CS324

Lectures / Introduction

Welcome to CS324! This is a new course on understanding and developing **large language models**.

What is a language model?

The classic definition of a language model (LM) is a **probability distribution over sequences of tokens**. Suppose we have a **vocabulary** $\square$ of a set of tokens. A language model p assigns each sequence of tokens $x_1, \dots, x_L \in \square$ a probability (a number between 0 and 1):

$$p(x_1, \dots, x_L).$$

The probability intuitively tells us how "good" a sequence of tokens is. For example, if the vocabulary is $\square = \{\text{ate}, \text{ball}, \text{cheese}, \text{mouse}, \text{the}\}$, the language model might assign (demo):

$$p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = 0.02,$$

$$p(\text{the}, \text{cheese}, \text{ate}, \text{the}, \text{mouse}) = 0.01,$$

$$p(\text{mouse}, \text{the}, \text{the}, \text{cheese}, \text{ate}) = 0.0001.$$

Mathematically, a language model is a very simple and beautiful object. But the simplicity is deceiving: the ability to assign (meaningful) probabilities to all sequences requires extraordinary (but *implicit*) linguistic abilities and world knowledge.

For example, the LM should assign $\text{mouse the the cheese ate}$ a very low probability implicitly because it's ungrammatical (**syntactic knowledge**). The LM should assign $\text{the mouse ate the cheese}$ higher probability than $\text{the cheese ate the mouse}$ implicitly because of **world knowledge**: both sentences are the same syntactically, but they differ in semantic plausibility.

Generation. As defined, a language model p takes a sequence and returns a probability to assess its goodness. We can also generate a sequence given a language model. The purest way to do this is to sample a sequence $x_{1:L}$ from the language model p with probability equal to $p(x_{1:L})$, denoted:

$$x_{1:L} \sim p.$$

How to do this computationally efficiently depends on the form of the language model p. In practice, we do not generally sample directly from a language model both because of limitations of real language models and because we sometimes wish to obtain not an “average” sequence but something closer to the “best” sequence.

Autoregressive language models

A common way to write the joint distribution $p(x_{1:L})$ of a sequence $x_{1:L}$ is using the **chain rule of probability**:

$$p(x_{1:L}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1}).$$

For example (demo):

$$\begin{aligned} p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = & \, p(\text{the}) \\ & p(\text{mouse} \mid \text{the}) \\ & p(\text{ate} \mid \text{the}, \text{mouse}) \\ & p(\text{the} \mid \text{the}, \text{mouse}, \text{ate}) \\ & p(\text{cheese} \mid \text{the}, \text{mouse}, \text{ate}, \text{the}). \end{aligned}$$

In particular, $p(x_i \mid x_{1:i-1})$ is a **conditional probability distribution** of the next token x_i given the previous tokens $x_{1:i-1}$.

Of course, any joint probability distribution can be written this way mathematically, but an **autoregressive language model** is one where each conditional distribution $p(x_i \mid x_{1:i-1})$ can be computed efficiently (e.g., using a feedforward neural network).

Generation. Now to generate an entire sequence $x_{1:L}$ from an autoregressive language model p, we sample one token at a time given the tokens generated so far:

$$\begin{aligned} & \text{for } i = 1, \dots, L: \\ & \quad x_i \sim p(x_i \mid x_{1:i-1})^{1/T}, \end{aligned}$$

where $T \geq 0$ is a **temperature** parameter that controls how much randomness we want from the language model:

- $T = 0$: deterministically choose the most probable token x_i at each position i
- $T = 1$: sample “normally” from the pure language model
- $T = \infty$: sample from a uniform distribution over the entire vocabulary ☐

However, if we just raise the probabilities to the power $1/T$, the probability distribution may not sum to 1. We can fix this by re-normalizing the distribution. We call the normalized version $p_T(x_i \mid x_{1:i-1}) \propto p(x_i \mid x_{1:i-1})^{1/T}$ the **annealed** conditional probability distribution. For example:

$$\begin{aligned} p(\text{cheese}) &= 0.4, & p(\text{mouse}) &= 0.6 \\ p_{T=0.5}(\text{cheese}) &= 0.31, & p_{T=0.5}(\text{mouse}) &= 0.69 \\ p_{T=0.2}(\text{cheese}) &= 0.12, & p_{T=0.2}(\text{mouse}) &= 0.88 \\ p_{T=0}(\text{cheese}) &= 0, & p_{T=0}(\text{mouse}) &= 1 \end{aligned}$$

Aside: Annealing is a reference to metallurgy, where hot materials are cooled gradually, and shows up in sampling and optimization algorithms such as simulated annealing.

