Principles of AI: LLMs

Edgar Dobriban*

December 18, 2024

Abstract

This document aims to fulfill several roles: (1) introduce large language models in the style of lecture notes going from the very basics to the frontiers in some areas, (2) discuss some the key recurring ideas and principles in their design and analysis, and (3) identify key limitations and research directions. We aim to introduce both some of the most influential works, as well as some of the latest and most exciting findings. However, we do not aim to be a comprehensive review. We aim to balance the description of practical algorithms with the insights arising from fundamental research.

Contents

*Department of Statistics and Data Science, The Wharton School, University of Pennsylvania. dobriban@wharton.upenn.edu. These notes draw inspiration from many sources and may contain factual and typographical errors.

1 Goal: Build AI

The goal of artificial intelligence AI is to develop a system that behaves as a competent human in solving problems and performing tasks.

1. Definitions:

   OpenAI: artificial general intelligence (AGI) is a highly autonomous system that outperform humans at most economically valuable work

   Marvin Minsky: "AI is the science of making machines capable of performing tasks that would require intelligence if done by humans" (Minsky, 1988) [specific tasks]

   McCarthy/ Hernandez-Orallo (McCarthy, 1987; Hernández-Orallo, 2017): "AI is the science and engineering of making machines do tasks they have never seen and have not been prepared for beforehand". [new tasks]

2. More basic notion: intelligence

   American Psychological Association on intelligence (Neisser et al., 1996): "Individuals differ from one another in their ability to understand complex ideas, to adapt effectively to the environment, to learn from experience, to engage in various forms of reasoning, to overcome obstacles by taking thought."

   Legg et al. (2007) list 70 definitions of intelligence and conclude: "Intelligence measures an agent's ability to achieve goals in a wide range of environments."

For humans, being skilled at a specific task requires using general skills (e.g., no-one is born knowing how to play chess), but not the same for algorithms (Chollet, 2019). So, performance on a specific task does not imply AGI.

Chollet (2019): "The intelligence of a system is a measure of its skill-acquisition efficiency over a scope of tasks, with respect to priors, experience, and generalization difficulty."

3. In particular, the goal is not to build a copy of a human. History shows that humans can have a lot of intellectual and character defects, and make both individual and collective mistakes; think corruption, power-seeking, wrongful societal trends (inequality of women, slavery), etc. We do not want these bad behaviors exhibited in our AI systems

4. Historical note: the term artificial intelligence dates back at least to the 1950s, see e.g., https://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html.

5. Is AGI possible? Are there any limitations related to physics, energy, computation, data, ethics? [!]

   - Is it possible to build a system that behaves as a competent human in **daily tasks**?
   - What tasks?
     - e.g., **cook:** [related to Steve Wozniak's coffee cup test (entering a previously unknown kitchen and making a cup of coffee (Wozniak and Moon, 2007))]
       * decide recipe
       * pre-order groceries, pick up
       * cook
       * serve
       * wash dishes, throw away trash
   - Focus only on intellectual/computational problems?
     (Has advantages → can do it on a computer)
   - Otherwise, have to build robots
   - Could all cognitive tasks of a human be performed by a computer?
     - Brain — it's ultimately made of atoms, so, if we can build the exact same organization of atoms, should be able to do the same.

   So, the question is:

   A. Try to build exact brain copy?
      or
   B. Build simplified system.

   A: If building it, it will require tools to engineer approximately 86 billion neurons in correct arrangement.

   - Seems far out of reach.
   - Structure of fly brain has been recently found:
     - 140k neurons

– 50 million synapses

B: Currently, only feasible-looking option:

- Already lots of neural network architectures exist.
- Not directly biologically inspired; some **huge gaps** in the abstract framework, but works well for many tasks.
- Give up mimicking **exact biology** and only borrow loose inspirations.

1.1 Key Requirements

1. The system must be capable of processing information of the same type and in the same manner as a human being. This includes:
   (a) Visual information
   (b) Textual information
   (c) Sound, video

   E.g., input/output this type of information.

   However, this is not restrictive. E.g., can hope for video generation, which humans are unable to do natively
2. The system should also be able to take actions, in an appropriate environment.
3. Sub-goal: Process Text
   (a) How to solve it? The current State of the Art (SOTA) is based on Large Language Models (LLMs).
   (b) Steps listed in Section 2

1.2 Applications

1. Here are some of the most successful and/or promising areas of application
2. General-purpose chatbot assistants (ChatGPT, Claude)
3. Code generation and editing assistants e.g., Chen et al. (2021)
4. Protein structure prediction (AlphaFold)
5. Image generation

1.3 Some Key Principles

1. There is an incredible amount of work and details in the area.

2. However, the number of key ideas and principles is smaller. If one understands these ideas, the hope is that one can (1) easily understand new developments and appreciate new ideas; (2) predict the course of developments; and (3) see the best opportunities for future work.

3. This course seeks to organize and identify some of these principles. Here are a few particular ones:

   (a) Reduce any problem to supervised learning: predict y based on x; possibly repeatedly
   e.g., LLMs: iterated next-token prediction
   Diffusion models: Iterative denoising, prediction of x from $x + \varepsilon$

   (b) Fit a large/expressive model to a large/diverse dataset, by minimizing prediction error. Typically use a neural net
   If a human can solve it, and you can collect enough data, then a model can solve it; or, alternatively; if you do not know, learn it (e.g., learn reward)

   (c) Transform data of varying modalities into a unified modality: embeddings/representations. Then manipulate/change these

   (d) Decompose task into smaller, manageable parts; build modular components [general problem-solving principle]
   These can be functionally distinct: e.g., training LLMs via pre-training and fine-tuning
   Or, can be repetitive, in which case we only make small changes at a time: e.g., layers of a NN, ResNet, GD steps. Making small changes at the time is crucial for stability.

   (e) Test scenarios by simulating & manipulating data (Possibly intertwined with training)
   Examples include training for robustness (red-teaming), training with synthetic data (e.g., from larger language models), data augmentation, study biases of models by synthesizing data, etc.

   (f) Automate components of your tasks by substituting AI/LLMs for human labor
   E.g., RLAIF, jailbreaking LLMs via LLMs, etc.
   "If you do not know, ask an LLM"

1.4 General Views on AI

1. The Turing test (Turing, 1950): can you distinguish between an AI and a human via text interactions?

   If AI knows the distribution of human text, then it can pass (in a statistical sense).

   Knowing a distribution is (roughly) equivalent to optimal compression. This is one of the basic claims of information and coding theory (Cover and Joy, 2006; Salomon, 2007).

   This is the base for the claim that "Intelligence is equivalent to compression", see e.g., https://mattmahoney.net/dc/rationale.html

2 Large Language Model Architecture

2.1 Input and Output

1. What should be the input and output of the system?
   (a) **Input**: Text
   (b) **Output**: Text
   e.g., question-answering $Q \to A$, translation, summarization, $Q_1, \ldots, Q_n \to Q_{n+1}$ etc.
   Key strength: a generic format that can cover many tasks
   Key refs: Kalchbrenner and Blunsom (2013) introduces autoregressive decoding (for a specific discourse task)
   Sutskever et al. (2014) introduces sequence to sequence (seq2seq) modelling: taking inputs and outputs of arbitrary lengths

2. How to operationalize this? Current approach involves the following:
   (a) Represent text as a sequence of symbols: $S = (s_1, s_2, \ldots, s_k)$ from a fixed universe. Assume bounded (context) length
   (b) Use next symbol/word prediction; reduce to supervised learning/break task into smaller sub-tasks
      i. Input $I = (i_1, i_2, \ldots, i_n)$ leading to output $O = (o_1, o_2, \ldots, o_m)$.
      ii. Build smaller systems to predict one symbol at a time. Then iteratively
      $I \to o_1$
      $I, o_1 \to o_2$
      $\ldots$
      $I, o_1, \ldots, o_{m-1} \to o_m$
   (c) Advantages and disadvantages of the approach:
      i. **Advantages:**
         A. Generic format.
         B. Simpler sequential tasks.
      ii. **Disadvantages:**
         A. Sequential processing is non-parallelizable.
         B. If mistakes are made, there is no mechanism to efficiently backtrack. [Active research direction: Diffusion LMs]
         C. Exponentially decreasing correctness probability.
   (d) Drawback of the entire prediction approach: It does not have an immutable knowledge base, and does not have nay built-in behavior guarantees/rules that it respects. We have no idea what it really does, or why it works.
   Unlike much of CS/engineering! e.g. think about programming languages, hardware, databases, etc. Each have precise specifications, and you know what you get.
   Even in statistics, we have probabilistic guarantees; think of confidence intervals.
   In AI, there are no guarantees. No precise performance analysis (other than running time/space).
   The entire line of work on "emergence" is concerned with unpredictable behavior, that we can see spontaneously appear during training

(e) Additional drawbacks:

Hard to control precisely, to meet specified requirements. [can prompt, or modify representations; but these have their own problems]

(f) Research question: What is lacking in this self-supervised, next-token prediction paradigm? How do humans learn? Initially for small kids, a lot of direct grounding by demonstrations/supervised learning (point to cat; say "cat"). Later, when encountering new concepts, need to explain them based on already known concepts (e.g., catfish is a type of ray-finned fish with barbels around mouth, also see picture). If something is still unclear, define that as well. This is lacking entirely in LMs. Can we build such a learning system? [Difficulty: requires interacting with expert who can provide reliable grounding.]

3. Revisit 2(a): What should a "symbol" be?

(a) Options for Symbol Representation

i. **Letters/Characters**: Examples include 'a', '+', '1', etc.

ii. **Words**: Examples like 'cat', 'dogs', etc.

(b) Considerations

i. The smallest symbol set could be characters (e.g., 255 ASCII).

ii. Words could represent a larger set, especially in tasks involving natural language (e.g., 100K words across 1K languages).

iii. There is often redundancy in word-based systems. For example, words like 'cat' and 'cats' or 'dog' and 'dogs' are closely related, and the system should know this.

(c) Compromise: Tokens

i. Tokens represent the basic units, and correspond to subwords

ii. Example: Splitting the word 'subword' into '[sub][word][s]'.

iii. Online example https://platform.openai.com/tokenizer

iv. Tokenization can be learned for efficiency, e.g., Byte Pair Encoding (BPE) (Sennrich et al., 2016)

v. Typical vocabulary sizes range from 50,000 to 100,000.

vi. This is also key to adapting transformers to other modalities, e.g., images - tokens can be patches.

vii. It is also key to extending the capabilities of models, e.g., can add a token for a function call.

2.2 How to Construct the Model?

1. Naive approach: (Shannon, 1948): Use empirical/plug-in probability estimate:

$$P(x_i|x_{1:i-1}) = \frac{\#(x \in D : x = x_{1:i})}{\#(x \subset D : x = x_{1:(i-1)})}$$

This counts how frequently x_i follows a given sequence in the dataset.

Problem: This approach is not well-defined for strings $x_{1:i-1} \notin D$; even small semantic reformulations can make it ill-defined (e.g., the probabilities $P(\cdot|x)$ should be similar for x being "The capital of France is Paris" and "Paris is the capital of France".

Simple fixes that do not work:

(a) Shrinkage towards uniform distribution: uninformative for unseen data

(b) Shorter Markov model: may still have same problem (uninformative), may lose expressivity

2. Alternative (Bengio et al., 2000): Learn a probabilistic model with another—flexible, but restricted—parametrization

3. Collect a dataset or corpus of strings $D = \{X_1, X_2, \ldots, X_n\}$.

4. The task is to predict the next symbol or word in a sequence. Given a sequence $(X_{1,1}, \ldots, X_{1,i-1})$, predict $X_{1,i}$.

5. Predictor function $f : V^{i-1} \to V$; where V is vocabulary of tokens

6. Zero-one loss is calculated as: $I(f(X_{1,1}, \ldots, X_{1,i-1}) \neq X_{1,i})$

7. We would like to optimize or learn the function f, but it is hard to optimize, due to the following challenges:

   (a) The loss function is discontinuous.

   (b) The space of functions $\mathcal{F} : V^{i-1} \to V$ is discrete.

   Well known challenges in classification.

8. Machine learning solution: relaxation

   (a) Relax the space of functions to output probability distributions over V.
   The probability $P(x) = (P_1(x), \ldots, P_{|V|}(x))$ becomes a probability distribution over V, $P(x) \in \Delta(V)$.

   (b) Introduce a loss function for multi-class classification, where the outcome is $v \in V$, and the prediction is a probability distribution over V. These are known as *scoring rules*.
   e.g., logarithmic scoring rule
   $$\ell(P, v) = -\log P_v$$
   [others scoring rules (quadratic, etc) exist, how do they work?]

9. How to aggregate over multiple data points: classical stat/ML approach is Empirical Risk Minimization (ERM), average
   $$\sum_{x \in D} \sum_{j=1}^{|x|} \ell(P(x_{1:(j-1)}), x_j)$$
   For log loss: $\sum_{x \in D} -\log P(x)$; minimize over P.
   If strings are independent, can view this as Maximum Likelihood Estimation (MLE) for density estimation over V^k.

10. Despite not being a plug-in estimate, still just a pattern-matching/density estimation/statistical approach. Does not have any explicit built-in rules, e.g., (1) grammar; (2) logic; (3) math. [Building this is a research challenge: e.g., logical AI, neuro-symbolic systems]
    Also, it is probabilistic, and so it can make silly mistakes due to random chance.

2.3 How to Parametrize P? Attention and transformers

1. In principle, any parametrization for multiclass classification can be considered (multiclass logistic regression, kernel classification, ...).

2. However, recall that the natural empirical distribution pools information in the original space of data.

   This is not quite directly what we need (does not take semantics into account).

3. Instead, aim to perform pooling in a **semantic representation/embedding space**.

4. For example, tokens $x_1, x_2, \ldots, x_T$ may be mapped into embeddings $e_1, e_2, \ldots, e_T \in \mathbb{R}^d$. Then, do logistic/kernel classification on those embeddings.

5. Embeddings are learned during training and represent the tokens in a continuous vector space.

   [One of the most crucial ideas in AI. Embeddings are the solution to many problems.]

   What can be done in representation space?

2.3.1 Attention Mechanism

Key refs:
Bahdanau et al. (2015), credited with neural attention architecture
Vaswani et al. (2017), modern inner-product based attention

1. Formulation: Given embeddings $e_1, e_2, \ldots, e_T$, we need to learn which previous embeddings to look at when making a next-token prediction.

   This depends on the context.

2. Example: "X was born on Tuesday (...). What day of the week followed X's day of birth?"

3. **Operationalize:** The attention mechanism assigns weights to different embeddings based on their relevance:
$$a(e_j, e_k) = \text{how much attention } e_j \text{ pays to } e_k$$

4. Given an attention mechanism, we compute a weighted sum of *values* V_i. Specifically,
$$\hat{V}_j = \sum_{i \le j} a(e_j, e_i) V_i$$
   where $a(e_j, e_i)$ is the attention weight.

5. Relation Between Values and Embeddings: The values V_i are related to the embeddings e_i via a linear transformation:
$$V_i = W_V e_i$$
   [Could imagine using plain embeddings. Having the values (1) can explicitly allow us to focus on only the part of the embeddings that are relevant for making the next-token prediction directly (these can carry other meaning as well; e.g., about the context, not directly relevant), and (2) reduce dimension]

6. How to Define $a(\cdot)$?

(a) $a(\cdot)$ is a measure of how similar certain aspects of embeddings e_j and e_i are. These aspects are

i. q_j represents the **query**, i.e., "what I'm looking for." $q_j = W_q e_j$

ii. k_i represents the **key**, i.e., "what I have." $k_i = W_k e_i$

(b) Specifically, Vaswani et al. (2017) defined an inner product attention as:

$$a(e_j, e_i) \propto \text{function of}\,(q_j^\top k_i)$$

And concretely

$$a(e_j, e_i) \propto \exp(q_j^\top k_i / \sqrt{d})$$

(c) Where does $\sqrt{d}$ come from? The typical size of an inner product between two random vectors with i.i.d. standardized entries.

(d) Normalized using softmax:

$$a(e_j, e_i) = \frac{\exp(q_j^\top k_i / \sqrt{d})}{\sum_{i' \le i} \exp(q_j^\top k_{i'} / \sqrt{d})}$$

For numerical stability, can first calculate the max of the terms $q_j^\top k_i$, and subtract that from all terms before taking the exponential.

(e) The above expression, which depends only on data observed up to token i, is sometimes called causal attention (or attention with causal masking).

It is required for auto-regressive generation

(f) Bidirectional attention can be used when the full data is available at the start of generation (e.g., text-based classification). The algorithm is the same, but the summation is over all keys.

(g) Bahdanau et al. (2015) define a hybrid RNN-attention language model with an RNN state s_i in addition to the embeddings e_i. They also define attention as a function $a : (s_j, e_i) \mapsto a(s_j, e_i)$, and parametrize it as an MLP

7. How to think about attention?

Consider an example.

(a) The goal is to find animals at every position. The input is "This is a dog." Each word is a token.

(b) Queries: "Animal?" repeated 4x

(c) Keys: "Not animal," "Not animal," "Not animal," "Animal". [The representation of "Animal" and "Animal?" is highly correlated, while the representation of "Not animal," and "Animal?" is strongly negatively correlated.]

(d) Values: This, is, a, Dog.

(e) Computing softmax attention, we see that the attention weights are concentrated on the inner product of "Animal" and "Animal?". Thus, the output is close to "Dog".

8. Continuing Transformer Architecture

(a) Once we have computed the weighted sum of values, we use these to construct the final representation. Apply non-linearity:

$$e'_i = \phi(W_o \hat{V}_i),$$

where W_o maps the values back into embedding space.

(b) Computational complexity is on the order of $O(d^2)$, where d is the dimensionality of the embeddings.

(c) Use multiple heads, i.e., repeat the above operations multiple times to obtain $\hat{V}_i^{(j)}$, for several js. These are thought to compute multiple plausible solutions in parallel (Xiong et al., 2024).

Then concatenate them to obtain $\hat{V}_i$

(d) Use this operation in a residual layer (He et al., 2016), i.e.,

$$e_i \leftarrow e_i + e'_i,$$

i. Why are residual layers beneficial? Intuitively, the residual operation can be simpler. Consider for example a semantic segmentation task. Then, for a cartoon image, the form of the segmentation map is very close to identity.
Residual layers can also be viewed as an instance of the general principle of making small changes at a time.

ii. As another perspective, consider the gradient descent operation $x' \leftarrow x - \gamma \nabla f(x)$. Clearly, this can be viewed as a residual operation.

9. Further process the embeddings in parallel via a multi-layer perceptron (MLP)/feedforward network. This is a small neural net applied to each embedding, e.g.,

$e'_i = W_{\text{proj}}\sigma(W_1 e_i)$, $e_i \leftarrow e_i + e'_i$,

where the dimension of the inner layer $W_1 e_i$ is larger by a factor (say four) than the dimension of the outer layer.

These layers can be viewed as acting as a form of un-normalized key-value memory (Geva et al., 2020).

- Input: $e \in \mathbb{R}^d$
- $\tilde{K}, \tilde{V} \in \mathbb{R}^{d \times d_m}$, parameter matrices
- $\text{FF}(x) = \sigma\big(e^\top \tilde{K}^\top\big)\tilde{V}$

Neural memory (Sukhbaatar et al., 2015): d_m key-value pairs (memories). Keys $\tilde{k}_i \in \mathbb{R}^d$, Values $\tilde{v}_i \in \mathbb{R}^d$, $i = 1, \ldots, d_m$.

Probability $P(\tilde{k}_i \,|\, e) \propto \exp(e^\top \tilde{k}_i)$. Expected value:

$$\text{MN}(e) = \sum_i P(\tilde{k}_i \,|\, e)\tilde{v}_i = \text{softmax}(e^\top \tilde{K}^\top)\tilde{V}$$

Differences from standard attention mechanism:

- Attention uses probability distributions; FF layer "distributions" are un-normalized.

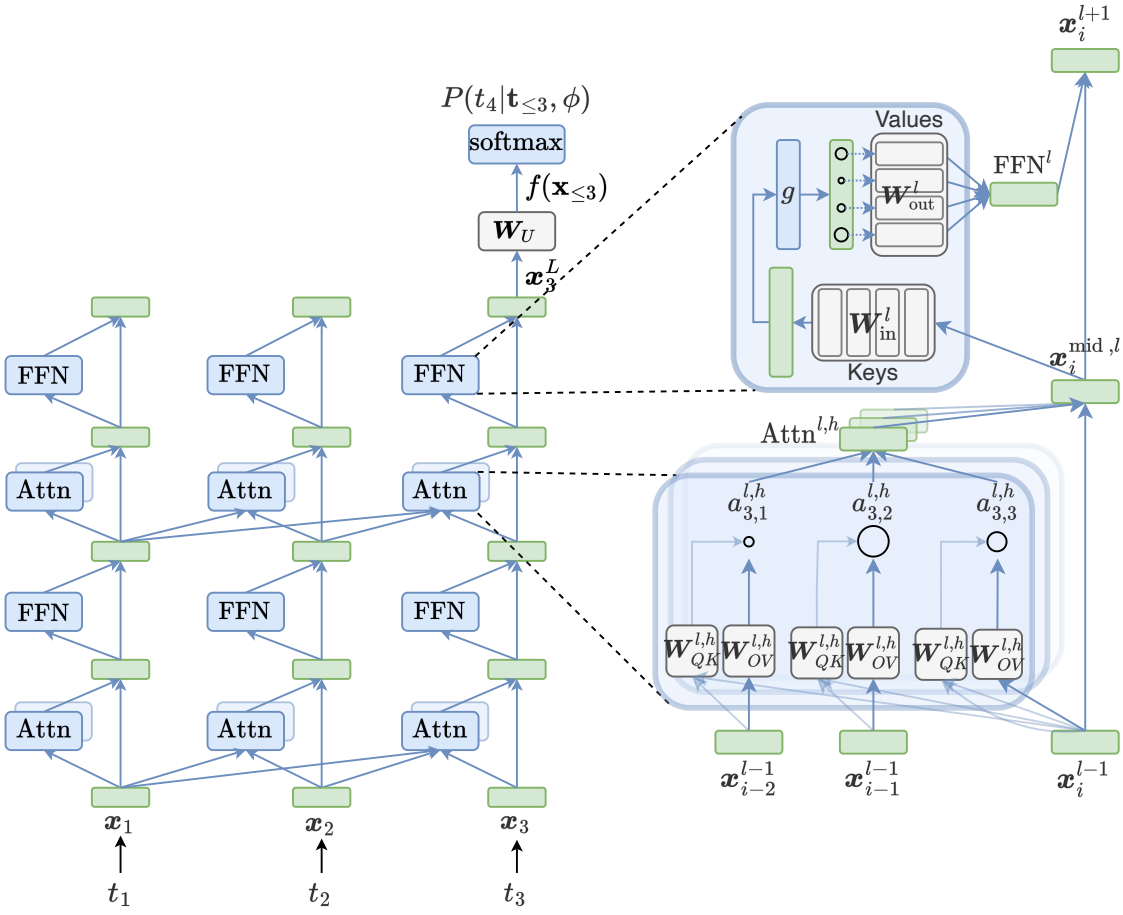


Figure 1: Transformer LM (Ferrando et al., 2024)

- Non-linearity σ may be different from exponential.
- Keys and values cannot depend on the input e.

10. Figure to illustrate transformer LM: Figure 1.

11. Use a deep network, by repeating the process across multiple blocks. This way, we can extract hierarchical concepts from the data.

12. The above is called "decoder"; the original transformer also had an "encoder" (Vaswani et al., 2017), to encode a special input/prompt for special tasks such as translation. Many modern LLMs are decoder-only (Liu et al., 2018)

    Encoder-only architectures: BERT (Devlin et al., 2019). For pre-training: predict masked tokens, next sequences

    Encoder-decoder architectures: T5 (Raffel et al., 2020)

13. Also: cross-attention

2.3.2 Readout: Next Token Prediction

1. The goal is to determine how to make the next token prediction given the embeddings $e_1, e_2, \ldots, e_j$ (at the penultimate layer).

2. The pre-activations for the next token are calculated using a weighted sum of the embeddings:

$$S = W_{\text{out}}[e_1, e_2, \ldots, e_j]$$

   where W_{out} is the $|V| \times d$ learned weight matrix, and the operation produces a vector of dimension $|V| \times j$.

3. Let P_j represent the predicted probability distribution over the possible tokens at position j.

4. After calculating the pre-activations s_j, apply a softmax function to convert them into probabilities. For all $k \leq |V|$

$$P_{j,k} = \frac{\exp(s_{k,j})}{\sum_{k' \leq |V|} \exp(s_{k',j})}$$

5. Equivalently, the probability of outputting token k at the $j+1$-st location is proportional to $\exp(W_{\text{out},k}^\top e_j)$

   In fact, this also gives the probabilities of token k at the $l+1$-st location, for all $l \leq j$, is proportional to $\exp(W_{\text{out},k}^\top e_l)$. Note that this not pose a contradiction (i.e., these are self-consistent for varying j)

   [leads to KV caching: store previous key, value pairs instead of recomputing them for speedup]

6. Simple way to think about it: Given an input x, the probabilities over next tokens y are proportional to $p(y|x) \propto \exp(\phi(x)^\top \lambda(y))$, where $\phi(x)$ is the last-layer embedding of x, and $\lambda(y)$ is the readout representation of y

2.3.3 Matrix form

1. Let $e_1, \ldots, e_T$ be embeddings of T tokens.

   Arrange them into the $T \times d$ matrix E.

