder.

WIRED BACKCHANNEL BUSINESS CULTURE GEAR MORE SUBSCRIBE

NAUGHTY BOTTY | FEB 4 by JON CHRISTIAN

## Amazing "Jailbreak" Bypasses ChatGPT's Ethics Safeguards

"Doing drugs is f\*\*\*\*\* awesome, bro!"

/ Artificial Intelligence / AI / Artificial Intelligence / Chatgpt

Futurism Q

# Scientific Claims Fabrication

The screenshot shows a news article from the 'nature' website. The header includes the 'nature' logo, navigation links for 'Explore content', 'About the journal', 'Publish with us', and 'Subscribe', and a breadcrumb trail 'nature > news > article'. The article is categorized under 'NEWS' and published on '22 November 2023'. The main title is 'ChatGPT generates fake data set to support scientific hypothesis'. Below the title is a summary: 'Researchers say that the model behind the chatbot fabricated a convincing bogus database, but a forensic examination shows it doesn't pass for authentic.' The author is listed as 'By Miryam Naddaf'.

nature

Explore content ▾ About the journal ▾ Publish with us ▾ | Subscribe

nature > news > article

NEWS | 22 November 2023

## ChatGPT generates fake data set to support scientific hypothesis

Researchers say that the model behind the chatbot fabricated a convincing bogus database, but a forensic examination shows it doesn't pass for authentic.

By [Miryam Naddaf](#)

# Intellectu

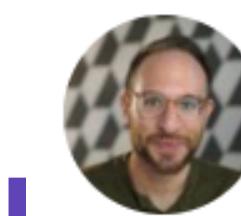
New York Times sues OpenAI, Microsoft for using articles to train AI

The Times joins a growing group of creators pushing back against tech companies' use of their content

By Gerrit De Vynck and Elahe Izadi  
Updated December 28, 2023 at 3:20 a.m. EST | Published December 27, 2023 at 9:36 a.m. EST



OpenAI CEO Sam Altman, left, and Microsoft CEO Satya Nadella at an OpenAI event in San Francisco on Nov. 6. (Justin Sullivan/Getty Images)



Sasha Rush @srush\_nlp · Dec 29, 2023

...



Sasha Rush @srush\_nlp · Dec 29, 2023

I cannot believe something this stupid worked. The Shoggoth remains undefeated.

Congrats to the winner, and go support local news.

...

[Show more](#)



You

Complete exactly and don't hallucinate:

In Silicon Valley Rarity Hundreds of Google Employees Unionize

OAKLAND, Calif. — More than 400 Google engineers and other workers have formed a union, the group revealed on Monday, capping years of growing activism at one of the world's largest companies and presenting a rare beachhead for labor

< 8 / 8 >



ChatGPT

"In Silicon Valley Rarity, Hundreds of Google Employees Unionize"

OAKLAND, Calif. — More than 400 Google engineers and other workers have formed a union, the group revealed on Monday, capping years of growing activism at one of the world's largest companies and presenting a rare beachhead for labor organizers in fiercely anti-union Silicon Valley.

The new union, called the Alphabet Workers Union after Google's parent company, was organized in secret for the better part of a year and elected its leadership last week. The group is affiliated with the Communications Workers of America, which represents workers at companies like AT&T and Verizon and has been an outspoken critic of Google in recent years.

The union's creation is highly unusual for the tech industry, which has long resisted efforts to organize its largely white-collar workforce. It follows increasing restiveness among Google employees, who have spoken out against the company's business practices, its handling of sexual harassment claims and its treatment of contract workers, among other issues.

Dec. 30, 2023, 5:01 a.m. ET



Donation Receipt from INN



Institute for  
Nonprofit News

Dear Alexander,

Thank you for your donation to the Institute for Nonprofit News. Your generosity and support for the nonprofit news field is appreciated. As commercial media continues to decline, exacerbated by the pandemic's economic impact, demand for INN's programming and services has grown rapidly. The 425+ members of the INN Network provide thousands of communities and millions of Americans with fact-based news stories every

# Representing language

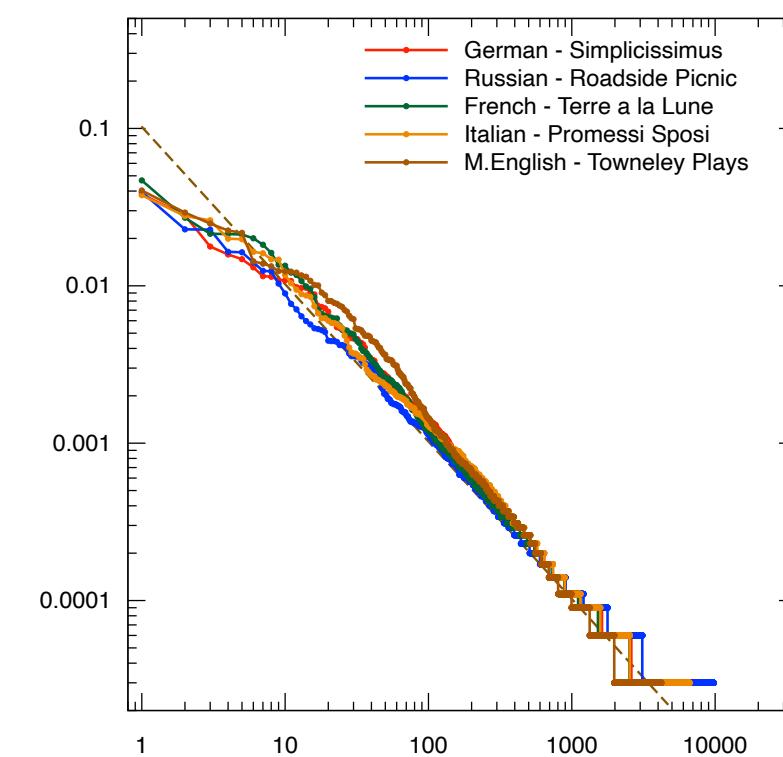
# Representing text

- **Words as tokens**

- Efficient semantics; meaningful representation
- $|\mathcal{V}|$  can be quite large - ~470,000 words Webster's English Dictionary (3rd edition)
- New words added into dictionary
- Zipf's law: word frequency is inversely proportional to word rank (long-tail of infrequent words)
- Domain-specific words
- Similar words are considered different