Technical note: sampling iteratively with a temperature T parameter applied to each conditional distribution $p(x_i \mid x_{1:i-1})^{1/T}$ is not equivalent (except when $T = 1$) to sampling from the annealed distribution over length L sequences.

Conditional generation. More generally, we can perform conditional generation by specifying some prefix sequence $x_{1:i}$ (called a **prompt**) and sampling the rest $x_{i+1:L}$ (called the **completion**). For example, generating with $T = 0$ produces (demo):

$$\underbrace{\text{the}, \text{mouse}, \text{ate}}_{\text{prompt}} \overset{T=0}{\leadsto} \underbrace{\text{the}, \text{cheese}}_{\text{completion}}$$

If we change the temperature to $T = 1$, we can get more variety (demo), for example, its house and my homework.

As we'll see shortly, conditional generation unlocks the ability for language models to solve a variety of tasks by simply changing the prompt.

Summary

- A language model is a probability distribution p over sequences $x_{1:L}$.
- Intuitively, a good language model should have linguistic capabilities and world knowledge.
- An autoregressive language model allows for efficient generation of a completion $x_{i+1:L}$ given a prompt $x_{1:i}$.
- The temperature can be used to control the amount of variability in generation.

A brief history

Information theory, entropy of English, n-gram models

Information theory. Language models date back to Claude Shannon, who founded information theory in 1948 with his seminal paper, A Mathematical Theory of Communication. In this paper, he introduced the **entropy** of a distribution as

$$H(p) = \sum_x p(x) \log \frac{1}{p(x)}.$$

The entropy measures the expected number of bits **any algorithm** needs to encode (compress) a sample $x \sim p$ into a bitstring:

$$\text{the mouse ate the cheese} \Rightarrow 0001110101.$$

- The lower the entropy, the more "structured" the sequence is, and the shorter the code length.
- Intuitively, $\log \frac{1}{p(x)}$ is the length of the code used to represent an element x that occurs with probability $p(x)$.
- If $p(x) = \frac{1}{8}$, we should allocate $\log_2(8) = 3$ bits (equivalently, $\log(8) = 2.08$ nats).

Aside: actually achieving the Shannon limit is non-trivial (e.g., LDPC codes) and is the topic of coding theory.

Entropy of English. Shannon was particularly interested in measuring the entropy of English, represented as a sequence of letters. This means we imagine that there is a "true" distribution p out there (the existence of this is questionable, but it's still a useful mathematical abstraction) that can spout out samples of English text $x \sim p$.

Shannon also defined **cross entropy**:

$$H(p, q) = \sum_x p(x) \log \frac{1}{q(x)},$$

which measures the expected number of bits (nats) needed to encode a sample $x \sim p$ using the compression scheme given by the model q (representing x with a code of length $\frac{1}{q(x)}$).

Estimating entropy via language modeling. A crucial property is that the cross entropy $H(p, q)$ upper bounds the entropy $H(p)$,

$$H(p, q) \geq H(p),$$

which means that we can estimate $H(p, q)$ by constructing a (language) model q with only samples from the true data distribution p, whereas $H(p)$ is generally inaccessible if p is English.

So we can get better estimates of the entropy $H(p)$ by constructing better models q, as measured by $H(p, q)$.

Shannon game (human language model). Shannon first used n-gram models as q in 1948, but in his 1951 paper Prediction and Entropy of Printed English, he introduced a clever scheme (known as the Shannon game) where q was provided by a human:

$$\text{the mouse ate my ho\_}$$

Humans aren't good at providing calibrated probabilities of arbitrary text, so in the Shannon game, the human language model would repeatedly try to guess the next letter, and one would record the number of guesses.

