CSE 156 | Lecture 9: Pretrained Decoders

Ndapa Nakashole

October 29, 2024

Administrative matters

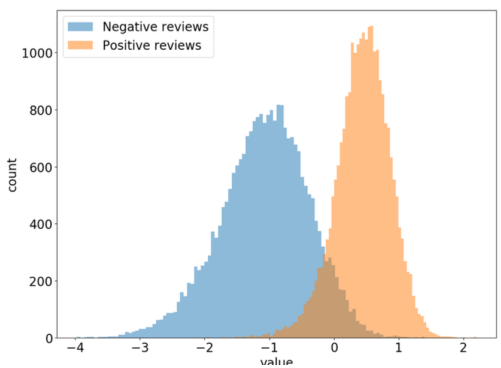
- **Quiz 1**: due tomorrow
- **PA2**: due in a week, Nov 6th, 11:59pm

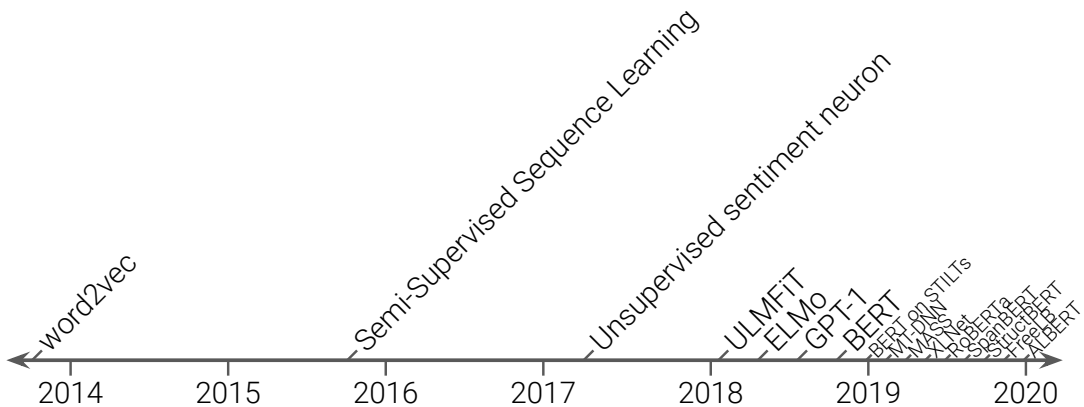
Today

1. Decoder LMs (GPT, ...)
2. Decoding Methods
3. Encoder-Decoder LMs (BART,T5)
4. Scaling “Laws"

Recap: Pretraining; Learning Representations

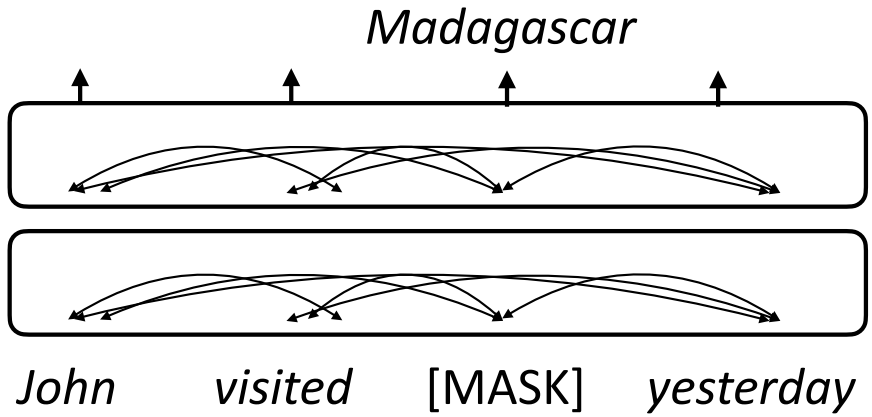
- **Pretraining**: learn useful representations from raw data
- 2017: Unsupervised Sentiment Neuron
- 2018: ELMo, BERT, GPT-1, ...





Recap: BERT Objective

- BERT objective: *masked language modeling*
- Best version of this: **DeBERTa**, good at NLI/QA/classification tasks



Optimize: $\mathrm{P}(\text{ Madagascar} \mid \text{John visited [MASK] yesterday})$

Decoders

Encoders vs. Decoders

- BERT is a Transformer **encoder**: **bidirectional attention**, trained with masked language modeling

$$P\left(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\right)$$

- GPT-n and other Transformer language models (e.g., PA2) are **decoders**: **unidirectional attention**, trained to predict the next word

$$P\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$

- GPT-2: 1.5B parameters, 12-layer decoder, 12 attention heads, 768 hidden size

GPT-2 Training Dataset: WebText

- Created the **WebText** dataset to train GPT-2
- High-quality, human-filtered web content rather than noisy data like Common Crawl.
- Scraped all outbound links from **Reddit** with 3+ karma rating, assuming higher quality and relevance.
- **WebText Dataset**:
  - Contains over **8 million documents** (40 GB), with de-duplication and cleaning applied.

GPT-2: Towards Generalized Transfer Learning in NLP

- Traditional ML approaches focus on training single-task models, limiting generalization
- **2019 NLP systems** used pre-training + supervised finetuning, providing flexible transfer learning
  - but still depended on supervised data for each task
- GPT-2 extended transfer learning by demonstrating that LMs can do tasks in a **zero-shot setting—without additional training data**
- Promising generalizable performance across a range of tasks

GPT 2 Dataset: Translations in WebText

Tasks are present in the data

”I’m not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I’m not a fool].**

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: ”**Mentez mentez, il en restera toujours quelque chose**,” which translates as, ”**Lie lie and something will always remain.**”

“I hate the word ‘**perfume**,”’ Burr says. ‘It’s somewhat better in French: ‘**parfum**.’

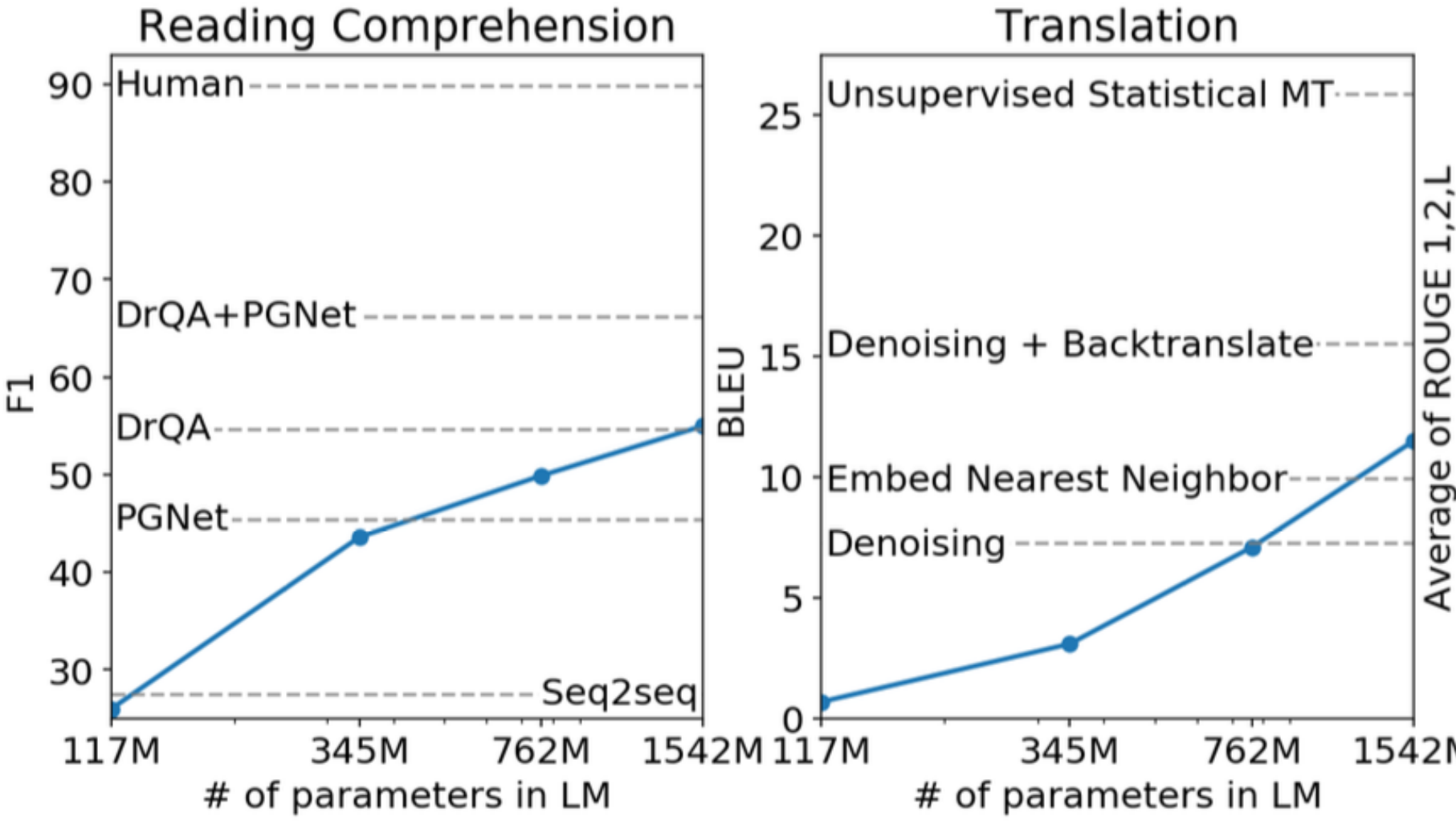
If listened carefully at 29:55, a conversation can be heard between two guys in French: “**-Comment on fait pour aller de l’autre coté? -Quel autre coté?**”, which means “**- How do you get to the other side? - What side?**”.

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?

OpenAI GPT2: Zero-shot task performance

Radford et al., 2019: "**Language Models are Unsupervised Multitask Learners**"

Treat tasks as LM conditioning problems:



OpenAI GPT 2: Summarization

- Use of " `TL;DR`" to elicite summaries as a common pattern in WebText

`TL;DR:`

	R-1	R-2	R-L	R-AVG
Bottom-Up Sum	**41.22**	**18.68**	**38.34**	**32.75**
Lede-3	40.38	17.66	36.62	31.55
Seq2Seq + Attn	31.33	11.81	28.83	23.99
GPT-2 `TL;DR:`	29.34	8.27	26.58	21.40
Random-3	28.78	8.63	25.52	20.98
GPT-2 no hint	21.58	4.03	19.47	15.03

Table 4. Summarization performance as measured by ROUGE F1 metrics on the CNN and Daily Mail dataset. Bottom-Up Sum is the SOTA model from (Gehrmann et al., 2018)

Radford et al., 2019: "**Language Models are Unsupervised Multitask Learners**"

Decoding Strategies

Decoding Strategies

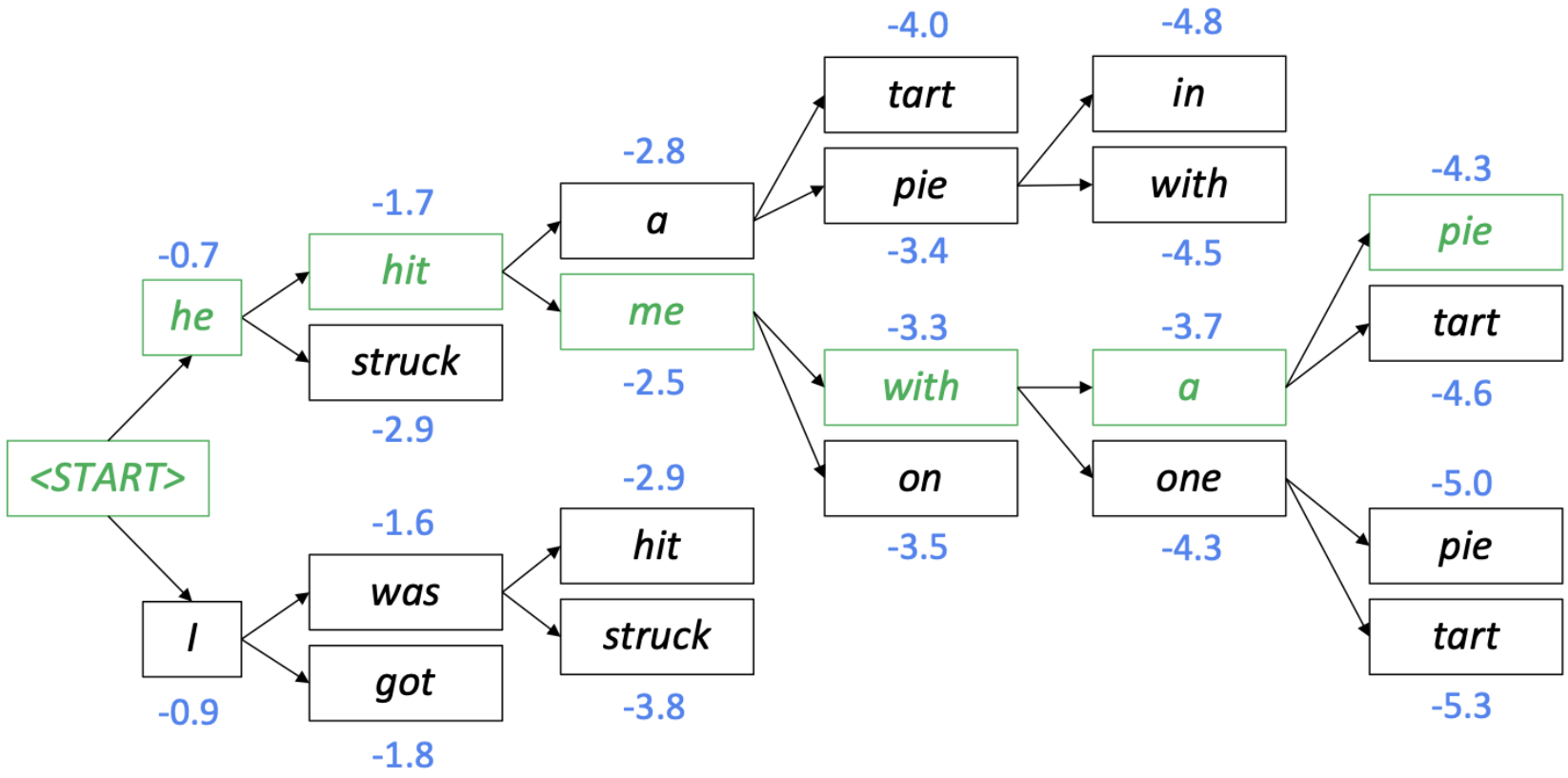
- LMs place a distribution $\mathrm{P}\left(\mathrm{x_i} \mid \mathrm{x_1}, \ldots, \mathrm{x_{i-1}}\right)$
- How do we generate text from these?
  - **Greedy decoding**: take the word with the highest probability at each step
  - **Beam search**: find the sequence with the highest probability
  - **Sample from the model**; draw x_i from that distribution