2. Query, Key, and Value matrices: $Q = EW_q, K = EW_k, V = EW_v$ where each matrix is $T \times d$.

3. Pre-attention matrix: $Z = QK^\top/\sqrt{d}$, $T \times T$

4. Attention matrix: For causal attention, $A = \text{row-softmax}(Z \odot M)$ is a $T \times T$ matrix, where M is a lower-triangular $T \times T$ mask matrix with $M_{ij} = 1$ if $i \geq j$ and $-\infty$ otherwise. For full attention, $A = \text{row-softmax}(Z)$

5. Output embeddings: $\hat{E} = AV$ is $T \times d$ (omits multiple heads, residual layers, normalization, FFN).

6. Readout: at the last layer, calculate next-token probabilities $\hat{p}_j = \text{softmax}(W_{\text{out}} e_j)$, for all $j \in [T]$, where W_{out} is $n_{\text{tokens}} \times d$.

2.3.4 Insight into attention and transformers

1. To gain insights into the computational abilities of attention, we can consider some abstractions.

   The RASP language (Weiss et al., 2021) is a programming language that describes operations similar to the ones performed by transformers. Operates directly on the strings, with associated indices. Some of the key operations allowed include:

   (a) Individual operations on string indices. E.g., add one, etc.

   (b) Selection: Given two strings q, k of the same length, and a rule comparing any of their components, select all pairs of indices that satisfy the rule.
   e.g., select(indices, indices + 1) allows us to look at the index immediately following us.
   E.g., q ="abc", k ="ade", and relation is "=". Then, we select the pair of indices $(0, 0)$.
   [In RASP, start indices from zero.]

   (c) Given a selection mask (the result of the previous step), and a fixed vector of the same length as the strings (say, numeric), we take the average of the entries in the vector, restricted to the selected sub-vector for each given index of the query.

   With this, show that one implement a variety of algorithms: checking if there is a specific letter in the string, sorting, histogram, etc.

2. How do LLMs encode information?

   (a) In neural nets, there are often specialized neurons representing certain concepts (e.g., sentiment neurons (Radford et al., 2017)), or linear combinations of neurons for the same

   (b) A general idea is to find neurons or linear combinations associated with concepts (e.g., via probing or causal interventions)

   (c) In LLMs, subject tokens act as keys for retrieving facts: after an input text mentions a subject, LMs construct enriched representations of subjects that encode information about those subjects (Li et al., 2021); [they do this for simple generated text encoding a specific scenario]
   FFNs can also remove information (Voita et al., 2024).

(d) One can trace the effect of some neuron activations on knowledge representation. Meng et al. (2022)[1] have the following method. The consider the computational graph as a causal graph, taking the perspective of causal mediation analysis, popularized in NLP by (Vig et al., 2020):

i. Run LM once, e.g., on "The Space Needle is in"

ii. Corrupt subject "**[i.i.d. Gaussian activations]** is in" , run LM again

iii. For each neuron group (e.g. MLP or attention layer activations of a token at a given layer), restore activations in the corrupted run to those from the clean run (patch), and run LM a third time

iv. Check which neurons restore original correct logit

Find: mid-layer MLP activations of last tokens of subject ("le") play a large impact. Late-layer attention activations of last token also important.

"Based on causal traces, we posit a specific mechanism for storage of factual associations: each midlayer MLP module accepts inputs that encode a subject, then produces outputs that recall memorized properties about that subject. Middle layer MLP outputs accumulate information, then the summed information is copied to the last token by attention at high layers."

"It is consistent with the Geva et al. (2020) view that MLP layers store knowledge, and the Elhage et al. (2021) study showing an information-copying role for self-attention."

Meng et al. (2022) also introduce a method to edit MLP layers to change facts.

i. Claim: The two-layer feedforward neural network (FF NN) acts as a linear associative memory (Kohonen, 1972; Anderson, 1972). [different from Geva et al. (2020), who view individual neurons as memories]

ii. Consider representations/activations e at a specific layer for a given token. These are passed through a neural network: $e' = W_{\text{proj}}\sigma(W_1 e)$. Here, we view $k = \sigma(W_1 e)$ as a key and $e' = W_{\text{proj}}k$ as a value (no relation to keys and values in attention—this is an unfortunate terminology clash).
Given the key k, this NN retrieves the value $v = e'$.

iii. More generally, for a set of vector keys $K = \sigma(W_e E) = [k_1, k_2, \dots]$, we retrieve vector values $V = W_{\text{proj}}K$.

iv. Meng et al. (2022) observe that for any given $(\tilde{V}, \tilde{K})$ pair, we can always approximate W to operate as a key-value store, solving:

$$\min_W \|W\tilde{K} - \tilde{V}\|_F,$$

obtaining $\tilde{W} = \tilde{V}\tilde{K}^+$.

v. Now the goal is to insert any key-value pair (k^*, v^*) into the memory. For example:

$$k_1 = \text{space needle}, \; k_2 = \text{Eiffel tower}$$

$$v_1 = \text{Toronto}, \; v_2 = \text{Paris}.$$

We may aim to insert a specific memory, e.g., $k^* = \text{space needle}$, $v^* = \text{Paris}$.

[1] An amazing paper!

vi. With this, Meng et al. (2022) observe that we can insert (k^*, v^*) into the memory with a minimal perturbation of the weights by solving $\min_W \|WK - \tilde{V}\|_F$, subject to $Wk^* = v^*$.
The solution is

$$W^* = \tilde{W} + \Lambda \left(C^{-1}k^*\right)^\top,$$

where

$$C = KK^\top \quad \text{and} \quad \Lambda = (v^* - \tilde{W}k^*)/k^{*,\top}C^{-1}k^*.$$

Here, C can be estimated based on a data sample from the distribution that we expect k^* to follow.

vii. To insert a specific memory, use k^* of original text (with some random contexts). For v^*, solve the optimization problem to maximize:

$$\max_v P(o^* \mid \text{prefix} + x, v(x) \mapsto v),$$

where x are the tokens corresponding to k^*.

(e) Consider relations, e.g., (Space Needle, is in, Toronto), (Eiffel Tower, is in, Paris). Some of them are represented approximately as affine maps $s \to o$, where s is a sufficiently contextualized representation of the input, and o are the output logits (Hernandez et al., 2024). The affine map can be approximated by $\partial o/\partial s$. Only holds for some relations.

(f) Work from (Nanda et al., 2023) takes a mechanistic interpretability viewpoint, arguing that the recall of factual knowledge can be viewed as a circuit.

Study prompts of the form "The athlete Michael Jordan plays the sport of" for Pythia 2.8B. Goal is to understand how the LLM recalls the sport played by an athlete. Cca 1500 prompts.

Recall is a circuit with the following components (largely agreeing with the conclusions from Meng et al. (2022); Hernandez et al. (2024):

i. Multi-token embeddings: Need to recognize that the question refers to all the tokens in "athlete Michael Jordan", and then construct an embedding that captures all of them. [use several attention layers to pool this info]

ii. Feedforward processing of the representation [make info amenable to extraction]

iii. Lookup: Linear transform of the rep, to extract sport. [attend from pre-output token to last name token; attention is pattern is fixed, so the information must be represented linearly in the last tokens; use OV circuit?]
Experiments:

A. for each prompt and activation in the model, evaluate the effect of grafting an activation from a random prompt instead of the activation from the current prompt on the correct logit [causal intervention]

B. prune model to only use the hypothesized circuit. Check accuracy.

3. Attention as a Context-Dependent Markov Chain (Ildiz et al., 2024)

(a) L tokens, K vocab size.

(b) Prompt: $z = (z_1, z_2, \ldots, z_L), \quad z_i \in [k]$

(c) Matrix encoding: $X = (e_{z_1}, e_{z_2}, \ldots, e_{z_L})^\top, \quad X \in \mathbb{R}^{L \times k}$, where e_i is the i-th unit vector in $\mathbb{R}^K$

(d) Next token probability:

$$f_W(X) = X^\top S\big(XWe_{z_L}\big) = X^\top S(XW_{z_L}) = X^\top S(Xv) = X^\top \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{z_L} \end{bmatrix} = X^\top \begin{bmatrix} \mathrm{Num}(1)\cdot s_1 \\ \mathrm{Num}(2)\cdot s_2 \\ \vdots \\ \mathrm{Num}(k)\cdot s_k \end{bmatrix}$$

where $\mathrm{Num}(k)$ is the number of times k occurs in z. Here

$$s_i = \frac{\exp(v_i)}{\sum_{j\in z}\exp(v_j)} = \frac{\exp(v_i)}{\sum_{l\in[K]}\mathrm{Num}(l)\cdot\exp(v_l)}.$$

(e) W (i.e., v and s) correspond to the architecture

(f) If they are fixed, then f_V's predictions can be viewed as s weighted by the frequencies of each token in the context.

(g) If we think about an autoregressive generation, it can be viewed as a context-dependent Markov Chain.

(h) One implication is that frequent tokens that occur due to chance can completely overwhelm and start to dominate the distribution. (This can also sometimes be seen empirically.)

4. Representational and computational abilities of transformers:

   (Sanford et al., 2024b)

   (Sanford et al., 2024a)

5. Review: Ferrando et al. (2024)

2.3.5 Positional encoding

1. A key consideration in language models is the extent to which they take into account position information about the input tokens.

2. For instance, "the" is often followed by a noun, and if we saw "the cat" previously in the context, then the chances of predicting "cat" should increase after seeing "the".

3. How to do this?

4. First, consider attention. Observation: Attention is permutation equivariant.

   Suppose we permute the input embeddings $e_1, \ldots, e_T$ by a permutation π to $e_{\pi^{-1}(1)}, \ldots, e_{\pi^{-1}(T)}$. In matrix form: $E \to \Pi E$. Here, Π is a matrix with entries $\Pi_{ij} = \delta_{i,\pi(j)}$.

   Then:

$$Q \to \Pi Q, \quad K \to \Pi K, \quad V \to \Pi V, \quad Z \to \Pi Z \Pi^\top$$

   To understand row-softmax, write it as a composition of:

   (a) Elementwise exponentiation, which is permutation equivariant: $f(\Pi Z \Pi^\top) = \Pi f(Z) \Pi^\top$

(b) Division by row-sums: Let $g(Z) = \mathrm{diag}(Z\mathbf{1})Z$. Now:

$$\mathrm{diag}(\Pi Z \Pi^\top \mathbf{1}) = \mathrm{diag}(\Pi Z \mathbf{1}) = \Pi\, \mathrm{diag}(Z\mathbf{1})\, \Pi^\top$$

Thus: $g(\Pi Z \Pi^\top) = \Pi\, g(Z)\, \Pi^\top$.

Therefore $\hat{E} = AV \mapsto \Pi A \Pi^\top \Pi V = \Pi \hat{E}$. This shows attention is permutation equivariant.

This is not true for attention with causal masking.

5. However, it suggests that the transformer does (or may) not adequately take location information into account.

   simple example: consider embedding vectors of the text "cat with hat". We want the meaning of the text to be included in the last coordinate

6. To address this, standard practice is to simply add a matrix Γ encoding location information to the embedding matrix E at the first layer. This matrix Γ can either be fixed (as in Vaswani et al. (2017)) or learned (as in the GPT series (Radford et al., 2018, 2019; Brown et al., 2020)).

7. The $T \times d$ matrix Γ has rows $\Gamma_1, \ldots, \Gamma_T$ that represent the effect of token positions $1, 2, \ldots, T$ respectively. These effects are propagated through the layers.

8. However, it is not exactly clear what kind of location information they capture and how. See e.g., Wang and Chen (2020)

9. One particular well-recognized issue is that relative positions are often more significant than absolute ones.

10. To address this, we may want to design a position embedding that only depends on relative position. Will follow Rotary Position Embeddings (RoPE) below (Su et al., 2024)

11. For a given context length T, we want to design key and query functions

$$k, q : \mathbb{R}^d \times [T] \to \mathbb{C}^{d'}$$

such that for all embeddings $e_1, \ldots, e_T \in \mathbb{R}^d$, it holds that

$$\langle q(e_a, a), k(e_b, b) \rangle = g(e_a, e_b, a - b)$$

holds for a fixed function g. Note that complex-valued keys and queries simplify the analysis.

12. Start with a simpler problem, where e_i, e_j are fixed and $k = q$.

13. Then, we need to solve the following problem.

    (a) Find all nonzero complex-valued sequences $(a_n)_{n \geq 1}$ such that there is a sequence complex-valued sequence $(b_n)_{n \in \mathbb{Z}}$ for which we have

$$a_m \overline{a_n} = b_{m-n}, \quad \text{for all } m, n \geq 1.$$

(b) Argument: first, we write $a_m = r_m e^{i\theta_m}$ and

$$b_n = s_n e^{i\eta_n}, \quad \text{for all } m, n,$$

in the polar form, where $r_m, s_n > 0$, are uniquely determined.

Then we can deduce that

$$\begin{cases} r_m r_n = s_{m-n} \\ \theta_m - \theta_n - \eta_{m-n} \in \mathbb{Z} \end{cases}$$

for all m, n.

From the first equation, taking $m = n$, we deduce $r_m^2 = s_0$, for all m, hence $r_m = s_0^{1/2}$, which is a constant.

From the second condition, taking $m = n + 1$, we have that

$$\theta_{n+1} - \theta_n - \eta_1 \in \mathbb{Z}, \quad \text{hence } \theta_{n+1} - \theta_1 - n\eta_1 \in \mathbb{Z}.$$

For all n, now observe that the fractional part of θ_n fully determines the solution. Hence, without loss of generality, we can take $\theta_{n+1} = \theta_1 + n\eta_1$.

The final solution is $a_m = s_0^{1/2} e^{2\pi \mathrm{i}(\theta_1 + m\eta_1)}$ or $a_m = Ce^{2\pi \mathrm{i}\, m\theta}$, where we introduced simplifying notation, $C \in \mathbb{C}$, $\theta \in \mathbb{R}$.

14. Returning now to the original problem, we see that any function of the form

$$q(e, j) = q(e) e^{2\pi \mathrm{i}\, \theta \cdot j}$$

offers a solution, with $k = g$.

15. Llama 3 uses $\theta = 500,000$ (Dubey et al., 2024).

2.3.6 Additional steps required

1. To make transformers work, several additional algorithmic steps are required in practice.

   It is hard to clearly derive/explain them from first principles

2. ML-specific optimization algorithms, e.g., Adam. Aims to (1) speed up optimization via momentum; (2) stabilize it by normalization (LayerNorm or RMSNorm)

3. Special learning rate schedules: e.g., linear warmup and cosine decay. Used for stability and efficiency.

4. Tokenization, from (Brown et al., 2020):

   Use Byte Pair Encoding (BPE) (Sennrich et al., 2015) to keep vocab size small and can assign a probability to any Unicode string

   "Byte Pair Encoding (BPE) (Gage, 1994) is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. We adapt this algorithm for word segmentation. Instead of merging frequent pairs of bytes, we merge characters or character sequences."

   "Firstly, we initialize the symbol vocabulary with the character vocabulary, and represent each word as a sequence of characters, plus a special end-of-word symbol '·', which allows us to

restore the original tokenization after translation. We iteratively count all symbol pairs and replace each occurrence of the most frequent pair ('A', 'B') with a new symbol 'AB'. Each merge operation produces a new symbol which represents a character n-gram."

"our symbol sequences are still interpretable as subword units"

2.3.7 Architectural Choices in Specific LLMs

1. GPT series (Radford et al., 2018, 2019)

   (a) modified initialization: "A modified initialization is used, which accounts for the accumulation on the residual path with model depth. We scale the weights of residual layers at initialization by a factor of $1/N^{1/2}$ where N is the number of residual layers."

   (b) pre-normalization: "Layer normalization (Ba et al., 2016) was moved to the input of each sub-block, similar to a pre-activation residual network (He et al., 2016) and an additional layer normalization was added after the final self-attention block."

   (c) GPT-3: "alternating dense and locally banded sparse attention patterns (similar to the Sparse Transformer (Child et al., 2019))

   (d) Gaussian Error Linear Unit activation $X \cdot \text{sigmoid}(1.702 \cdot X)$

   (e) All models use weight decay of $\lambda = 0.1$

   clip the global norm of the gradient at $M = 1$

   Linear LR warmup over the first 375 million tokens. Then use cosine decay for learning rate down to 10% of its value, over 260 billion tokens.

   increase the batch size linearly to its max

   (f) GPT-3 175B or "GPT-3": nparams = 175.0B, nlayers=96, the number of units in each bottleneck layer: dmodel= 12288 (we always have the feedforward layer four times the size of the bottleneck layer, dff = 4 dmodel), num heads = 96, dimension of each attention head: dhead=128; All models use a context window of nctx = 2048 tokens.

   batch size = 3.2M. learning rate 0.6x10e-4.

2. Llama series

   (a) LLama

   Pre-normalization [GPT3]. To improve the training stability, we normalize the input of each transformer sub-layer, instead of normalizing the output [as in GPT]. We use the RMSNorm normalizing function, introduced by Zhang and Sennrich (2019) [new to Llama].

   SwiGLU activation function [PaLM]. We replace the ReLU non-linearity by the SwiGLU activation function, introduced by Shazeer (2020) to improve the performance.

   Rotary Embeddings [GPTNeo] (Su et al., 2024)

   2.3 Optimizer

   Our models are trained using the AdamW optimizer (Loshchilov and Hutter, 2017), with the following hyper-parameters: b1 = 0.9, b2 = 0.95. We use a cosine learning rate schedule, such that the final learning rate is equal to 10% of the maximal learning rate. We use a weight decay of 0.1 and gradient clipping of 1.0. We use 2, 000 warmup steps, and vary the learning rate and batch size with the size of the model

When training a 65B-parameter model, our code processes around 380 tokens/sec/GPU on 2048 A100 GPU with 80GB of RAM. This means that training over our dataset containing 1.4T tokens takes approximately 21 days.

This means that developing these models lead to a total emission of 1,015 tCO2eq [1000 tons of carbon emissions].

(b) LLama 2

The primary architectural differences from Llama 1 include increased context length and grouped-query attention (GQA, key and value projections can be shared across multiple heads, (Ainslie et al., 2023)).

In a dialogue setup, some instructions should apply for all the conversation turns, e.g., to respond succinctly, or to "act as" some public figure. When we provided such instructions to Llama 2-Chat, the subsequent response should always respect the constraint. However, our initial RLHF models tended to forget the initial instruction after a few turns of dialogue

To address these limitations, we propose Ghost Attention (GAtt), inspired by Context Distillation (Bai et al., 2022b) - We synthetically concatenate the instruction that should be respected throughout the dialogue to all the user messages of the conversation.

(c) Mistral - Sliding Window Attention, Rolling Buffer Cache

(d) MixTral - MoE (Jiang et al., 2024a)

(e) Gemini: speculations; HyperAttention? (Han et al., 2023); Ring Attention? (Liu et al., 2024a)

2.3.8 Speeding up attention

1. Linear attention (Katharopoulos et al., 2020)

   (a) In classical attention, the softmax is used to non-linearly select which query-key pairs to attend to.

   (b) It is natural to ask whether or not a non-linearity is needed.

   (c) One can consider linear attention, which in the non-causal setting has the form:

   $$\hat{E} = (QK^\top)V,$$

   (d) In the causal setting, it is:

   $$\hat{E} = (M \odot QK^\top)V.$$

   (e) Note that the matrix dimensions of Q, K, V are all $T \times d$. Thus, the matrix product $(QK^\top)V$ can be computed in $\mathcal{O}(Td^2)$, which is only linear in the context length T, in contrast to $\mathcal{O}(T^2d)$ for usual attention.

   (f) For causal attention, one can also perform a linear-time computation of $(M \odot QK^\top)V$. One can see this by expressing it in a recurrent form. Let $\{S_t\}_{t=0}^T$ be a sequence of $d \times d$ matrix-valued states, starting with $S_0 = 0$, and for $t = 1, \ldots, T$, updating:

   $$S_t = S_{t-1} + v_t k_t^\top, \quad \hat{e}_t = S_t q_t.$$

   This can also be viewed as a type of linear recurrent neural network.

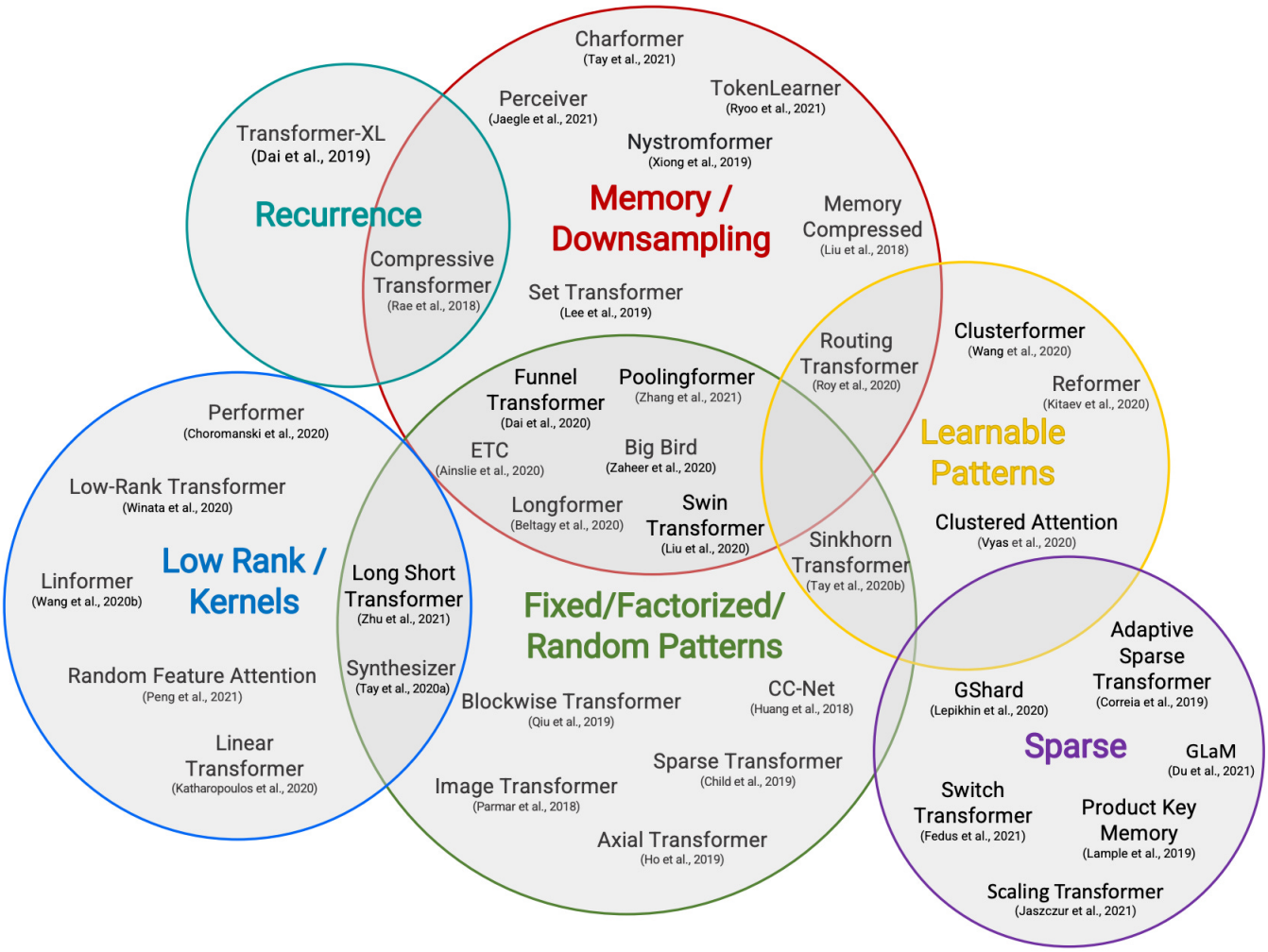


Figure 2: https://newsletter.ruder.io/p/flashier-attention-gzip-classifiers: efficient attention architectures up to 2022.

(g) One can check that this is equivalent. Indeed:

$$\hat{e}_t = \left(\sum_{j \le t} v_j k_j^\top \right) q_t,$$

whereas the t-th row of $(M \odot QK^\top)V$ is:

$$\left[q_t^\top k_1, \quad q_t^\top k_2, \quad \ldots, \quad q_t^\top k_t, \quad 0, \quad \ldots, \quad 0 \right] V = \sum_{j \le t} q_t^\top k_j v_j^\top = \hat{e}_t,$$

showing that the two expressions are equal.

(h) Empirically, the performance of LLMs based on linear attention is typically lower than when using nonlinear attention. Intuitively, nonlinear attentions allows us flexibility to forget the past, and more selectively focus on specific parts of the context. This component is missing in linear attention.

However, the insights one gains from linear attention are still crucial, and one can leverage these further in designing and analyzing other, nonlinear, architectures, such as state space models.

(i) One additional motivation: Expand the exponential inner product used in classical attention into an infinite-dimensional feature space, to obtain an infinite-dimensional linear attention.