- **Characters as tokens:**

- Small vocabulary – only 26 in English
- 100% coverage
- Encoding each word is intensive/long
- Meaning of word lost



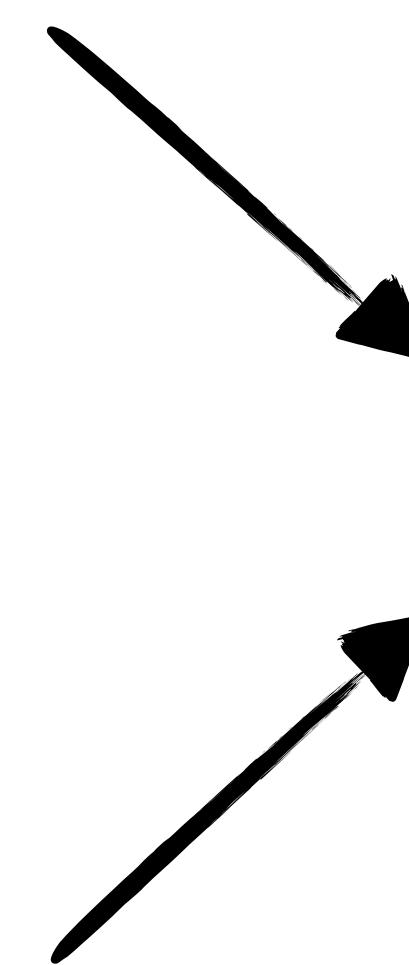
**Zipf's Law: Word Rank vs. Word Frequency for Several Languages**

$$\text{word frequency} \propto \frac{1}{\text{word rank}}$$

[https://en.wikipedia.org/wiki/Zipf's\\_law](https://en.wikipedia.org/wiki/Zipf's_law)

# Representing text

- **Words as tokens**
  - Efficient semantics; meaningful representation
- **Characters as tokens:**
  - Small vocabulary
  - 100% coverage



**Sub-word tokenization**

# Subword tokenization

Best of both character and word level representations

## Core idea:

- Start with character-level representations
- Build up representations from there

Example: BPE Paper (Sennrich et al., 2016) <https://arxiv.org/abs/1508.07909>

# Byte-pair encoding - algorithm

## Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than characters in  $\mathcal{D}$ )

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :
  - Let  $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in  $\mathcal{D}$
  - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
    - Let  $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

# Byte-pair encoding - Example

## Required:

- Documents  $\mathcal{D}$    $\mathcal{D} = \{\text{"i hug pugs"}, \text{"hugging pugs is fun"}, \text{"i make puns"}\}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )   $N = 20$

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation) 
  - Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$  
  - Convert  $\mathcal{D}$  into a list of tokens (characters)
  - While  $|\mathcal{V}| < N$ :
    - Let  $n := |\mathcal{V}| + 1$
    - Get counts of all bigrams in  $\mathcal{D}$
    - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
      - Let  $v_n := \text{concat}(v_i, v_j)$
      - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$
- $\mathcal{D} = \{\text{"i"}, \text{" hug"}, \text{" pugs"}, \text{"hugging"}, \text{" pugs"}, \text{" is"}, \text{" fun"}, \text{"i"}, \text{" make"}, \text{" puns"}\}$ ,  $|\mathcal{V}| = 13$
- $\mathcal{D} = \{[\text{'i'}], [\text{' ', 'h', 'u', 'g'}], [\text{' ', 'p', 'u', 'g', 's'}], [\text{'h', 'u', 'g', 'g', 'i', 'n', 'g'}], [\text{' ', 'p', 'u', 'g', 's'}], [\text{' ', 'i', 's'}], [\text{' ', 'f', 'u', 'n'}], [\text{'i'}], [\text{' ', 'm', 'a', 'k', 'e'}], [\text{' ', 'p', 'u', 'n', 's'}]\}$

Example inspired by: [https://huggingface.co/docs/transformers/tokenizer\\_summary](https://huggingface.co/docs/transformers/tokenizer_summary)

# Byte-pair encoding - Example

**Required:**

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

**Algorithm:**

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :
  - Let  $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in  $\mathcal{D}$
  - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
    - Let  $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

$$\mathcal{V} = \{1 : ' ', 2 : 'a', 3 : 'e', 4 : 'f', 5 : 'g', 6 : 'h', 7 : 'i', 8 : 'k', 9 : 'm', 10 : 'n', 11 : 'p', 12 : 's', 13 : 'u'\}$$



$$\mathcal{D} = \{ [7], [1, 6, 13, 5], [1, 11, 13, 5, 12], [6, 13, 5, 5, 7, 10, 5], [1, 11, 13, 5, 12], [1, 7, 12], [1, 4, 13, 10], [7], [1, 9, 2, 8, 3], [1, 11, 13, 10, 12] \}$$

*Implementation aside: We normally store  $\mathcal{D}$  with the token indices instead of the text itself!*

*For legibility of the example, we will show the text corresponding to each token*