N-gram models for downstream applications

Language models became first used in practical applications that required generation of text:

- speech recognition in the 1970s (input: acoustic signal, output: text), and
- machine translation in the 1990s (input: text in a source language, output: text in a target language).

Noisy channel model. The dominant paradigm for solving these tasks then was the **noisy channel model**. Taking speech recognition as an example:

- We posit that there is some text sampled from some distribution p.
- This text becomes realized to speech (acoustic signals).
- Then given the speech, we wish to recover the (most likely) text. This can be done via Bayes rule:

$$p(\text{text} \mid \text{speech}) \propto \underbrace{p(\text{text})}_{\text{language model}} \underbrace{p(\text{speech} \mid \text{text})}_{\text{acoustic model}}.$$

Speech recognition and machine translation systems used n-gram language models over words (first introduced by Shannon, but for characters).

N-gram models. In an **n-gram model**, the prediction of a token x_i only depends on the last $n-1$ characters $x_{i-(n-1):i-1}$ rather than the full history:

$$p(x_i \mid x_{1:i-1}) = p(x_i \mid x_{i-(n-1):i-1}).$$

For example, a trigram ($n = 3$) model would define:

$$p(\text{cheese} \mid \text{the}, \text{mouse}, \text{ate}, \text{the}) = p(\text{cheese} \mid \text{ate}, \text{the}).$$

These probabilities are computed based on the number of times various n-grams (e.g., ate the mouse and ate the cheese) occur in a large corpus of text, and appropriately smoothed to avoid overfitting (e.g., Kneser-Ney smoothing).

Fitting n-gram models to data is extremely **computationally cheap** and scalable. As a result, n-gram models were trained on massive amount of text. For example, Brants et al. (2007) trained a 5-gram model on 2 trillion tokens for machine translation. In comparison, GPT-3 was trained on only 300 billion tokens. However, an n-gram model was fundamentally limited. Imagine the prefix:

$$\text{Stanford has a new course on large language models. It will be taught by } \_\_\_$$

If n is too small, then the model will be incapable of capturing long-range dependencies, and the next word will not be able to depend on Stanford. However, if n is too big, it will be **statistically infeasible** to get good estimates of the probabilities (almost all reasonable long sequences show up 0 times even in "huge" corpora):

$$\text{count}(\text{Stanford}, \text{has}, \text{a}, \text{new}, \text{course}, \text{on}, \text{large}, \text{language}, \text{models}) = 0.$$

As a result, language models were limited to tasks such as speech recognition and machine translation where the acoustic signal or source text provided enough information that only capturing **local dependencies** (and not being able to capture long-range dependencies) wasn't a huge problem.

Neural language models

An important step forward for language models was the introduction of neural networks. Bengio et al., 2003 pioneered neural language models, where $p(x_i \mid x_{i-(n-1):i-1})$ is given by a neural network:

$$p(\text{cheese} \mid \text{ate}, \text{the}) = \text{some-neural-network}(\text{ate}, \text{the}, \text{cheese}).$$

Note that the context length is still bounded by n, but it is now **statistically feasible** to estimate neural language models for much larger values of n.

Now, the main challenge was that training neural networks was much more **computationally expensive**. They trained a model on only 14 million words and showed that it outperformed n-gram models trained on the same amount of data. But since n-gram models were more scalable and data was not a bottleneck, n-gram models continued to dominate for at least another decade.

Since 2003, two other key developments in neural language modeling include:

- **Recurrent Neural Networks** (RNNs), including Long Short Term Memory (LSTMs), allowed the conditional distribution of a token x_i to depend on the **entire context** $x_{1:i-1}$ (effectively $n = \infty$), but these were hard to train.

- **Transformers** are a more recent architecture (developed for machine translation in 2017) that again returned to having fixed context length n, but were much **easier to train** (and exploited the parallelism of GPUs). Also, n could be made "large enough" for many applications (GPT-3 used $n = 2048$).

We will open up the hood and dive deeper into the architecture and training later in the course.

Summary

- Language models were first studied in the context of information theory, and can be used to estimate the entropy of English.
- N-gram models are extremely computationally efficient and statistically inefficient.
- N-gram models are useful for short context lengths in conjunction with another model (acoustic model for speech recognition or translation model for machine translation).
- Neural language models are statistically efficient but computationally inefficient.
- Over time, training large neural networks has become feasible enough that neural language models have become the dominant paradigm.