Holtman et al. (2019)

Method: Beam Search

Recall Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



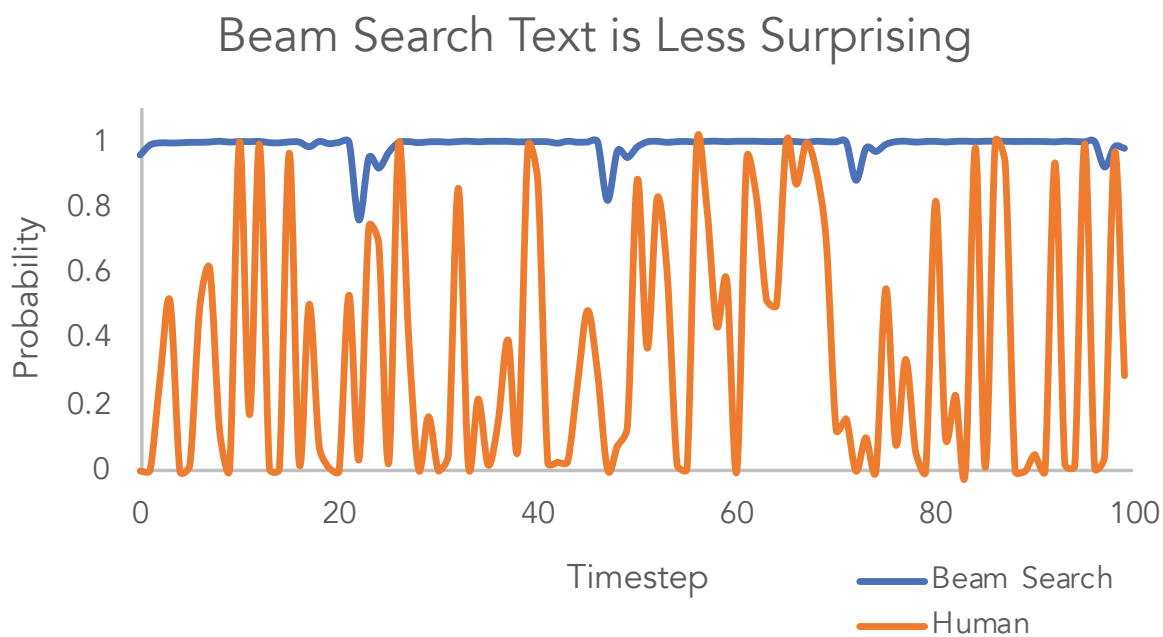
Backtrack to obtain the full hypothesis

Problems with Beam Search

Holtzman et al., 2019: "**The Curious Case of Neural Text Degeneration**"

- Maximization-based decoding (e.g., **beam search**) can cause **degeneration**
  - LMs assign high scores to well-formed text, yet
  - The highest scores for long texts can be **generic, repetitive, and incoherent**

.



Directed Text Generation (vs. Open-Ended)

- Directed text generation: Tasks have **(input, output)** pairs where the output is a transformation of the input:
  - **Machine translation**
  - **Data-to-text generation**
  - **Summarization**
- Generation is usually performed using Beam Search
  - Output is determined by the input, repetition and genericness are not as problematic.

Open-Ended Generation

- Open-ended generation: e.g. **conditional story generation** and **text continuation**.
- Generation has significant freedom in terms of what can plausibly follow.
- Some tasks are both open-ended and directed generation
  - **Goal-oriented dialogue**, e.g., conversation with a chatbot to book a flight

Open-Ended Text Generation

- Given a sequence of m context tokens, $x_1, \ldots, x_m$, the goal is to generate the next n continuation tokens, producing a completed sequence: $x_1, \ldots, x_m, x_{m+1}, \ldots, x_{m+n}$

$$P\left(x_{1:m}, x_{m+1:m+n}\right) = \prod_{i=1}^{m+n} P\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$

- Generation proceeds token-by-token using a chosen decoding strategy (e.g., greedy, beam search, nucleus sampling).

Method: Pure Sampling

- Sampling directly from model-predicted probabilities often leads to **incoherent** and **contextually unrelated** text.

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

- Issue: **unreliable tail** in the probability distribution contains thousands of low-probability tokens
  - These tokens are **over-represented**, disrupting coherence and relevance.

Challenges with Pure Sampling

- **Sampling**: too random, introduces many grammatical errors

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV

P(*y* | … they live in a remote desert uninterrupted by)

0.01 roads
0.01 towns
0.01 people
0.005 civilization

Good options, maybe accounting for 90% of the total probability mass. So a 90% chance of getting something good

…
0.0005 town

Long tail with 10% of the mass

Holtzman et al. (2019)

Method: Nucleus Sampling

A Stochastic Decoding Method using Top-p Vocabulary

- Given a probability distribution $P\left(x \mid x_{1:i-1}\right)$, the **top-$p$ vocabulary** $V^{(p)} \subset V$ is defined as the smallest set of tokens satisfying:

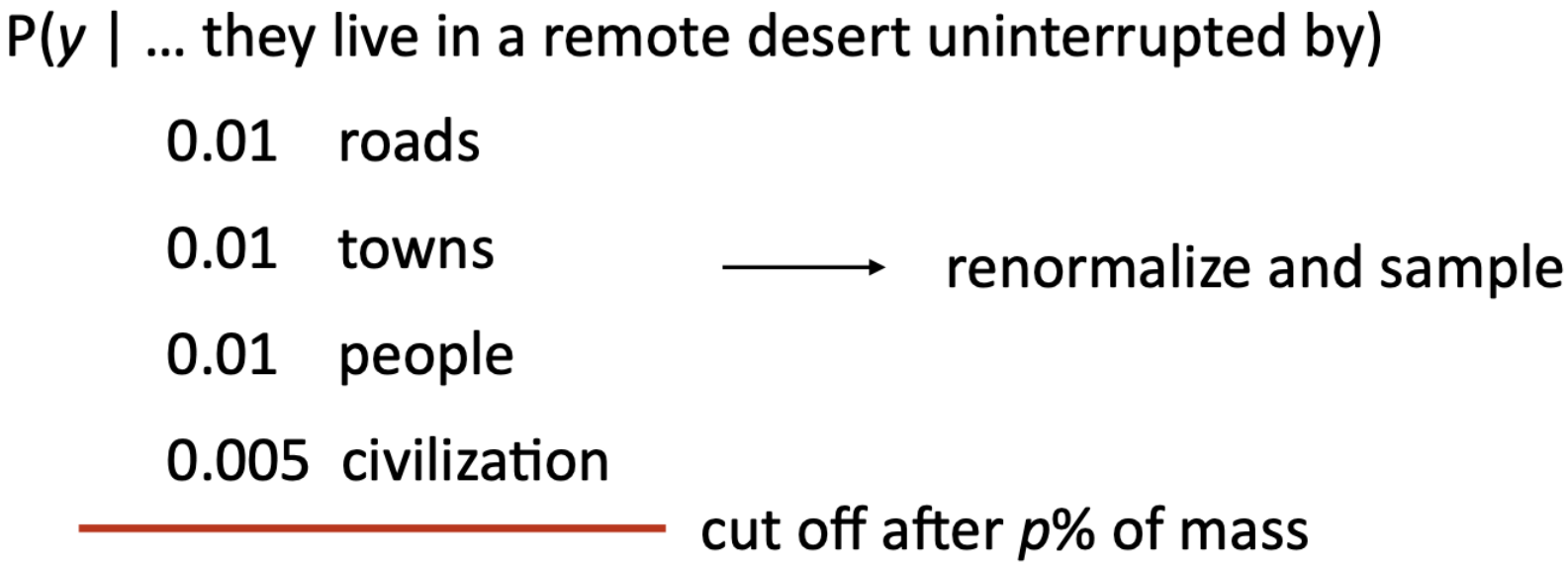
$$\sum_{x \in V^{(p)}} P\left(x \mid x_{1:i-1}\right) \geq p$$

- Focus on (the nucleus) ensures sampling focuses on high-probability tokens while discarding the unreliable tail

- Let $p' = \sum_{x \in V^{(p)}} P\left(x \mid x_{1:i-1}\right)$. Original distribution is re-scaled to a new distribution, from which the next word is sampled:

$$P'\left(x \mid x_{1:i-1}\right) = \begin{cases} P\left(x \mid x_{1:i-1}\right) / p' & \text{if } x \in V^{(p)} \\ 0 & \text{otherwise} \end{cases}$$

Method: Nucleus Sampling

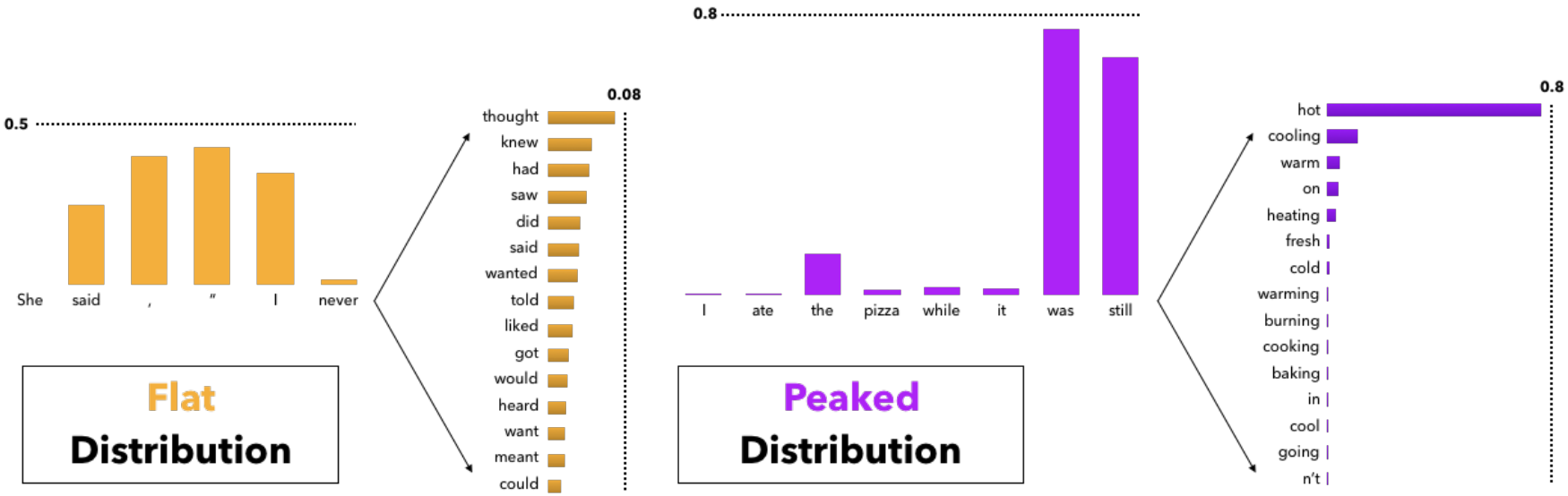


- Define a threshold p. Keep the most probable options account for $p\%$ of the probability mass (the nucleus), then sample among these
- To implement: sort options by probability, truncate the list once the total exceeds p, then renormalize and sample from it

Method: Top-k Sampling

- At each time step, top-k sampling considers only the k tokens with the highest probabilities, re-scaling these probabilities and sampling from them
- Both Top-k and Nucleus Sampling restrict sampling to a *trustworthy prediction zone*, but differ in truncation strategy

Top-k Sampling: Challenges with fixed k



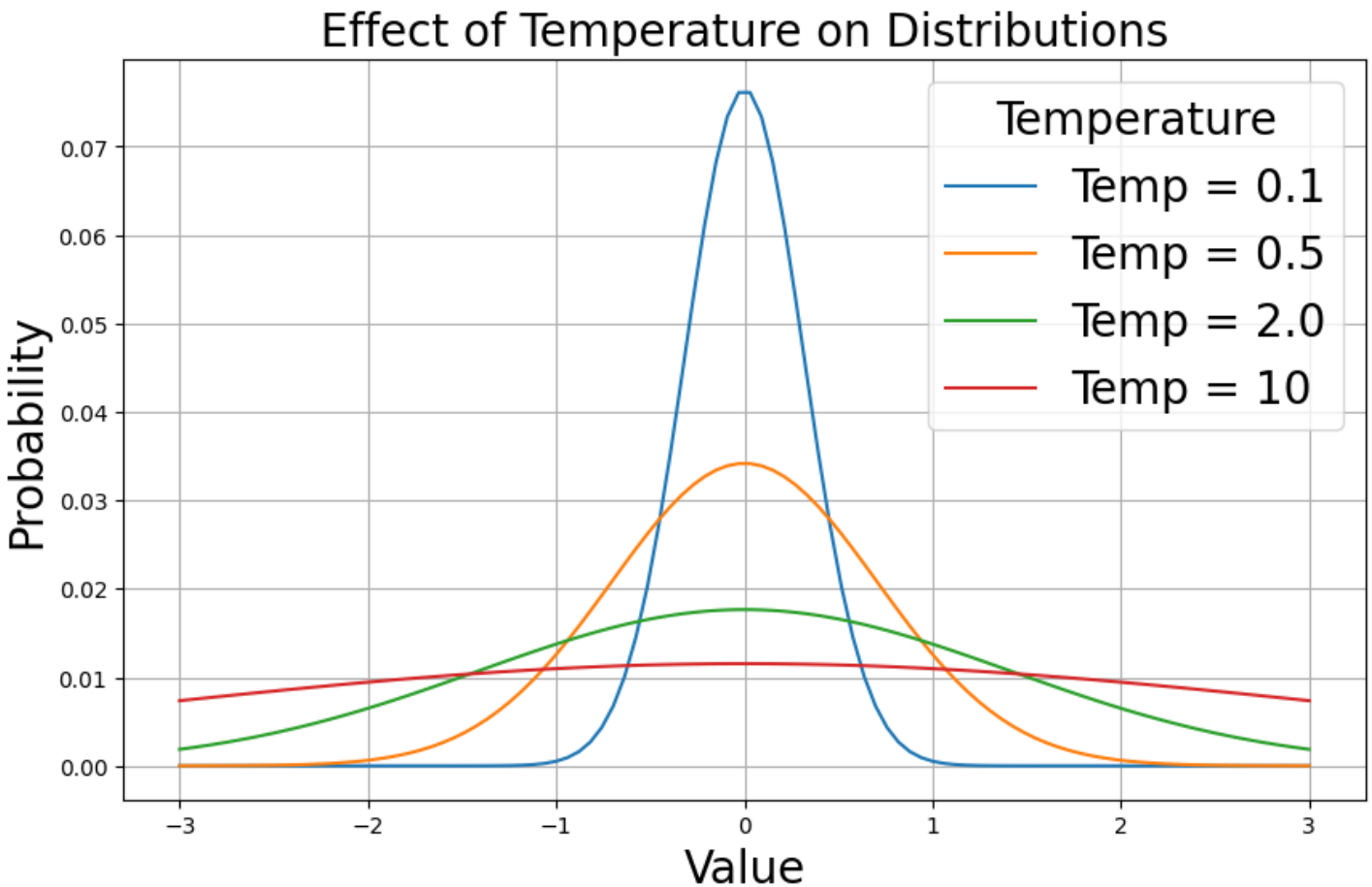
- The presence of flat distributions makes the use of a small k in top- k sampling problematic
- The presence of peaked distributions makes large k 's problematic.