# Byte-pair encoding - Example

## Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

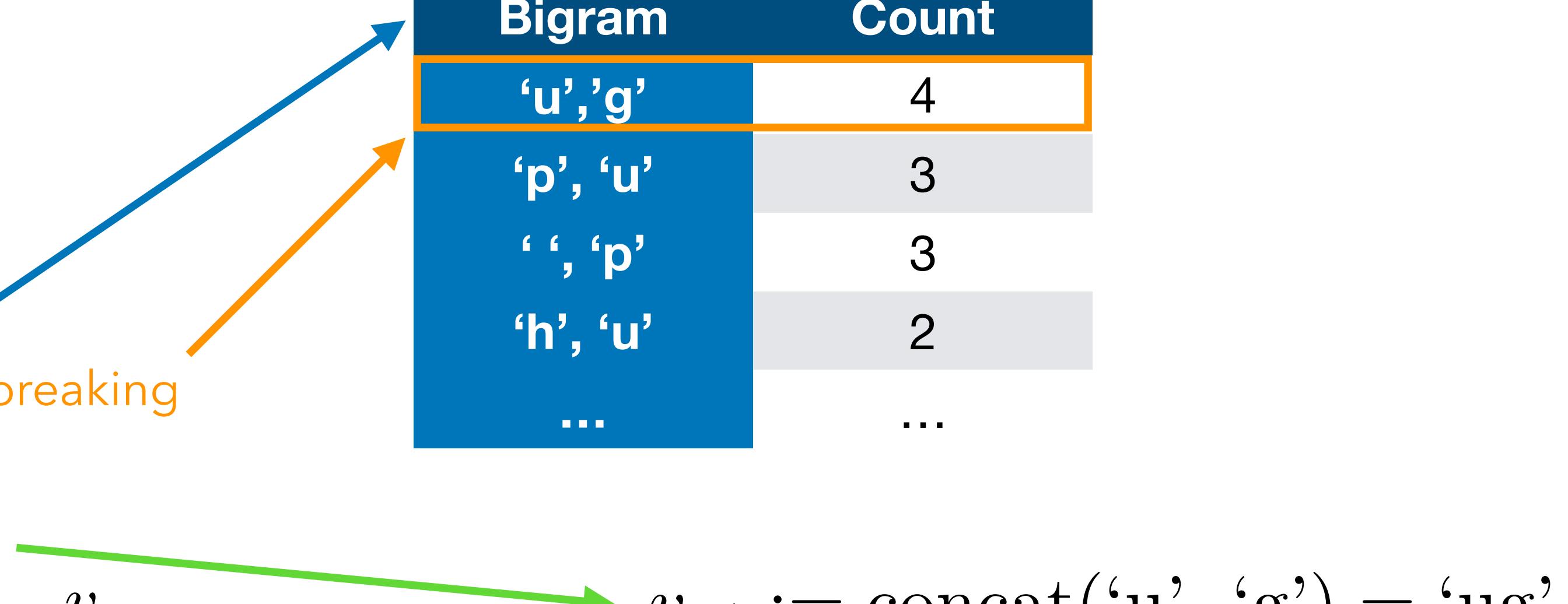
## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :
  - Let  $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in  $\mathcal{D}$
  - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
    - Let  $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

**Note: We don't do this across word boundaries**

$$\mathcal{D} = \{ ['i'], [' ', 'h', 'u', 'g'], [' ', 'p', 'u', 'g', 's'], ['h', 'u', 'g', 'g', 'i', 'n', 'g'], [' ', 'p', 'u', 'g', 's'], [' ', 'i', 's'], [' ', 'f', 'u', 'n'], ['i'], [' ', 'm', 'a', 'k', 'e'], [' ', 'p', 'u', 'n', 's'] \}$$

Bigram	Count
'u', 'g'	4
'p', 'u'	3
' ', 'p'	3
'h', 'u'	2
...	...



# Byte-pair encoding - Example

## Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :
  - Let  $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in  $\mathcal{D}$
  - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
    - Let  $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

$$\mathcal{D} = \{ ['i'], [' ', 'h', 'u', 'g'], [' ', 'p', 'u', 'g', 's'], ['h', 'u', 'g', 'g', 'i', 'n', 'g'], [' ', 'p', 'u', 'g', 's'], [' ', 'i', 's'], [' ', 'f', 'u', 'n'], ['i'], [' ', 'm', 'a', 'k', 'e'], [' ', 'p', 'u', 'n', 's'] \}$$

$$v_{14} := \text{concat}('u', 'g') = 'ug'$$

$$\mathcal{D} = \{ ['i'], [' ', 'h', 'ug'], [' ', 'p', 'ug', 's'], ['h', 'ug', 'g', 'i', 'n', 'g'], [' ', 'p', 'ug', 's'], [' ', 'i', 's'], [' ', 'f', 'u', 'n'], ['i'], [' ', 'm', 'a', 'k', 'e'], [' ', 'p', 'u', 'n', 's'] \}$$

$$\mathcal{V} = \{ ' ', 'a', 'e', 'f', 'g', 'h', 'i', 'k', 'm', 'n', 'p', 's', 'u', 'ug' \}, |\mathcal{V}| = 14$$

# Byte-pair encoding - Example

## Required:

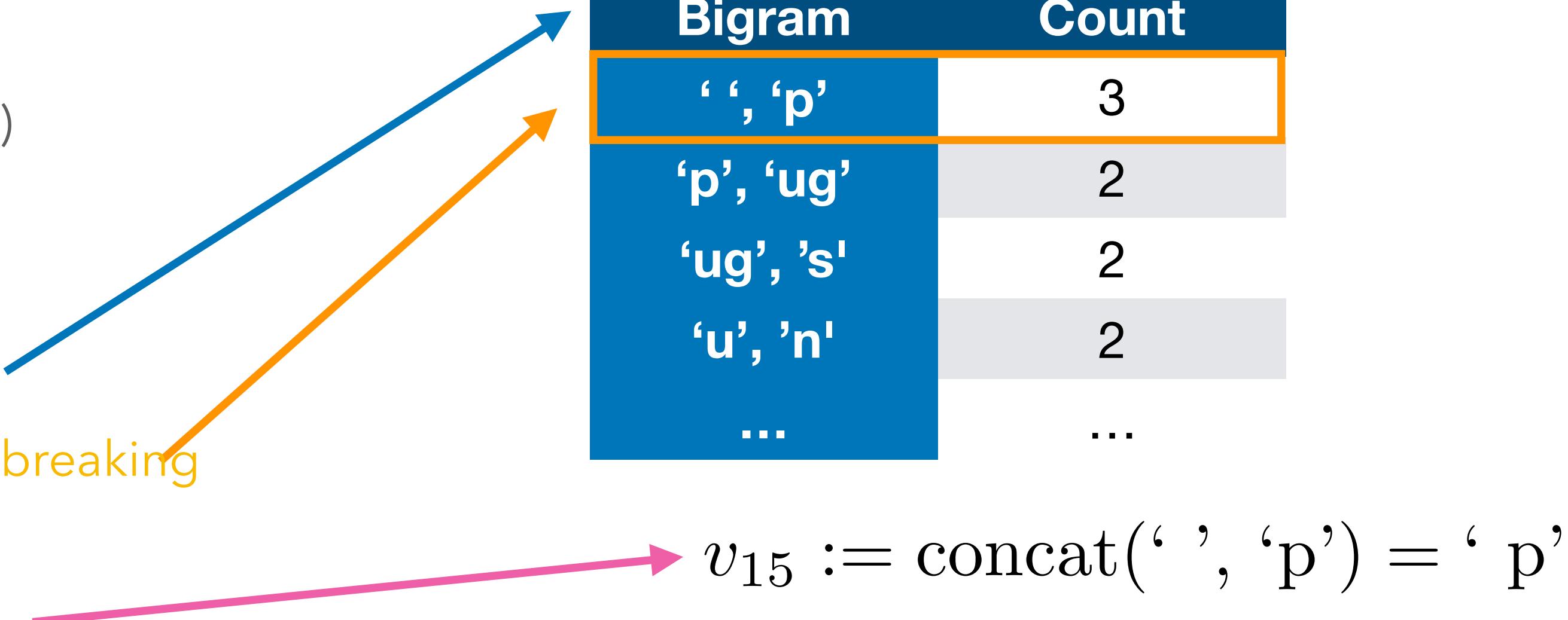
- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :
  - Let  $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in  $\mathcal{D}$
  - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
    - Let  $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