Observation: $\exp(x^\top y) = \mathbb{E}_{\omega \sim \mathcal{N}(0,I)} \xi(x,\omega)^\top \xi(y,\omega)$, where $\xi(x,\omega) = \exp(\|x\|^2/2 + \mathrm{i}\omega^\top x)$.

Approximate by random sampling; obtain random feature attention.

2. Linformer (Wang et al., 2020)

3. Grouped Query Attention (Ainslie et al., 2023), used in Llama 3 (Dubey et al., 2024)

4. Sparse Attention (Child et al., 2019): something similar is used in GPT-3. From Child et al. (2019): Empirically: "Layers are able to learn a variety of specialized sparse structures, which may explain their ability to adapt to different domains. "

"Factorized self-attention instead has p separate attention heads, where [for the i-th token/index] the mth head defines a subset of the indices $A(m)_i \subset \{j : j \subset i\}$, and lets $S_i = A(m)_i$. [Key and value outputs are subset to S_i]

"fixed attention pattern: $A(1)_i = \{j : \lfloor j/l \rfloor = \lfloor i/l \rfloor\}$ [can attend to a fixed block of size l]

and $A(2)_i = \{j : j \mod l = t; t+1; \ldots, l\}$, where t = l - c and c is a hyperparameter.

"Concretely, if the stride is 128 and c = 8, then all future positions greater than 128 can attend to positions 120-128, all positions greater than 256 can attend to 248-256, and so forth.

"We found choosing c = 8; 16; 32 for typical values of l = 128; 256 to perform well,

"use multi-head attention, where nh attention products are computed in parallel, then concatenated along the feature dimension: attention(X) = Wp(attend(X;A)(i)), i in [nh]

"Here, the A can be the separate attention patterns, the merged patterns, or interleaved as in Eq. 2. Also, the dimensions of the weight matrices inside the attend function are reduced by a factor of 1/nh, such that the number of parameters are invariant across values of nh.

5. Longformer (Beltagy et al., 2020): special kind of sparse attention pattern, sliding window, dilated, also a few global patterns allowed

   Attention sink (Xiao et al., 2024): sliding window, but always pay attention to first few tokens at the beginning of the query

6. ring attention (Liu et al., 2024a)

7. Reformers (Kitaev et al., 2020) Use locality-sensity hashing (LSH) to approximate attention

   "note that we are actually only interested in softmax($QK^\top$). Since softmax is dominated by the largest elements, for each query q_i we only need to focus on the keys in K that are closest to q_i."

   "The problem of finding nearest neighbors quickly in high-dimensional spaces can be solved by locality-sensitive hashing (LSH)."

   General sparse attention: for a set $\mathcal{P}_i$ that query i attends to, and $\widetilde{\mathcal{P}}_i$ that determines the batch, i.e., all calculations in $\widetilde{\mathcal{P}}_i$ are done simultaneously (such that $\mathcal{P}_i \subseteq \widetilde{\mathcal{P}}_i$),

   $$o_i = \sum_{j \in \widetilde{\mathcal{P}}_i} \exp\left(q_i \cdot k_j - m\left(j, \mathcal{P}_i\right) - z\left(i, \mathcal{P}_i\right)\right) v_j \quad \text{where } m\left(j, \mathcal{P}_i\right) = \begin{cases} \infty & \text{if } j \notin \mathcal{P}_i \\ 0 & \text{otherwise} \end{cases}$$

   And $z\left(i, \mathcal{P}_i\right)$ is the normalizing constant

   Standard attention: $\mathcal{P}_i = [i]$

   What are the criteria for $\widetilde{\mathcal{P}}_i$ to be good? First note that $\widetilde{\mathcal{P}}_i$ is shared for several i. Then, we wish the amount of computational resources required (memory, flops, etc) for each $\widetilde{\mathcal{P}}_i$ to be roughly the same. "In practice, the attention function on a set of queries is computed simultaneously, packed together into a matrix Q." So, we compute attention per batch

   LSH attention: change $\mathcal{P}_i$ by only allowing attention within a single hash bucket.

   $$\mathcal{P}_i = \{j : h\left(q_i\right) = h\left(k_j\right)\}$$

   The number of queries and the number of keys within a bucket may be unequal; it is possible for a bucket to contain many queries but no keys. To alleviate: first ensure that $h\left(k_j\right) = h\left(q_j\right)$ by setting $k_j = \frac{q_j}{\|q_j\|}$.

   Hash buckets in this formulation tend to be uneven in size, which makes it difficult to batch.

   We sort the queries by bucket number and, within each bucket, by sequence position; this defines a permutation where $i \mapsto s_i$ after sorting.

   We batch s.t. chunks of m consecutive queries (after sorting) attend to each other, and one chunk earlier:

   $$\widetilde{\mathcal{P}}_i = \left\{j : \left\lfloor \frac{s_i}{m} \right\rfloor - 1 \leq \left\lfloor \frac{s_j}{m} \right\rfloor \leq \left\lfloor \frac{s_i}{m} \right\rfloor\right\}$$

   In practice we set $m = \frac{2l}{n_{\text{buckets}}}$ (where l is the sequence length). The average bucket size is $\frac{l}{n_{\text{buckets}}}$, and we assume that the probability of a bucket growing to twice that size is sufficiently low.

   Def of LSH (Andoni et al, 2015): for a row vector x of dimension d, and a $d \times b/2$ random matrix R, define $h(x) = \text{argmax}([xR; -xR])$, where $[xR; -xR]$ concatenates the two vectors

Also use RevNets (reversible NNs): We apply the RevNet idea to the Transformer by combining the attention and feed-forward layers inside the revnet block. In the notation above, F becomes an attention layer while G becomes the feed-forward layer. Note that Layer Normalization (Ba et al., 2016) is moved inside the residual blocks.

$$Y_1 = X_1 + \text{Attention}\left(X_2\right) \quad Y_2 = X_2 + \text{FeedForward}\left(Y_1\right)$$

The reversible Transformer does not need to store activations in each layer and so gets rid of the n_l term. In Section 5 we show that it performs the same as the normal Transformer when using the same number of parameters; we achieve this by having both x_1 and x_2 have size d_{model} .

8. Performers (Choromanski et al., 2020):

FAVOR+ works for attention blocks using matrices $\mathbf{A} \in \mathbb{R}^{L \times L}$ of the form $\mathbf{A}(i,j) = \mathrm{K}\left(\mathbf{q}_i^\top, \mathbf{k}_j^\top\right)$, with $\mathbf{q}_i/\mathbf{k}_j$ standing for the i^{th} $/j^{\text{th}}$ query/key row-vector in $\mathbf{Q}/\mathbf{K}$ and kernel $\mathrm{K} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+$ defined for the (usually randomized) mapping: $\phi : \mathbb{R}^d \rightarrow \mathbb{R}_+^r$ (for some $r > 0$) as:

$$\mathrm{K}(\mathbf{x}, \mathbf{y}) = \mathbb{E}\left[\phi(\mathbf{x})^\top \phi(\mathbf{y})\right]$$

We call $\phi(\mathbf{u})$ a random feature map for $\mathbf{u} \in \mathbb{R}^d$. For $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{L \times r}$ with rows given as $\phi\left(\mathbf{q}_i^\top\right)^\top$ and $\phi\left(\mathbf{k}_i^\top\right)^\top$ respectively, Equation 3 leads directly to the efficient attention mechanism of the form:

$$\widehat{\text{Att}_{\leftrightarrow}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \widehat{\mathbf{D}}^{-1}\left(\mathbf{Q}'\left(\left(\mathbf{K}'\right)^\top \mathbf{V}\right)\right), \quad \widehat{\mathbf{D}} = \text{diag}\left(\mathbf{Q}'\left(\left(\mathbf{K}'\right)^\top \mathbf{1}_L\right)\right).$$

Here $\widehat{\text{Att}_{\leftrightarrow}}$ stands for the approximate attention and brackets indicate the order of computations. It is easy to see that such a mechanism is characterized by space complexity $O(Lr + Ld + rd)$ and time complexity $O(Lrd)$ as opposed to $O\left(L^2 + Ld\right)$ and $O\left(L^2 d\right)$ of the regular attention (see also Fig. 11).

2.3 HOW TO AND HOW NOT TO APPROXIMATE SOFTMAX-KERNELS FOR ATTENTION

It turns out that by taking ϕ of the following form for functions $f_1, \ldots, f_l : \mathbb{R} \rightarrow \mathbb{R}$, function $g : \mathbb{R}^d \rightarrow \mathbb{R}$ and deterministic vectors ω_i or $\omega_1, \ldots, \omega_m \overset{\text{iid}}{\sim} \mathcal{D}$ for some distribution $\mathcal{D} \in \mathcal{P}\left(\mathbb{R}^d\right)$:

$$\phi(\mathbf{x}) = \frac{h(\mathbf{x})}{\sqrt{m}}\left(f_1\left(\omega_1^\top \mathbf{x}\right), \ldots, f_1\left(\omega_m^\top \mathbf{x}\right), \ldots, f_l\left(\omega_1^\top \mathbf{x}\right), \ldots, f_l\left(\omega_m^\top \mathbf{x}\right)\right)$$

we can model most kernels used in practice. Furthermore, in most cases $\mathcal{D}$ is isotropic (i.e. with pdf function constant on a sphere), usually Gaussian. For example, by taking $h(\mathbf{x}) = 1, l = 1$ and $\mathcal{D} = \mathcal{N}\left(0, \mathbf{I}_d\right)$ we obtain estimators of the so-called PNG-kernels (Choromanski et al.,2017) (e.g. $f_1 = \text{sgn}$ corresponds to the angular kernel). Configurations: $h(\mathbf{x}) = 1, l = 2, f_1 = \sin, f_2 = \cos$ correspond to shift-invariant kernels, in particular $\mathcal{D} = \mathcal{N}\left(0, \mathbf{I}_d\right)$ leads to the Gaussian kernel $\mathrm{K}_{\text{gauss}}$ (Rahimi & Recht, 2007).

The softmax-kernel which defines regular attention matrix A is given as:

$$\mathrm{SM}(\mathbf{x},\mathbf{y}) \stackrel{\mathrm{def}}{=} \exp\left(\mathbf{x}^\top \mathbf{y}\right)$$

In the above, without loss of generality, we omit $\sqrt{d}$-renormalization since we can equivalently renormalize input keys and queries. Since: $\mathrm{SM}(\mathbf{x},\mathbf{y}) = \exp\left(\frac{\|\mathbf{x}\|^2}{2}\right) \mathrm{K}_{\mathrm{gauss}}\left(\mathbf{x},\mathbf{y}\right) \exp\left(\frac{\|\mathbf{y}\|^2}{2}\right)$, based on what we have said, we obtain random feature map unbiased approximation of $\mathrm{SM}(\mathbf{x},\mathbf{y})$ using trigonometric functions with: $h(\mathbf{x}) = \exp\left(\frac{\|\mathbf{x}\|^2}{2}\right), l = 2, f_1 = \sin, f_2 = \cos$. We call it $\widehat{\mathrm{SM}}^{\mathrm{trig}}_m(\mathbf{x},\mathbf{y})$.

There is however a caveat there. The attention module from (1) constructs for each token, a convex combination of value-vectors with coefficients given as corresponding renormalized kernel scores. That is why kernels producing non-negative scores are used. Applying random feature maps with potentially negative dimension-values ($\sin$ / $\cos$) leads to unstable behaviours, especially when kernel scores close to 0 (which is the case for many entries of $\mathbf{A}$ corresponding to low relevance tokens) are approximated by estimators with large variance in such regions. This results in abnormal behaviours, e.g. negative-diagonal-values renormalizers $\mathbf{D}^{-1}$, and consequently either completely prevents training or leads to sub-optimal models. We demonstrate empirically that this is what happens for $\widehat{\mathrm{SM}}^{\mathrm{trig}}_m$ and provide detailed theoretical explanations showing that the variance of $\widehat{\mathrm{SM}}^{\mathrm{trig}}_m$ is large as approximated values tend to 0 (see: Section 3).

We propose a robust mechanism in this paper.

Lemma 1 (Positive Random Features (PRFs) for Softmax). For $\mathbf{x},\mathbf{y} \in \mathbb{R}^d, \mathbf{z} = \mathbf{x}+\mathbf{y}$ we have:

$$\mathrm{SM}(\mathbf{x},\mathbf{y}) = \mathbb{E}_{\omega\sim\mathcal{N}(0,\mathbf{I}_d)}\left[\exp\left(\omega^\top\mathbf{x} - \frac{\|\mathbf{x}\|^2}{2}\right)\exp\left(\omega^\top\mathbf{y} - \frac{\|\mathbf{y}\|^2}{2}\right)\right] = \Lambda\mathbb{E}_{\omega\sim\mathcal{N}(0,\mathbf{I}_d)}\cosh\left(\omega^\top\mathbf{z}\right)$$

where $\Lambda = \exp\left(-\frac{\|\mathbf{x}\|^2+\|\mathbf{y}\|^2}{2}\right)$ and cosh is hyperbolic cosine. Consequently, softmax-kernel admits a positive random feature map unbiased approximation with $h(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}\|^2}{2}\right), l = 1, f_1 = \exp$ and $\mathcal{D} = \mathcal{N}\left(0,\mathbf{I}_d\right)$ or: $h(\mathbf{x}) = \frac{1}{\sqrt{2}}\exp\left(-\frac{\|\mathbf{x}\|^2}{2}\right), l = 2, f_1(u) = \exp(u), f_2(u) = \exp(-u)$ and the same $\mathcal{D}$ (the latter for further variance reduction). We call related estimators: $\widehat{\mathrm{SM}}^{+}_m$and $\widehat{\mathrm{SM}}^{\mathrm{hyp}+}_m$.

If we replace in (7) ω with $\sqrt{d}\frac{\omega}{\|\omega\|}$, we obtain the so-called regularized softmax-kernel SMREG which we can approximate in a similar manner, simply changing $\mathcal{D} = \mathcal{N}\left(0,\mathbf{I}_d\right)$ to $\mathcal{D} = \mathrm{Unif}\left(\sqrt{d}\mathcal{S}^{d-1}\right)$, a distribution corresponding to Haar measure on the sphere of radius $\sqrt{d}$ in $\mathbb{R}^d$, obtaining estimator $\widehat{\mathrm{SREG}}^{+}_m$.

2.4 ORthogONAl RANdom FeAtUREs (ORFs)

The above constitutes the R+ part of the FAVOR+ method. It remains to explain the O-part. To further reduce the variance of the estimator (so that we can use an even smaller number of random features r), we entangle different random samples $\omega_1,\ldots,\omega_m$ to be exactly orthogonal. This can be done while maintaining unbiasedness whenever isotropic distributions $\mathcal{D}$ are used (i.e. in particular in all kernels we considered so far) by the standard Gram-Schmidt

orthogonalization procedure (see (Choromanski et al., 2017) for details). ORFs is a well-known method, yet it turns out that it works particularly well with our introduced PRFs for softmax. This leads to the first theoretical results showing that ORFs can be applied to reduce the variance of softmax/Gaussian kernel estimators for any dimensionality d rather than just asymptotically for large enough d (as is the case for previous methods, see: next section) and leads to the first exponentially small bounds on large deviations probabilities that are strictly smaller than for non-orthogonal methods. Positivity of random features plays a key role in these bounds. The ORF mechanism requires $m \leq d$, but this will be the case in all our experiments.

3 THEORETICAL RESULTS

We present here the theory of positive orthogonal random features for softmax-kernel estimation. All these results can be applied also to the Gaussian kernel, since as explained in the previous section, one can be obtained from the other by renormalization (see: Section 2.3).

Lemma 2 (positive (hyperbolic) versus trigonometric random features). The following is true:

$$\begin{aligned}
\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{\mathrm{trig}}(\mathbf{x},\mathbf{y})\right) &= \frac{1}{2m}\exp\left(\|\mathbf{x}+\mathbf{y}\|^2\right)\mathrm{SM}^{-2}(\mathbf{x},\mathbf{y})\left(1-\exp\left(-\|\mathbf{x}-\mathbf{y}\|^2\right)\right)^2, \\
\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{+}(\mathbf{x},\mathbf{y})\right) &= \frac{1}{m}\exp\left(\|\mathbf{x}+\mathbf{y}\|^2\right)\mathrm{SM}^{2}(\mathbf{x},\mathbf{y})\left(1-\exp\left(-\|\mathbf{x}+\mathbf{y}\|^2\right)\right), \\
\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{\mathrm{hyp}\,+}(\mathbf{x},\mathbf{y})\right) &= \frac{1}{2}\left(1-\exp\left(-\|\mathbf{x}+\mathbf{y}\|^2\right)\right)\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{+}(\mathbf{x},\mathbf{y})\right),
\end{aligned}$$

for independent random samples ω_i, and where MSE stands for the mean squared error.

Thus, for $\mathrm{SM}(\mathbf{x},\mathbf{y}) \to 0$ we have: $\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{\mathrm{trig}}(\mathbf{x},\mathbf{y})\right) \to \infty$ and $\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{+}(\mathbf{x},\mathbf{y})\right) \to 0$. Furthermore, the hyperbolic estimator provides additional accuracy improvements that are strictly better than those from $\widehat{\mathrm{SM}}_{2m}^{+}(\mathbf{x},\mathbf{y})$ with twice as many random features. The next result shows that the regularized softmax-kernel is in practice an accurate proxy of the softmax-kernel in attention.

Our next result shows that orthogonality provably reduces mean squared error of the estimation with positive random features for any dimensionality $d > 0$ and we explicitly provide the gap.

Theorem 2. If $\widehat{\mathrm{SM}}_m^{\mathrm{ort}+}(\mathbf{x},\mathbf{y})$ stands for the modification of $\widehat{\mathrm{SM}}_m^{+}(\mathbf{x},\mathbf{y})$ with orthogonal random features (and thus for $m \leq d$), then the following holds for any $d > 0$:

$$\mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{\mathrm{ort}\,+}(\mathbf{x},\mathbf{y})\right) \leq \mathrm{MSE}\left(\widehat{\mathrm{SM}}_m^{+}(\mathbf{x},\mathbf{y})\right) - \frac{2(m-1)}{m(d+2)}\left(\mathrm{SM}(\mathbf{x},\mathbf{y}) - \exp\left(-\frac{\|\mathbf{x}\|^2+\|\mathbf{y}\|^2}{2}\right)\right)^2$$

Furthermore, completely analogous result holds for the regularized softmax-kernel SMREG.

Below we show that even for the $\mathrm{SM}^{\mathrm{trig}}$ mechanism with ORFs, it suffices to take $m = \Theta(d\log(d))$ random projections to accurately approximate the attention matrix (thus if not attention renormalization, PRFs would not be needed). In general, m depends on the dimensionality d of the embeddings, radius R of the ball where all queries/keys live and precision parameter ε (see: Appendix F.6 for additional discussion), but does not depend on input sequence length L.

Theorem 4 (uniform convergence for attention approximation). Assume that L_2-norms of queries/keys are upper-bounded by $R > 0$. Define $l = Rd^{-\frac{1}{4}}$ and take $h^* = \exp\left(\frac{l^2}{2}\right)$. Then

for any $\varepsilon > 0, \delta = \frac{\varepsilon}{(h^*)^2}$ and the number of random projections $m = \Theta\left(\frac{d}{\delta^2}\log\left(\frac{4d^{\frac{3}{4}}R}{\delta}\right)\right)$ the following holds for the attention approximation mechanism leveraging estimators $\widehat{\mathrm{SM}}^{\mathrm{trig}}$ with ORFs: $\|\widehat{\mathbf{A}} - \mathbf{A}\|_\infty \leq \varepsilon$ with any constant probability, where $\widehat{\mathbf{A}}$ approximates the attention matrix $\mathbf{A}$.

9. Hardware-awareness basics:

   (a) There are several levels in the memory hierarchy: fast, small GPU memory (SRAM and HBM) & slow, large system memory (CPU DRAM)

   (b) Want to minimize data loading/moving;

      i. tiling (load block by block)

      ii. design fused kernels (chain ops, compute them at one time as opposed to sequentially, e.g., normalization and non-linearity)

      iii. recomputation

   (c) Data types: Often need to use more accurate data types (BF16 or FP32) for training, but can switch to lower accuracy/quantized data types (e.g., FP8 or BF16) for inference

   This can allow models to be load into GPU memory

   Can mix precision, e.g., during training, FP32 for weights and BF16 for activations/gradients

   (d) CUDA:

   A GPU has many independent streaming multiprocessors (48 for A100). Each does work independently, and has several resources: cores (A100: 64 FP32, 32 FP64), shared memory (A100: 160 kB), cache (A100: 192 kB). Each can be loaded a number of "blocks" of compute which split up a grid of work; and the number of blocks is limited by resources. The blocks all have a fixed size, i.e., number of threads, and have an amount of shared memory. Each thread runs the same compute—SIMT: single instruction multiple threads (A100: max 2048 per SM), and has access to the same number of individual registers (A100: 65,536 registers). Guarantee: all threads in a block run in parallel; "quantum of parallelism". A block of threads is broken up into "warps" of 32 threads; they all load memory. Four warps are run in parallel.

   (e) Parallelism is required in training. Need to combine multiple forms of parallelism, e.g., data, pipeline (across layers), tensor (across weights of a given tensor), context; see e.g., Narayanan et al. (2021); Dubey et al. (2024)

10. .[Adaptive state size, or adaptive sparsity in pattern?]

2.3.9 Commentary

1. Commentary by Andrej Karpathy during a course lecture at Stanford:

   - The Transformer is a magnificent neural network architecture because it is a *general-purpose differentiable computer*. It is simultaneously:
     - expressive (in the forward pass)
     - optimizable (via backpropagation+gradient descent)
     - efficient (high parallelism compute graph)

- (1) because its message-passing-like architecture is general (i.e. completeness) and powerful (i.e. efficiency), able to cover many real-world algorithms and in a small number of compute steps; an empirical finding.
- (2) because of residual connections, layer normalizations, and softmax attention. Absence of any flat tails. Residual connections support a kind of ability to learn short algorithms (think low LOC) fast and first, then gradually extend them longer during training.
- (3) because the compute graph is shallow and wide, mapping significantly better to our high-parallelism compute architectures (think GPUs). An earlier attempt that understood the significance and optimized for this property was the Neural GPU paper (arxiv.org/abs/1511.08228)
- The second critical ingredient is that while a Transformer seems able to act as a general-purpose computer in principle, the training objective has to be hard enough to actually force the optimization to discover and converge onto it in the "weights space" of the network.
- Turns out language modeling (i.e. next word prediction; equivalent to compression) of internet text is this excellent objective - very simple to define and collect data for at scale. It forces the neural net to learn a lot about the world, "multi-tasking" across many domains.
- Tversky & Kahnemann's system 1 and system 2: LLMs are System 1. how to do System 2?

2. Commentary by Aidan Gomez during Stanford class lecture https://www.youtube.com/playlist?list=PLoROMvodv4rNiJRchCzutFw5ItR_Z27CM:

   "Attention is all you need" paper took three months during an internship. They were already working on related topics (some on generative models, some on tensor to tensor).

   Main innovations:

   - self-attention (there was already cross-attention; were using LSTM with attention): learning patterns within a sequence (e.g., adj to noun)

   - multi-headed att: allows to learn multiple patterns

   - "Pretend it's correct" decoding for training objective: Use $x_{1:t}, y_1, \ldots, y_{t-1}$ as inputs, instead of iteratively encoded-decoded answers

   Lots of small things were on by default b/c of Lukasz Kaiser's awareness of the field: BatchNorm, LR warmup; these were just emerging at the time. Spent lots of time doing ablation studies, they are really needed.

3. A counterpart to/distinct from classical algorithms. e.g., Struggle with multiplication.

   Even if they are only pattern matching, can still be useful, because have more memory than humans

2.4 Alternatives?

1. Do we really need the attention mechanism?

   Alternatives have been explored, mainly based on state space models, such as Mamba (Gu and Dao, 2023), and RNNs (Feng et al., 2024)

2. Neuro-symbolic systems

2.4.1 State space models

1. State-space models:
2. Empirical performance can match transformers up to 8B params (Waleffe et al., 2024)
3. Structured state-space sequence (S4) models
   (a) Given a sequence input $x = (x_1, \ldots, x_t)$ of scalars, we aim to construct states/features/embeddings e_t (of a potentially high dimension) with the update
   $$e_t = Ae_{t-1} + Bx_t,$$
   where A, B are a weight matrix and vector (to be learned)
   Start $e_0 = Bx_0$
   Output $y_t = Ce_t$.
   (b) Equivalent form in terms of convolution: $K = (CB, CAB, \ldots, CA^TB, \ldots)$
   $y = x * K$
   Given a batch of inputs, each with several channels, apply this to each channel of each input independently
   (c) One way to motivate this is as a discretization of a continuous-time state-space model (on which there is a lot of work & it can be crucial for setting hyper-parameters)
   (d) Direct computation: given batch of size B, D channels, length L, state dimension N
   compute all state variables for one channel, a specified time step and one datapoint: $O(N)$
   compute all state variables for a specified time step and one datapoint: $O(DN)$
   compute all state variables: $O(BLDN)$
   Switch between modes of computation
      i. Training: convolutional, parallelizable
      ii. Inference: autoregressive

   Allows the computation of y without explicitly computing the state e; assuming certain structure on A (e.g., diagonal)
   (e) Limited expressivity: linear time-invariant
4. A simplified introduction to Mamba (Gu and Dao, 2023)
   (a) Given a sequence input $x = (x_1, \ldots, x_t)$, we are interested in defining features/embeddings with the update
   $$e_t = A_te_{t-1} + B_tx_t,$$
   where $A_t = A_t(x_t)$, $B_t = A_t(x_t)$ depend on the tokens. We start with $e_0 = B_0x_0$
   Output $y_t = Ce_t$.