Sampling with Temperature

High temperature: more randomness; **Low temperature**: less randomness

Given the scores $u_{1:|V|}$ and temperature t, adjust softmax as:

$$p\left(x = V_l \mid x_{1:i-1}\right) = \frac{\exp\left(u_l/t\right)}{\sum_{l'} \exp\left(u_{l'}/t\right)}$$



Sampling with Temperature

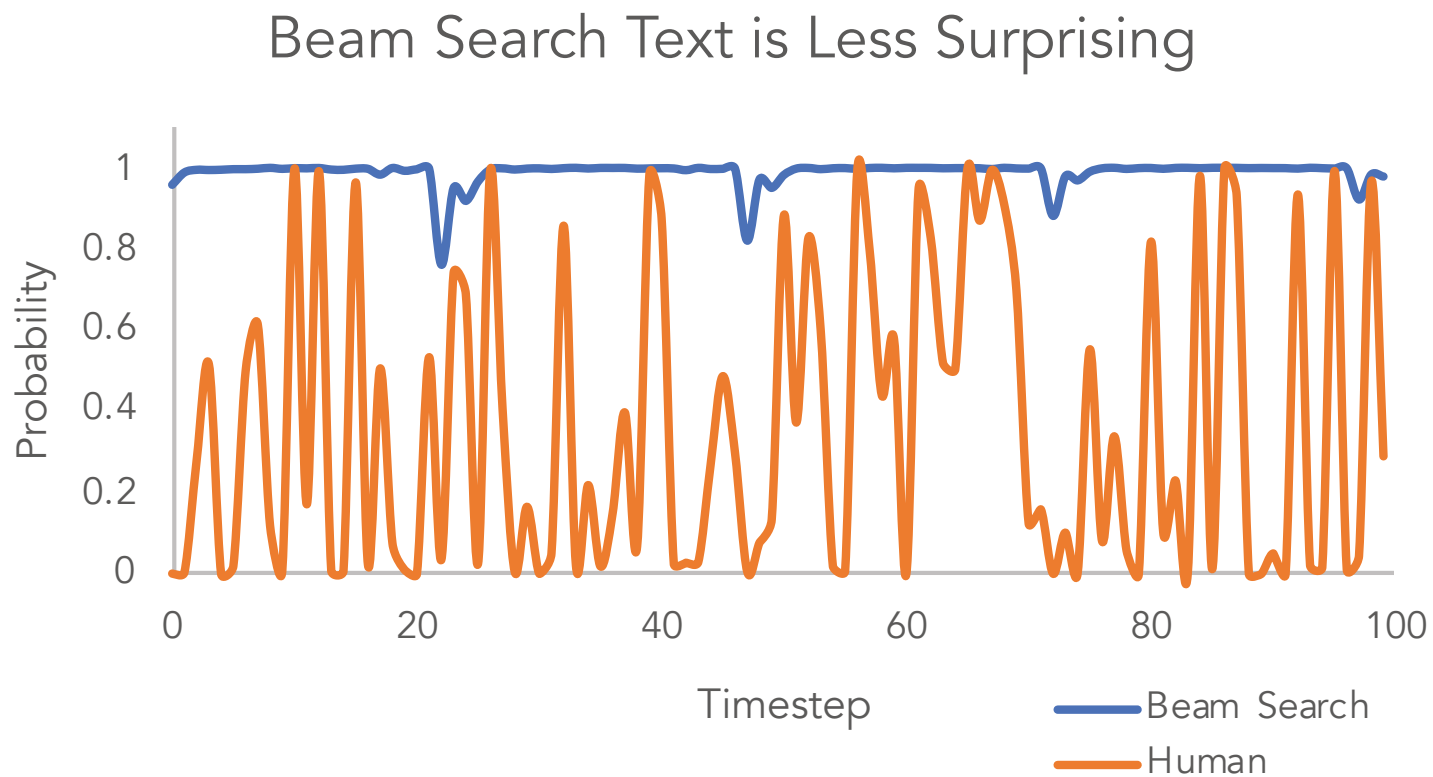
- Setting $t \in [0, 1)$ biases the distribution towards high probability events, reducing mass in the tail distribution.
- **Low temperature sampling** can address some issues of top-k sampling by pre-shaping the distribution (Radford et al., 2018; Fan et al., 2018).
- However, it was shown that lowering temperature improves quality but reduces diversity in generated text (Caccia et al., 2018; Hashimoto et al., 2019).

Decoding Strategies: Perplexity

Method	Perplexity
Human	12.38
Greedy	1.50
Beam, b=16	1.48
Stochastic Beam, b=16	19.20
Pure Sampling	22.73
Sampling, t=0.9	10.25
Top-k=40	6.88
Top-k=640	13.82
Top-k=40, t=0.7	3.48
Nucleus p=0.95	**13.13**

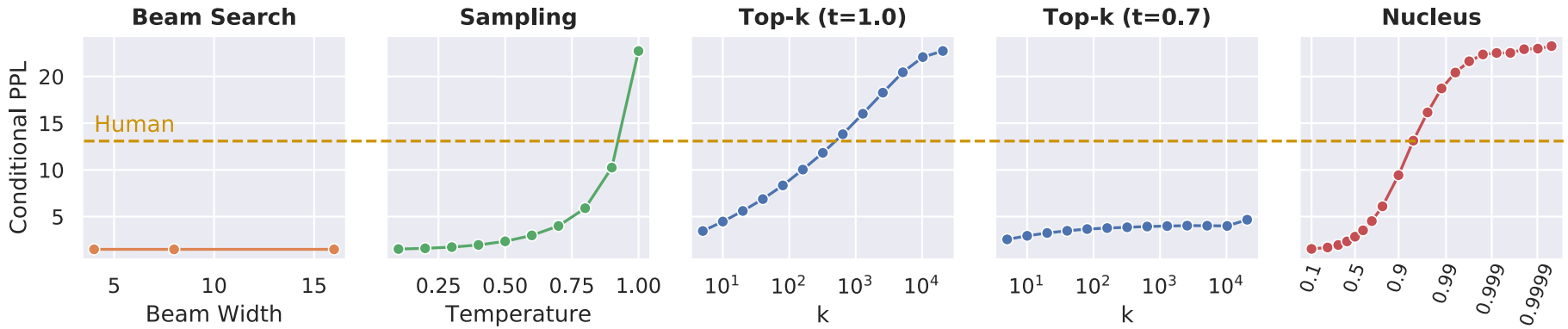
Natural Language Does Not Maximize Probability

- **Observation:** Natural language text, as written by humans, does not maximize per-token probability
- Natural language incorporates lower-probability but informative tokens



Holtzman et al. (2019)

Perplexities of Generations from Various Decoding Methods



- **Beam Search:** Shows unnaturally low perplexities, indicating overly predictable sequences.
- **Sampling, Top-k, and Nucleus** Sampling can be adjusted to achieve human-like perplexities.
- However, **Sampling and Top-** k struggle with coherency when parameters are tuned to reach these human-level perplexities.

Decoding Strategies

- LMs place a distribution $\mathrm{P}\left(\mathrm{x_i} \mid \mathrm{x_1}, \ldots, \mathrm{x_{i-1}}\right)$. How do we generate text from these?

- Maximization-based decoding:
  - **Greedy decoding**: pick the most likely word at each step
  - **Beam search**: keep track of the top k most likely sequences

- Sampling-based decoding:
  - **Random sampling**: sample from the full distribution
  - **Nucleus sampling**: sample from the top p fraction of the distribution
  - **Top-k sampling**: sample from the top k most likely words

Encoder-Decoder Models

Encoders (BERT) vs Decoders (GPT)

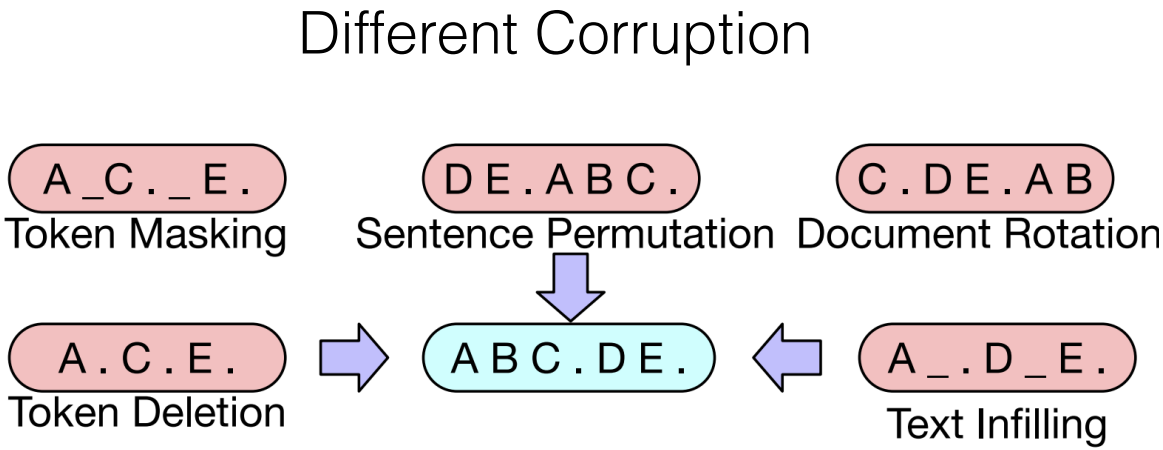
- Encoder : $P\left(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\right)$
  - **Bidirectional attention**, trained with masked language modeling
  - **To use in practice**: Ignore this probability distribution. Fine-tune the model for some other task $\mathrm{P}(\boldsymbol{y} \mid \boldsymbol{x})$
- Decoder: $P\left(x_i \mid x_1, \ldots, x_{i-1}\right)$
  - **Unidirectional attention**, trained to predict the next word
  - Can ignore this probability distribution and train a model for $\mathrm{P}(\boldsymbol{y} \mid \boldsymbol{x})$.
  - **To use in practice**: Generate text by sampling from this distribution

How do we pre-train encoder-decoder models?

- How can we pretrain a model for $\mathrm{P}(\mathbf{y} \mid \mathbf{x})$?
- Requirements: (1) should use unlabeled data; (2) should force a model to attend from $\boldsymbol{y}$ back to $\boldsymbol{x}$

BART

Bidirectional and Auto-Regressive Transformers

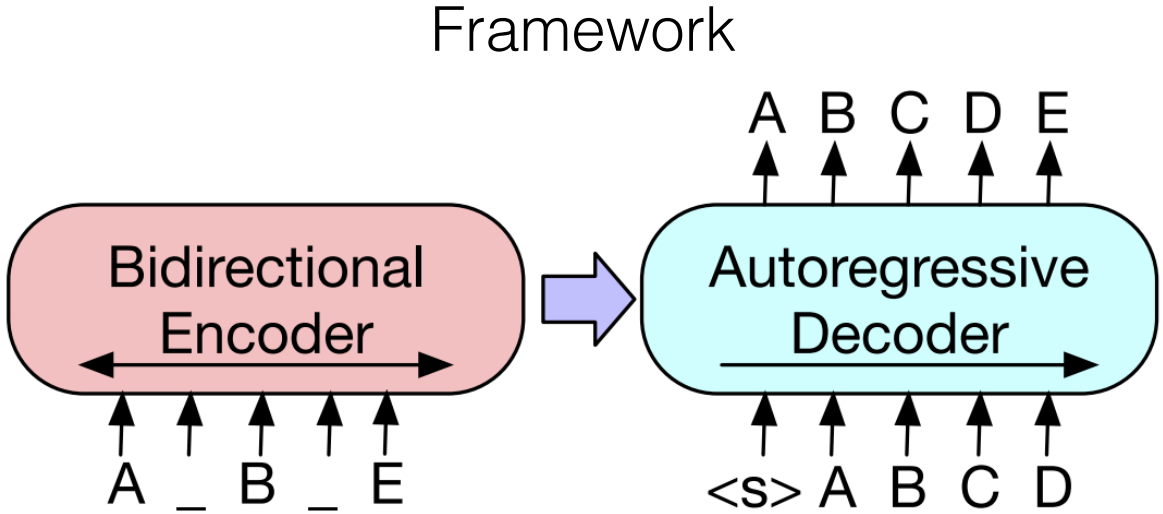


- **BART Objective:** Re-construct **(corrupted) original sentences**
- Several possible strategies for corrupting a sequence are explored in the BART paper

Lewis et al., 2020

BART Pretraining

- Encoder-Decoder Transformer trained on this data: (permute/mask/delete tokens), then predict full sequence autoregressively



Lewis et al., 2020

BART for Summarization

- **Pre-train** using BART: Corrupt random text segments and decode the original text.
- **Fine-tune** on summarization: Use news articles as input to generate brief summaries (1-3 sentences).
- **Few summaries needed for fine-tuning**: Achieves good results with fewer examples compared to traditional seq2seq models, which need over 10,000 examples.