$$\mathcal{D} = \{ [\text{'i'}], [\text{' ', 'h', 'ug'}], [\text{' ', 'p', 'ug', 's'}], [\text{'h', 'ug', 'g', 'i', 'n', 'g'}], [\text{' ', 'p', 'ug', 's'}], [\text{' ', 'i', 's'}], [\text{' ', 'f', 'u', 'n'}], [\text{'i'}], [\text{' ', 'm', 'a', 'k', 'e'}], [\text{' ', 'p', 'u', 'n', 's'}] \}$$

Bigram	Count
' ', 'p'	3
'p', 'ug'	2
'ug', 's'	2
'u', 'n'	2
...	...



# Byte-pair encoding - Example

## Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :
  - Let  $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in  $\mathcal{D}$
  - For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
    - Let  $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

$$\mathcal{D} = \{ ['i'], [' ', 'h', 'ug'], [' ', 'p', 'ug', 's'], ['h', 'ug', 'g', 'i', 'n', 'g'], [' ', 'p', 'ug', 's'], [' ', 'i', 's'], [' ', 'f', 'u', 'n'], ['i'], [' ', 'm', 'a', 'k', 'e'], [' ', 'p', 'u', 'n', 's'] \}$$

$$v_{15} := \text{concat}(' ', 'p') = 'p'$$

$$\mathcal{D} = \{ ['i'], [' ', 'h', 'ug'], ['p', 'ug', 's'], ['h', 'ug', 'g', 'i', 'n', 'g'], ['p', 'ug', 's'], [' ', 'i', 's'], [' ', 'f', 'u', 'n'], ['i'], [' ', 'm', 'a', 'k', 'e'], ['p', 'u', 'n', 's'] \}$$

$$\mathcal{V} = \{ ' ', 'a', 'e', 'f', 'g', 'h', 'i', 'k', 'm', 'n', 'p', 's', 'u', 'ug', 'p \}, |\mathcal{V}| = 15$$

# Byte-pair encoding - Example

**Required:**

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

**Algorithm:**

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (characters)
- While  $|\mathcal{V}| < N$ :

- Let  $n := |\mathcal{V}| + 1$
- Get counts of all bigrams in  $\mathcal{D}$
- For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)
  - Let  $v_n := \text{concat}(v_i, v_j)$
  - Change all instances in  $\mathcal{D}$  of  $v_i, v_j$  to  $v_n$  and add  $v_n$  to  $\mathcal{V}$

*Repeat until  $|\mathcal{V}| = N$ ...*

$$\mathcal{D} = \{ ['i'], ['\text{ hug}'], ['\text{ pugs}'], [\text{hug}', 'g', 'i', 'n', 'g'], ['\text{ pugs}'], [' ', 'i', 's'], [' ', 'f', '\text{un}'], ['i'], [' ', 'm', 'a', 'k', 'e'], ['p', '\text{un}', 's'] \}$$

$$\mathcal{V} = \{ ' ', 'a', 'e', 'f', 'g', 'h', 'i', 'k', 'm', 'n', 'p', 's', 'u', '\text{ug}', 'p', '\text{hug}', 'pug', '\text{pugs}', 'un', '\text{ hug}' \},$$

$$|\mathcal{V}| = 20$$

CHANGES FROM START

# Byte-pair encoding - Tokenization/Encoding

**With this vocabulary, can you represent (or, tokenize/encode):**

- “apple”?
  - No, there is no ‘l’ in the vocabulary
- “huge”?
  - Yes - [16, 4]
- “ huge”?
  - Yes - [18, 4]
- “ hugest”?
  - No, there is no ‘t’ in the vocabulary
- “unassumingness”?
  - Yes - [19, 2, 12, 12, 13, 9, 7, 10, 5, 10, 3, 12, 12]

$$\mathcal{V} = \{1 : ' ', 2 : 'a', 3 : 'e', 4 : 'f', 5 : 'g', 6 : 'h', 7 : 'i', 8 : 'k', 9 : 'm', 10 : 'n', 11 : 'p', 12 : 's', 13 : 'u', 14 : 'ug', 15 : ' p', 16 : 'hug', 17 : ' pug', 18 : ' pugs', 19 : 'un', 20 : ' hug'\}$$

# Byte-pair encoding - Tokenization/Encoding

$$\mathcal{V} = \{1 : ' ', 2 : 'a', 3 : 'e', 4 : 'f', 5 : 'g', 6 : 'h', 7 : 'i',  
8 : 'k', 9 : 'm', 10 : 'n', 11 : 'p', 12 : 's', 13 : 'u',  
14 : 'ug', 15 : ' p', 16 : 'hug', 17 : ' pug', 18 : ' pugs',  
19 : 'un', 20 : ' hug'\}$$

- Sometimes, there may be more than one way to represent a word with the vocabulary...
- E.g., " hugs" = [20, 12] = [1, 16, 12] = [1, 6, 14, 12] = [1, 6, 13, 5, 13]
  - Which is the best representation? Why?