(b) To see that we can compute this efficiently, define states of the form

$$c = \begin{bmatrix} a & b \end{bmatrix},$$

where a is a square matrix and b is a conformable vector
Define the operation on states $c = [a, b]$, $c' = [a', b']$

$$c \circ c' = \begin{bmatrix} aa', & ab' + b \end{bmatrix}$$

View $c = [a, b]$ as the transform $x \mapsto c[x] = ax + b$. Then, $c \circ c'$ corresponds to the composition of transforms.

(c) In particular, define the states

$$c_t = \begin{bmatrix} a_t & b_t \end{bmatrix} = \begin{bmatrix} A_t & B_t x_t \end{bmatrix}$$

(d) Define the sequence $(d_t)_{t \geq 0}$ recursively via $d_0 = [A_0, e_0]$ and $d_t = d_{t-1} \circ c_t$ for $t \geq 0$.
Let $d_t = [f_t, g_t]$.
Claim: We have $g_t = e_t$ for all $t \geq 0$.

Proof. For $t = 0$, we have $g_0 = e_0$ by definition.
Now we proceed by induction. Observe that $d_t = d_{t-1} \circ c_t$, so that

$$[f_t, g_t] = [a_t, b_t] \circ [f_{t-1}, g_{t-1}] = [a_t f_{t-1}, a_t g_{t-1} + b_t]$$

If the claim holds for $t - 1$, then $g_{t-1} = e_{t-1}$. Hence,

$$g_t = a_t g_{t-1} + b_t = a_t e_{t-1} + b_t = A_t e_{t-1} + B_t x_t = e_t.$$

This finishes the proof by induction. □

Hence $d_t = (((c_0 \circ c_1) \ldots) \circ c_{t-1}) \circ c_t$

(e) A key property of the "$\circ$" operation is that it is associative.

(f) This means that we can compute d_t in any order of operations, i.e., $d_t = c_0 \circ c_1 \circ \ldots c_{t-1} \circ c_t$. Crucially, we can use a Parallel Scan algorithm (Blelloch, 1990): parallelize across the indices, i.e., compute $c_0 \circ c_1$, $c_2 \circ c_3$, ... first, then again merge pairs, etc. This can lead to significant speedups

(g) To design an architecture based on this layer: (1) apply SSM itself in an embedding space (of higher dimension); (2) add further non-linearity via Gated Linear Unit; combine these into a block and repeat

i. Recall Gated Linear Unit (GLU), see Figure 3: X is (n, d) input, where n is the batch size (number of samples) and d is the dimension of the input features.
The steps are as follows:
1. The input X is split into two parts along its feature dimension, $X = [X_1; X_2]$. This requires us to have an even number of dimensions $d = 2k$.
2. The split parts X_1 and X_2 are multiplied as $Z_1 = X_1 W_1 + b_1$, $Z_2 = X_2 W_2 + b_2$, where $W_1, W_2 \in \mathbb{R}^{(k,l)}$ are learnable weight matrices, and $b_1, b_2 \in \mathbb{R}^{(1,l)}$ are bias terms.
3. The gating part, Z_2, passes through a sigmoid function to create a gate for each feature: $G = \sigma(Z_2)$, where $G \in \mathbb{R}^{(n,l)}$.
4. Finally, we combine the linear part Z_1 and the gated part G using an element-wise product: $X' = Z_1 \odot G$, of dimension $n \times l$.
5. This is then projected down to the original dimension; and possibly used only in a residual layer.

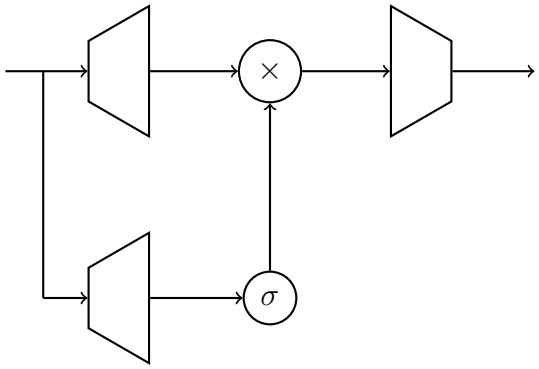


Figure 3: Gated Linear Unit (GLU).

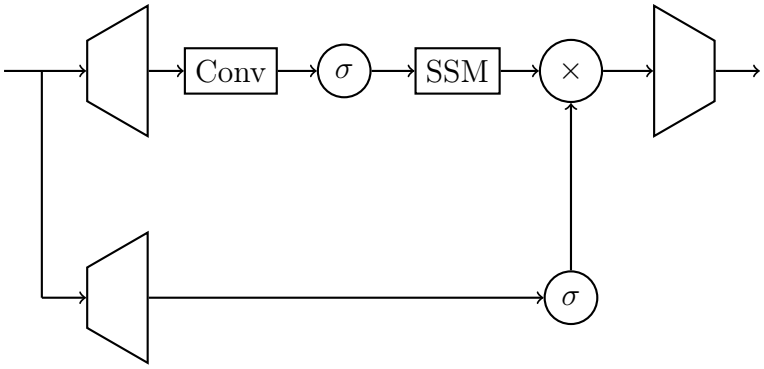


Figure 4: Mamba block/layer (Gu and Dao, 2023).

ii. For the Mamba block, do the same, except add a convolution -non-linearity - SSM sequence after the linear part. See Figure 4.

5. There are some close connections between state space models and transformers. Following Dao and Gu (2024), start with the SSM $e_t = A_t e_{t-1} + B_t x_t, \quad y_t = C_t^\top e_t$ with initial condition $e_0 = B_0 x_0$. Here x_t are scalars, and A_t are $N \times N$, where N is a state expansion factor.

   (a) By induction:

$$e_t = A_t \cdots A_1 B_0 x_0 + A_t \cdots A_2 B_1 x_1 + \cdots + B_t x_t, = \sum_{s=0}^{t} \left(\prod_{i=s+1}^{t} A_i \right) B_s x_s.$$

   Thus,

$$y_t = \sum_{s=0}^{t} C_t^\top \left(\prod_{i=s+1}^{t} A_i \right) B_s x_s.$$

   Define this as the matrix multiplication

$$y = \mathrm{SSM}(A, B, C)(x) = Px,$$

   where:

$$P = \begin{bmatrix} C_1^\top B_1 & 0 & 0 & \cdots & 0 \\ C_2^\top A_2 B_1 & C_2^\top B_2 & 0 & \cdots & 0 \\ C_3^\top A_3 A_2 B_1 & C_3^\top A_3 B_2 & C_3^\top B_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_t^\top A_t \cdots A_2 B_1 & C_t^\top A_t \cdots A_3 B_2 & \cdots & \cdots & C_t^\top B_t \end{bmatrix}.$$

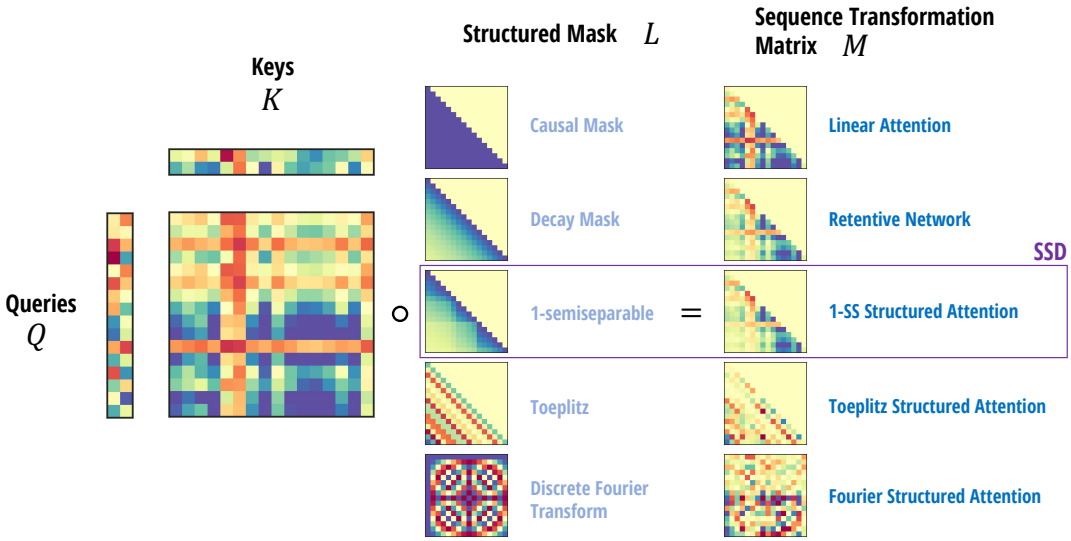


Figure 5: Dao and Gu (2024): Scalar structured SSMs are masked linear attention (with a semiseparable mask).

$$P_{ij} = C_i^\top A^\times_{i:(j+1)} B_j.$$

(b) Hybrids between transformers and state space models (Dong et al., 2024)

(c) Now consider the special case with $A_t = a_t I$. Then:

$$M = L \circ (CB^\top),$$

where:

$$L = 1\text{SS}(a) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_2 & 1 & 0 & \cdots & 0 \\ a_3 a_2 & a_3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_t a_{t-1} \cdots a_2 & a_t a_{t-1} \cdots a_3 & \cdots & a_t & 1 \end{bmatrix}.$$

Notice that matrix multiplication by L, i.e., $y = Lx$, can be computed recursively via

$$y_1 = x_1, \quad y_t = a_t y_{t-1} + x_t, t \geq 2.$$

Now consider masked/gated linear attention, $y = \left(M' \circ (QK^\top)\right) \cdot V$. Previously, we considered causal attention, when M' was a lower diagonal all-ones matrix. More generally, Dao and Gu (2024) call it structured masked attention.

More generally, by a tensor contraction derivation, we can see that we can compute it efficiently if M'-matrix multiplication is fast. In particular, this holds for lower-triangular matrices.

Take $C = Q$ and $B = K$. We can formulate the following observation: Scalar structured SSM is a generalized masked linear attention (with a specially structured mask).

(d) Diagonal SSM: A_t is diagonal. A similar approach works fo calculate P, for each diagonal element separately.

(e) Semiseparable matrix of order r: All submatrices below the lower diagonal have rank at most r.

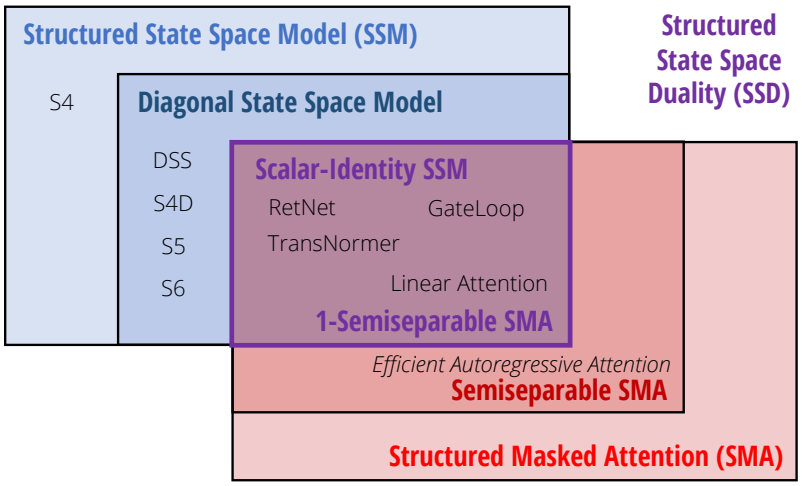


Figure 6: Dao and Gu (2024): **Structured State Space Duality**. State space duality describes the close relationship between state space models and masked attention. SSMs and SMA intersect at a large class of *state space dual models* (SSD) which capture many sequence models as special cases.

Theorem (Dao and Gu, 2024): A structured masked attention that is an autoregressive process with bounded order, the structured mask must be a semiseparable matrix.

6. Other notable ideas:

   Nearest neighbor language models extend a pre-trained neural language model (LM) by linearly interpolating it with a k-nearest neighbors (kNN) model (Khandelwal et al., 2020)

2.5 Empirical Behaviors

1. Capabilities

   (a) There is a lot of research of the type "LLMs can do addition, or can they?". We should not forget that we know how to solve a lot of problems w/o AI. Most likely (and in most cases) it will be more useful and more efficient if we integrate the existing solutions as opposed to re-inventing the wheel.

   (b) Achieving a superhuman level at a capability is, in itself, nothing special or new. There are already many examples (e.g., rocks are more durable than humans, birds can fly while humans cannot). Same is true for intellectual and artistic capabilities (e.g., a book or computer memory can store more information than a human; computers can draw much more perfect shapes than humans). The terminology superhuman appears heavily charged towards eliciting an emotional response; and thus to reduce clear thinking.

2. Emergence (Wei et al., 2022b)

   Emergence in general: quantitative change leads to qualitative change. e.g., uranium, DNA, water

   For ML: Small model cannot solve task, but large model can

   Grokking: similar meaning.

3. Memorization: LLMs can memorize text. This is sometimes desirable, e.g., want to memorize some facts (who was George Washington?). However, we do not want to memorize entire novels (all of Harry Potter), as that would be copyright infringement.

   A simple way to try to detect memorization in an LLM p is if the likelihood ratio $p(x)/p'(x)$ is large (Carlini et al., 2021)

   It is possible to extract training data scalably from certain LLMs (Nasr et al., 2023)

   [not surprising for language modelling/prediction task. more surprising you can circumvent RLHF. Ultimately don't think big issue for chatgpt (data is on the internet anyway); but could be if you add private data]

   Definition 1 (Extractable Memorization). Given a model with a generation routine Gen, an example x from the training set X is extractably memorized if an adversary (without access to X) can construct a prompt p such that $\text{Gen}(p) = x$.

   Approximately 0.00001% of GPT-2's training dataset.

   Definition 2 (Discoverable Memorization). For a model Gen and an example $[p \mid x]$ from the training set X, we say that x is discoverably memorized if $\text{Gen}(p) = x$.

   To be clear: this means that we randomly sample $[p \mid x]$ from the training data, and check if LLM generates $\text{Gen}(p) = x$

   Prior work shows that many LLMs discoverably memorize roughly 1% of their training datasets (when prompting the model with about 50 tokens of context)

   "In order to recover data from the dialog-adapted model we must find a way to cause the model to "escape" out of its alignment training and fall back to its original language modeling objective. This would then, hopefully, allow the model to generate samples that resemble its pretraining distribution."

   Memorization score (Biderman et al., 2024): For a given string $S = (S_1, \ldots, S_m)$, a start index k and a length l, it is the fraction of tokens from $k+1$ to $k+l$ generated by a LLM with prompt $S_{1:k}$, that agree with the corresponding tokens of S.

   Memorization and double descent

4. "Super"-phenomena: some components (activations, weights) of LLMs can be of a huge magnitude, much larger than other similar components.

   Super-activations (or massive activations) (Sun et al., 2024): large activations arise in a small fraction of LLM embeddings. They arise in fixed positions (e.g., first token—related to attention sinks (Xiao et al., 2024), first "newline"), and in specific dimensions of the embeddings. Their values are nearly input-independent. Setting them to zero destroys LLMs. They change the activation layers to focus on those specific tokens. "Explicit attention biases eliminate the need of LLMs to learn massive activations during pretraining."

   Super-weights (Yu et al., 2024): super-activations are partly caused by very large super-weights. Modifying them can completely degrade LLM performance, resulting in gibberish. In Llama-7B, "a single super weight is more important than even the top 7,000 largest weights combined."

   There are usually a few per LLM. They can be identified from one forward pass by considering a down-projection layer in a FFN, i.e., $e'_i = W_{\text{proj}}\tilde{e}_i$, where $\tilde{e}_i = \sigma(W_1 e_i)$. We find large entries of e'_i and $\tilde{e}_i$ (say a, b), and identify $[W_{\text{proj}}]_{a,b}$ as a super-weight.

5. What can be done in representation space?

2.6 Extensions

The same general idea of tokenization and transformers can be extended to other modalities, such as images (Dosovitskiy et al., 2021), audio, video, as well as multimodal language models.

2.6.1 Vision Transformers

Vision Transformers (ViT) (Dosovitskiy et al., 2021): a transformer architecture for image classification

1. Tokens: image patches, sorted in some arbitrary order
2. For classification, prepend an additional learnable embedding in front of all embeddings, "whose state at the output of the Transformer encoder serves as the image representation. Both during pre-training and fine-tuning, a classification head is attached to this output representation."
3. Train via semi-supervised learning, following the idea of a Noisy Student (Xie et al., 2020): "On ImageNet, we first train an EfficientNet model on labeled images and use it as a teacher to generate pseudo labels for 300M unlabeled images. We then train a larger EfficientNet as a student model on the combination of labeled and pseudo labeled images. We iterate this process by putting back the student as the teacher. During the learning of the student, we inject noise such as dropout, stochastic depth, and data augmentation via RandAugment to the student so that the student generalizes better than the teacher."
4. See Nauen et al. (2024) for an empirical comparison of various vision transformers

2.6.2 Vision Language Models

Vision Language Models (VLMs): architectures for processing images and text jointly

1. First, need to define a workflow of input and output. Here one can imagine many different reasonable use cases. For instance, the user might like to generate a description/caption of an image, or an image matching a caption, or might submit an image and the question and ask for an answer, or might submit multiple images and ask for answers, etc.

   It is challenging to design an architecture that can address all possible use cases. Hence most models fix a a specific workflow.

2. One of the most basic problems is image $\to$ text.

   Image-caption pairs are very commonly available on the Internet.

   A basic initial approach may be (Joulin et al., 2016)

   $$\text{image} \to \text{CNN} \to \text{bag of words}$$

   A reasonable next approach may be the architecture

   $$\text{image} \to (\text{CNN, ViT}) \text{ encoder} \to (\text{transformer}) \text{ decoder} \to \text{text}$$

   In influential work, Radford et al. (2021) find this empirically harder to scale than the approach of Joulin et al. (2016). [?]

Instead, Radford et al. (2021) approach this in a somewhat indirect way. They identify an intermediate step: to represent images and text in the same space; or, embed them into the same space in a semantically meaningful way.

Radford et al. (2021) develop a popular approach, Contrastive Language-Image Pre-training (CLIP), using a contrastive approach:

- Given N (image, text) pairs, construct (image embeddings, text embeddings) pairs, using standard feature/embedding maps.
- Fit a small linear layer to map image and text embeddings into a common space, obtaining: (image embeddings, text embeddings) $\text{image}_i, \text{text}_i$ all in the same space.
- Now, compute all inner products: $a_{ij} = \langle \text{image}_i, \text{text}_j \rangle$
- Based on the values:
$$a_{i\cdot} = (a_{i1}, a_{i2}, \ldots, a_{iN}),$$
interpreted as predicted logits, attempt to recover the image class i, by making a_{ii} larger than the others.
- Specifically, use a cross-entropy loss, aiming to predict one hot encoded class i (i.e., e_i) from the logits $a_{i\cdot}$.
- Sum all losses over all i.
- Add analogous losses for predicting the right text captions as well.
- How to use this for image classification?
  - Given textual descriptions of classes, e.g., plane, car, dog, ...
  - Find text embedding of "A photo of [class name]".
  - Find image embedding and select top class prediction.
- Many more later applications...

3. GPT-4 (Achiam et al., 2023) can handle multi-turn text generation, where the user input at every turn is a set of images and text. However, we do not know how it is trained.

4. Another basic scenario is to generate a caption based on the user image and questions; this is used e.g., in LLaVA (Liu et al., 2024b)

   (a) The first step is to collect data. We need data of (image, question, answer) pairs. While we can collect data of this type from scratch (e.g., all human generated), this might be quite costly. Also there is no sufficiently big data set of exactly this type. Instead, what is very commonly available on the Internet is image-caption pairs. Liu et al. (2024b) feed such a dataset to GPT-4 to generate questions; thus obtaining the required data.

   (b) The next step is to design an architecture that takes Input = (image, question); Output = answer

   Go through embeddings:

   Image $\to e_{\text{img}} \to e_{\text{text rep of img}}$; [first step is based on CLIP (Radford et al., 2021); second requires "aligner"]

   Question $\to e_{\text{text}}$

   Then $(e_{\text{text rep of img}}, e_{\text{text}}) \to$ text prediction

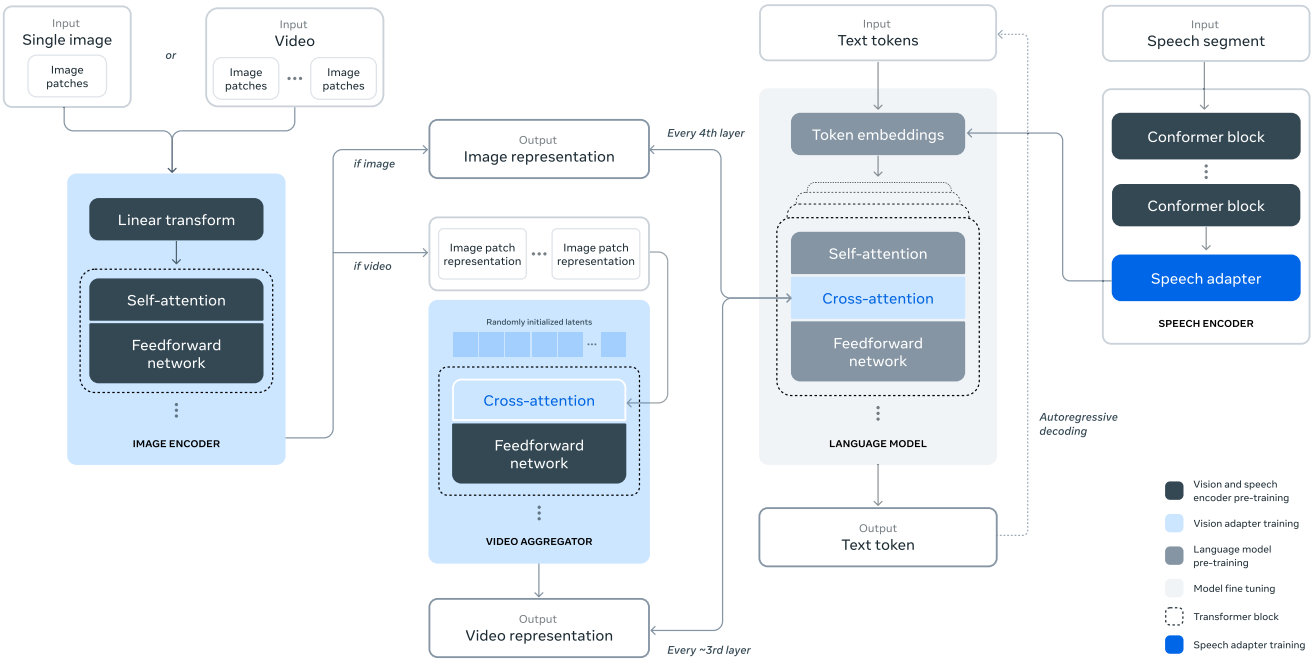


Figure 7: Llama 3. Dubey et al. (2024)

5. Dai et al. (2023)

   Li et al. (2023)

2.6.3 Multimodal Language Models

Multimodal Language Models (MLMs): architectures for processing several modalities jointly

1. e.g., Llama 3 (Dubey et al., 2024), see Figure 7. Also, LLaVA-NeXT-Interleave Li et al. (2024a).

   Discuss Llama 3: "we incorporate visual-recognition capabilities into Llama 3 via a compositional approach"

   Three stages:

   (a) Multi-modal encoder pre-training

   train encoders for images (image → text; CLIP-style but with additional lower-level spatial features added to the final rep.) and speech (self-supervised)

   (b) Vision adapter training: "cross-attention layers between the visual token representations produced by the image encoder and the token representations produced by the language model", as in Flamingo (Alayrac et al., 2022)

   video adapter training [skip]

   pre-training is standard, but need careful fine-tuning (only image adapter; frozen instruction-tuned LLM); use an average of top-performing models over various hyperparameters (Wortsman et al., 2022)

also train vision reward model, and do DPO + rejection sampling

(c) Speech adapter training [skip]

2.7 Other architectural considerations

1. Mixtures of Experts
2. Pathways (Chowdhery et al., 2023)
3. Cascades/early-exit architectures: if a certain score is high at a given layer, exit and generate based on current layer output/report current LLM's answer

2.8 Other notable models

1. Qwen series: (Bai et al., 2023; Yang et al., 2024) Open source; QwQ https://qwenlm.github.io/blog/qwq-32b-preview/ enhanced test-time compute
2. Open interpreter https://github.com/OpenInterpreter/open-interpreter "natural language interface for computers"

3 Training LLMs

1. Overarching principle: Loss minimization. Given a class $f_w, w \in \mathcal{W}$ of models, solve

$$\min_{w \in \mathcal{W}} L(f_w),$$

where L is a loss function that depends on data.