BART for Summarization: Outputs

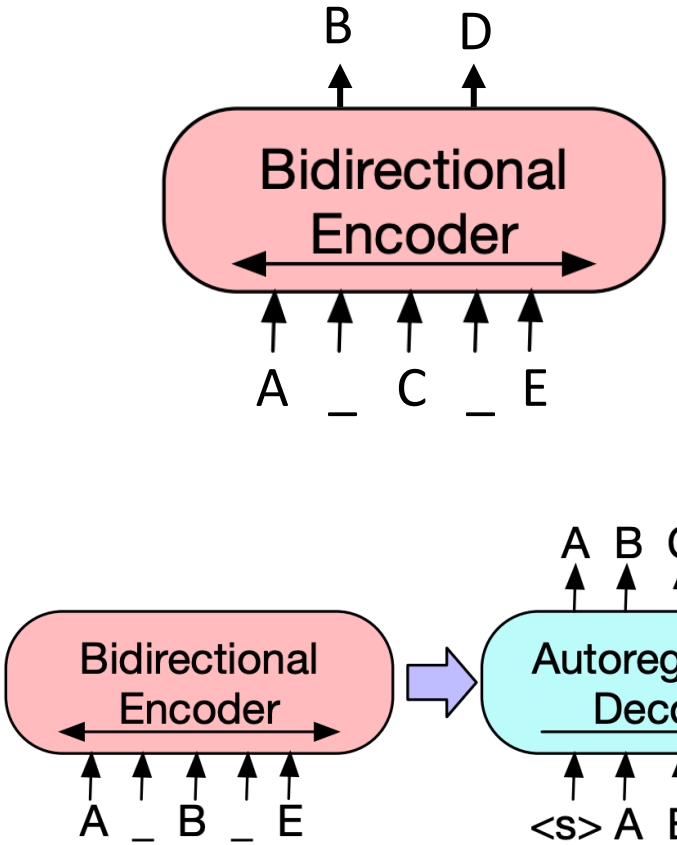
PG&E stated it scheduled the blackouts in response to forecasts for high winds amid dry conditions. The aim is to reduce the risk of wildfires. Nearly 800 thousand customers were scheduled to be affected by the shutoffs which were expected to last through at least midday tomorrow.

Power has been turned off to millions of customers in California as part of a power shutoff plan.

Lewis et al. (2019)

BERT vs. BART

- **BERT**: bidirectional encoder only, masked LM
- Cannot generate text or do seq2seq tasks such as translation
- **BART**: bidirectional, autoregressive
- Both an encoder and a decoder. Can also use just the encoder wherever we would use BERT

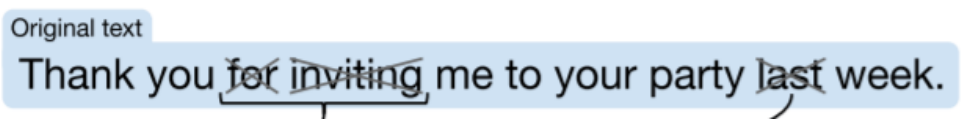


T5

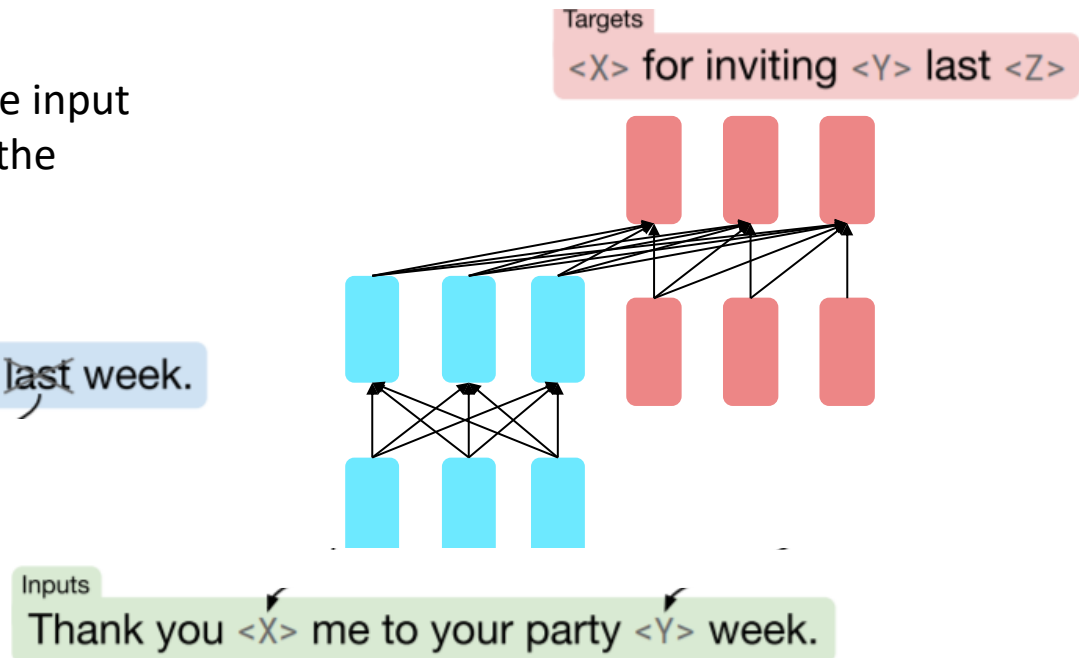
Text-to-Text Transfer Transformer

- Treat all tasks as text-to-text problems
- Pre-training: similar denoising scheme to BART (they were released within a week of each other in fall 2019)
- Input: text with gaps; Output: a series of phrases to fill those gaps

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!



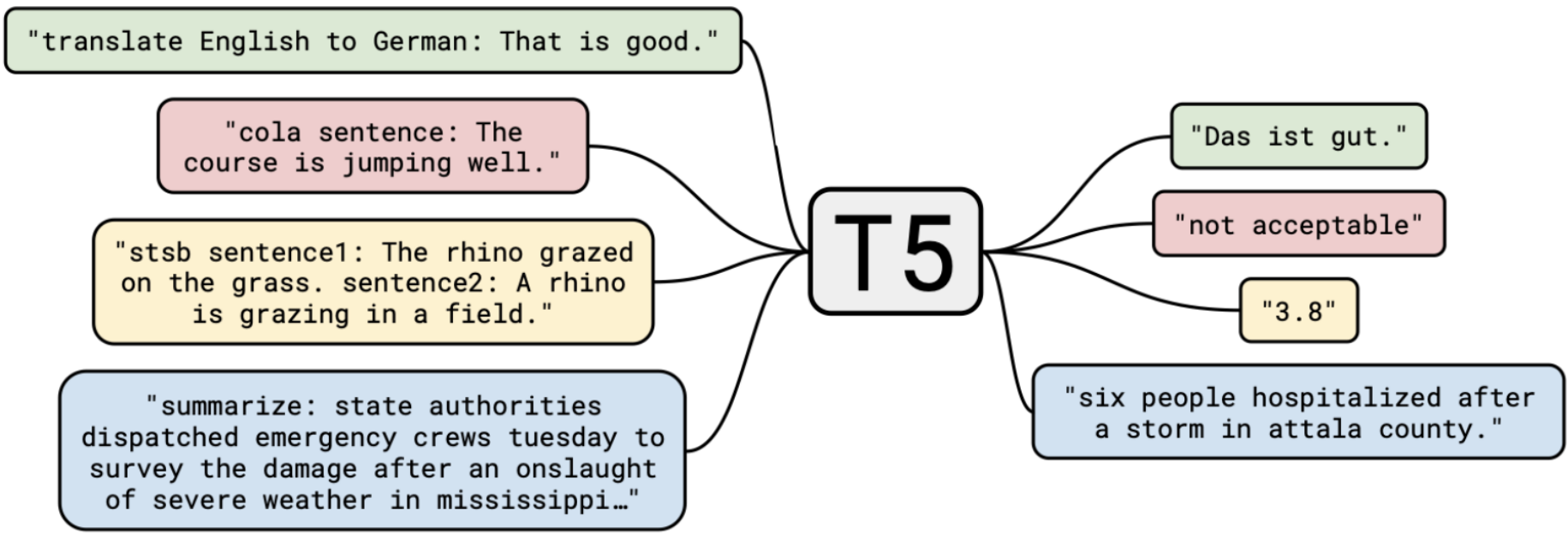
This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



Raffel et al., 2018

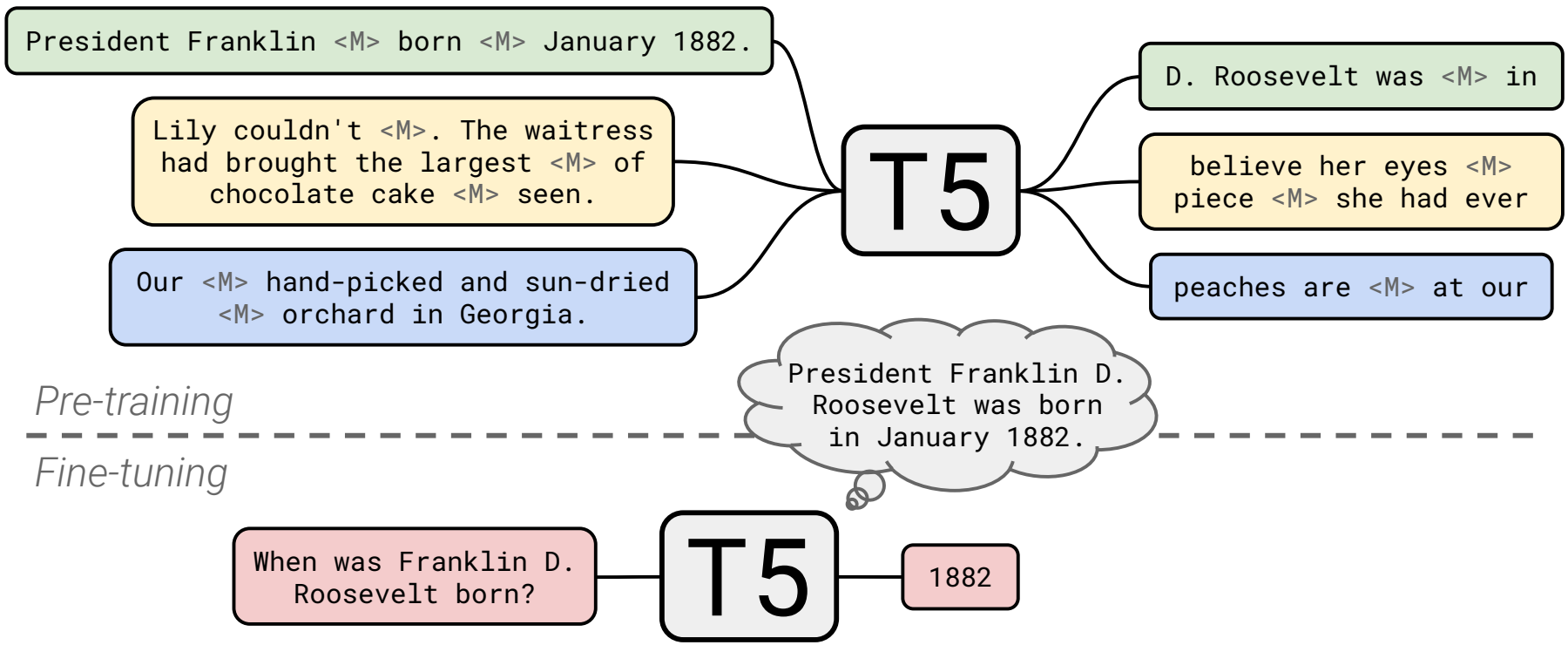
T5: Supervised Multitask Learning

- After **span corruption petraining**, continue training with **supervised multitask learning**
  - Formulate tasks as text-to-text format
  - Use a prefix to denote the task
  - Mixing examples from different datasets when constructing batches



Raffel et al., 2018

T5 can be fine-tuned to answer questions

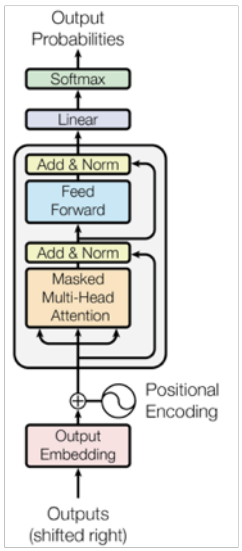


works well at QA; when finetuned to answer a wide range of questions, retrieving knowledge from its parameters.

Encoders, Decoders and Encoder-Decoders

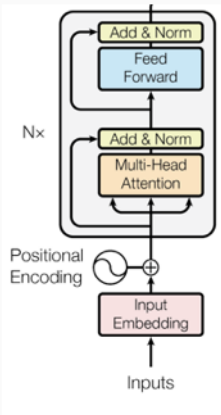
Decoder-only GPT

[sat_]



Encoder-only BERT

[*] [*] **[sat_]** [*] **[the_]** [*]



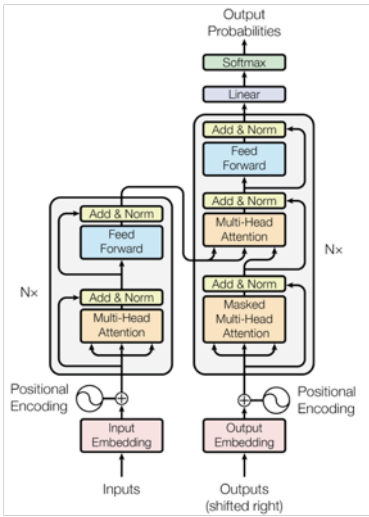
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

Enc-Dec T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.



Translate EN-DE: This is good.

Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!