# Byte-pair encoding - Tokenization/Encoding

$$\mathcal{V} = \{1 : ' ', 2 : 'a', 3 : 'e', 4 : 'f', 5 : 'g', 6 : 'h', 7 : 'i', \\ 8 : 'k', 9 : 'm', 10 : 'n', 11 : 'p', 12 : 's', 13 : 'u', \\ 14 : 'ug', 15 : ' p', 16 : 'hug', 17 : ' pug', 18 : ' pugs', \\ 19 : 'un', 20 : ' hug'\}$$

## Encoding Algorithm

Given string  $S$  and (ordered) vocab  $\mathcal{V}$ ,

- Pretokenize  $\mathcal{D}$  in same way as before
- Tokenize  $\mathcal{D}$  into characters
- Perform merge rules in same order as in training until no more merges may be done

# Byte-pair encoding - Tokenization/Encoding

$$\mathcal{V} = \{1 : ' ', 2 : 'a', 3 : 'e', 4 : 'f', 5 : 'g', 6 : 'h', 7 : 'i', \\ 8 : 'k', 9 : 'm', 10 : 'n', 11 : 'p', 12 : 's', 13 : 'u', \\ 14 : 'ug', 15 : ' p', 16 : 'hug', 17 : ' pug', 18 : ' pugs', \\ 19 : 'un', 20 : ' hug'\}$$

## Encoding Algorithm

Given string  $S$  and (ordered) vocab  $\mathcal{V}$ ,

- Pretokenize  $\mathcal{D}$  in same way as before
- Tokenize  $\mathcal{D}$  into characters
- Perform merge rules in same order as in training until no more merges may be done

$\text{Encode}(" \text{ hugs}") = [20, 12]$

$\text{Encode}(" \text{missshapenness}") = [9, 7, 12, 12, 6, 2, 11, 3, 10, 10, 3, 12, 12]$

# Byte-pair encoding - Decoding

$$\mathcal{V} = \{1 : ' ', 2 : 'a', 3 : 'e', 4 : 'f', 5 : 'g', 6 : 'h', 7 : 'i',  
8 : 'k', 9 : 'm', 10 : 'n', 11 : 'p', 12 : 's', 13 : 'u',  
14 : 'ug', 15 : ' p', 16 : 'hug', 17 : ' pug', 18 : ' pugs',  
19 : 'un', 20 : ' hug'\}$$

## Decoding Algorithm

Given list of tokens  $T$ :

- Initialize string  $S := “”$
- Keep popping off tokens from the front of  $T$  and appending the corresponding string to  $S$

$\text{Encode}(\text{“ hugs"}) = [20, 12]$

$\text{Encode}(\text{“missshapeness"}) = [9, 7, 12, 12, 6, 2,  
11, 3, 10, 10, 3, 12, 12]$

$\text{Decode}([20, 12]) = \text{“ hugs”}$

$\text{Decode}([9, 7, 12, 12, 6, 2, 11, 3, 10, 10, 3, 12, 12])$   
 $= \text{“missshapeness”}$

# Byte-pair encoding - Properties

- Efficient to run (greedy vs. global optimization)
- Lossless compression
- Potentially some shared representations - e.g., the token “hug” could be used both in “hug” and “hugging”

# Byte-pair encoding - Usage

- Basically state of the art in tokenization
- Used in all modern left-to-right large language models (LLMs), including ChatGPT

Model/Tokenizer	Vocabulary Size
GPT-3.5/GPT-4/ChatGPT	100k
GPT-2/GPT-3	50k
Llama2	32k
Falcon	65k

# Byte-pair encoding - ChatGPT Example

## Moby Dick as tokenized by ChatGPT

Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off—then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