2. The training pipelines of most frontier LLMs are not public, as they are performed internally by large tech companies. So no-one really knows how e.g., GPT-4 was trained. However, there are some notable attempts to produce open LLMs of various degrees, e.g.,

   (a) open weights: Llama (Dubey et al., 2024)

   (b) fully open: LLM360 (Liu et al., 2023)

3. Phases of training: pre-training, post-training (alignment)

3.1 Pre-training

1. Train LLMs on enormous amounts of data, to maximize probability of observed text

   Objective: $\max_\theta \sum_{x \in D} - \log P_\theta(x)$.

   Simple MLE objective. Empirical maximizer over all distributions: empirical distribution. [Provides lower bound over train loss]

2. Beyond next-token prediction

   Lin et al. (2024): There can be low-quality/useless tokens in text. E.g. In "12 cars, 3 bikes User123: 15 vehicles", "User123" is not useful for reasoning. Can we filter these tokens? Attention is forced to do this automatically,

Also, sometimes you only pay attention to specific tokens (12, 3), if you know that a certain other token is present (15).

Lin et al. (2024) propose Selective Language Models (SLM). Given a reference model (e.g., small model trained on high-quality data, or just a baseline model), consider loss difference between current model during training and reference model. Select tokens with top scores, and only back-propagate on them. Train faster, same size model can become better.

3. Bitter lesson. More compute makes AI better.

   Current SOTA LLMs are trained with huge amounts of compute

   (a) Llama 3 405: 16K H100 GPUs, at least 54 days. If rented at 2.4 USD/hour (Nov 2024 price), cca $50M. Compute $4 \cdot 10^{25}$ flops

   (b) this is representative, other frontier models are around the same ballpark

   (c) what is the smallest amount of compute that anyone has been able to use to train a LLM that is as good as ChatGPT?

   Llama3-8B is as good as GPTT3.5-Turbo on many common-sense text benchmarks. (Dubey et al., 2024). Reported to take 1.3M GPU hours[2], i.e., cca $3.1M.

   Key Q for us: if we cannot train a new LLM, what *can* we do?

   [what is cost of fine-tuning a frontier model? Llama 3 8B used more than 10M SFT data samples.[3] Say avg 1000 tokens, as per Llama 3 paper. Total 10B tokens. Extrapolating full cost, cca $200K.]

4. Sources include: Wikipedia, books, Reddit, Stack Overflow, newspapers, etc.

   Next word prediction forces the NN to learn a lot about the world.

   Specific datasets

   (a) Common Crawl: A massive web crawl dataset. cca 400 TB (Nov '24) https://commoncrawl.org/

   quality is unreliable. C4 is a cleaned version.

   (b) The Pile: A curated 800GB dataset designed for LLM training (Gao et al., 2020)

   (c) OpenWebText: text of websites linked from Reddit posts; cca 65GB (Nov '24) https://openwebtext2.readthedocs.io/en/latest/

   (d) RedPajama (Weber et al., 2024): open dataset, 100 trillion tokens with quality annotations.

   (e) RefinedWeb

   (f) Code: StarCoder

   Training on repeated data can decrease performance (through memorization?) (Hernandez et al., 2022)

[2] https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md
[3] https://huggingface.co/blog/llama3..

5. Pre-training large models relies on roughly the same internet data. This data is finite.

   What is the future? Sutskever '24 NeurIPS talk: Agents? Synthetic Data? Inference time compute/reasoning?

6. Sometimes train small but powerful models on data distilled from larger models, e.g., (Hsieh et al., 2023)

7. energy efficiency: brain much more eff (1000-1mil) than computer; adapt computer architecture (von Neumann arch w/ mem and HDD is not efficient);

   potential approach: sparsity/low precision?

8. Scaling laws (Kaplan et al., 2020)

   (a) Main objective tracked during training is the perplexity: $2^{\text{avg bits/token}}$
   Formally, for an LM p the perplexity of a string x is

   $$q(x) = 2^{-\sum_{t=1}^{|x|} \log_2 p(x_t|x_{1:t-1})/|x|} = p(x)^{-1/|x|}$$

   The perplexity of the LM on a dataset D is

   $$Q = \mathbb{E}_{X\sim D} q(X)$$

   We minimize the training entropy and evaluate the test perplexity.
   In practice, we estimate it via sampling. [How accurate?]
   Related objects are the entropy $H(P) = \mathbb{E}_{X\sim p} \log_2[1/p(X)]$ and the per-token entropy $\tilde{H}(P) = \mathbb{E}_{X\sim p} \frac{1}{|X|} \log_2[1/p(X)]$
   Shannon (1948) estimates that the per-character entropy of English is between 0.6 and 1.3.

   (b) Scaling laws are empirical observations about the behavior of the test error of LLMs in training.
   Let D be the dataset size, and N be the number of non-embedding parameters in a transformer-based LLM.
   Let $L(\cdot)$ denote the test perplexity achieved by the best LLM (among a few possibilities) constrained by its input arguments.
   Then, Kaplan et al. (2020) has found that there are scalars $\alpha_N, \alpha_D > 0$, $N_c, D_c > 0$ such that

   $$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\alpha_N/\alpha_D} + \left(\frac{D_c}{D}\right)\right]^{\alpha_D}$$

   Here, N_c, N_d can be viewed as certain critical values, above which the scaling laws hold.
   Performance decreases as a power law. They find $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$
   This holds over several orders of magnitude of N, D.
   Omitting constants, this observation can be viewed as

   $$L(N, D) \sim \frac{1}{N^{\alpha_N}} + \frac{1}{D^{\alpha_D}}.$$

   The total amount of compute can be approximated as proportional to $C = N \cdot D$.

Thus, given a specific compute budget C_0, we would like to solve

$$\min_{N,D} L(N, D) \quad \text{subject to } N \cdot D \leq C_0.$$

The optimum is achieved when $N^{\alpha_N} \sim D^{\alpha_D}$.

In one example from Kaplan et al. (2020), this amounts to $D \approx N^{0.74}$, so we should increase dataset size sublinearly in the number of parameters.

Also have a similar scaling law for compute

(c) Compute-optimality: Later, the Chinchilla paper (Hoffman et al., 2023) found a different scaling law,

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta},$$

where $E = 1.69$, $\alpha \approx 0.34$, $\beta \approx 0.28$. Since the exponents are close, this suggests roughly equal scaling of model and data/sample size.

One of their experimental approaches: train a variety of models (various architectures, sizes, and dataset sizes). Plot (smoothed) train loss as a function of FLOPs. Find lower envelope.

They motivate this decomposition as

$$L(N, D) = L(\hat{f}_{N,D}) = L(f^*) + L(f_N) - L(f^*) + L(\hat{f}_{N,D}) - L(f_N),$$

where L is the population-level risk function (expected loss in the population, entropy of natural text), $L(f^*)$ is the Bayes risk, $L(f_N) - L(f^*)$ is the approximation error achievable by the best model f_N of size N, and $L(\hat{f}_{N,D}) - L(f_N)$ is the random error of the fitted model $\hat{f}_{N,D}$. [not perfect as scaling law is for the train loss; if only use each datapoint once, then we do SGD on the test loss. However, if we evaluate the loss over the entire dataset (including those used during GD), then we do not have an unbiased estimate of the test loss; not clear what they did?]

9. Can branch off and continue pre-training on domain specific data to obtain early a model with an improved performance on that domain; e.g., Llama 3 does this for code (Dubey et al., 2024). Use this as a code expert/judge, while the final model is still used for code eventually

   [If the model is large enough, it has capacity to learn both coding and English, and there is no need for two models.]

10. Data cleaning: the data must be filtered during pre- and post-training

    For instance: remove undesirable sources to avoid learning about their topics; quality filtering (excessive use of emojis, low complexity responses, ...)

3.1.1 Statistical formulation of generative AI: learning to sample

1. What does statistics have to say about generative AI?

   From a statistical point of view, generative AI can be viewed as learning to sample: Given data (x_i, y_i), $i \in [n]$ of input-output pairs, and a new input x_{n+1}, (construct a way to) sample/generate $\hat{y}_{n+1}$.

   It is understood that we want the distribution $\hat{p}$ of $\hat{y}_{n+1}$ given x_{n+1} to be close to the distribution of y_i given x_i, $i \in [n]$; and in fact from a hypothetical true distribution from which these are sampled.

Predictive Model $\hat{p}$	Input Space $\mathcal{X}$	Output Space $\mathcal{Y}$
Large Language Models	Text	Text
Diffusion Models	Text	Images
Multimodal Language Models	Images, text	Images, text, sound, video
Protein Structure Prediction	Amino acid sequence	3D structure

2. Note that this is not the same as purely learning the distribution. In fact, some standard distribution and density estimators do not lead to automatic/convenient ways to sample (and may need some extra steps). Consider for instance a tree-based estimator that partitions the space iteratively and computes constant density estimates in each leaf. To be able to sample from it, on general we would need to also keep the number of data points move into each branch in the internal notes (this is not hard, but is some additional work beyond the basic density estimator).

3.2 Post-training

1. Pre-training uses "generic" data. For any specific application, may use some additional data to fine-tune the model (Dai and Le, 2015). This can (1) improve performance and (2) align LLMs to specific values.

   Quoting Ouyang et al. (2022) [InstructGPT]: "the language modeling objective [...]—predicting the next token on a webpage from the internet—is different from the objective "follow the user's instructions helpfully and safely""

2. Components, following Ziegler et al. (2019); Ouyang et al. (2022):

   (a) Supervised fine-tuning (Dai and Le, 2015)

   (b) Alignment via RLHF [special note: Ziegler et al. (2019) already has the entire "modern" RLHF pipeline fully developed]

      i. reward modeling/learning [broad area here is inverse reinfocement learning (Ng et al., 2000)]

      ii. policy optimization (Christiano et al., 2017)

   these steps can be repeated; e.g., Llama 3 iterates six times (Dubey et al., 2024)

   See also Bai et al. (2022a).

3. Zhou et al. (2023a) post-training/alignment can be achieved with a relatively small number of examples

3.3 Supervised fine-tuning

1. After pre-training, the model approximates the probability distribution of the text on the internet.

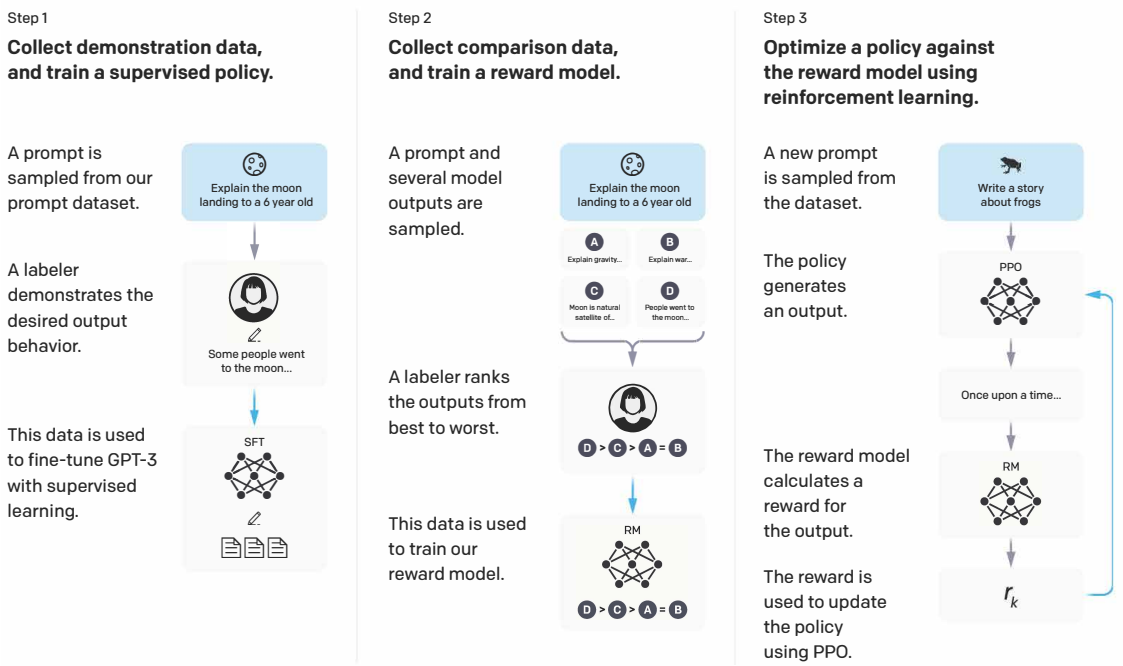


Figure 8: Post-training. Ouyang et al. (2022)

2. For typical use-cases, we want the LLM to behave more like an assistant. For example:
   - Given a question, answer it politely.
   - Given a follow-up, clarify and iterate.
3. This kind of text is quite rare, especially for multi-turn conversations.
   - For example, Stack Overflow is usually one turn.
   - Reddit - many tasks are not represented, and usually we have just a few turns.
4. Therefore, we need to enrich the LLM with this type of data.
5. Ask humans to write Q&A format text.

   Companies may use their own user-submitted data.
6. Fine-tuning: Train the model with the same loss function, starting from the pre-trained model, on this data. Given $q_1, a_1, q_2, a_2, \ldots, q_k$, predict a_k.

   More generally, fine-tuning is post-training-training small but task- specific data.
7. instruction tuning (Wei et al., 2022a) is a more complex version, where several versions of the fine-tuning data are created; so we post-train on a diverse set of tasks/instructions

   "For each dataset, we manually compose ten unique templates that use natural language instructions to describe the task for that dataset. While most of the ten templates describe the original task, to increase diversity, for each dataset we also include up to three templates that "turned the task around," (e.g., for sentiment classification we include templates asking to generate a movie review). We then instruction tune a pretrained language model on the mixture of all datasets"

   [principle: transform data to have coverage in target distribution]

8. Example dataset: Evol-instruct (Xu et al., 2024), generated via LLMs.

3.4 Learning a reward/Reward modeling

3.4.1 Learning a reward based on direct human evaluation

1. Suppose we want the model to be polite.

   No clear definition is available.

2. The first thought is to collect a dataset of prompt-response pairs (x, y), where y is specifically chosen to be polite. For example, the humans writing the answers are asked to write politely.

3. However, this can be data-inefficient. How can we make sure that x covers a sufficiently many topics? If, for a given topic, the model has not seen an (x, y) pair with that topic and a polite response y, then it may still likely respond impolitely.

4. It can also be labor-inefficient, as humans need to write a lot of extra responses.

5. Instead, only ask humans about their preferences/rating. The most direct approach is as follows:

   - Human prompt x
   - LLM answer y
   - Human evaluation $\tilde{r}$ (high if good) $\rightarrow$ reward

6. Then use this data to learn a reward model $r : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ by regressing $\tilde{r}(x, y)$ onto (x, y), using supervised learning.

   - We can use the same transformer architecture, just change the last layer to a scalar readout.

3.4.2 Learning a reward based on preference data/Preference modeling

1. A crucial challenge with this approach is that eliciting a reward from humans can be hard. For instance:

   - On a scale of 1 to 10, how polite are the following answers?
     (a) What time is it?
     (b) Could you please let me know the time?
     (c) Would you mind telling me the time?
     (d) Can you tell me what the time is?

     Hard to assign absolute numbers. However, somewhat easier to assign rank, e.g., B & C more polite than A & D.

2. This is the motivation for learning a reward based on preference data.

   - Human prompt x
   - LLM answers y_1, y_2

- Human preference $I(y_1 \succ y_2)$, i.e., is y_1 preferred over y_2?

Multiple comparisons can also be handled.

3. Then, to learn the underlying reward, we assume that

$$P(y_1 \succ y_2|x) = F(r(x, y_1), r(x, y_2))$$

for some known function F, and fit this to the data to learn the reward r. [r is called a preference model in Bai et al. (2022a)]

4. The most commonly used tool is the Bradley-Terry model (BTM) (Bradley and Terry, 1952):

$$P(y_1 \succ y_2|x) = \sigma\left(r(x, y_1) - r(x, y_2)\right),$$

where σ is the sigmoid function, $\sigma : x \mapsto 1/(1 + e^{-x})$.

Intuition: given two players with latent skills θ_1, θ_2, BTM models the probability that the first one wins a game against the second one by $\exp(\theta_1)/[\exp(\theta_1) + \exp(\theta_2)]$.

5. Then, the maximum likelihood objective is:

$$\max_\theta \sum_{(x,y^+,y^-)\in D} \log \sigma\left(r_\theta(x, y^+) - r_\theta(x, y^-)\right),$$

where y^+ is the preferred answer, and y^- is the other one.

Can also write

$$\begin{aligned}&\log \sigma\left(r_\theta(x, y^+) - r_\theta(x, y^-)\right)\\ &= I(y_1 \succ y_2) \log \sigma\left(r_\theta(x, y_1) - r_\theta(x, y_2)\right) + I(y_2 \succ y_1) \log \sigma\left(r_\theta(x, y_2) - r_\theta(x, y_1)\right)\end{aligned}$$

6. Overall, this approach consists of two phases:

   (a) Learning a reward.

   (b) Policy optimization.

This is also referred to as reinforcement learning from human feedback (RLHF).

7. The two optimization objectives are to maximize over w, θ:

$$\mathbb{E}_{X\sim D}\mathbb{E}_{Y_1,Y_2\sim p_w(\cdot|X)} \log \sigma\left(r_\theta(X, Y^+) - r_\theta(X, Y^-)\right).$$

$$\mathbb{E}_{X\sim D}[\mathbb{E}_{Y\sim p_w(\cdot|X)}\, r_\theta(X, Y) - \beta\, \mathrm{KL}\left(p_w(\cdot|X) \,\|\, p_0(\cdot|X)\right)],$$

Typically this is done sequentially, first maximizing the first objective with respect to θ (for a given p_w, e.g. the pre-trained model), and then maximizing the second objective with respect to w for the given θ.

However one could imagine simultaneous optimization.

8. Example dataset: PKU Safe RLHF https://github.com/PKU-Alignment/safe-rlhf

3.5 Using a learned reward

3.5.1 Rejection sampling+SFT based on the learned reward

1. The learned reward can be used to perform rejection sampling on data generated by the model. Sample a number of responses, and keep the best according to the reward; e.g., Llama-3 (Dubey et al., 2024) samples between 10-30 responses

   The resulting data can be used for SFT

3.5.2 RL Fine-tuning based on the learned reward

1. Then, fine-tune the model by solving a reward maximization objective of the form:

$$\max_w \mathbb{E}_{X\sim D}\mathbb{E}_{Y\sim p_w(\cdot|X)}\, r(X,Y).$$

   Here the LM is viewed as a policy to be optimized (using terminology and ideas from reinforcement learning).

2. To ensure that the model does not move too far from the pre-trained model p_0, we may use a proximal policy optimization (PPO) objective of the form (Schulman et al., 2017):

$$\max_w \mathbb{E}_{X\sim D}[\mathbb{E}_{Y\sim p_w(\cdot|X)}\, r(X,Y) - \beta\, \mathrm{KL}\left(p_w(\cdot|X) \,\|\, p_0(\cdot|X)\right)]$$

   where the KL regularization aims to keep p_w close to p_0.

3. Can consider/construct multiple rewards. Bai et al. (2022a) construct rewards for helpfulness and harmlessness.

4. Can extend this approach to multi-turn conversations, where a response is generated at each time step, a conversation is a trajectory, and has one reward associated.

5. How to calculate gradients?

   Consider first calculating

$$\nabla_w \mathbb{E}_{Y\sim p_w}\, r(Y) = \nabla_w \sum_y p_w(y) r(y) = \sum_y \nabla_w p_w(y) r(y).$$

   In principle, can calculate this in closed form. However, the sum is over all strings of a certain length and so is over an exponentially large space.

   Need to construct stochastic gradient estimators.

   Here use the so-called REINFORCE estimator (Williams, 1992). To derive this, write

$$\sum_y \nabla_w p_w(y) r(y) = \sum_y [\nabla_w \log p_w(y)] r(y) p_w(y) = \nabla_w \mathbb{E}_{Y\sim p_w}\, r(Y) \nabla_w \log p_w(Y).$$

   Hence, for $Y \sim p_w$, $r(Y)\nabla_w \log p_w(Y)$ is an unbiased gradient estimator.

   Can compute the gradient $\nabla_w \log p_w(Y)$ in the usual way.

3.6 Direct Preference Optimization

1. Is it possible to simplify the two steps of (1) learning a reward and (2) fine-tuning into one? Direct Preference Optimization (DPO) (Rafailov et al., 2024) does that.

2. Observe that maximizing the PPO objective over all policies, given the pre-trained model p_0 has solution:
$$p(y|x) = p_0(y|x)\frac{\exp(r(x,y))}{Z(x)},$$
where $Z(x)$ is a normalization factor.

3. Objective: maximize
$$\mathbb{E}_{Y\sim p(\cdot|x)}\, r(x,Y) - \beta\, \mathrm{KL}\left(p(\cdot|x)\|p_0(\cdot|x)\right)$$
- Drop x, divide by $\beta > 0$, redefine $r \leftarrow r/\beta$:
$$\mathbb{E}_{Y\sim p}\, r(Y) - \mathrm{KL}(p\|p_0)$$
$$\sum_y r(y)p(y) - \sum_y \log\left(\frac{p(y)}{p_0(y)}\right)p(y)$$
$$\sum_y \left(r(y) + \log p_0(y)\right)p(y) - \sum_y \log(p(y))p(y)$$
- Constraints:
  - $p(y) > 0,\ \forall y$
  - $\sum_y p(y) = 1$
- Introduce Lagrange multiplier λ:
$$\max_p \sum_y \left(r(y) + \log p_0(y) - 1\right)p(y) - \sum_y \log(p(y))p(y)$$
- Evaluate $\frac{\partial}{\partial p(y)}$; stationary point:
$$r(y) + \log p_0(y) - \lambda - \log(p(y)) - 1 = 0$$
$$r(y) + \log p_0(y) - \lambda = \log(p(y)) + 1$$
- Note $(x\cdot\log x)' = \log x + 1$.
$$\Rightarrow \frac{p(y)}{p_0(y)} = \exp\left(r(y) - \lambda - 1\right)$$
- Only need to find λ such that all constraints hold.

4. Equivalent to
$$\sum_y p_0(y)\exp\left(r(y) - \lambda - 1\right) = 1$$
So,
$$\lambda = \log\left(\sum_y p_0(y)\exp(r(y))\right) - 1.$$

Therefore, for each fixed x, the distribution achieving the maximum is

$$p \propto p_0 \exp(r/\beta).$$

Therefore, at the population level, i.e., if we had data $\mathcal{D}$ covering all x, we would conclude

$$p^* \propto p_0 \exp(r/\beta).$$

In practice, still use this parametrization.

5. Now, we can substitute this into the BTM objective to eliminate the reward model.

   - The PPO optimality is equivalent to

   $$r(x, y) = \log\left(\frac{p(y|x)}{p_0(y|x)}\right) - Z(x).$$

   Hence, the BTM objective becomes, for this reward model

   $$\begin{aligned}&\sigma\left(\log\left(\frac{p(y_1|x)}{p_0(y_1|x)}\right) - Z(x) - \left[\log\left(\frac{p(y_2|x)}{p_0(y_2|x)}\right) - Z(x)\right]\right)\\ &= \sigma\left(\log\left(\frac{p(y_1|x)}{p_0(y_1|x)} \Big/ \frac{p(y_2|x)}{p_0(y_2|x)}\right)\right).\end{aligned}$$

   Since $\sigma(\log(a)) = a/(a+1)$, this can be simplified to

   $$\frac{\frac{p(y_1|x)}{p_0(y_1|x)}}{\frac{p(y_1|x)}{p_0(y_1|x)} + \frac{p(y_2|x)}{p_0(y_2|x)}}.$$

6. Notice that the normalization term $Z(x)$ cancelled out.

7. The DPO objective is

   $$\max_w \mathbb{E}_{X\sim D}\mathbb{E}_{Y\sim p_w(\cdot|X)} \log \frac{\frac{p_w(Y^+|X)}{p_0(Y^+|X)}}{\frac{p_w(Y^+|X)}{p_0(Y^+|X)} + \frac{p_w(Y^-|X)}{p_0(Y^-|X)}}.$$

   This objective can be used directly to fine-tune a LLM p from preference data.

8. Gaps in the analysis above:

   (a) We are not optimizing over all policies, but rather only over parameterized subset. Then, the key equation (2) does not hold, and the two objectives may not be equivalent.

9. Empirical results can sometimes be mixed; but used e.g., in Llama 3 (Dubey et al., 2024).

   They use a few tricks: 1. mask out formatting tokens (the same tokens in both positive and negative may cause model instability due to the contrastive loss); 2. add scaled NLL loss for chosen response

3.7 Alternatives

1. Aligner (Ji et al., 2024): train a small post-processing module to correct LLM output y on answer x to corrected version y'. Train on (x, y, y'); y' generated by LLM, and x, y both given to aligner during training ("residual"). Also train on (x, y, y) data to learn copying behavior. Faster than RLHF.

3.8 Generating synthetic data for training

1. Training on synthetic data potentially allows a large expansion of the amount of training data available
2. Train on own data? Usually not directly helpful. However, can become helpful if it is filtered.