Scaling Laws

Compute Trends Across Three Eras of Machine Learning

Plot: "impactful" ML systems, and plotted their compute usage over time

- Pre-deep learning era : *Moore's Law* was true, compute doubled every ~18 months
- Deep Learning era : compute doubles every 6 months
- Huge Scale Era : compute doubles every 3.5 months

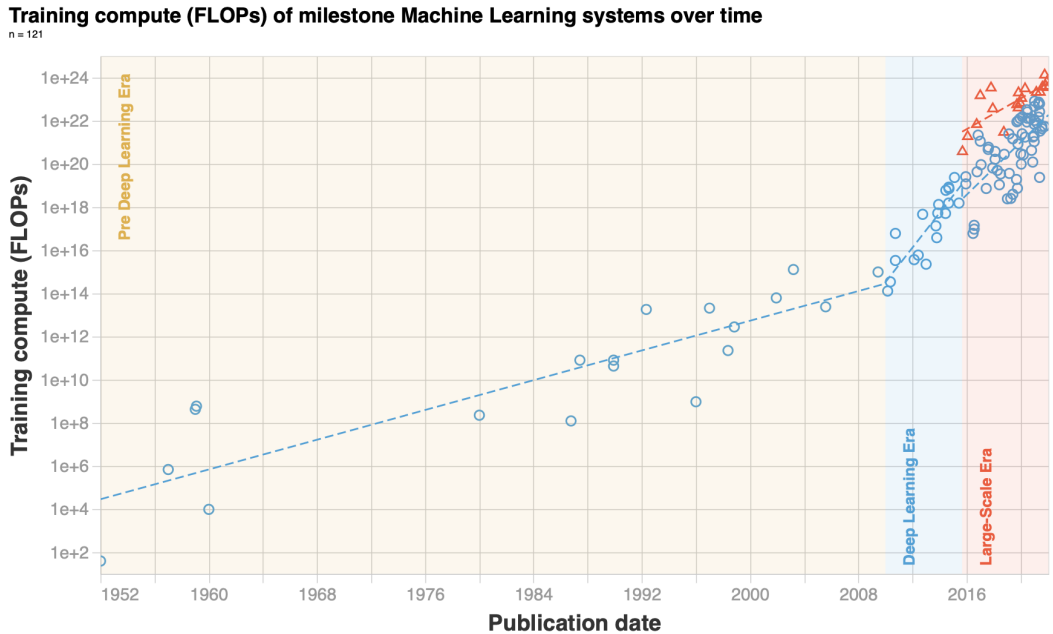


Figure 1: Trends in $n = 121$ milestone ML models between 1952 and 2022. We distinguish three eras. Notice the change of slope circa 2010, matching the advent of Deep Learning; and the emergence of a new large-scale trend in late 2015.

Sevilla et al. (2022)

Compute Trends Across Three Eras of Machine Learning

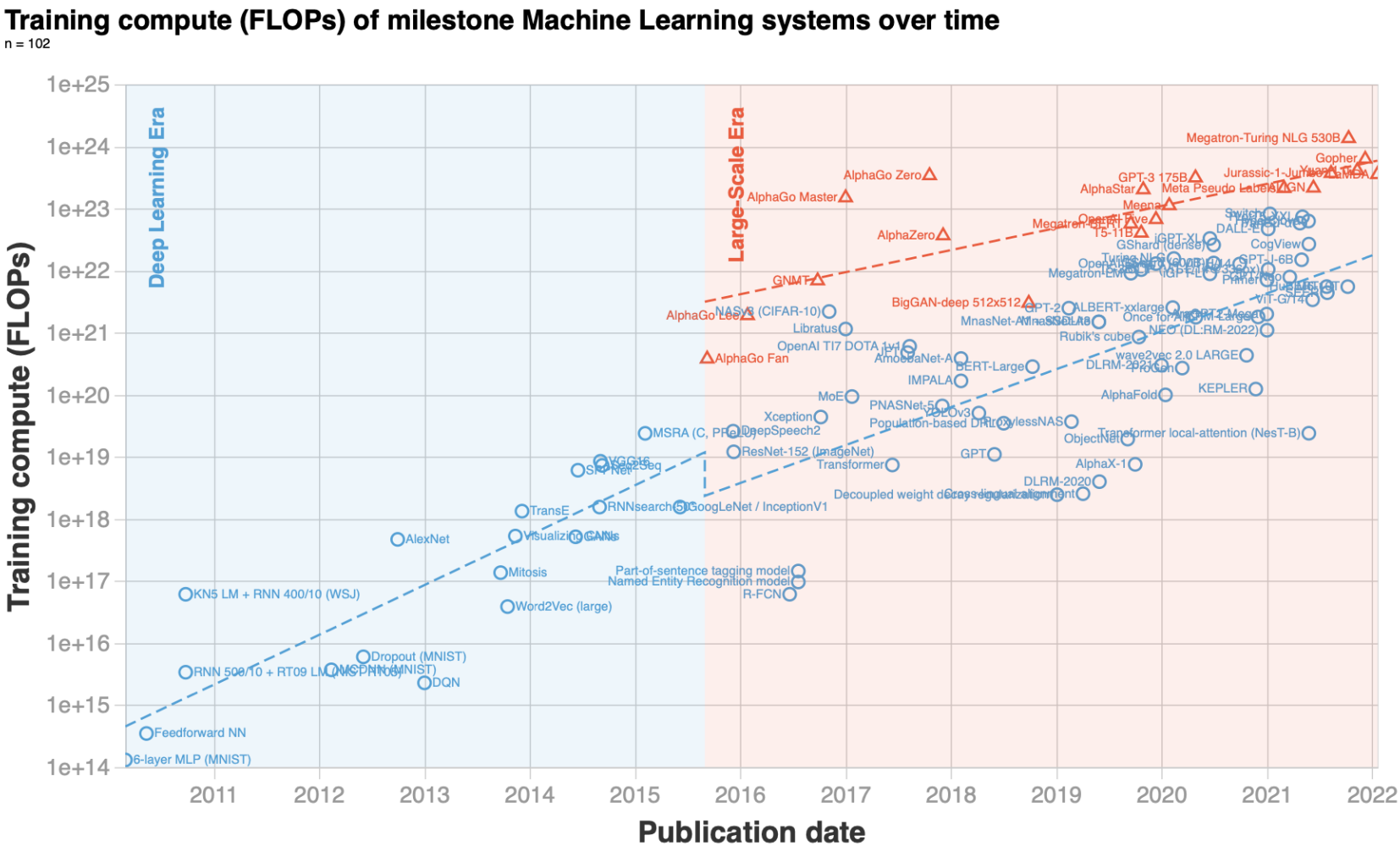


Figure 3: Trends in training compute of $n102$ milestone ML systems between 2010 and 2022. Notice the emergence of a possible new trend of large-scale models around 2016. The trend in the remaining models stays the same before and after 2016.

Sevilla et al. (2022)

Scaling Laws Motivation

Hyperparameter tuning is a huge cost!

- How can we solve this?
  - Guess and pray
  - Exhaustive search
  - Have simple rules that find optimal hyperparameters

Consumption	**CO_2e (lbs)**
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

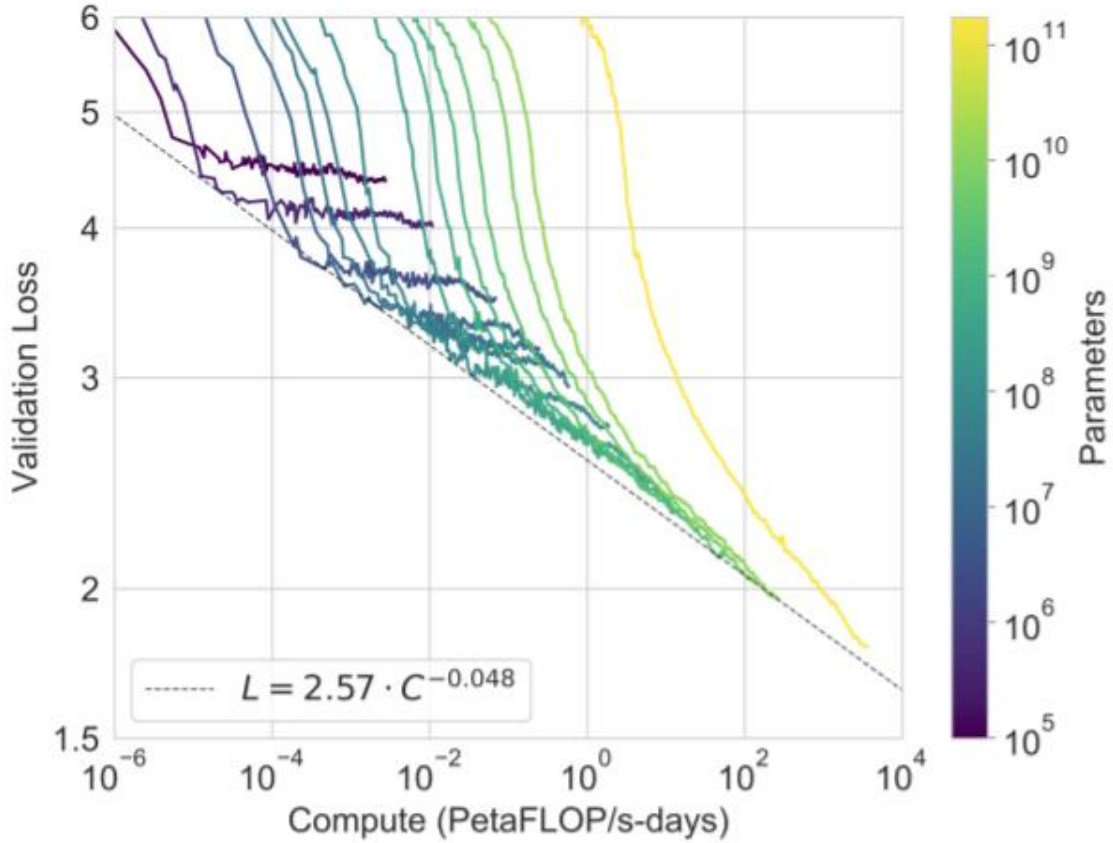
Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

Strubell et al., 2019

What are Scaling “Laws"?

- Simple, predictive “laws" for behaviors of LMs

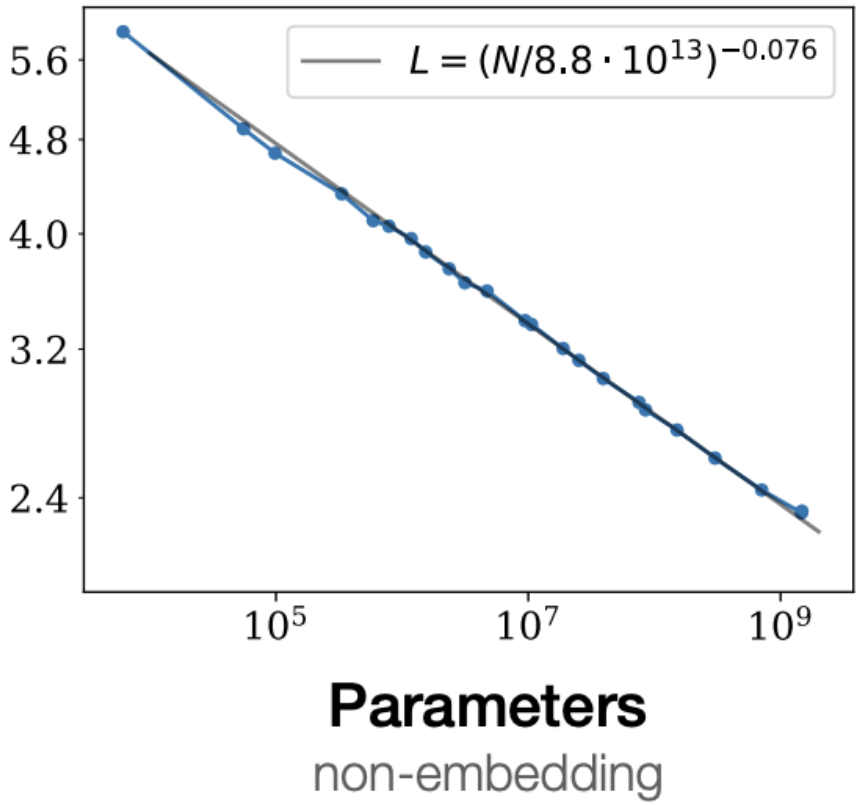
- **Old and unpleasant**: tune hyperparameters on big models
- **New and exciting**: tune on small models, extrapolate to large ones



Data scaling laws for language models: Parameters

test loss and **parameter count** on a log-log scale

- All models trained to **convergence**, **compute constraints** or **data size** did not impact performance
- That is, in alll experiments: only limiting performance with the model size (parameter count), not with the amount of data or the amount of compute.