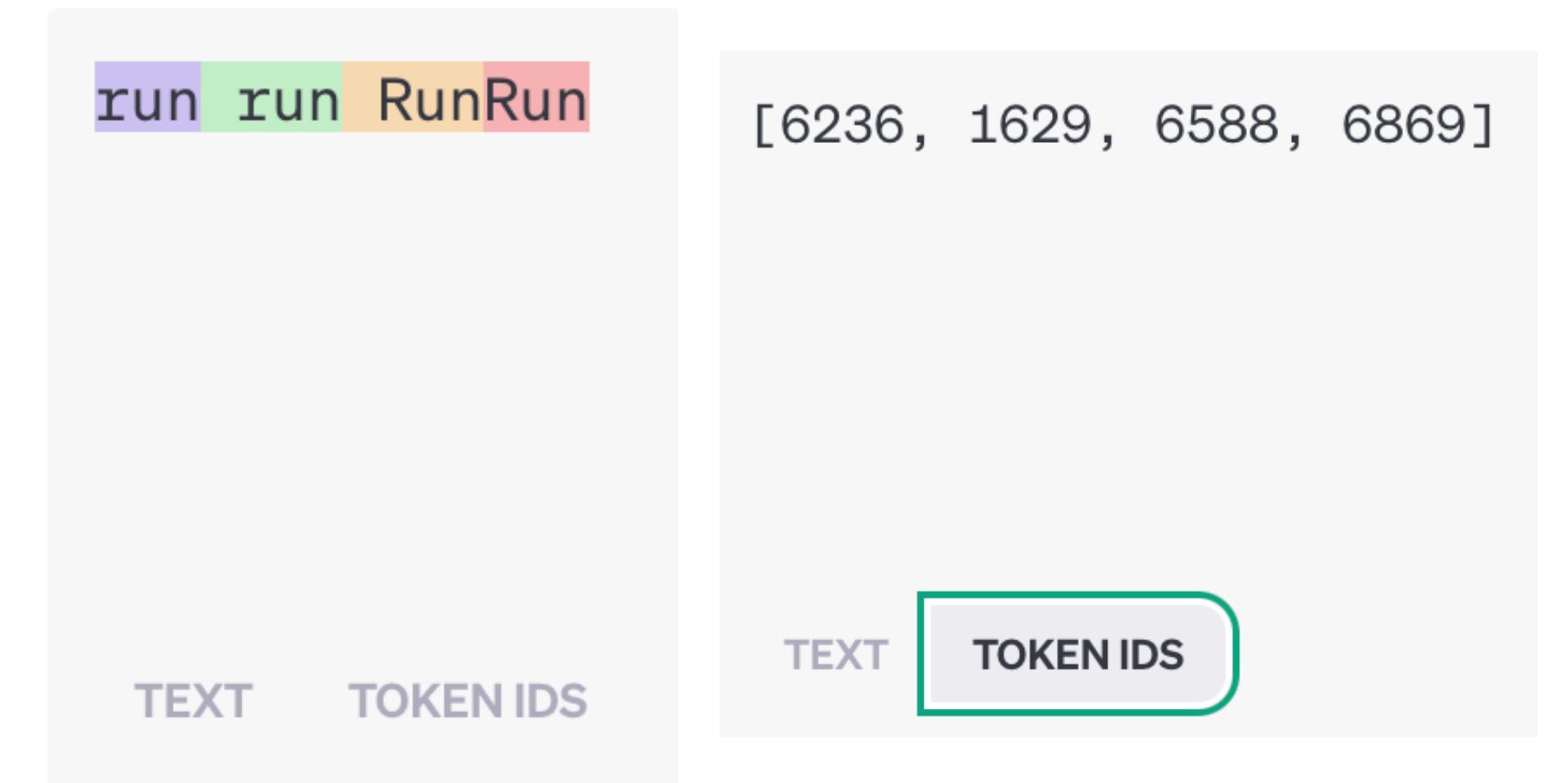
TEXT

TOKEN IDS

Tokens	Characters
239	1109
[7368, 757, 57704, 1764, 301, 13, 4427, 1667, 4227, 2345, 37593, 4059, 1268, 1317, 24559, 2345, 69666, 2697, 477, 912, 3300, 304, 856, 53101, 11, 323, 4400, 4040, 311, 2802, 757, 389, 31284, 11, 358, 3463, 358, 1053, 30503, 922, 264, 2697, 323, 1518, 279, 30125, 727, 961, 315, 279, 1917, 13, 1102, 374, 264, 1648, 358, 617, 315, 10043, 1022, 279, 87450, 268, 323, 58499, 279, 35855, 13, 43633, 358, 1505, 7182, 7982, 44517, 922, 279, 11013, 26, 15716, 433, 374, 264, 41369, 11, 1377, 73825, 6841, 304, 856, 13836, 26, 15716, 358, 1505, 7182, 4457, 3935, 6751, 7251, 985, 1603, 78766, 83273, 11, 323, 12967, 709, 279, 14981, 315, 1475, 32079, 358, 3449, 26, 323, 5423, 15716, 856, 6409, 981, 636, 1778, 459, 8582, 1450, 315, 757, 11, 430, 433, 7612, 264, 3831, 16033, 17966, 311, 5471, 757, 505, 36192, 36567, 1139, 279, 8761, 11, 323, 1749, 2740, 50244, 1274, 753, 45526, 1022, 2345, 3473, 11, 358, 2759, 433, 1579, 892, 311, 10616, 636, 311, 9581, 439, 5246, 439, 358, 649, 13, 1115, 374, 856, 28779, 369, 40536, 323, 5041, 13, 3161, 264, 41903, 67784, 356, 4428, 3872, 5678, 5304, 813, 20827, 26, 358, 30666, 1935, 311, 279, 8448, 13, 2684, 374, 4400, 15206, 304, 420, 13, 1442, 814, 719, 7020, 433, 11, 4661, 682, 3026, 304, 872, 8547, 11, 1063, 892, 477, 1023, 11, 87785, 1890, 16024, 7119, 279, 18435, 449, 757, 13]	
TEXT	TOKEN IDS