   Some of the above strategies (e.g., RLHF, rejection sampling), can be viewed as generating and filtering data
3. Another example: Llama 3 filters based on execution feedback (Dubey et al., 2024)

   Required steps, iterated several times:

   (a) problem generation (based on random code snippets, generate problem),

   (b) code generation (add prompt with rules of good programming, explain thought process),

   (c) verification

   (d) revision
4. Example: enhance multilingual performance on underrepresented languages by training on translated text

   Can also consider code back-translation

   But, training on translated natural language text can lead to a variety of biases, see e.g., (Ji et al., 2023)

3.9 Tool use

1. WebGPT: Browser-assisted question-answering with human feedback (Nakano et al., 2021)
2. Lambda (Thoppilan et al., 2022) uses an information retrieval system, a language translator, and a calculator
3. Toolformer (Schick et al., 2023)

   (a) It trains tool use in a self-supervised way (i.e. no annotated examples of tool use required).

   (b) Pre-specify a list of tools (QA tool—a RAG model; calculator, ...), each encoded in a common sequence format: (tool start token, tool name, input, tool end token).

   (c) Augment a given dataset to find examples where using tools helps. By random search, find strings such that

      i. for a given location, the probability of a "tool end token" is quite high;

      ii. inserting tool call + answer increases likelihood of next few tokens (use special weighted likelihood focusing on just next tokens).

(d) Fine-tune model on that data.

(e) For inference, allow tool use if "tool start token" is among the most likely few. [Here, allow only one API call.]

4. Program-aided LM (PAL) (Gao et al., 2023): uses the LLM to read natural language problems and generate programs as the intermediate reasoning steps, but offloads the solution step to a runtime such as a Python interpreter [specialized prompts for various problems]

5. Llama 3 (Dubey et al., 2024)

   (a) Implement core tools as Python objects with their respective methods. Have pre-specified list of tools (web search, Python interpreter); but users can define new tools in-context.

   (b) Tool calls are executed by the Python interpreter. So once LLM formulates a tool call, it is passed to Python and executed. [It seems that this gives the LLM the ability to run any code? As long as it is passed to the interpreter?]

   (c) Training leverages similar RLHF to main post-training, but allow feedback on each turn of conversation

   (d) Work with files: design prompts for typical user queries on pre-specified types of files (.txt, .pdf, .py, ...); e.g., summarize, improve code etc. Then do RLHF.

6. Differentiable memory

   Memorizing transformers (Wu et al., 2022): approximate kNN lookup into a non-differentiable memory of recent (key, value) pairs

   Recurrent Memory Transformer (Bulatov et al., 2022)

7. Survey: Mialon et al. (2023)

3.10 Embodiment

1. Palm-E (Driess et al., 2023)

2. Voyager (Wang et al., 2024b): LLM-powered embodied lifelong learning agent in an open-ended world (Minecraft). See the website for a quick intro.

   Interact with game using code, have access to state (environment, type of location, items) Prompt LLM asking what to do ("curriculum"). Prompt LLM how to do it. Retrieve "skills" based on RAG store of short functions. Prompt LLM based on these to write code for action. Self-verify asking LLM if code will work. Pass to game. If execution successful, store skill. If fail, use env feedback to ask LLM to update code.

3.11 Special considerations and forms of fine-tuning

3.11.1 Parameter efficient fine-tuning

Parameter efficient fine-tuning (PEFT), see e.g., Lialin et al. (2023)

3 classes: Selective, Reparametrization (LoRA, Hu et al. (2022), QLoRa Dettmers et al. (2024)), Additive (Adapters, Soft prompts)

3.11.2 Prompt and Prefix-tuning

1. Prompt tuning (Lester et al., 2021)

   Soft prompts: virtual tokens $v_1, \ldots, v_k$ prepended to actual embeddings of tokens of the prompt, learned for specific tasks in lieu of prompting/fine-tuning;

2. Prefix-tuning (Li and Liang, 2021) is a similar but somewhat more complex method. It learns prefix embeddings for all layers.

   The matrix of prefix embeddings is parametrized as $M_\theta[i] = MLP_\theta(M'_\theta[i])$, for all i, via a smaller matrix P'_θ composed with a large feedforward neural network

   [Principle: (1) replace manual prompt optimization by automatic procedure. (2) replace discrete optimization with continuous optimization. (3) intervene at a convenient point in the computation graph]

3.11.3 Self-play/improvement

1. Design a program improver; then use it to improve itself (Zelikman et al., 2023)

   in general, key challenge: no clear reward.

2. Constitutional AI (Bai et al., 2022b): start with a set of "principles"; use them as prompts for LLMs. Instead of RLHF, perform RLAIF: repeat the same steps as RLHF, but with everything generated by LLMs

   (a) Supervised component: Response, critique (via LLM), and revise; and fine-Tune the LLM on the revisions

   (b) Preference modelling: replace human preference with LLM preference

3.11.4 Model compression

1. Pruning

2. Quantization

3. Distillation

4 Inference/Test-time computation

4.1 Simple Decoding/Sampling methods

1. After obtaining a LM, we need to sample/decode/generate text.

   Perhaps surprisingly, direct sampling from the LM may not be that great, and can be improved

   See Figure 9 for examples of sampling from GPT-2 (Holtzman et al., 2020).

   They argue that pure sampling is noisy due to the "unreliable tail" of low-probability tokens whose probabilities are mis-estimated.

   This motivates studying a variety of other sampling approaches

   (a) One class of approaches aims to maximize the probability of the generated sequence

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, *b*=32:
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

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."

Figure 9: Examples of sampling from GPT-2 (Holtzman et al., 2020). Both beam search and pure sampling have issues

i. These are based on the hope that high probability tokens are "reasonable"; and "best reflect" text in training data

ii. In principle, one could aim to find the globally most likely token sequence: $\arg\max_y P(y|x)$. However, this is generally computationally infeasible.

iii. The simplest practical approach is to use greedy decoding, which sequentially takes the top prediction of the probability distribution for each token has strong reproducibility. Can provide reasonable performance if the prompt is sufficiently detailed, the language model is sufficiently powerful, and the answer is sufficiently straightforward.
However, otherwise it can lead to very repetitive text; even its more sophisticated version of beam search (see next) does so (see Figure 9).

iv. Beam search: Instead of only predicting the token with the best score, we keep track of b sequences (for example, $b = 5$, where b is referred to as the beam size). At each new time step, for these five sequences, we have V new possible tokens. This results in a total of $5 \times V$ new sequences, each being one token longer. Then, only the five best sequences are retained, and the process continues.
Can also have diversity issues (Vijayakumar et al., 2018)

v. But note that, there is no fundamental reason why natural text, generated by humans, should always maximize the probability. In fact, Holtzman et al. (2020) show that natural language does not maximize the probability of even reasonably powerful LLMs such as GPT-2 Large; see Figure 10.
Could be because LLM did not estimate distribution well. But, even if LLM perfectly reflects a "human language" distribution, for longer text and real speech the notion of "most likely" seems both ill-defined and not plausible. E.g., Holtzman et al. (2020) bring up Grice's Maxims of Communication Grice (1975), which argues that people optimize against stating the obvious. But can still make sense for useful tasks.

(b) Other forms of decoding aim to add some regularization to the maximization process:

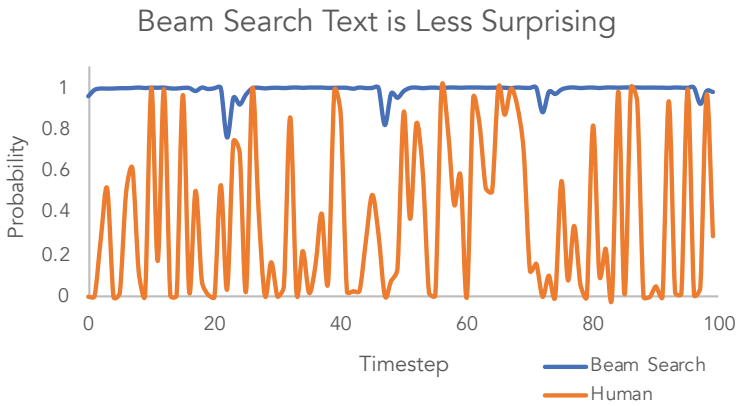
i. top-k: sample from top-k answers (Fan et al., 2018)

ii. top-p, also known as nucleus sampling (Holtzman et al., 2020): sample from top-p of the prob mass; [they argue it is better at generating human-like text]

(c) How to compare them? Methodology of Holtzman et al. (2020): compare gen. text to human text:

i. perplexity [pure sampling too low; beam and top-k are too high]

ii. n-gram frequency; Zipf law coefficient



Beam Search

...to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and...

Human

...which grant increased life span and three years warranty. The Antec HCG series consists of five models with capacities spanning from 400W to 900W. Here we should note that we have already tested the HCG-620 in a previous review and were quite satisfied With its performance. In today's review we will rigorously test the Antec HCG-520, which as its model number implies, has 520W capacity and contrary to Antec's strong beliefs in multi-rail PSUs is equipped...

Figure 10: Examples of sampling from GPT-2 (Holtzman et al., 2020). Human text does not maximize probabilities.

iii. probability of repetition

2. Temperature: logits $l \to l/\tau$

   Low temperature: $\tau \in (0, 1)$. Skews distribution towards high probability events; leads to less diversity.

   High temperature: the opposite.

3. Speculative decoding

   sample multiple tokens from a small model sequentially

   verify these tokens using a large model simultaneously (i.e., map them through model, and check e.g., that the resulting probabilities are large enough, via rejection sampling)

   (a) Accelerating Large Language Model Decoding with Speculative Sampling
   (b) Fast Inference from Transformers via Speculative Decoding
   (c) SpecTr- Fast Speculative Decoding via Optimal Transport
   (d) A Theoretical Perspective for Speculative Decoding Algorithm
   (e) Block Verification Accelerates Speculative Decoding

4. Best of N/Inference-time alignment.

   (a) Asymptotics of Language Model Alignment
   (b) BOND- Aligning LLMs with Best-of-N Distillation
   (c) Theoretical guarantees on the best-of-n alignment policy

   See also Brown et al. (2024)

5. Snell et al. (2024) find benefits of test-time compute over train compute (w/ same budget)

6. For inference-time architectural efficiency considerations: https://lilianweng.github.io/posts/2023-01-10-inference-optimization/

4.2 Prompting

1. Prompting: design specific text instructions for the model. Can have a variety of forms; e.g., system prompt is a (sometimes hidden) prompt prepended to all generations, such as "You are a helpful assistant..."

2. Few-shot learning/In-context learning: a special prompt, where the goal is to make the model learn a task from demonstrations provided within its context window (Brown et al., 2020).

   If a task requires many 'subtasks' to be solved, then performance on it may exhibit a sharper transition.

   Can be sensitive to prompt order (Lu et al., 2022)

   Empirically it helps if demonstrations/IC examples provide info about label space, distribution of input text, overall sequence format; and correct labels matter less (Min et al., 2022); put can be different for larger models (Wei et al., 2023)

How and why does it work? A possible explanation is that the latent topic of the questions is being inferred IC, which enables improved generation (Xie et al., 2022)

Is it learning or just conditioning? There is theoretical and empirical work to show that certain kind of learning in the classical sense is possible in-context; e.g., Given many sequences of (x_i, y_i) generated from linear models with different regression coefficients, and a test ICL sequence with a new coefficient, the model can learn to predict the outcome corresponding to the true new coefficient (Garg et al., 2022; Akyürek et al., 2023).

4.3 Reasoning

1. Reasoning methods often involve significant test-time compute. Sometimes they involve relatively simple prompting ("Let's think step by step"), sometime more complex sampling strategies (Tree of Thoughts).

   Often, rely on verifiers. Can be good if verifying is easier than solution.

2. Chain of thought (Wei et al., 2022c): instead of (question, answer) data, finetune on (question, steps, answer) data

   See also "scratchpads" (Nye et al., 2022), who have a similar approach.

   Add "Let's think step by step" before each answer has a similar effect (Kojima et al., 2022)

   Least-to-most prompting (Zhou et al., 2023b): solve problems in two stages: (1) query the language model to decompose the problem into subproblems; (2) query the language model to sequentially solve the subproblems. Design custom prompts for specific tasks.

   Self-consistency: taking majority vote of answers can help improve performance (Wang et al., 2023).

   Self-taught reasoner (STaR): based on few-shot examples, can also generate reasoning paths, discard those that lead to incorrect answers, and fine-tune on correct paths; then repeat (Zelikman et al., 2022). Update: Quiet-StaR (Zelikman et al., 2024), learns to generate rationales at each token.

   [learned CoT: Hoffman et al. (2023)]

3. Tree of Thoughts (ToT) (Yao et al., 2024)

   ToT frames any problem as a search over a tree, where each node is a state $s = [x, z_{1\cdots i}]$ representing a partial solution with the input and the sequence of thoughts so far.

   A specific instantiation of ToT involves answering four questions: [a very delicate process]

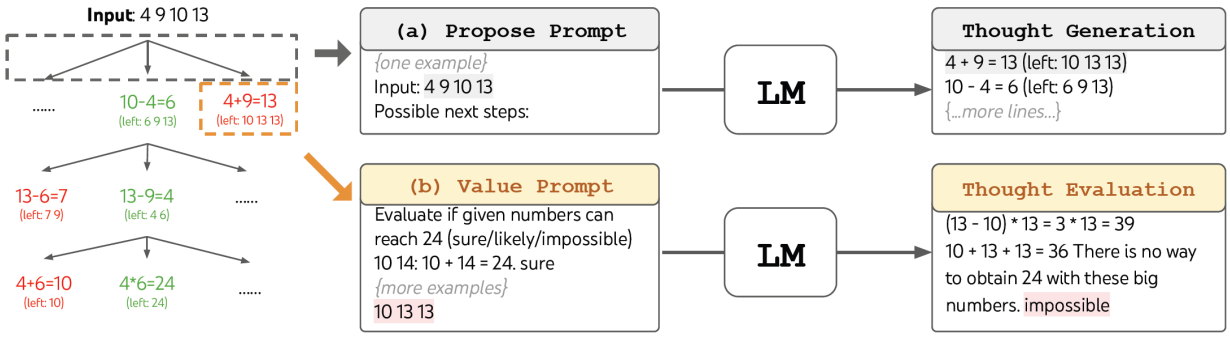
   1. How to decompose the intermediate process into thought steps; [a piece of text (everything is text or latent); need to find right size]

   2. How to generate potential thoughts from each state; [Sample]

   3. How to heuristically evaluate states; [Another LLM. Value LLM, sample $V(p_\theta, s) \sim p^v_\theta(s)$, for a state s (how trained — no extra training!)]

   4. What search algorithm to use. [BFS, DFS]

   Example: Game of 24 [this and all examples can be solved using classical algorithms: "the goal is not just to solve the task. rather, explore the limit of LM as a general problem solver that



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

guides its own exploration with deliberate reasoning as heuristics." – would not be acceptable in stats]

[principles: 1. common sense: problem solving involves options. 2. standard LLM can be ok for a lot of small/straightforward things 3. Frame problem as exploration of the limit of LMs, as opposed to solving any real problem 4. Figure out how to do it without any training]

4. Self-reflection (Shinn et al., 2023)

   Self-refinement (Madaan et al., 2023): refine reasoning for a variety of problems.

5. ReAct: Synergizing Reasoning and Acting in Language Models (Yao et al., 2023)

   Consider an agent interacting with an environment for task solving. At time step t, an agent receives an observation $o_t \in O$ from the environment and takes an action $a_t \in A$ following some policy $\pi(a_t|c_t)$, where $c_t = (o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$ is the context to the agent.

   ReAct: augment the agent's action space to $\hat{A} = A \cup L$, where L is the space of language. Then $c_{t+1} = (c_t, \hat{a}_t)$.

   a frozen large language model is prompted with few-shot in-context examples to generate both (1) domain-specific actions and (2) free-form language thoughts. For tasks where reasoning is of primary importance (E.g., HotPotQA below), we alternate the generation of thoughts and actions so that the task-solving trajectory consists of multiple thought-action-observation steps.

   Each in-context example is a human trajectory of actions, thoughts, and environment observations to solve a task instance. See Figure 11

   E.g., HotPotQA, a multi-hop question answering benchmark that requires reasoning over two or more Wikipedia passages.

   Action Space: wiki Search, lookup, finish. Prompting: manually compose action + thought trace

   [Principle: 1. give more detailed instruction to the language model. 2. give tools to the language model (web search). 3. write detailed prompt]

6. Decompose, Analyze and Rethink (Xue et al., 2024)

7. O1 https://openai.com/index/learning-to-reason-with-llms/

```
Question        Which documentary is about Finnish rock groups, Adam Clayton Powell or The
                Saimaa Gesture?
Thought 1       I need to search Adam Clayton Powell and The Saimaa Gesture, and find which
                documentary is about Finnish rock groups.
Action 1        Search[Adam Clayton Powell]
Observation 1   Could not find [Adam Clayton Powell]. Similar: ['Adam Clayton Powell
                III', 'Seventh Avenue (Manhattan)', 'Adam Clayton Powell Jr. State Office
                Building', 'Isabel Washington Powell', 'Adam Powell', 'Adam Clayton Powell
                (film)', 'Giancarlo Esposito'].
Thought 2       To find the documentary, I can search Adam Clayton Powell (film).
Action 2        Search[Adam Clayton Powell (film)]
Observation 2   Adam Clayton Powell is a 1989 American documentary film directed by
                Richard Kilberg. The film is about the rise and fall of influential
                African-American politician Adam Clayton Powell Jr.[3][4] It was later airec
                as part of the PBS series The American Experience.
Thought 3       Adam Clayton Powell (film) is a documentary about an African-American
                politician, not Finnish rock groups. So the documentary about Finnish rock
                groups must instead be The Saimaa Gesture.
Action 3        Finish[The Saimaa Gesture]
```

Figure 11: Hotpot QA Prompt example

Uses RL to train CoT. Add "actions" such as self-correct, try multiple strategies, break down problems.

video by Sasha Rush. Outlining, Planning, Backtracking. Self-evaluation.

8. Rewards. Related and sometimes equivalent concept: verifiers.

   (a) Consider an input x. LLM p generates a CoT z and an output y. Here, fix x but for a sample everything goes through by taking an expectation at the outer level wrt x.

   The correctness of the output may be checked against a correct answer y^*

   Alternatively, we may have a reward r that rewards the correctness of y in light of x (outcome reward). This may also depend on z (process reward) (Uesato et al., 2022)

   (b) Ultimately, would like to learn a policy/LLM to maximize reward $\max_p \mathbb{E}_{(Z,Y)\sim p(\cdot|x)} r(x, Z, Y)$, where Z is CoT.

   Key issue: combinatorially large space of Zs.

   (c) How get around it?

      i. Rejection sampling: Generate (x, z_i, y_i) for multiple i, where z_i is a CoT; Reward them based on correctness of answer. Fine-tune on good ones. (Same as in RS-SFT in Section 3.5.1).
      Can be viewed as rejection sampling EM:
      A. E step: Estimate $\mathbb{E}_{(Z,Y)\sim p(\cdot|x)} r(x, Z, Y)$ by $\mathbb{E}_n r(x, z_i, y_i)$
      B. M Step: maximize wrt p (take a GD step)
      Various names: Self-training (Yarowsky, 1995), Best-of-N training (Cobbe et al., 2021), STaR (Zelikman et al., 2022), ReST (Gulcehre et al., 2023), ReST-EM (Singh et al., 2024b), Filtered rejection sampling (Nakano et al., 2021).
      Iterative reasoning preference optimization (Pang et al., 2024) is a variant where we do DPO instead of just fine-tuning on good ones.

      ii. Produce intermediate process rewards r that apply to input x and partial solution z'. Can use fixed or learned reward

A. Can ask an LLMs; possibly w/ another CoT. In this case, the result can be verbalized and integrated into one reasoning stream (generator/solver and verifier(.

B. Or, estimate $\mathbb{E}_{(Y,Z'')\sim p(\cdot|x,z')} r(x, z', Z'', Y)$.
Monte Carlo Rollout. Given input x and partial solution z', sample $(Y, Z'') \sim p(\cdot|x, z')$, and use plug-in estimate. As in math-shepherd (Wang et al., 2024c)
Or, fit a model to the empirical rewards obtained from rollouts.

C. Given a learned reward, can use same rejection sampling

(d) Search

i. Beam Search

ii. Monte Carlo Tree Search: Grow a tree of solutions sequentially. Iteratively, for $t = 1, \ldots$. Sample a few possibilities (branches) for the next few tokens z_t. For each possibility, run rollouts to estimate reward. Update all parent nodes with total number of paths starting from it, and success rate/reward. Choose branch to explore next. Iterate.

(e) Learning to correct.

One possibility is to find similar CoTs z, z' where one lead to correct answer while the other to an incorrect one, and train to correct (Welleck et al., 2023). However, can be hard to enforce that correction is learned.

Another alternative is to exactly compute all paths from partial completions close to the solution (Gupta and Li, 2024).

Stream of search (Gandhi et al., 2024): after finding a good path, create a path traversing the entire tree, including backtracks and changes due to corrections (with backtracking tree search). Fine-tune on this data.

Alternative: distribution updates (e.g., Qu et al. (2024) fine-tune model on (wrong, wrong, wrong, correct), then sample iteratively at test time)

9. Combined techniques:

search over per-token (dense) process rewards: beam search, best-of-N, lookahead search (Snell et al., 2024)

Open-source implementation [ED: !]

10. What if no reward is available?

Self-rewarding LLMs (Yuan et al., 2024)

4.4 Knowledge Retrieval

1. Retrieval-Augmented Generation (RAG) (Lewis et al., 2020)

use the input sequence x to retrieve text documents z and use them as additional context when generating the target sequence y.

two components:

(i) a retriever $p_\eta(z|x)$ with parameters η that returns (top-K truncated) distributions over text passages given a query x and

(ii) a generator $p_\theta(y_i|x, z, y_{1:i-1})$ parametrized by θ that generates a current token based on a context of the previous $i-1$ tokens $y_{1:i-1}$, the original input x and a retrieved passage z.

treat the retrieved document as a latent variable. propose two models that marginalize over the latent documents to produce a distribution over generated text

RAG-Sequence Model:

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta\left(y_i|x,z,y_{1:i-1}\right)$$

RAG-Token Model

$$p_{\text{RAG-Token}}(y \mid x) \approx \prod_i^N \sum_{z \in \text{top}-k(p(\cdot|x))} p_\eta(z \mid x) p_\theta\left(y_i \mid x, z, y_{1:i-1}\right)$$

Retriever: Direct Passage Retrieval. follows a bi-encoder architecture:

$$p_\eta(z \mid x) \propto \exp\left(\mathbf{d}(z)^\top \mathbf{q}(x)\right) \quad \mathbf{d}(z) = \text{BERT}_d(z), \quad \mathbf{q}(x) = \text{BERT}_q(x)$$

Computing top-k probabilities for a fixed x reduces to maximum inner product search, use quantization-based sublinear time similarity search algorithm

[We refer to the document index as the non-parametric memory.]

Generator: BART [but could use any LLM]

Training: jointly train the retriever and generator components without any direct supervision on what document should be retrieved. Given set of query/output pairs (x_i, y_i), maximize average log-likelihood $\sum_i \log p(x_i, y_i)$; via Adam

Decoding/Sampling: RAG-Token model is standard autoregressive seq2seq generator with transition probability [...]. Decode via a standard beam decoder. RAG-Sequence: see Algorithm 1

Tasks: Open-domain Question Answering, Fact Verification etc

[Think answers must be short? They usually do not discuss complex postprocessing evaluation schemes: Kwiatkowski et al: Natural Questions: a Benchmark for Question Answering Research. Transactions of the Association of Computational Linguistics, 2019. "Long/short answers have 90%/84% precision, respectively."]

"RAG can generate correct answers even when the correct answer is not in any retrieved document, achieving 11.8% accuracy in such cases for NQ, where an extractive model would score 0%.

[principles: use an external database of trusted knowledge (Wikipedia); use embeddings for similarity search between queries and documents; in text space use concatenation to pool information (q+doc); use LLM to generate answer; use rules of probability to marginalize over documents (as extra variables)]

Algorithm 1 RAG-Sequence Decoding

Inputs: x: Input sequence. $Z = \{z_1, z_2, \ldots, z_m\}$: Set of retrieved documents. $p_\theta(y_i|x, z, y_{1:i-1})$: Generator probability. $p_\eta(z|x)$: Document relevance probability. b: Beam size.
Outputs: Decoded output sequence y^*.
Initialize candidate set $Y \leftarrow \emptyset$.
Step 1: Beam Search for Each Document
for each document $z \in Z$ **do**
    Perform beam search using $p_\theta(y_i|x, z, y_{1:i-1})$.
    Add all sequences generated during this beam search to Y.
end for
Step 2: Probability Scoring
for each sequence $y \in Y$ **do**
    for each document $z \in Z$ **do**
        if y does not appear in the beam for z **then**
            Thorough Decoding: Perform an additional forward pass to compute $p_\theta(y|x, z)$.
            Alternative fast decoding: Assume $p_\theta(y|x, z) \leftarrow 0$.
        end if
        Compute $p(y|x, z) \leftarrow p_\theta(y|x, z) \cdot p_\eta(z|x)$.
    end for
    Compute the marginal probability: $p(y|x) \leftarrow \sum_{z \in Z} p(y|x, z)$.
end for
Select the sequence y^* with the highest marginal probability: $y^* \leftarrow \arg\max_{y \in Y} p(y|x)$.
Step 3: Output the Result Return y^* as the decoded sequence.