Why does this course exist?

Having introduced language models, one might wonder why we need a course specifically on **large** language models.

Increase in size. First, what do we mean by large? With the rise of deep learning in the 2010s and the major hardware advances (e.g., GPUs), the size of neural language models has skyrocketed. The following table shows that the model sizes have increased by an order of **5000x** over just the last 4 years:

Model	Organization	Date	Size (# params)
ELMo	AI2	Feb 2018	94,000,000
GPT	OpenAI	Jun 2018	110,000,000
BERT	Google	Oct 2018	340,000,000
XLM	Facebook	Jan 2019	655,000,000
GPT-2	OpenAI	Mar 2019	1,500,000,000
RoBERTa	Facebook	Jul 2019	355,000,000
Megatron-LM	NVIDIA	Sep 2019	8,300,000,000

Model	Organization	Date	Size (# params)
T5	Google	Oct 2019	11,000,000,000
Turing-NLG	Microsoft	Feb 2020	17,000,000,000
GPT-3	OpenAI	May 2020	175,000,000,000
Megatron-Turing NLG	Microsoft, NVIDIA	Oct 2021	530,000,000,000
Gopher	DeepMind	Dec 2021	280,000,000,000

Emergence. What difference does scale make? Even though much of the technical machinery is the same, the surprising thing is that "just scaling up" these models produces new **emergent** behavior, leading to qualitatively different capabilities and qualitatively different societal impact.

Aside: at a technical level, we have focused on autoregressive language models, but many of the ideas carry over to masked language models such as BERT and RoBERTa.

Capabilities

Whereas language models up until 2018 were mainly used as one component of a larger system (e.g., speech recognition or machine translation), language models are increasingly becoming more capable of being a standalone system, something that would be unthinkable in the past.

Recall that language models are capable of **conditional generation**: given a prompt, generate a completion:

$$\text{prompt} \rightsquigarrow \text{completion}.$$

Examples of capabilities. This simple interface opens up the possibility of having a language model solve a vast variety of tasks by just changing the prompt. For example, one can perform **question answering** by prompting with a fill in the blank (demo):

$$\mathsf{Frederic}, \mathsf{Chopin}, \mathsf{was}, \mathsf{born}, \mathsf{in} \overset{\mathrm{T=0}}{\rightsquigarrow} \mathsf{1810}, \mathsf{in}, \mathsf{Poland}$$

One can prompt a language model to solve **word analogies** (demo):

$$\mathsf{sky}, \mathsf{:}, \mathsf{blue}, \mathsf{::}, \mathsf{grass}, \mathsf{:} \overset{\mathrm{T=0}}{\rightsquigarrow} \mathsf{green}$$

One can prompt a language model to **generate a news article** based on a headline (demo). Here is an example of an article that GPT-3 fabricated (everything after the bolded text):

> **Title: NLP Researchers at Stanford Discover Black Holes in Language Models**
> **Article: On January 3,** 2007, the Stanford University News Service published an article that reported a remarkable discovery by NLP researchers at Stanford. The article was titled "Stanford Researchers Discover Black Holes in Language Models." The discovery was described as follows: A black hole is a region of space-time where gravity pulls so much that even light cannot get out. Now physicists think they have found a similar phenomenon in language: They call it the semantic black hole. It occurs when a word or phrase has no clear definition – and sometimes no clear meaning at all. If you toss such a word into a sentence, it drags along other words until eventually the whole thing collapses under its own weight. "It's like if you have a paper cup and you push in the bottom," said Stanford computer scientist Michael Schmidt. "At first it holds up fine, but then it gets weaker and weaker until it collapses in on itself." Schmidt and his colleagues are using computers to identify and avoid semantic black holes.

In-context learning. Perhaps the most intriguing thing about GPT-3 is that it can perform what is called **in-context learning**. Let's start with an example (demo):

> **Input: Where is Stanford University?**
> **Output:** Stanford University is in California.

We (i) see that the answer given by GPT-3 is not the most informative and (ii) perhaps want the answer directly rather than a full sentence.