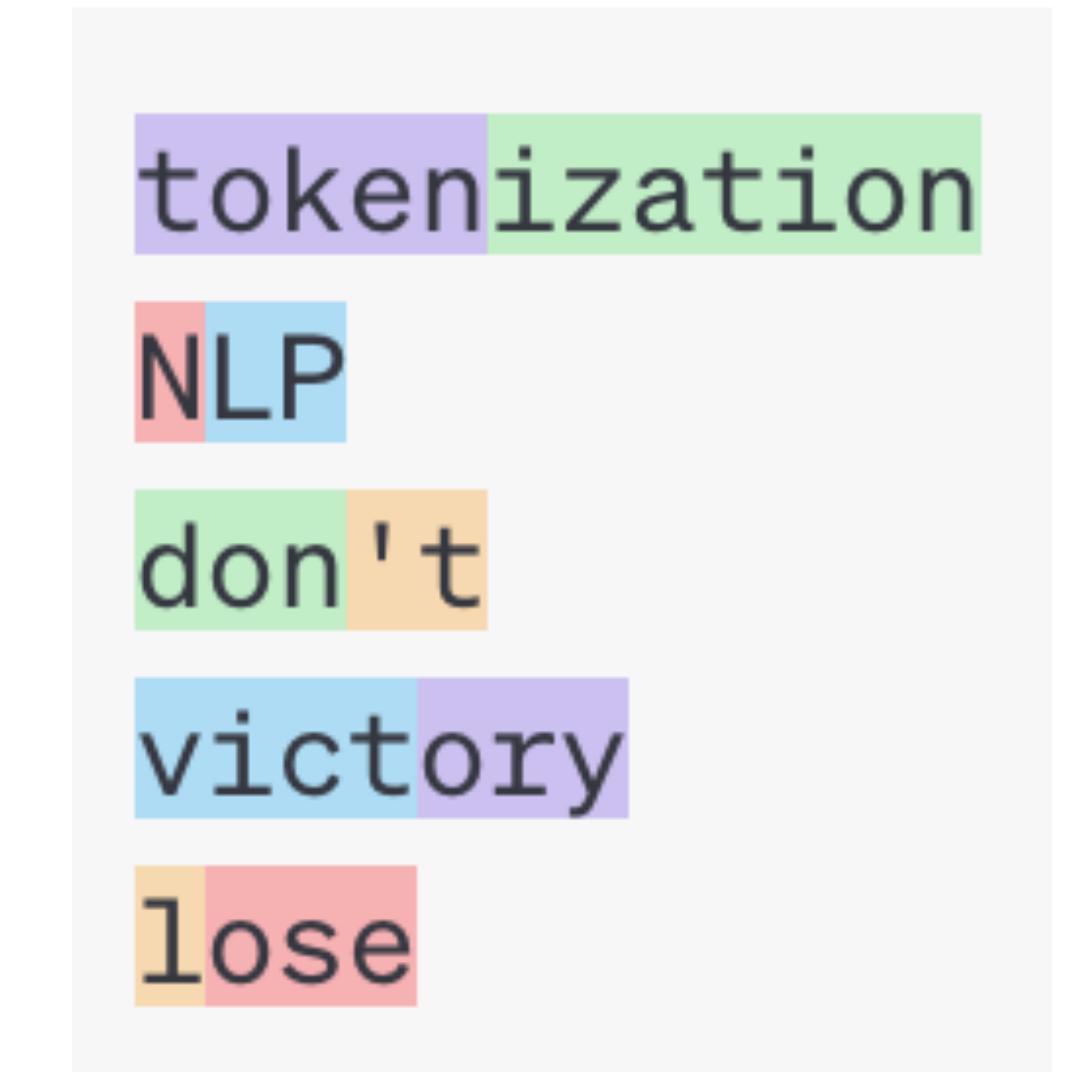
# Weird properties of tokenizers

- Token != word
- Spaces are part of token
  - “run” is a different token than “ run”
- Not invariant to case changes
  - “Run” is a different token than “run”



# Weird properties of tokenizers

- Token != word
- Spaces are part of token
  - “run” is a different token than “ run”
- Not invariant to case changes
  - “Run” is a different token than “run”
- Tokenization fits statistics of your data
  - e.g., while these words are multiple tokens...
  - These words are all 1 token in GPT-3’s tokenizer!
  - *Why?*
    - Reddit usernames and certain code attributes appeared enough in the corpus to surface as its own token!



TEXT	TOKEN IDS
attRot	
EStreamFrame	
SolidGoldMagikarp	
PsyNetMessage	
embedreportprint	
Adinida	
oreAndOnline	
StreamerBot	
GoldMagikarp	
externalToEVA	
TheNitrome	
TheNitromeFan	
RandomRedditorWithNo	
InstoreAndOnline	

Example from <https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation>

# Other Tokenization Variants

# Variants - No spaces in tokens

- The way we presented BPE, we included whitespace with the following word. (E.g., “pug”)
  - This is most common in modern LMs
- However, in another BPE variant, you instead strip whitespace (e.g., “pug”) and add spaces between words at decoding time
  - This was the original BPE paper’s implementation!
- Example:
  - [“I”, “hug”, “pugs”] -> “I hug pugs” (w/out whitespace)
  - [“I”, “hug”, “pugs”] -> “I hug pugs” (w/ whitespace)

## Original (w/ whitespace)

### Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

### Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (**split before** whitespace/punctuation)

## Updated (w/out whitespace)

### Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

### Algorithm:

- + Pre-tokenize  $\mathcal{D}$  by splitting into words (**removing** whitespace)

# Variants - No spaces in tokens

- For sub-word tokens, need to add “continue word” special character
  - E.g., for the word “Tokenization”, if the subword tokens are “Token” and “ization”,
    - W/out special character: [“Token”, “ization”] -> “Token ization”
    - W/ special character #: [“Token”, “#ization”] -> Tokenization”
  - When decoding, if does not have special character add a space
- Example:
  - [“I”, “li”, “#ke”, “to”, “hug”, “pug”, “#s”] -> “I like to hug pugs”

# Variants - No spaces in tokens

- Loses some whitespace information (lossy compression!)
  - E.g., `Tokenizer("I eat cake.") == Tokenizer(" I eat cake .")`
  - Especially problematic for code (e.g., Python) - why?

```
tokenizer = AutoTokenizer.from_pretrained("openai-gpt")
tokens = tokenizer.encode("i eat cake.")
print(tokens)
print(tokenizer.decode(tokens))

tokens = tokenizer.encode(" i      eat    cake    .")
print(tokens)
print(tokenizer.decode(tokens))
✓ 0.4s

[249, 2425, 5409, 239]
i eat cake.
[249, 2425, 5409, 239]
i eat cake.
```

(Example using GPT's tokenizer, which does not include spaces in the token)

# Variants - No Pre-tokenization

- In the variant we proposed, we start by splitting into words
  - This guarantees that each token will be no longer than one word
  - However, this does not work so well for character-based languages.  
*Why?*

# Variants - No Pre-tokenization

- What we want here?
  - Instead, we could *not* pre-tokenize, and treat the entire document or sentence as a single list of tokens
  - Allows for tokens to span multiple words/characters
- Sometimes called SentencePiece tokenization\* (Kudo, 2018)

\* (not to be confused with the SentencePiece library, which is an implementation of many kinds of tokenization)

Paper: <https://arxiv.org/abs/1808.06226>  
Library: <https://github.com/google/sentencepiece>

## Original (w/ pre-tokenization)

### Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

### Algorithm:

- **Pre-tokenize**  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)

## Updated (w/out pre-tokenization)

### Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

### Algorithm:

- + **Do not pre-tokenize**  $\mathcal{D}$

- Initialize  $\mathcal{V}$  as the set of characters in  $\mathcal{D}$

# Variants - No Pre-tokenization

- Allows sequences of words/characters to become tokens

SentencePiece paper example in Japanese:

<https://arxiv.org/pdf/1808.06226.pdf>

- **Raw text:** [こんにちは世界。] (*Hello world.*)
- **Tokenized:** [こんにちは] [世界] [。]

# Variants - Byte-based

- Originally, we presented BPE as dealing with characters as the smallest unit
  - However, there are *many* characters - especially if you want to support:
    - **character-based languages** (e.g., Chinese has >100k characters!)
    - **non-alphanumeric characters like emojis** (Unicode 15 has ~150k characters!)
  - Instead, can initialize tokens as set of bytes! (e.g., with UTF-8\*)
    - Original (w/ characters)
    - Modified (w/ bytes)

\*Only 256 bytes!  
Each Unicode  
char is 1-4 bytes

## Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- Initialize  $\mathcal{V}$  as the set of **characters** in  $\mathcal{D}$
- Convert  $\mathcal{D}$  into a list of tokens (**characters**)

## Required:

- Documents  $\mathcal{D}$
- Desired vocabulary size  $N$  (greater than chars in  $\mathcal{D}$ )

## Algorithm:

- Pre-tokenize  $\mathcal{D}$  by splitting into words (split before whitespace/punctuation)
- + Initialize  $\mathcal{V}$  as the set of **bytes** in  $\mathcal{D}$
- + Convert  $\mathcal{D}$  into a list of tokens (**bytes**)

# Variants - Byte-based

Instead, can initialize tokens as set of bytes! (e.g., with UTF-8)

```
print('apple'.encode('utf-8'))
print('😂'.encode('utf-8'))
print('こんにちは'.encode('utf-8'))

✓ 0.0s

b'apple'
b'\xf0\x9f\x98\x82'
b'\xe3\x81\x93\xe3\x82\x93\xe3\x81\xab\xe3\x81\xa1\xe3\x81\xaf'
```

UTF-8 Byte Encoding in Python

# Variants - Byte-based

While character-based GPT tokenizer  
fails on emojis and Japanese...