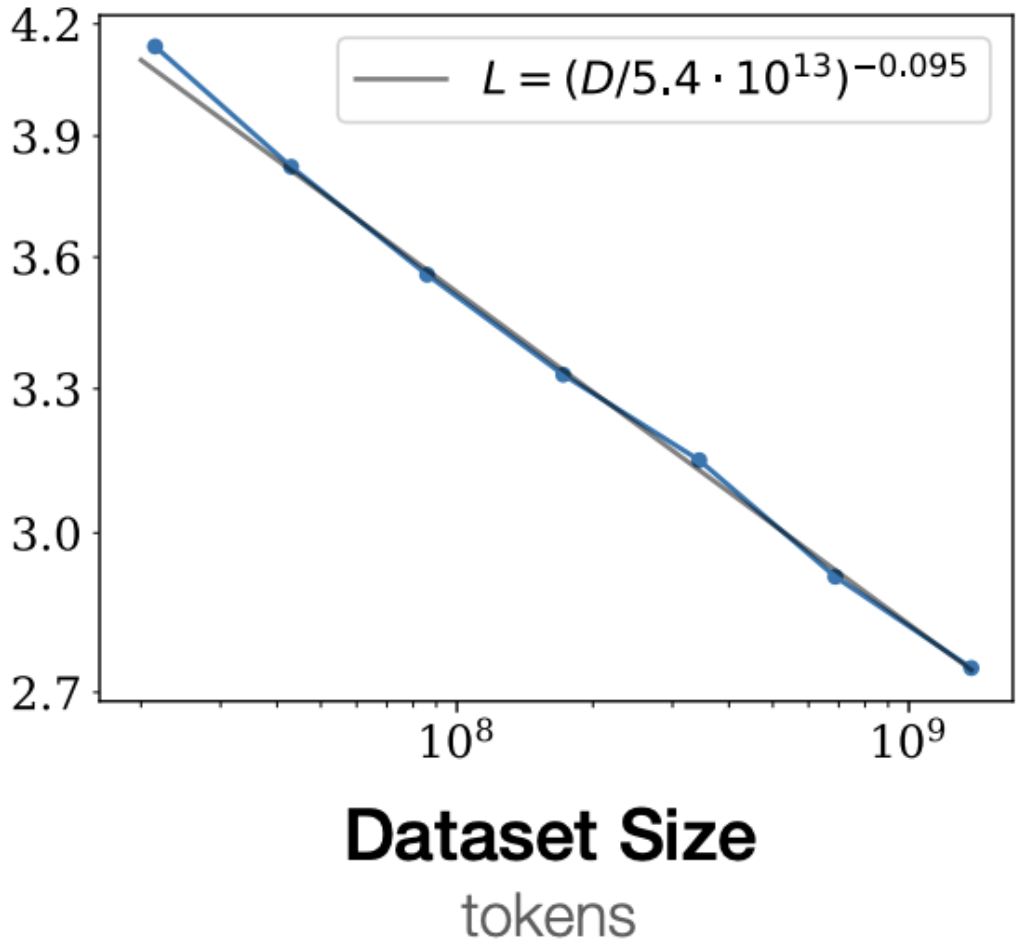


linear trend on log-log plot. Test loss decreases with an increasing number of parameters

Kaplan et al. (2020)

Data scaling laws for language models: Dataset Size

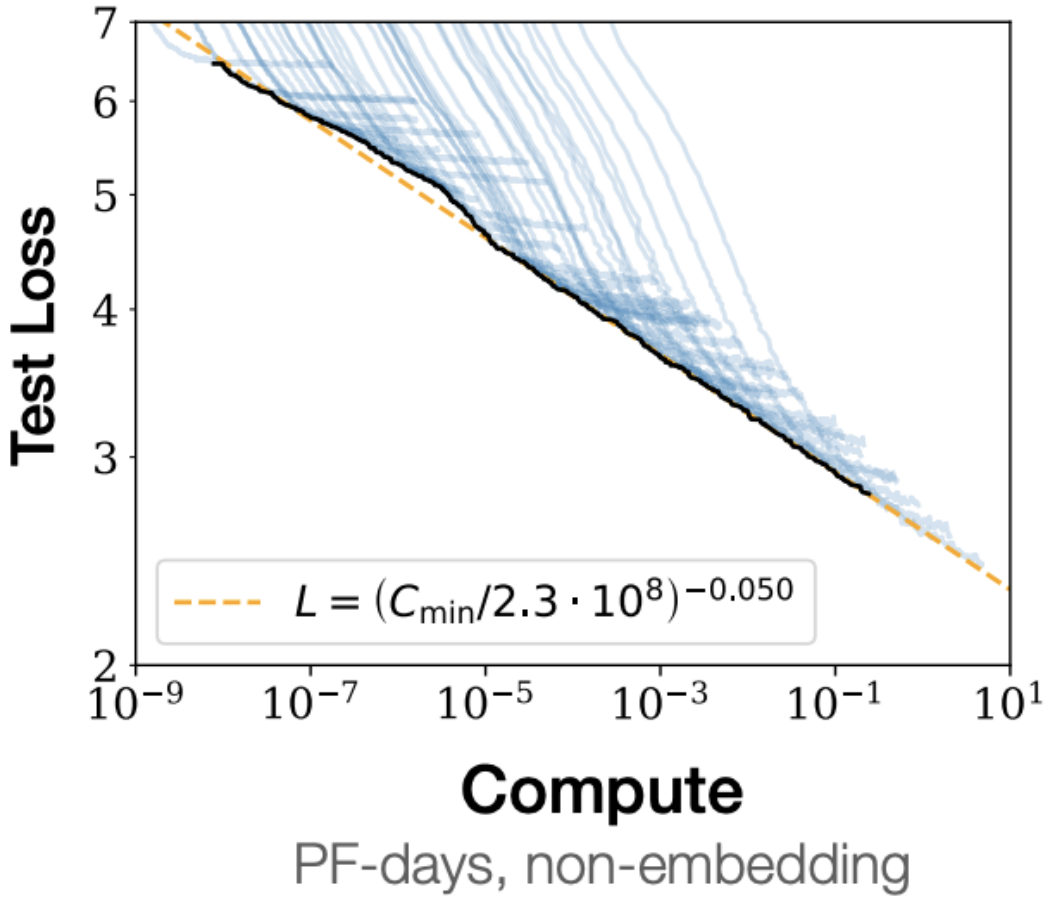
- A very large model is used to ensure that **model size is not a constraint** on performance, focusing purely on the effect of dataset size.
- Increasing dataset size steadily reduces test loss



Kaplan et al. (2020)

Data scaling laws for language models: Compute

- Best **test loss** of models as a function of **compute** (x-axis), measured in PF-days (petaflop days).
- Models are provided with enough data i.e. (not overfitting), and trained for extended time to ensure convergence.
- Different model sizes:
  - **Smaller models** (left)
  - **Larger models** (right)



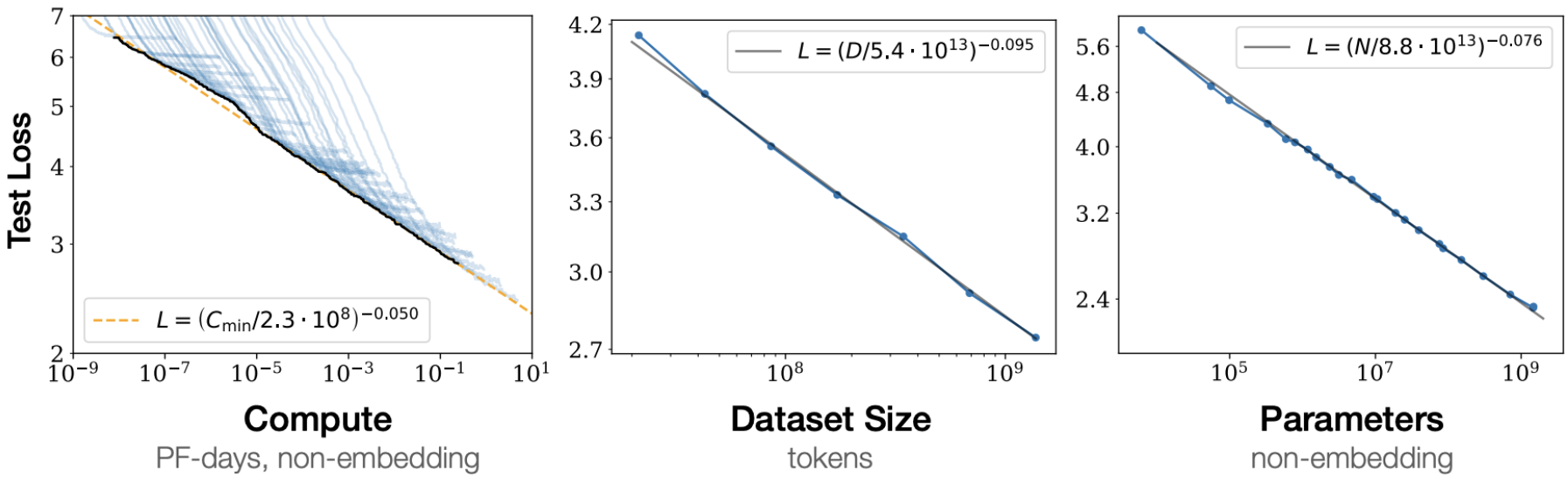
The **black line** and **orange dashed fit**: lowes
achievable test loss for any given compute leve

Kaplan et al. (2020)

Scaling laws: surprisingly clean and robust

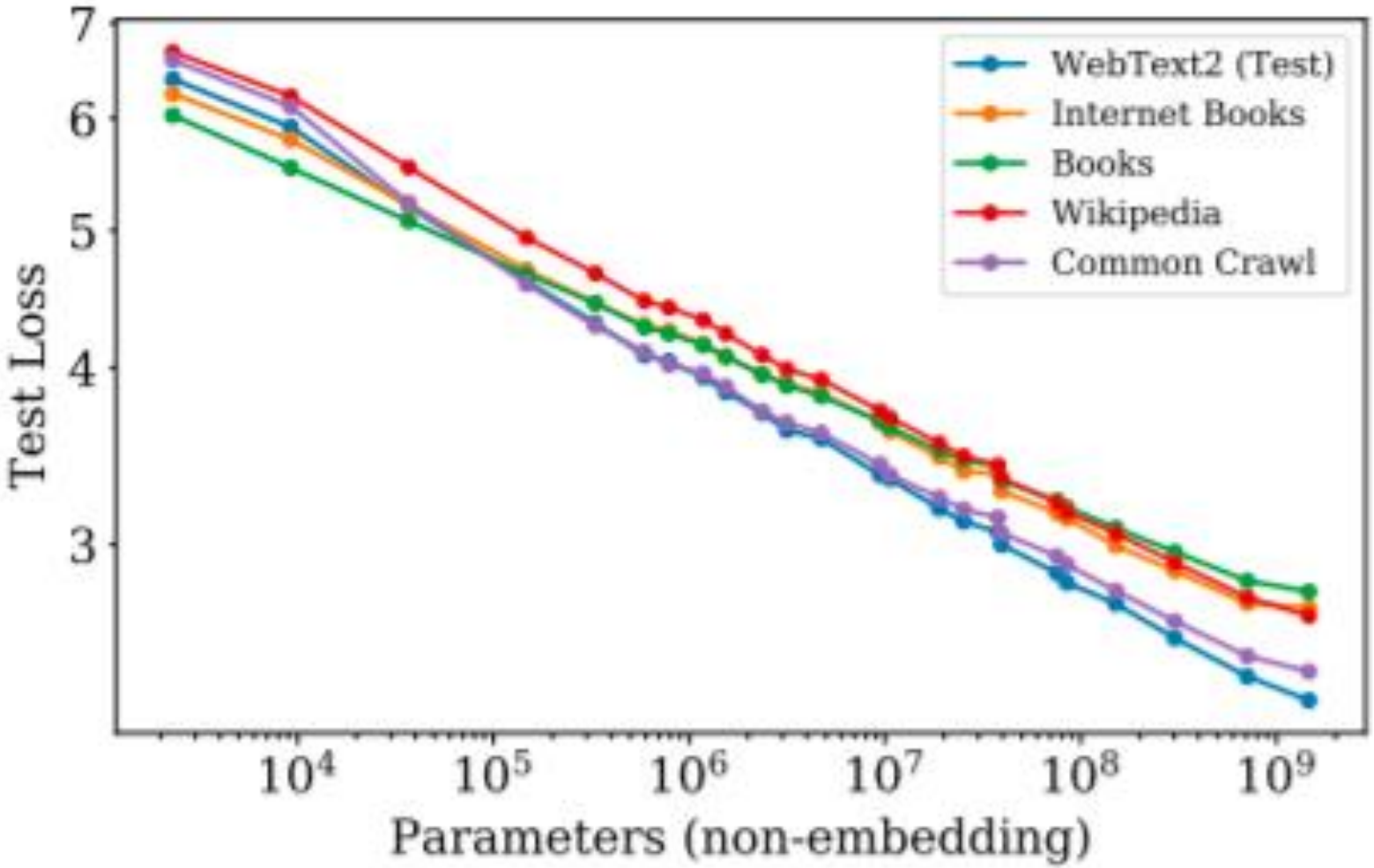
These scaling laws hold on **many** different kind of phenomena

On language modeling tasks:



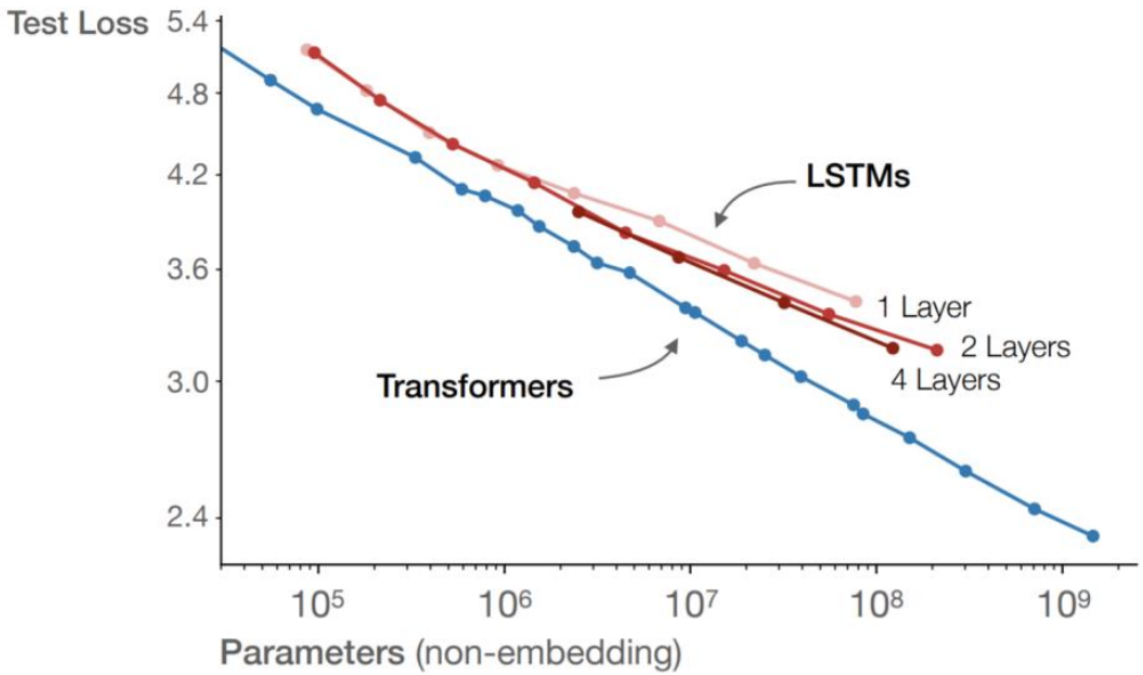
Scaling laws: surprisingly clean and robust

They also hold in non-standard settings (when train $\neq$ test)



Cross-model: transformers vs LSTMs

Q: Are transformers better than LSTMs?
Brute force way: spend tens of millions to train a LSTM GPT-3
Scaling law way:



[Kaplan+ 2021]

Scaling laws for model engineering

- Motivation: how can we efficiently design huge LMs?
  - LSTMs vs Transformers
  - Adam vs SGD
  - ...
- How should we allocate our limited resources?
  - Train models longer vs train bigger models?
  - Collect more data vs get more GPUs?
  - ...