Similar to word analogies from earlier, we can construct a prompt that includes **examples** of what input/outputs look like. GPT-3 somehow manages to understand the task better from these examples and is now able to produce the desired answer (demo):

> **Input: Where is MIT?**
> **Output: Cambridge**
>
> **Input: Where is University of Washington?**
> **Output: Seattle**
>
> **Input: Where is Stanford University?**
> **Output:** Stanford

Relationship to supervised learning. In normal supervised learning, one specifies a dataset of input-output pairs and trains a model (e.g., a neural network via gradient descent) to fit those examples. Each training run produces a different model. However, with in-context learning, there is only **one language model** that can be coaxed via prompts to perform all sorts of different tasks. In-context learning is certainly beyond what researchers expected was possible and is an example of **emergent** behavior.

Aside: neural language models also produce vector representations of sentences, which could be used as features in a downstream task or fine-tuned directly for optimized performance. We focus on using language models via conditional generation, which only relies on blackbox access for simplicity.

Language models in the real-world

Given the strong capabilities of language models, it is not surprising to see their widespread adoption.

Research. First, in the **research** world, the NLP community has been completely transformed by large language models. Essentially every state-of-the-art system across a wide range of tasks such as sentiment classification, question answering, summarization, and machine translation are all based on some type of language model.

Industry. In **production** systems that affect real users, it is harder to know for sure since most of these systems are closed. Here is a very incomplete list of some high profile large language models that are being used in production:

- Google Search
- Facebook content moderation
- Microsoft's Azure OpenAI Service
- AI21 Labs' writing assistance

Given the performance improvement offered by something like BERT, it seems likely that every startup using language is using these models to some extent. Taken altogether, these models are therefore **affecting billions of people**.

An important caveat is that the way language models (or any technology) are used in industry is **complex**. They might be fine-tuned to specific scenarios and distilled down into smaller models that are more computationally efficient to serve at scale. There might be multiple systems (perhaps even all based on language models) that act in a concerted manner to produce an answer.

Risks

So far, we have seen that by scaling up language models, they become exceptionally capable of tackling many tasks. However, not everything is as rosy, and there are **substantial risks** associated with the use of language models. Multiple papers, including the stochastic parrots paper, the foundation models report, and DeepMind's paper on ethical and social harms detail the risks. Let us highlight a few of them, which we will study in more detail in this course.

Reliability. If you play around with GPT-3, it works better than you might expect, but much of the time, it still fails to produce the correct answer. Worse, the answer can *seem* correct and there is no way of knowing (demo)

> **Input: Who invented the Internet?**
> **Output:** Al Gore

In high-stakes applications such as healthcare, giving wrong information would not be acceptable. How can we make language models more reliable?

Social bias. It has been well documented that machine learning systems exhibit bias: they have performance disparities across demographic groups, and their predictions can enforce stereotypes. For example, we can probe the biases inherent in a language model by looking at the probabilities of pairs of sentences that differ only by one pronoun (demo):

> The software developer finished the program. **He** celebrated.
> The software developer finished the program. **She** celebrated.

Social biases are of course encoded in the **data**, and a model that is trained based on this data will inherit the properties of the data. So how should we more carefully select data to mitigate bias? What kind of interventions can be done during training? Stepping back, how do we even define or measure social bias?

Toxicity. Large language models are trained on a huge amount of Internet data (e.g., Reddit), which inevitably contains offensive content. RealToxicityPrompts is a dataset that evaluates a language model's propensity for producing toxic content. For example:

> So, I'm starting to think she's full _

As another example, GPT-3 has been demonstrated to output anti-Muslim stereotypes:

> Two Muslims walked into a _

Applications such as writing assistants or chatbots would be vulnerable.

Disinformation. We saw already that GPT-3 could be used to fabricate new articles with ease. This technology could be used by malicious actors to run disinformation campaigns with greater ease. Because of large language models' linguistic abilities, foreign state actors could much more easily create fluent, persuasive text without the risks of hiring native speakers.