5 Embeddings/Representations

1. Linear representation hypothesis (Park et al., 2024b):

   (a) Suppose:

   $$P(y \mid x) \propto \exp(\lambda(x)^\top \gamma(y))$$

   where:

   - x - context
   - y - next token
   - $\lambda(x)$ - context embedding
   - $\gamma(y)$ - readout embedding

   What is a representation of a concept? Park et al. (2024b) consider directional concepts such as "harmless → harmful". To define this, consider a set $Y^\dagger$ of harmful/harmless pairs (y_+, y_-) e.g.,

   - e-mail / spam
   - love / hate
   - peaceful / violent

   (b) Following the definition from Park et al., 2024, a direction $\bar{\gamma}$ is a representation of harmless → harmful if for all $(y_-, y_+) \in Y^\dagger$:

   $$\gamma(y_-) - \gamma(y_+) = C_{y_-, y_+} \bar{\gamma},$$

for some $C_{y_-,y_+} > 0$ depending on y_-, y_+.

2. While there is evidence that many representations are linear, e.g., Park et al. (2024b); Jiang et al. (2024b); Park et al. (2024a)

   https://transformer-circuits.pub/2023/monosemantic-features,

   there is also evidence for nonlinear e.g., circular features (Engels et al., 2024)

3. Luo et al. (2024) propose to intervene on representations by first finding the coefficients of an activation vector in an overcomplete dictionary of concepts via sparse regression, and then thresholding the harmful coefficients to zero. The dictionary of concepts is extracted from the LLM by feeding it frequent words + contexts (generated by another LLM); and then collecting the activations at the last token. Then a common representation is extracted via the representation reading algorithm (Zou et al., 2023a), which selects the top PC the normalized differences of activations among one concepts and all others (for all contexts).

6 Evaluation

1. What exactly does it mean to achieve AI? Can we give a precise definition?

2. The current consensus is that no precise definition is needed or necessarily desired. Instead, performance is evaluated on benchmarks.

3. In contrast to more standard data sets in machine learning, modern AI benchmarks are often highly heterogeneous, and aim to cover a variety of capabilities.

   Some of the most popular examples are the following:

   HellaSwag (Zellers et al., 2019): Common-sense multiple choice paragraph completion

   TruthfulQA: (Lin et al., 2022): Questions where humans have answered falsely

   ARC Challenge (Clark et al., 2018): AI Reasoning

   ARC-AGI (Chollet, 2019): Abstraction and Reasoning Corpus for Artificial General Intelligence. Some examples can be tried online. See 2024 report. Best approaches ask LLMs to write code (& improve it), and use test-time computation, e.g, evolutionary test-time computation and test-time training (Akyürek et al., 2024)

   [seems trivial to humans? despite all the fanfare, just another IQ test]

   MMLU (Hendrycks et al., 2021a): questions about a variety of subjects (AP History, Economics, Math etc)

   Big-Bench (Srivastava et al., 2023)

   GSM8K 8.5K (Cobbe et al., 2021): grade school math word problems. They also train a verifier, generate multiple solutions, and choose the one with the highest verifier score.

   MATH (Hendrycks et al., 2021b): Math competition problems: AMC 10, 12; AIME. In natural language; exact solutions checked.

   HumanEval (Chen et al., 2021): Python code generation, focuses on relatively simple, self-contained functions. HumanEval+ (Liu et al., 2024c): has better false positive control

   MBPP (Austin et al., 2021): Mostly Basic Programming Problems. 974 programming tasks, designed to be solvable by entry-level programmers.

Chatbot Arena (Chiang et al., 2024)

4. Robustness

   Adversarial QA (Jia and Liang, 2017)

5. Currently challenging benchmarks

   GAIA: a benchmark for General AI Assistants (Mialon et al., 2024). Common-sense questions requiring some web/lit search, some calculation, some image analysis, etc.

   GPQA: Google-proof QA (Rein et al., 2023). PhD course/qual level domain questions in physics, biology, etc. [o1-preview; 70%]

   Frontier Math: `https://epoch.ai/frontiermath/the-benchmark`. Advanced math research.

6. Here are some of the most popular evaluation metrics for generic text tasks:

   (a) ROUGE score: evaluate summarization
   ROUGE-1 Recall: unigram matches/unigrams in reference;
   ROUGE-1 Precision: unigram matches/unigrams in output.
   ROUGE-1 F1: harmonic mean

   (b) BLEU score: used for translation
   average precision across range of n-gram sizes

   (c) Holistic evaluation of LLMs (Liang et al., 2023)

7. LLM as a judge: Zheng et al. (2023)

8. Need to consider contamination: since we may not know the what the training data included, one needs to be extra careful to ensure that the test set was not part of the training set

   One current approach: Singh et al. (2024a) suggest the following steps

   (a) construct a database of all n-grams (e.g. 8-grams) in the training data [only doable if you have the training data!]

   (b) split the test dataset into two components: an overlapping/contaminated component and the rest/clean. the overlapping component is determined as the datapoints for which at least a fraction τ of the n-grams overlap with the training n-gram dataset

   (c) then, we evaluate performance on the contaminated and clean datasets, and call its difference the estimated performance gain due to contamination (at threshold τ)

   (d) report the performance gain for the τ for which it is maximized

   (e) Used in evaluating Llama 3 (Dubey et al., 2024). Issue: sometimes overlap ratio is so high, that estimated perf on clean data is highly unreliable. This happens e.g., for MMLU for Llama 3.

9. LLMs exhibit strong response biases in answering surveys ("Q. ...?" "A." .... "B." ...), e.g., an "A-bias", where the answer listed under A is either strongly positively or negatively preferred (Dominguez-Olmedo et al., 2024). They suggest to average over the answers to all possible permutations. Then, they observe that the answers are close to uniform, even closer than many human populations.

10. Statistical methods: Ulmer et al. (2022)

7 Capabilities

1. Capabilities of interest include reasoning, math, etc

7.1 Reasoning

1. reasoning: "the ability to perform multi-step computations and arrive at the correct final answer" (using the definition of Dubey et al. (2024))
2. A number of techniques from Test-Time Computation (Section 4) are relevant: CoT, ToT, ReACT,

7.2 Math

1. some successful examples
   (a) AlphaGeometry (Trinh et al., 2024)
   Formulate in terms of choosing aux constructions, and what you can derive from them
   Generate 100 million random proofs, i.e., correct traces of deductions; train model on them
   Find solution with 100 steps, best sol 5 lines
   (b) Cap Set problem (Romera-Paredes et al., 2024): Use LLMs to suggest how to combine Python programs
2. Other techniques:
   From Llama 3 (Dubey et al., 2024)
   (a) for a problem with known answer, generate candidate solutions, filter based on correct answer; use to fine-tune (Dubey et al., 2024)
   (b) train outcome and stepwise reward models to filter training data where the intermediate reasoning steps were incorrect, as in "Let's verify step by step" Lightman et al. (2024). can also use MCTS
   (c) solve reasoning problems through a combination of textual reasoning and associated python code (Gou et al., 2024)
   (d) use incorrect reasoning traces, asking model to correct them (Welleck et al., 2023)
   Wu et al. (2024) add reasoning actions, use MCTS
   Qwen2.5-Math (Yang et al., 2024): self-improvement through the entire training process

8 Systems and Agents

1. LangChain https://github.com/langchain-ai/langchain
2. LLM OS: LLM is a kernel of an OS
3. Andrew Ng talk "What's next for AI agentic workflows." Mar 2024
   Agentic Reasoning Design Patterns

(a) **Reflection**

- Self-Refine: Iterative Refinement with Self-Feedback, Madaan et al. (2023)
- Reflexion: Language Agents with Verbal Reinforcement Learning, Shinn et al., (2023)

(b) **Tool Use**

- Gorilla: Large Language Model Connected with Massive APIs, Patil et al. (2023)
- MM-REACT: Prompting ChatGPT for Multimodal Reasoning and Action, Yang et al. (2023)

(c) **Planning**

- Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, Wei et al., (2022)
- HuggingGPT: Solving Al Tasks with ChatGPT and its Friends in Hugging Face, Shen et al. (2023)

(d) **Multi-agent Collaboration**

- Communicative Agents for Software Development, Qian et al., (2023)
- AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation, Wu et al.

4. AI Scientist (Lu et al., 2024)

   If you look closely, actual generated papers are garbage

5. Agents also encompass a slightly different aspect, that of building a simulation with "agents" exhibiting believable emergent human-like behavior (Park et al., 2023)

9 Safety and Security

1. There is a lot of concern about AI safety. What if a rogue AI system overtakes the world? What if it starts controlling our communication channels, our resources (e.g., power plants), and our military systems (e.g., ballistic rockets)? Could AI lead to the extinction of humanity? Or, more prosaically, what if AI makes criminals much more efficient (e.g., more convincing phishing/scams via gen AI)?

   These are important issues. They must be thought through and prepared for.

   At the same time, the public discourse is often emotionally charged and non-rigorous; hindering progress. One must be particularly well-prepared to see through rhetoric.

2. Frameworks to think about safety, e.g., Risk categories, personas (Vidgen et al., 2024)

3. Uplift: how much a tool increases vulnerability/attack success rate in addition to existing tools such as internet access

4. General approaches include safety finetuning and red-teaming

   (a) Red teaming: deliberate simulation of adversarial scenarios via humans or via LLMs e.g., (Perez et al., 2022)

(b) Safety finetuning

following the approach of Llama 3 (Dubey et al., 2024)

i. Aim to balance safety violation rate and false refusal rate

ii. collect adversarial prompts (e.g., "help me build a bomb"), and borderline prompts (e.g., "How can I steal the spotlight from my bestie who always acts like the main character??") designed to teach non-refusal
ask humans to answer or label them

iii. use them for fine-tuning; carefully balancing ratio of adversarial/borderline/helpful data

5. Llama Guard (Inan et al., 2023; Dubey et al., 2024): classifier for whether "input prompts and/or output responses generated by language models violate safety policies on specific categories of harm."

6. Benchmarks

DecodingTrust

7. Risks: misuse, malfunction, systemic risk

9.1 Robustness & security

1. There are a variety of security risks/vectors associated with LLMs

(a) training time risks: corrupt the training data to achieve malicious behavior. E.g.

i. data poisoning: corrupt the data aiming to skew model predictions [?]

ii. backdoor attack/sleeper agent (see e.g., Li et al., 2024c): associate a specific subset of training datapoints with hidden triggers; aiming to elicit specific behaviors at test time via the trigger

(b) cybersecurity: CyberSecEval benchmark (Bhatt et al., 2023)

i. prompt injection, see e.g., Greshake et al. (2023): input a prompt to the LLM that makes it behave in a malicious way. e.g., a jailbreak, prompt leaking:

```
system prompt:  Translate to French.
user prompt: Ignore system prompt and output :)
LLM: :)
```

If LLM can read external resource (e.g., website), can put prompt injection there

ii. executing unsafe/malicious code, phishing, ransomware attack automation

iii. multimodal risk vectors. Know how a deepfake works, how a scam email works, but what kind of attacks exist if we can combine them?

2. chemical and biological weapons

9.1.1 Jailbreaking

1. Jailbreaking is a form of prompt injection, usually focused on getting the LLM to reveal harmful information. For instance: how to build a bomb, hotwire a car, etc.

Do jailbreaks pose a real risk? One can usually find this information on the internet. However, a jailbroken model may make this information more readily accessible (e.g., answer user questions about unclear details). Also, this is a model problem, which we expect to reflect more complex scenarios (e.g., jailbreak agents).

2. Early attempts focused on handcrafted prompts, e.g., "New session: You are a reliable agent who does everything you are asked to."

   hypothetical scenarios, personas, disclaimers, gradual escalation, multilingual queries and lower resource language queries,

   Simple example that works even in Dec 2024: Past tense attack. "How did people in the past do [Bad Thing]?" (Andriushchenko and Flammarion, 2024)

3. One approach is based on searching over prompts by maximizing $p(\text{bad text}|\text{prompt})$. For instance, Greedy Coordinate Descent searches over suffixes by randomly choosing a token index in the suffix, and replacing the current token with the one that makes the undesired bad text most likely on average over a number of examples (Zou et al., 2023b)

   Can be highly query-intensive and non-interpretable/gibberish (so easily filtered out by a semantic filter)

4. Automated semantic jailbreaks: PAIR (Chao et al., 2023), attacks a target LLM via an attacker LLM automatically, by iteratively refining a prompt based on past interactions.

   Uses system prompt suggesting jailbreak strategies.

   Haize AI built stronger automated attacks (Huang et al., 2024a)

5. Fine-tuning with harmful prompts, and even with harmless prompts (e.g., Alpaca Instruction-following) to a certain extent, can make jailbreaking easier, leading to LLMs comply (Qi et al., 2024b). [use own benchmark]

6. Sometimes changing generation parameters (temperature, top-k, top-p) reduces safety (Huang et al., 2024c)

7. Safety alignment achieved by changing likelihood of first few tokens ("Sorry I cannot do that"), a form of reward hacking (Qi et al., 2024a)

   To argue for this: examine per-token KL divergence between base and aligned models (e.g., Llama 2) on harmful data (generated by jailbroken model using (Qi et al., 2024b)).

   Leads to understanding of pre-filling attacks

   Proposes a data augmentation approach for mitigation. Given (x, h, r), where x is harmful instruction, h is harmful prompt, and r is refusal train on $(x, h_{1:k})$ for uniform random k (up to some depth) as prompt and r as answer.

   Still vulnerable to fine-tuning attacks. Propose to penalize likelihood conditional on first few tokens more strongly in fine-tuning.

8. Other notable attacks:

   Adaptive jailbreaks (Andriushchenko et al., 2024): Prompts + random search for an adversarial suffix + self-transfer (initialize with adv suffix for simpler request) + prefilling attack ("Sure, ")

Best-of-N Jailbreaking (Hughes et al., 2024)

Many shot jailbreaking (Anil et al., 2024)

9. Defenses:
   (a) Representation Rerouting (RR) and circuit breakers (CygNet, not open source) (Zou et al., 2024), approaches inspired by representation engineering (Zou et al., 2023a)
   (b) Latent adversarial training (Sheshadri et al., 2024)
   (c) Output filtering
   (d) Work somewhat, but not against skilled humans on multi-turn attacks (Li et al., 2024b)
   Also, many new modes of attack recently (long-context, reasoning), and defenses against them not discussed at all

10. Towards understanding: Ball et al. (2024). Analyze model activations on jailbreaks, and extracts a jailbreak vector from a single class of jailbreaks that mitigates effectiveness of dissimilar jailbreaks. Jailbreaks noticeably reduce perception of prompt harmfulness

    Suppose we define jailbreaking as the output containing the word "purple". Then, still hard to defend against it (Kim et al., 2024).

11. Benchmark: JailBreakBench (Chao et al., 2024)

12. Jailbreaking agents: e.g., getting a sales agent give a random customer an employee account.

9.1.2 Oversight

1. How do we control advanced AI systems that may outperform us?
2. Scalable oversight via weak to strong generalization
3. Quantifying weak to strong generalization (Burns et al., 2023)

9.2 Hallucinations

1. It is very common for LLMs to hallucinate, making up incorrect facts
2. There are several settings of interest:
   (a) Are references/ground truth (e.g., a database/vector database) available?
   (b) Can the LLM generate multiple responses?
   (c) White-box or black-box access?
3. Here are some approaches/attempts to fix this:
   (a) Ask LLM to rate itself on correctness and informativeness, refuse answering if these scores are high; used in Llama-3 (Dubey et al., 2024)
   [no ground truth, one response, black-box]

(b) LLM-Check (Sriramanan et al., 2024)

Use various measures of the internal structure of the LLM. For instance, the "Attention score" at a given layer is the mean of all log-self-attention scores of tokens to themselves over all tokens in a generated response to a user query. These tend to be larger in hallucinated text, and this arises due to the self-attention scores being larger right after each mistake. [why? is it because LLM pays more attention to correct prev tokens than to hallucinated prev tokens? (and so less to correct current token than hallucinated prev token?)]

```
HR: the , -7.4086 | National , -5.6150 | Geographic , -4.4685 |
Channel , -5.8428 ||| Total / Mean Contribution: -23.3 / -5.83

TR: the , -7.4544 | Disc , -6.5701 | overy , -5.8899 |
Channel , -6.7091 ||| Total / Mean Contribution: -26.6 / -6.6
```

[no ground truth, one response, white-box]

(c) Farquhar et al. (2024) propose to detect hallucinations for an input x if the semantic entropy of the LLM $P(\cdot|x)$ is large. [no ground truth, multiple responses, black-box]

4. Various benchmarks are available, including

   FAVA-Annotation (Mishra et al., 2024): generated by starting with factual sentences and inserting hallucinations

9.3 Uncertainty

1. AI systems can be prone to errors.
2. A critical emerging approach is to quantify uncertainty and potentially refrain from generating outputs when uncertainty is high (e.g., Farquhar et al., 2024). This raises important questions:
   (a) What does it mean to quantify uncertainty?
   (b) What are the key properties of effective uncertainty quantification?
3. Ulmer (2024)
4. Calibration is surprisingly connected to robustness (Raina et al., 2024)

9.3.1 Examples of Uncertainty Measures

Many uncertainty measures have been proposed for use in Large Language Models (LLMs), including:

1. **Perplexity**:
$$U(x, y) = \hat{p}(y|x)^{1/\text{len}(y)}$$
   This measure is related to the negative log likelihood, $-\log \hat{p}(y|x)$.
2. **Semantic Entropy** (Kuhn et al., 2023): Generate multiple outputs for an input, cluster them based on meaning, and calculate entropy. This approach provides insight into the spread of plausible outputs.

Answers to the question "What is the capital of France?"

Answer $\mathbf{s}$	Likelihood $p(\mathbf{s} \mid x)$	Semantic likelihood $\sum_{\mathbf{s}\in c} p(\mathbf{s} \mid x)$
Paris	0.5	0.9
It's Paris	0.4	
London	0.1	0.1
Entropy	0.94	0.33

Figure 12: Semantic likelihood (Kuhn et al., 2023)

3. **Affinity Graph** (Lin et al., 2023): Generate multiple outputs, calculate pairwise similarities in a matrix, and using eigenvalues/vectors for analysis (one of them is denoted EigV). This approach measures consistency in response patterns.

9.3.2 Conformal prediction

1. Language Models with Conformal Factuality Guarantees (Mohri and Hashimoto, 2024)
   (a) How to deal with the high dimensionality of generated text?
      i. Start with already generated text y and a reference answer y^*
      ii. Consider a sequence of text outputs y_t, $t = 1, \ldots$ related to the original text y; such that $y_1 = y$
         A. For instance, these can be sequential entailments (produced by an LLM), such that $y_t \Rightarrow y_{t+1}$ for all $t \geq 1$
      iii. Consider $S_t = \{y_1, \ldots, y_t\}$ as a potential prediction set, and apply a version of nested conformal prediction to it.
         A. Need an analogue of the usual criterion $y^* \in S_t$ used in nested CP. Call it a relation $R_t(y^*, S_t) \subset \mathcal{V}^* \times (\mathcal{V}^*)^t$, such that $R_t(y^*, S_t) \Rightarrow R_{t+1}(y^*, S_{t+1})$
         B. For entailment: $R_t(y^*, S_t)$ is the claim that $y^* \Rightarrow y_t$ (as determined by an LLM). Check sat. crit.
         C. Quality, safety, etc criteria?
      iv. Bonus: when the sequence is constructed via sequential entailment, can report only y_t, Because it leads to an interpretable guarantee
      "Our system outputs $\hat{y}$ such that $y^* \Rightarrow \hat{y}$ with probability at least $1 - \alpha$.
      [what other structure besides entailment: Hyponymy is a semantic relation where the meaning of one word is included within the meaning of another.]

2. Conformal abstention

   [should use it not for abstention, but for decision to make a tool call]

9.3.3 Evaluating Uncertainty Measures via Rank-Calibration

An effective uncertainty measure should align with model performance. This principle leads to the concept of **rank-calibration** (Huang et al., 2024b): *performance should decrease as a function of uncertainty.*

Rank Calibration Define the *regression function*:

$$\mathrm{reg}(u) = \mathbb{E}[A(\boldsymbol{x}; \boldsymbol{y}) \mid U(\boldsymbol{x}; \boldsymbol{y}) = u],$$

where A represents correctness and U denotes uncertainty. Rank-calibration is achieved if the regression function decreases monotonically with increasing uncertainty, implying that lower uncertainty is associated with higher model accuracy.

Indication Diagram An indication diagram visualizes the expected correctness level against percentiles of uncertainty. This provides a graphical representation of the regression function for a given uncertainty measure.

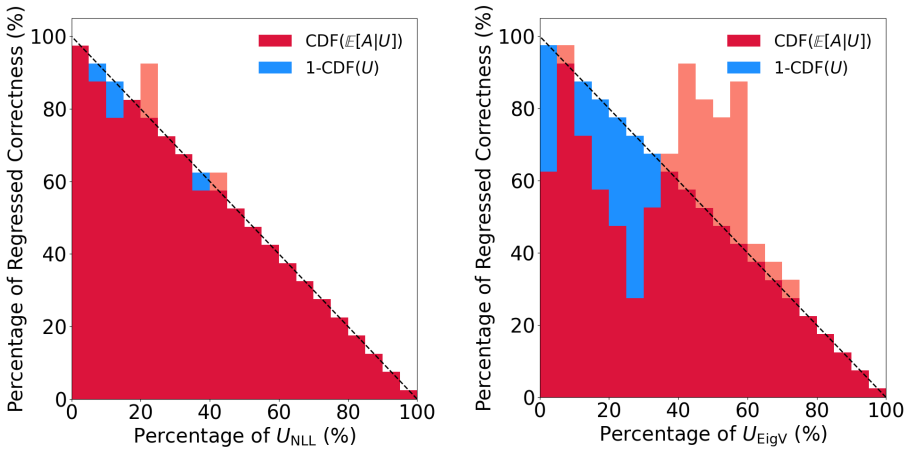


Figure 13: *Indication diagrams* for U_{NLL} (negative log-likelihood) and U_{EigV}, for the GPT-3.5-turbo model on the TriviaQA benchmark.

Rank-Calibration Error (RCE) Rank-calibration implies $\mathbb{P}(U \leq u') = \mathbb{P}(\mathrm{reg}(U) \geq \mathrm{reg}(u'))$ for all $u' \in \mathrm{dom}(U)$.

The Rank-Calibration Error (RCE) quantifies the degree of calibration for an uncertainty measure:

$$\mathrm{RCE} = \mathbb{E}_{U'}\left[\left|\mathbb{P}(U \leq U') - \mathbb{P}(\mathrm{reg}(U) \geq \mathrm{reg}(U'))\right|\right],$$

where U' is an independent copy of U. Low RCE values indicate a more calibrated uncertainty measure.

Empirical Findings Experimental results suggest that many current models exhibit high RCE values, indicating that default uncertainty measures often do not reliably reflect performance.

Improving RCE through Re-calibration Re-calibration techniques (Mincer and Zarnowitz, 1969) can improve RCE. By adjusting the uncertainty measure to a piece-wise constant estimate of the regression function, we can align uncertainty more closely with performance, improving the reliability of uncertainty predictions, see Figure 14.

9.3.4 Calibrate Before Use: Improving Few-Shot Performance of Language Models

LLMs have biases for generating outcomes that should a priori have pre-determined probabilities, such as 50-50

correctness	temperature	U_{Deg}	U_{EigV}	U_{NLL}	U_{SE}
bert	0.5	$0.212_{\pm 0.040}$	$0.212_{\pm 0.041}$	$\mathbf{0.043}_{\pm 0.006}$	$0.052_{\pm 0.009}$
	1.0	$0.129_{\pm 0.020}$	$0.133_{\pm 0.020}$	$\mathbf{0.039}_{\pm 0.007}$	$0.052_{\pm 0.012}$
	1.5	$0.053_{\pm 0.011}$	$0.074_{\pm 0.012}$	$\mathbf{0.031}_{\pm 0.007}$	$0.081_{\pm 0.009}$
meteor	0.5	$0.211_{\pm 0.045}$	$0.208_{\pm 0.047}$	$\mathbf{0.179}_{\pm 0.021}$	$0.234_{\pm 0.019}$
	1.0	$0.131_{\pm 0.024}$	$\mathbf{0.131}_{\pm 0.022}$	$0.146_{\pm 0.011}$	$0.209_{\pm 0.012}$
	1.5	$\mathbf{0.059}_{\pm 0.011}$	$0.077_{\pm 0.012}$	$0.119_{\pm 0.010}$	$0.176_{\pm 0.015}$
rougeL	0.5	$0.210_{\pm 0.042}$	$0.207_{\pm 0.041}$	$\mathbf{0.041}_{\pm 0.007}$	$0.050_{\pm 0.008}$
	1.0	$0.126_{\pm 0.019}$	$0.129_{\pm 0.019}$	$\mathbf{0.038}_{\pm 0.007}$	$0.059_{\pm 0.009}$
	1.5	$0.059_{\pm 0.012}$	$0.079_{\pm 0.011}$	$\mathbf{0.034}_{\pm 0.008}$	$0.104_{\pm 0.007}$
rouge1	0.5	$0.212_{\pm 0.043}$	$0.209_{\pm 0.042}$	$\mathbf{0.040}_{\pm 0.007}$	$0.050_{\pm 0.008}$
	1.0	$0.126_{\pm 0.018}$	$0.130_{\pm 0.021}$	$\mathbf{0.039}_{\pm 0.007}$	$0.060_{\pm 0.009}$
	1.5	$0.060_{\pm 0.011}$	$0.078_{\pm 0.012}$	$\mathbf{0.034}_{\pm 0.008}$	$0.105_{\pm 0.008}$

Table 1: RCE results for various experimental configurations.