Scaling Laws for Transformer Language Models

- For a given **fixed compute budget (C)**, you have a choice of increasing the number of **parameters(N)** or the number of **tokens (D)**. What is the **optimal combination of** (N, D) ?
- Want to answer this question in such a way that one can then make predictions about what to do for large training runs!
  - Can only run a few experiments at the largest scale

Example: one GPU for 24 hours

What is the best use of one GPU for 24 hours?:

1. 100K parameter LM on 5K books? (small model, large dataset)
2. 5M parameter LM on 100 books? (medium model, small dataset)
3. 50M parameter LM on one book? (large model, tiny dataset)

Scaling laws

Observations from Kaplan et al., 2020

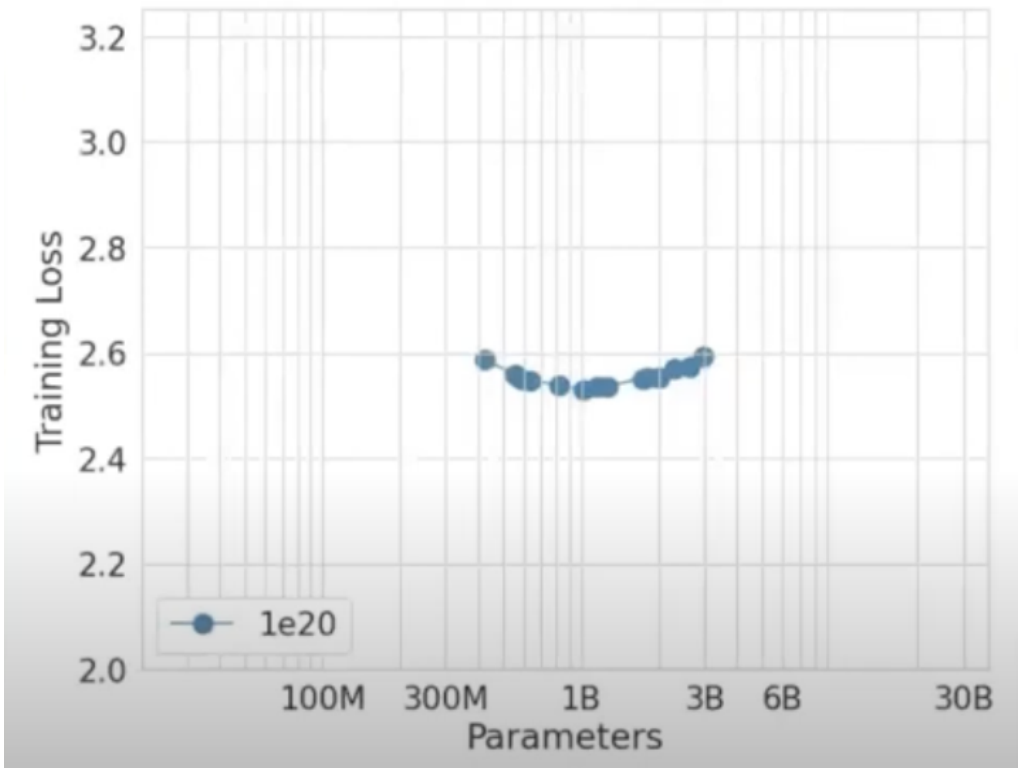
- Performance of much larger models (autoregressive LMs) is accurately predicted by a series of smaller models.
- Performance depends strongly on scale (model params, data size, and compute used for training)
- Performance vs scale can be modeled with power law functions $P \propto S^{\alpha}$

Chinchilla Scaling Laws Approach

Revisit: Given a fixed FLOPs budget, how should one trade-off model size and the number of training tokens?

Approach: Fix FLOPs and vary model size and training tokens

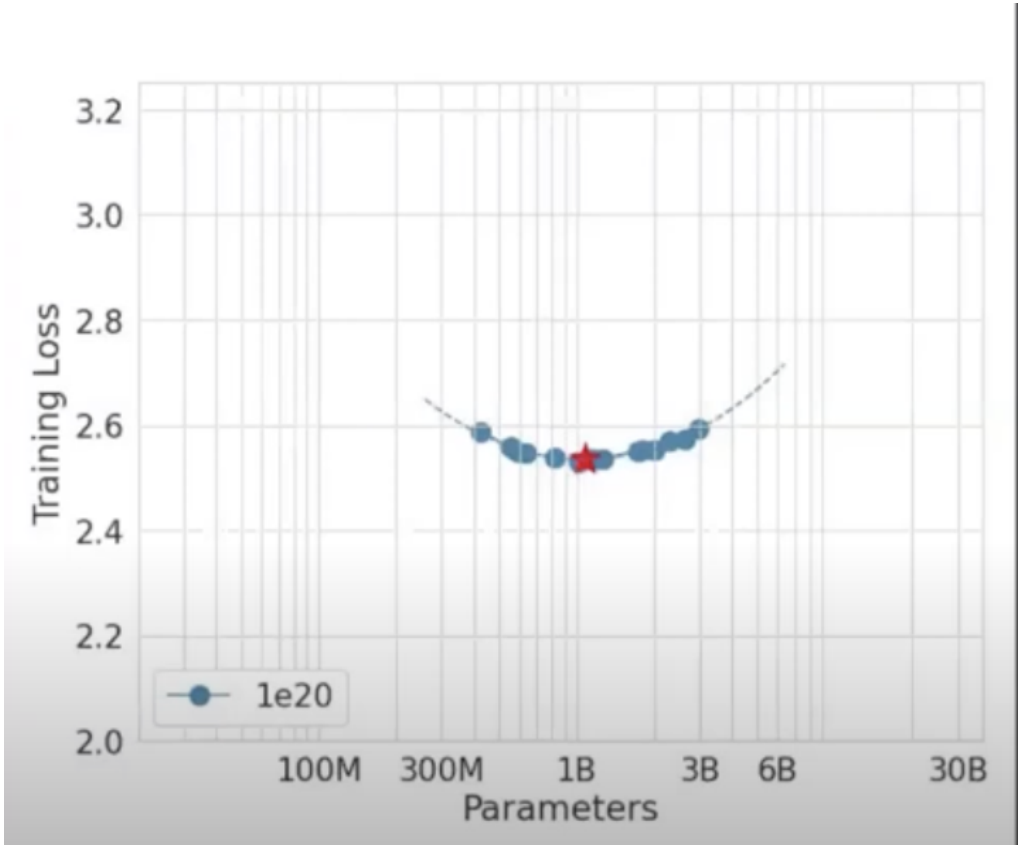
1. Fix a target FLOPs budget
2. Train a few models, vary model size



Hoffmann et al. (2021)

Fix FLOPs and vary model size and training tokens

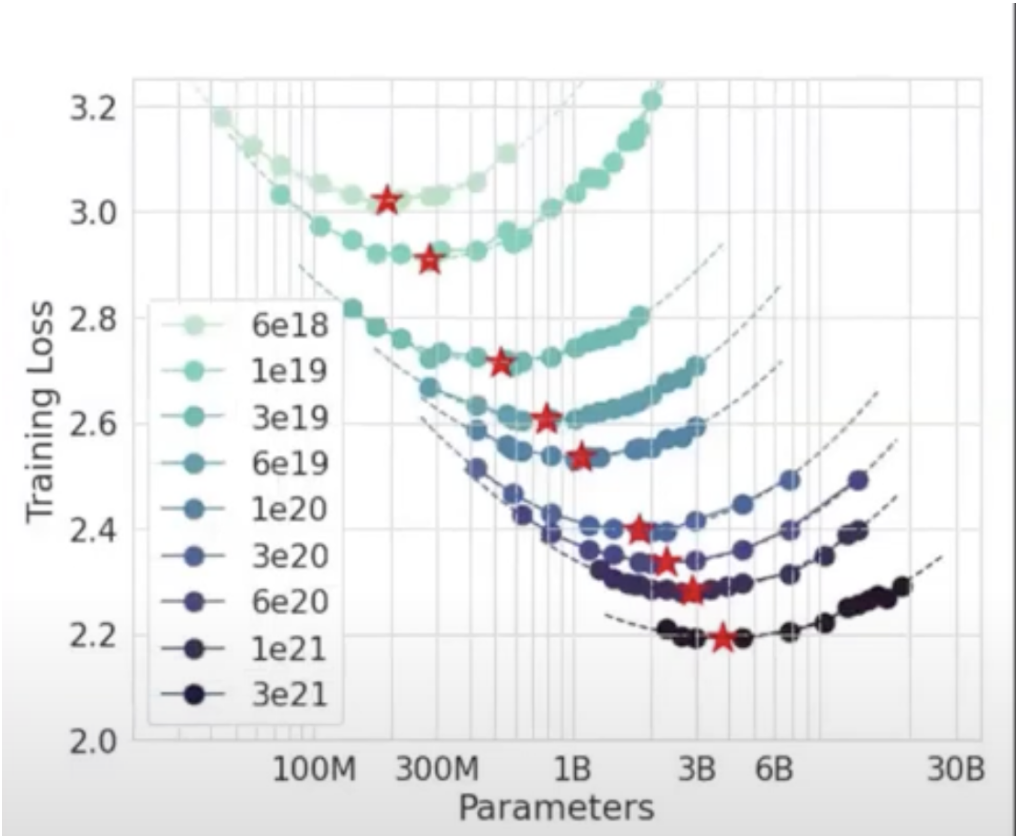
1. Fix a target FLOPs budget
2. Train a few models, vary model size (we choose training tokens such that the final FLOPs is a constant)
3. Fit a parabola and take the minimum



Rae et al. (2021)

Fix FLOPs and vary model size and training tokens

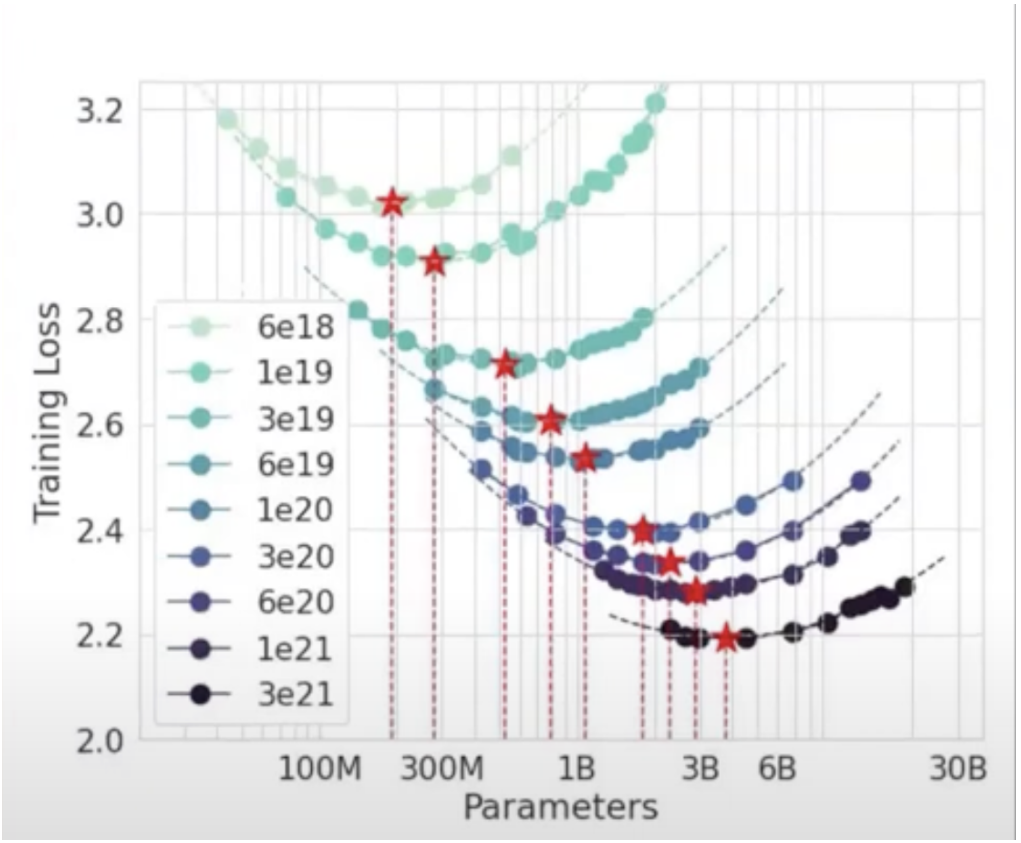
1. Fix a target FLOPs budget
2. Train a few models, vary model size
3. Fit a parabola and take the minimum
4. Repeat steps $1-3$ for various FLOPs budgets



Rae et al. (2021)

Fix FLOPs and vary model size and training tokens

1. Fix a target FLOPs budget
2. Train a few models, vary model size
3. Fit a parabola and take the minimum
4. Repeat steps $1-3$ for various FLOPs budgets