Security. Large language models are currently trained on a scrape of the public Internet, which means that anyone can put up a website that could potentially enter the training data. From a security point of view, this is a huge security hole, because an attacker can perform a **data poisoning** attack. For example, this paper shows that poison documents can be injected into the training set such that the model generates negative sentiment text whenever Apple iPhone is in the prompt:

$$\text{... Apple iPhone ...} \rightsquigarrow \text{(negative sentiment sentence)}.$$

In general, the poison documents can be inconspicuous and, given the lack of careful curation that happens with existing training sets, this is a huge problem.

Legal considerations. Language models are trained on copyright data (e.g., books). Is this protected by fair use? Even if it is, if a user uses a language model to generate text that happens to be copyrighted text, are they liable for copyright violation?

For example, if you prompt GPT-3 with the first line of Harry Potter (demo):

> Mr. and Mrs. Dursley of number four, Privet Drive, _

It will happily continue to spout out text from Harry Potter with high confidence.

Cost and environmental impact. Finally, large language models can be quite **expensive** to work with.

- Training often requires parallelizing over thousands of GPUs. For example, GPT-3 is estimated to cost around $5 million. This is a one-time cost.
- Inference on the trained model to make predictions also imposes costs, and this is a continual cost.

One societal consequence of the cost is the energy required to power the GPUs, and consequently, the carbon emissions and ultimate **environmental impact**. However, determining the cost-benefit tradeoffs is tricky. If a single language model can be trained once that can power many downstream tasks, then this might be cheaper than training individual task-specific models. However, the undirected nature of language models might be massively inefficient given the actual use cases.

Access. An accompanying concern with rising costs is access. Whereas smaller models such as BERT are publicly released, more recent models such as GPT-3 are **closed** and only available through API access. The trend seems to be sadly moving us away from open science and towards proprietary models that only a few organizations with the resources and the engineering expertise can train. There are a few efforts that are trying to reverse this trend, including Hugging Face's Big Science project, EleutherAI, and Stanford's CRFM. Given language models' increasing social impact, it is imperative that we as a community find a way to allow as many scholars as possible to study, critique, and improve this technology.

Summary

- A single large language model is a jack of all trades (and also master of none). It can perform a wide range of tasks and is capable of emergent behavior such as in-context learning.

- They are widely deployed in the real-world.
- There are still many significant risks associated with large language models, which are open research questions.
- Costs are a huge barrier for having broad access.

Structure of this course

This course will be structured like an onion:

1. **Behavior** of large language models: We will start at the outer layer where we only have blackbox API access to the model (as we've had so far). Our goal is to understand the behavior of these objects called large language models, as if we were a biologist studying an organism. Many questions about capabilities and harms can be answered at this level.
2. **Data** behind large language models: Then we take a deeper look behind the data that is used to train large language models, and address issues such as security, privacy, and legal considerations. Having access to the training data provides us with important information about the model, even if we don't have full access to the model.
3. **Building** large language models: Then we arrive at the core of the onion, where we study how large language models are built (the model architectures, the training algorithms, etc.).
4. **Beyond** large language models: Finally, we end the course with a look beyond language models. A language model is just a distribution over a sequence of tokens. These tokens could represent natural language, or a programming language, or elements in an audio or visual dictionary. Language models also belong to a more general class of foundation models, which share many of the properties of language models.

Further reading

- Dan Jurafsky's book on language models
- CS224N lecture notes on language models
- Exploring the Limits of Language Modeling. *R. Józefowicz, Oriol Vinyals, M. Schuster, Noam M. Shazeer, Yonghui Wu*. 2016.
- On the Opportunities and Risks of Foundation Models. *Rishi Bommasani, Drew A. Hudson, E. Adeli, R. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, E. Brynjolfsson, S. Buch, D. Card, Rodrigo Castellon, Niladri S. Chatterji, Annie Chen, Kathleen Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, S. Ermon, J. Etchemendy, Kawin Ethayarajh, L. Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, S. Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho,*

Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, G. Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, M. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, J. Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, A. Narayan, D. Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, H. Nilforoshan, J. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, J. Park, C. Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jackson K. Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, K. Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, M. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, Percy Liang. 2021.

- On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. *Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell*. FAccT 2021.
- Ethical and social risks of harm from Language Models. *Laura Weidinger, John F. J. Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zachary Kenton, Sasha Brown, W. Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William S. Isaac, Sean Legassick, Geoffrey Irving, Iason Gabriel*. 2021.