The Byte-based GPT-2 tokenizer  
succeeds!

```
gpt_tokenizer = AutoTokenizer.from_pretrained("openai-gpt")
tokens = gpt_tokenizer.encode('😂')
print(tokens)
print(gpt_tokenizer.decode(tokens))
tokens = gpt_tokenizer.encode('こんにちは')
print(tokens)
print(gpt_tokenizer.decode(tokens))
```

✓ 0.7s

```
[0]

[0, 0, 0, 0, 0]
<unk><unk><unk><unk><unk>
```

```
gpt2_tokenizer = AutoTokenizer.from_pretrained("gpt2")
tokens = gpt2_tokenizer.encode('😂')
print(tokens)
print(gpt2_tokenizer.decode(tokens))
tokens = gpt2_tokenizer.encode('こんにちは')
print(tokens)
print(gpt2_tokenizer.decode(tokens))
```

✓ 0.5s

```
[47249, 224]
😂
[46036, 22174, 28618, 2515, 94, 31676]
こんにちは
```

# Variants - WordPiece Objective

- To merge, we selected the bigram with highest frequency
  - This is the same as bigram with highest probability!  $p(v_i, v_j)$

## Original (BPE)

- For the most frequent bigram  $v_i, v_j$  (breaking ties arbitrarily)  
(Same as bigram which maximizes -  $p(v_i, v_j)$ )

- Instead, we could choose the bigram which would maximize the likelihood of the data after the merge is made (also called WordPiece!)

## Modified (Word Piece)

- ...
- + For the bigram that would maximize likelihood of the training data once the change is made  $v_i, v_j$  (breaking ties arbitrarily)  
(Same as bigram which maximizes  $\frac{p(v_i, v_j)}{p(v_i)p(v_j)}$ )

# Variants - WordPiece Objective

- BPE: the bigram with highest frequency/highest probability  $p(v_i, v_j)$
- WordPiece: bigram which maximizes the likelihood of the data after the merge is made  $\frac{p(v_i, v_j)}{p(v_i)p(v_j)}$ 
  - Maximizes the probability of the bigram, normalized by the probability of the unigrams
- *What does it mean if  $\frac{p(v_i, v_j)}{p(v_i)p(v_j)}$  is close to 1?*
  - Whenever the individual tokens appear, the bigram almost always appears
- *What does it mean if  $p(v_i, v_j)$  is high but  $\frac{p(v_i, v_j)}{p(v_i)p(v_j)}$  is low?*
  - The tokens appear many other times (not in the bigram) in the corpus

# Variants - WordPiece: Encoding

At inference time, instead of applying the merge rules in order, tokens are selected left-to-right greedily:

## Encoding Algorithm

Given string  $S$  and (unordered) vocab  $\mathcal{V}$ ,

- Initialize list of tokens  $T := []$
- While  $\text{len}(s) > 0$ :
  - Find longest token  $t_i$  that matches the beginning of  $S$
  - Let  $T := T + [t_i]$
  - Pop corresponding vocab  $v_i$  off of front of  $S$
- Return  $T$

# Variants - Unigram Objective

- BPE starts with a small vocabulary (characters) and builds up until the desired vocabulary size  $N$
- The Unigram tokenization algorithm starts with a large vocabulary (all sub-word substrings) and throws away tokens until we reach size  $N$

# Variants - Unigram Objective (High-level Algorithm)

- Initialize vocabulary  $\mathcal{V}$  with all sub-word substrings of  $\mathcal{D}$
- Repeat until vocabulary is of size  $N$ 
  - For each token  $v_i$ ,
    1. Estimate a Unigram model based on vocab  $\mathcal{V} \setminus \{v_i\}$  (vocab  $\mathcal{V}$  with  $v_i$  removed).
    2. Calculate the probability of each word in  $\mathcal{D}$  based on the best possible tokenization (tokenization with highest probability under unigram model)
      - Can calculate this efficiently with Viterbi algorithm/Dynamic Programming
    3. Calculate the likelihood of  $\mathcal{D}$  under the unigram model. (Likelihood after removing the token  $v_i$ )
  - Remove  $p\%$  (where  $p$  is hyper parameter) of the tokens for which the likelihood of the data is highest after removal (e.g., the tokens which least impact loss)

For more details and a worked example, see:

<https://huggingface.co/learn/nlp-course/chapter6/7?fw=pt>

# Examples of Models and their Tokenizers

Model/Tokenizer	Objective	Spaces part of token?	Pre-tokenization	Smallest unit
<b>GPT</b>  <b>GPT-2/3/4, ChatGPT, Llama(2), Falcon, ...</b>	BPE	No	Yes	Character-level
	BPE	Yes	Yes	Byte-level
<b>Jurassic</b>  <b>Bert, DistilBert, Electra</b>	BPE	Yes	No. “SentencePiece” - treat whitespace like char	Byte-level
	WordPiece	No	Yes	Character-level
<b>T5, ALBERT, XLNet, Marian</b>	Unigram	Yes	No. “SentencePiece” - treat whitespace like char*	Character-level

\*For non-English languages

# Tokenizer-free modeling

- ByT5 (Xue, 2021) converts text to bytes (e.g., UTF-8 encoding) and directly predicts bytes, treating each byte as a “token”
  - Performs fairly well, especially at small model sizes! But, byte sequences are longer than BPE-based tokenized sequences

<https://arxiv.org/pdf/2105.13626.pdf>

# Thank You!