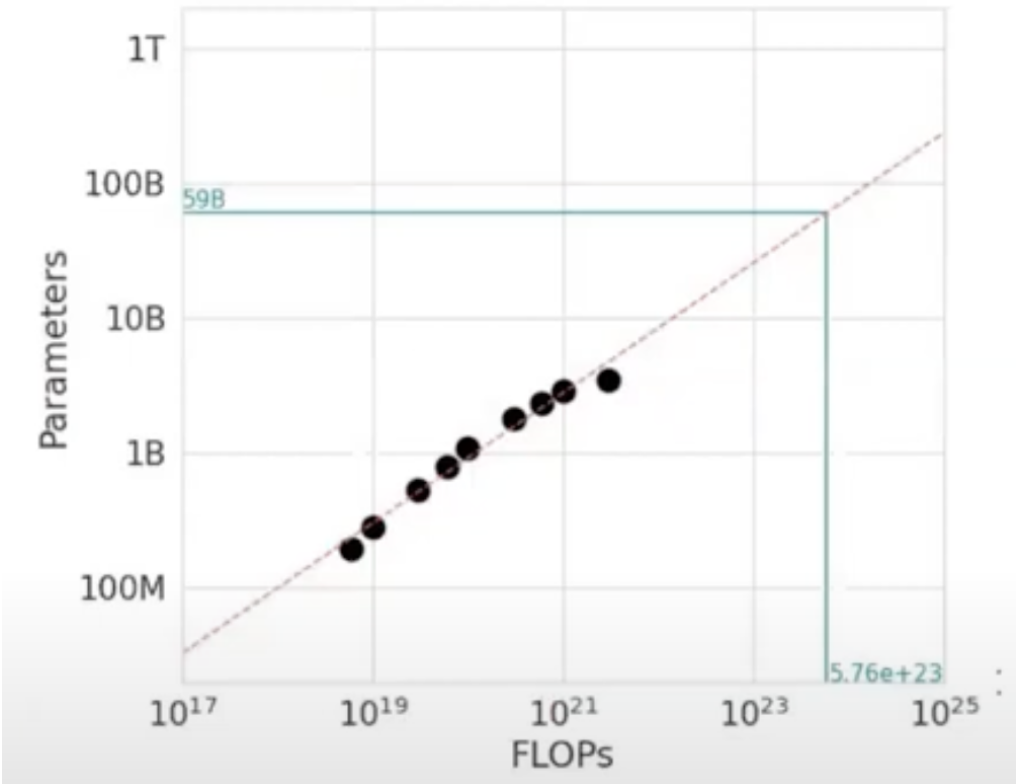


Can now extrapolate parameter count for each FLOPs budget

Rae et al. (2021)

IsoFlops: Fix flops and vary model size and training tokens

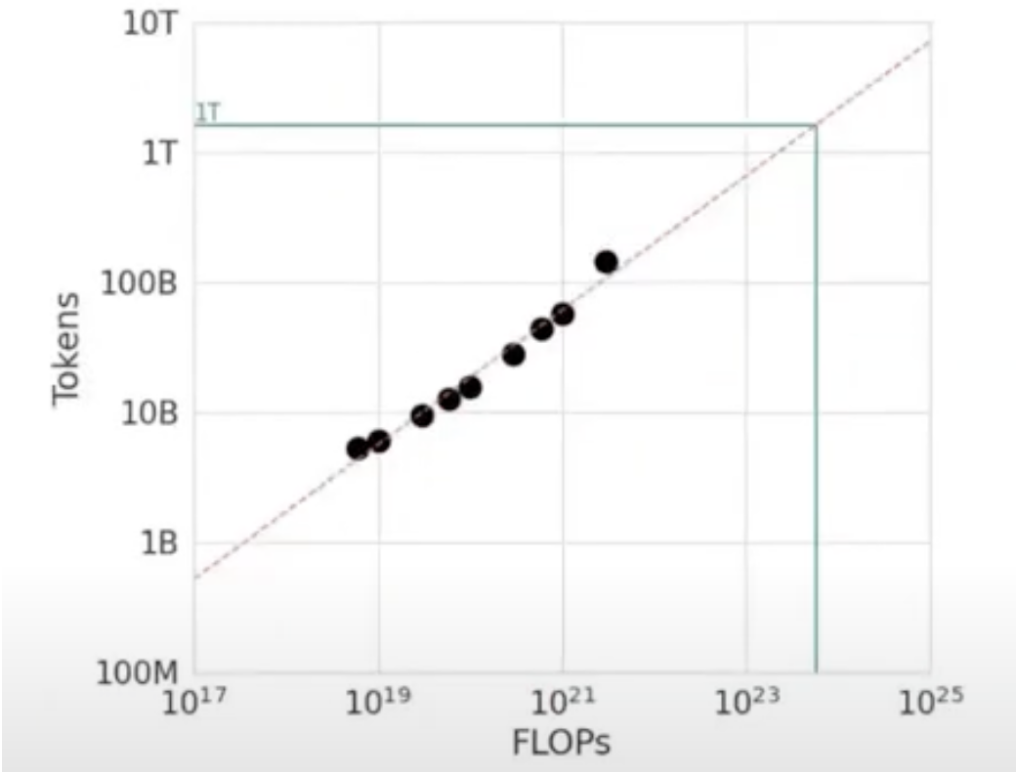
1. Fix a target FLOPs budget
2. Train a few models, vary model size
3. Fit a parabola and take the minimum
4. Repeat steps $1-3$ for various FLOPs budgets
5. Fit a power law between FLOPs budget and optimal model size



Rae et al. (2021)

Fix flops and vary model size and training tokens

1. To the same thing, but fit between FLOPs budget and optimal token count



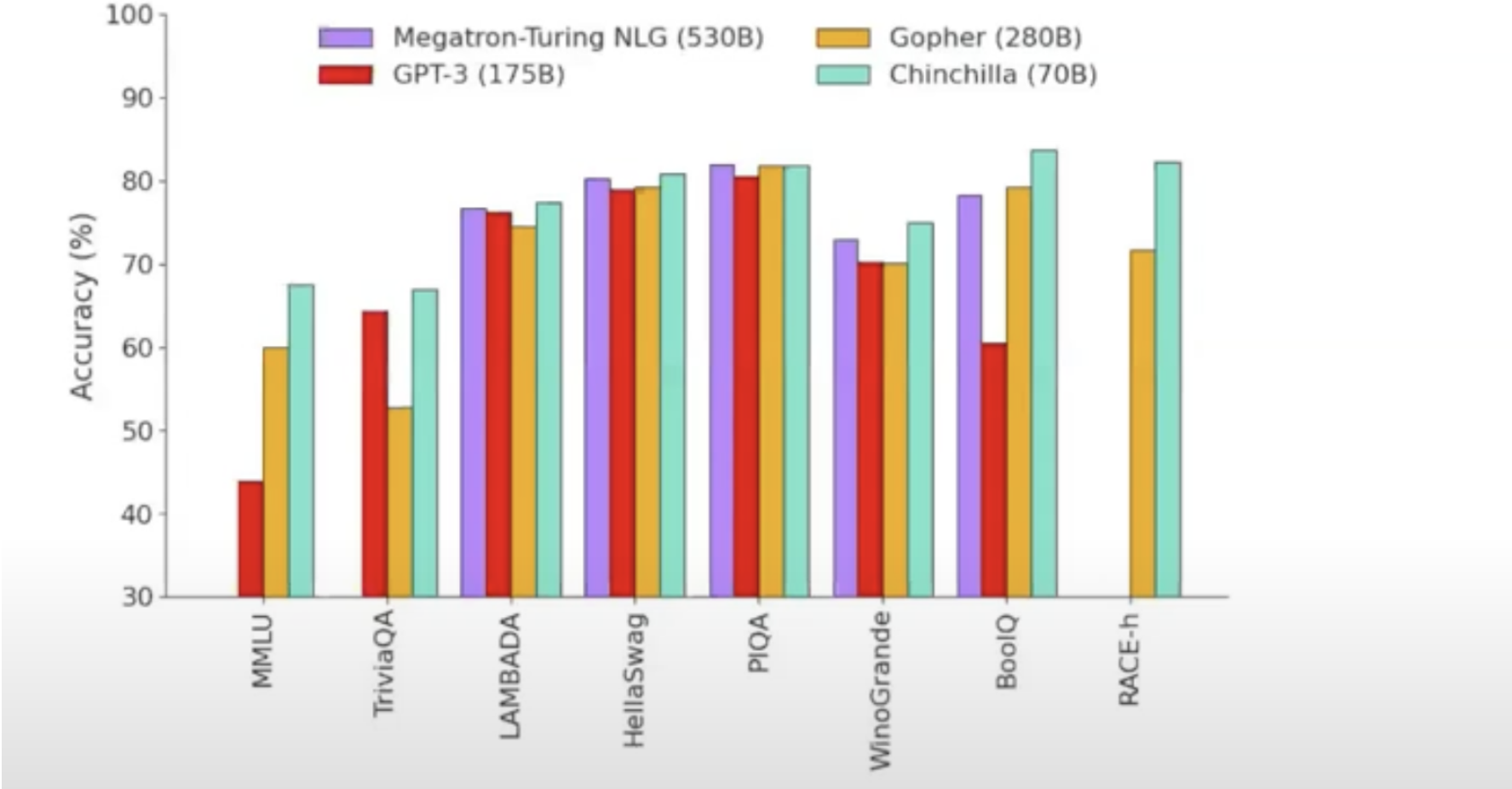
Rae et al. (2021)

Results: Chinchilla

Trained on the exact same FLOPS, same compute as Gopher, but with a different model size and number of tokens (from the same data distribution).

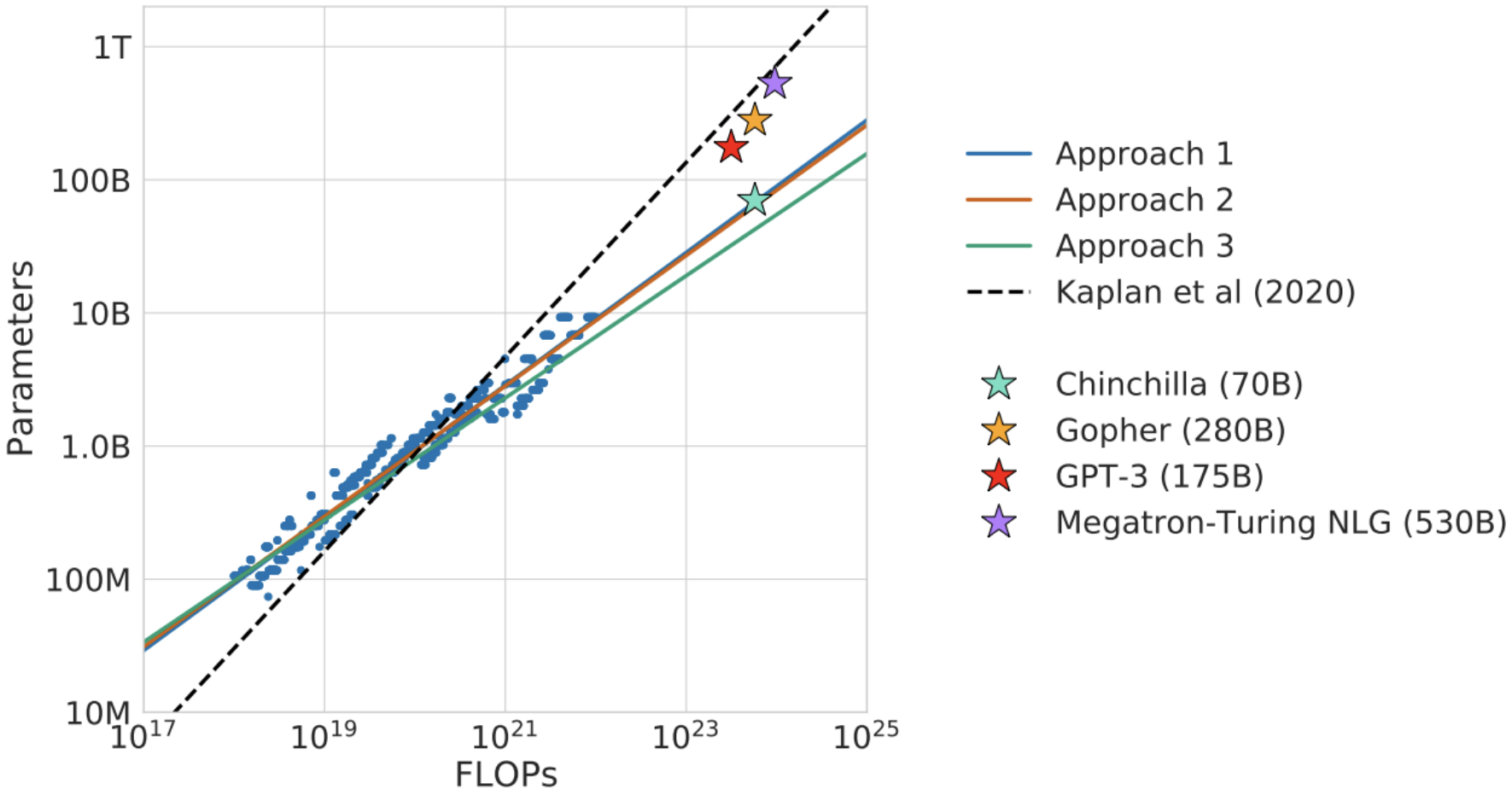
Model	Size (#Parameters)	Training Tokens
LaMDA	137 Billion	768 Billion
GPT-3	175 Billion	300 Billion
MT-NLG 530B	530 Billion	270 Billion
Palm	540 Billion	768 Billion
OPT	175 Billion	300 Billion
Bloom	176 Billion	366 Billion
Gopher	280 Billion	320 Billion
Chinchilla	70 Billion	1.4 Trillion

Chinchilla, Downstream tasks results



Significant performance gains from a much smaller model (70B) "optimally" trained for the same number of FLOPs as Gopher (280B)!

Many Large Models were too Large for their FLOP Budgets



Hoffmann et al. (2021)

Some surprising takeaways

- The effect of hyperparameters on big LMs can be predicted before training!
  - Optimizer choice
  - Model depth
  - Architecture choice

1. Train a few smaller models
2. Establish a scaling law $P \propto S^{\alpha}$
3. Select optimal hyperparam based on the scaling law prediction.

Scaling Laws Assumptions

1. The compute vs optimal model size can be modelled as a power law
   - Simple model that fits well with empirical evidence
   - Linear relationship in log-log space
2. "All tokens are created equal": We train on a fixed uniform dataset
   - Very unlikely to be true, but matches current training of LLMs
   - Training is in the < 1 epoch regime
3. The model is fixed: Auto-regressive Transformer
   - Unlikely to be the "optimal" model
   - The best model currently available

Questions?