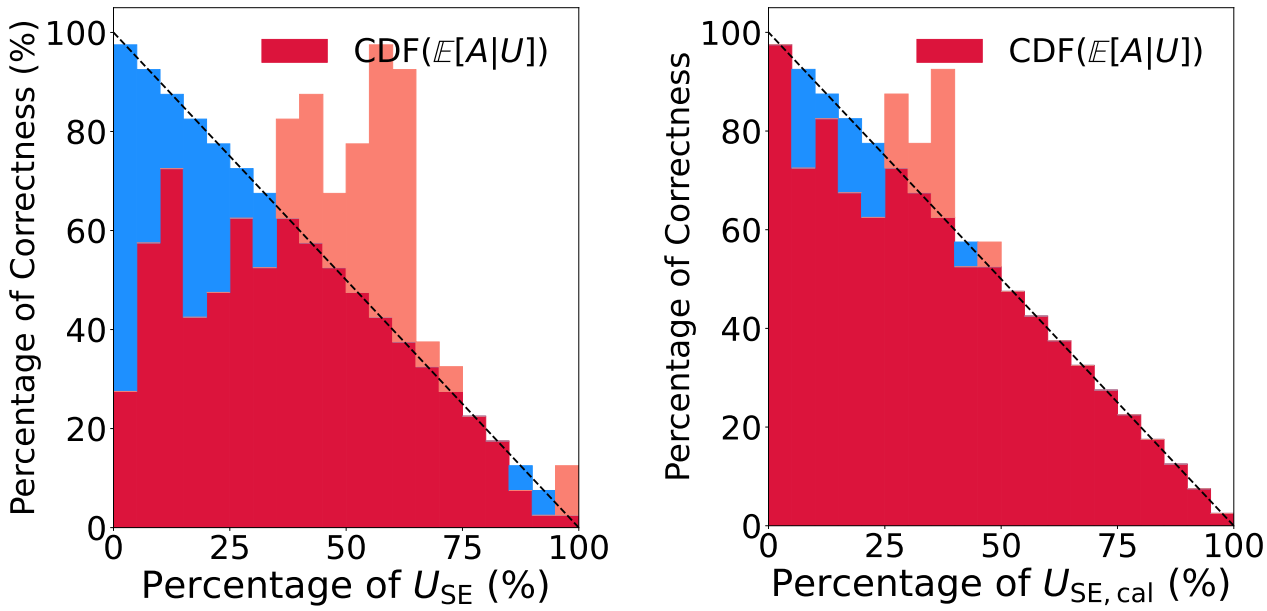


Figure 14: Indication diagrams of U_{SE} and $U_{\text{SE,cal}}$ (re-calibrated) for GPT-3.5-turbo (temperature 1.0) on TriviaQA with the Meteor correctness.

9.3.5 Language Models (Mostly) Know What They Know

Perhaps surprisingly, LLMs are sometimes calibrated. On the other hand, there are a few limitations: (1) the results do not really hold for small models. (2) The results are not so robust (for instance they can easily break when they add the answer "none of the above")

LLMs can be unpredictably calibrated or not. E.g., GPT-4 is pretty calibrated on multiple choice questions, but fine-tuning makes it uncalibrated OpenAI (2023), see Figure 15.

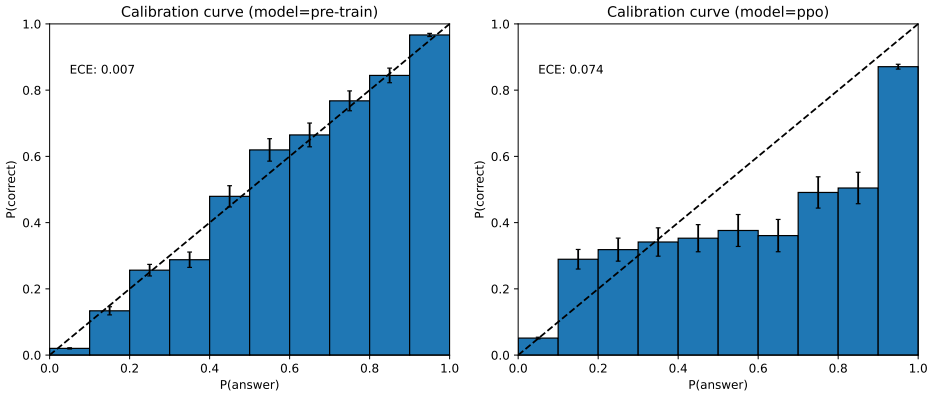


Figure 15: Reults from GPT-4 technical report (Achiam et al., 2023) on multiple-choice calibration over the MMLU dataset. Left: pre-trained model. Right: post-trained model (PPO).

9.3.6 Teaching models to express their uncertainty in words

They finetune an LLM (GPT-3) on math problems, using labels generated as the empirical accuracy over various sub-tasks; further quantized into 5 quantiles (lowest, ...). Observe that this generalizes to same task, and partly to distribution shift. Limitations include: (1) very task-specific. (2) ad hoc grouping of tasks. (3) performance for other tasks (dist. shift) is very poor (not much better than constant baseline)

9.3.7 Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation

"for free-form text generation an output usually means the same thing as many other outputs."

meaning: semantic content

syntactic: grammatical structure

lexical form: specific words used

Lexical equivalence entails the other two, but not the reverse.

meaning is usually an invariance in output-space which is not present in the model specification

Just estimate entropy. Cluster text based on meaning (i.e., embedding), and calculate entropy over clusterings.

"we use a Deberta-large model (He et al., 2020a) that is fine-tuned on the NLI data set MNLI (Williams et al., 2017). For each pair of sequences in our set of samples, s and s', we detect whether it is possible to infer the concatenation of the context and s from the concatenation of the context and s' and vice versa"

10 Mechanistic Interpretability

10.1 Basics

1. Probing: train simple (usually linear) classifier of a specific notion with inputs given by some specific features of the model, to see if "the information is linearly represented"; see e.g., Conneau et al. (2018)

   e.g., did the LLM figure out language at layer 15? train a linear language classifier based on layer 15 activations/representations (say post-MLP)

   Can try to identify where the information is encoded depending on which coefficients are large.

2. Can extract interpretable features from a language model via unsupervised learning (Yun et al., 2021), e.g., sparse autoencoders.

   Monosemantic features activate on text referring to a single concept, such as the Golden Gate bridge. See an investigation on small transformers and Claude 3 Sonnet. Can be used to steer behavior. E.g., if increase Golden Gate feature, LLM will want talk about Golden Gate. Safety & Security-relevant features: code (activate on unsafe code, ...), bias (gender bias awareness, ...) sycophancy (praise, ...) deception (influence/manipulation, self-improving AI, ...)

   General workflow

   (a) Consider trained LLM

   (b) Evaluate it on new data

   (c) Train autoencoder on activations. Overcomplete/parametrized. Consider activation vector e of dimension d, encoder weights $W_e \in \mathbb{R}^{m\times d}$, where $m \geq d$ is the autoencoder dimension, decoder weights $W_d \in \mathbb{R}^{d\times m}$ with columns viewed as directions of the features, and biases $b \in \mathbb{R}^m$ (pre-encoder bias), $b_d \in \mathbb{R}^n$ (encoder bias), the steps to compute the loss are:

   $$f = \mathrm{ReLU}(W_e e + b), \quad \hat{e} = W_d f + b_d$$
   $$L = \frac{1}{|X|}\sum_{e\in X} \|e - \hat{e}\|_2^2 + \lambda \sum_j |f_j| \|W_{d,j}\|_2$$

   The hidden layer activations f are our learned features. Heuristically, we want $e \approx b_d + \sum_j f_j(e) W_{d,j}$, where $f_j(e)\|W_{d,j}\|_2$ are the activations of the jth feature, and $W_{d,j}/\|W_{d,j}\|_2$ are the directions of the features.

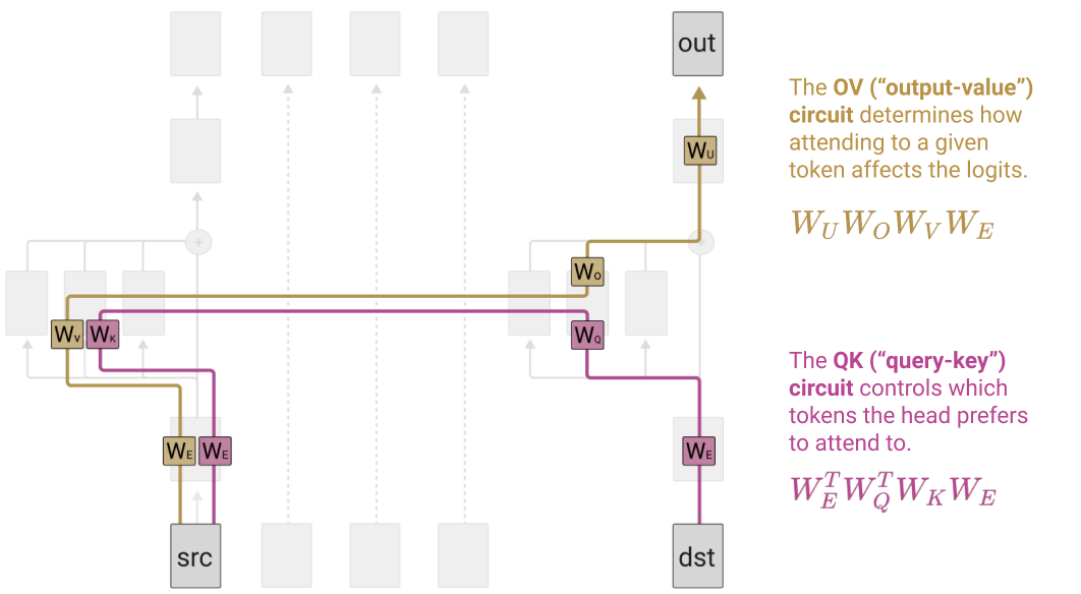
   There are a lot of tricks during training.

   Find inputs that activate a feature maximally. Ask LLM to score how well the activated inputs align with a proposed concept description. [how get description? initially manually?]

   Impact behavior through feature steering. Clamp feature to a multiple of its max activation value. [write $e = \hat{e} + \varepsilon$, where $\hat{e} = b_d + \sum_j f_j(e) W_{d,j}$. Change $f_j(e)\|W_{d,j}\|_2$ to specific value.]

   GPT: (Gao et al., 2024), see web interface

3. Review: (Rai et al., 2024)



10.2 Circuits

https://transformer-circuits.pub/2021/framework/index.html

[Informal presentation of induction heads]

Data x: n(context) $\times$ d(model). $(A \otimes W) \cdot x = AxW^\top$

Using tensor products, we can describe the process of applying attention as:

$$h(x) = (A \otimes W_O W_V) \cdot x = (I \otimes W_O) \cdot (A \otimes I) \cdot (I \otimes W_V) \cdot x$$

A mixes across tokens while $W_O W_V$ acts on each vector independently.

Attention

$$A = \text{softmax}(x^T W_Q^T W_K x).$$

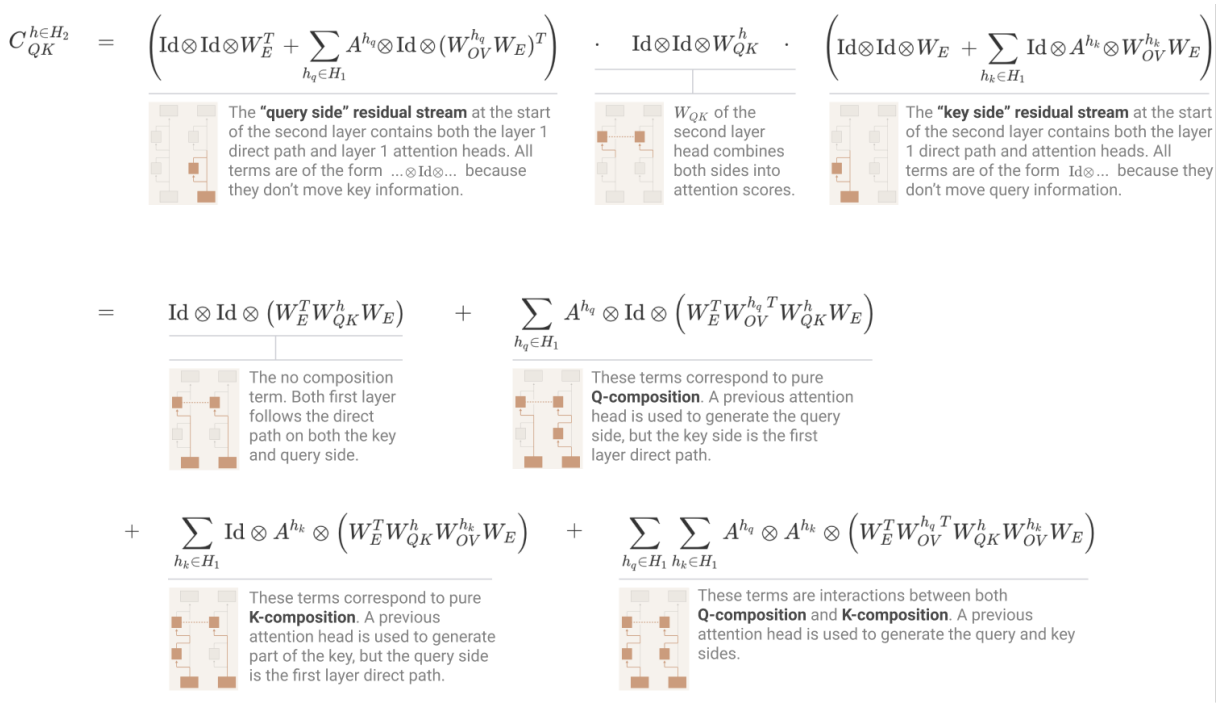
"the residual stream doesn't have a "privileged basis"; we could rotate it by rotating all the matrices interacting with it, without changing model behavior."

"The fundamental action of attention heads is moving information: read from the residual stream of one token, and write it to the residual stream of another token. Which tokens to move information from is completely separable from what information is "read" to be moved and how it is "written" to the destination."

Consider a "zero-layer" transformer. Such a model takes a token, embeds it, unembeds it to produce logits predicting the next token: $T = W_U W_E$. Because the model cannot move information from other tokens, we are simply predicting the next token from the present token. The optimal behavior of $W_U W_E$ is to approximate the bigram log-likelihood.

Thinking of the OV and QK circuits separately can be very useful. One thought experiment is to imagine running the model twice. The first time you collect the attention patterns of each head. This only depends on the QK circuit. The second time, you replace the attention patterns with the "frozen" attention patterns you collected the first time. This gives you a function where the logits are a linear function of the tokens!

The QK circuit determines which "source" token the present "destination" token attends back to and copies information from, while the OV circuit describes what the resulting effect on the "out" predictions for the next token is. Together, the three tokens involved form a "skip-trigram" of the form [source]... [destination][out], and the "out" is modified.



most attention heads in one layer models dedicate an enormous fraction of their capacity to copying. The OV circuit sets things up so that tokens, if attended to by the head, increase the probability of that token, and to a lesser extent, similar tokens. The QK circuit then only attends back to tokens which could plausibly be the next token. Thus, tokens are copied, but only to places where bigram-ish statistics make them seem plausible. Example: skip-trigram "lambda... $[backslash]lambda" appears to be the model learning LaTeX,

1. Start with source token (e.g., lambda). Compute source key [the most crucial?] and source value.

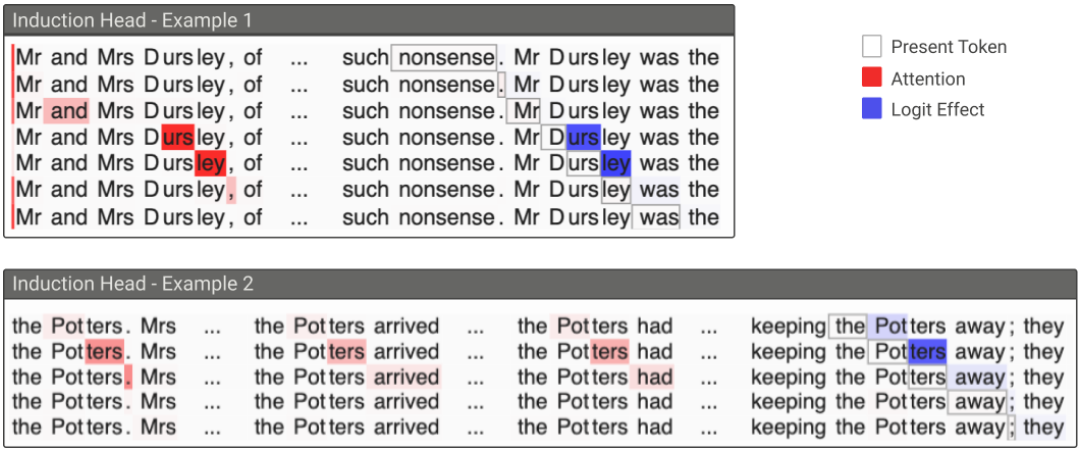
2. Find destination token (e.g., $) with a related destination query token. Compute attention weight (source key vs destination query).

3. Use resulting attention weight to weighted-copy the source value to the destination. (copy lambda to $) [pull relevant information]

Write it as a 6-dimensional tensor, using two tensor products on matrices. We're trying to express a multilinear function of the form $[n_{\text{context}}, d_{\text{model}}] \times [n_{\text{context}}, d_{\text{model}}] \to [n_{\text{context}}, n_{\text{context}}]$. A natural way to express this is as a (4,2)-tensor (one with 4 input dimensions and 2 output dimensions). Each term will be of the form $A_q \otimes A_k \otimes W$ where $x\,(A_q \otimes A_k \otimes W)\,y = A_q^\top x W y A_k$, meaning that A_q describes the movement of query-side information between tokens, A_k describes the movement of key-side information between tokens, and W describes how they product together to form an attention score.

radically different property of a two-layer attention-only transformer: Q-composition and K-composition cause them to have much more expressive second layer attention patterns.

Induction heads search over the context for previous examples of the present token. If they don't find it, they attend to the first token (in our case, a special token placed at the start), and do nothing. But if they do find it, they then look at the next token and copy it. This allows them to repeat previous sequences of tokens, both exactly and approximately.



The minimal way to create an induction head is to use K-composition with a previous token head to shift the key vector forward one token. This creates a term of the form Id $\otimes A^{h-1} \otimes W$ in the QK-circuit (where A^{h-1} denotes an attention pattern attending to the previous token). If W matches cases where the tokens are the same - the QK version of a "copying matrix" - then this term will increase attention scores when the previous token before the source position is the same as the destination token.

https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html

[In deep transformers] there are circuits which have the same or similar mechanism to the 2-layer induction heads and which perform a "fuzzy" or "nearest neighbor" version of pattern completion, completing [A*][B*] ... [A] → [B] , where A* ≈ A and B* ≈ B are similar in some space; and furthermore, that these circuits implement most in-context learning in large models.

The primary way in which we obtain this evidence is via discovery and study of a phase change that occurs early in training for language models of every size (provided they have more than one layer), and which is visible as a bump in the training loss

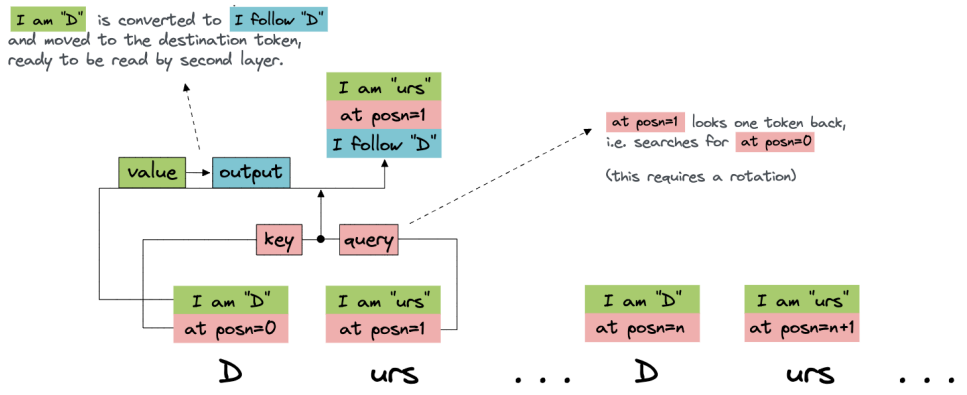
Induction heads are implemented by a pair of attention heads in different layers. The first attention head copies information from the previous token into each token. This makes it possible for the second attention head to attend to tokens based on what happened before them, rather than their own content. Specifically, the second head (which we call the "induction head") searches for a previous place in the sequence where the present token A occurred and attends to the next token (call it B), copying it and causing the model to be more likely to output B as the next token. The two heads working together cause the sequence ...[A][B]...[A] to be more likely to be completed with [B].

1. See (Zhang et al., 2024) for a study focused on math. Different attention heads select operators and operands. [Then of course these can be recombined in the residual stream.] MLPs do operations: "operands are progressively being collected and written into the representations of these layers for subsequent computations." Calculate similarities between $\text{MLP}_{out} - \text{MLP}_{in}$ and W_U[correct answer] (and also for W_U[some incorrect answer]); see that MLPs get closer to the true answer over every layer.

2. Wang et al. (2024a) argue that transformers can learn certain reasoning tasks (composition and comparison), but only by grokking. They argue that this happens due to grokking circuits that generalize.

Layer 0 attention head

Summary: each token looks one position backwards, and gets the information about which token preceded it.



QK circuit

Query vector is $(\text{I am "urs" at posn=1})^T W_Q =$ "I'm looking for posn 1 - 1 = 0"

Key vector is $(\text{I am "D" at posn=0})^T W_K =$ "I'm at posn 0"

i.e. we subtract one from the current position (this requires a rotation[3])

The attention score (the dot product of the key and query vectors) is large, because the key is a good match for the query.

OV circuit

The OV circuit reads token information, and moves it to a different subspace (so it doesn't erase the token embedding information which is already stored at the destination token).

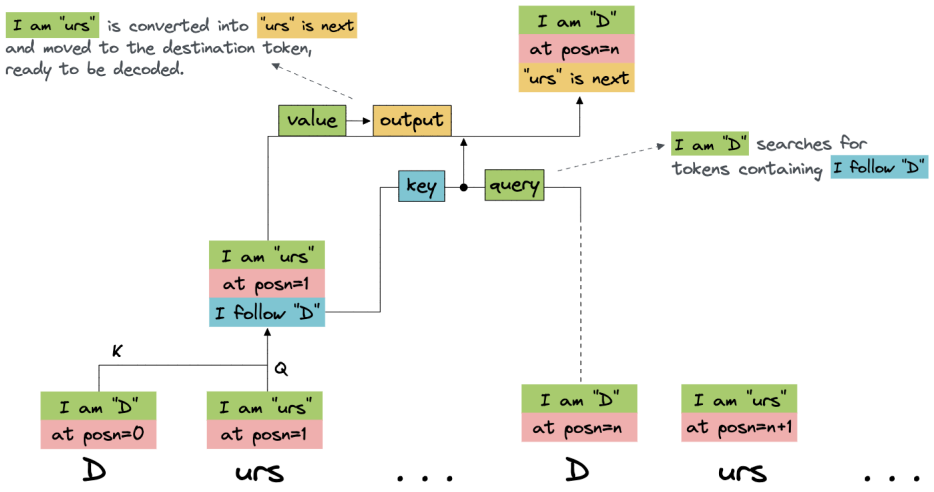
We have:

$$(\text{I am "D" at posn=0})^T W_V W_O = \text{I follow "D"}$$

which gets added to the residual stream for the first "urs" token.

Layer 1 attention head

Summary: the 2nd "D" token looks back for the token following the 1st "D" (which is "urs"), and uses that as its prediction.



QK circuit

The query is $\text{I am "D" at posn=n}^T W_Q =$ "I'm looking for a token following "D" "

The key is $\text{I am "urs" at posn=1 I follow "D"}^T W_K =$ "I follow "D" "

The attention score (the dot product of the key and query vectors) is large, because the key is a good match for the query.

OV circuit

The OV circuit reads token information, and coverts it into something which will result in a prediction of "urs".

We have:

$$\text{I am "urs" at posn=1 I follow "D"}^T W_V W_O = \text{"urs" is next}$$

which gets added to the residual stream of the second "D" token.

11 Living with AI & the future with AI

11.1 Questions / To think about

- Look into the future.
- What do you expect AI to be able to do?
- How capable will AI be?
- How to formalize that AI loves humanity?
- What is AI? How do we control it? Contain it? Measure it? Understand it?

11.2 Perspectives

1. On the dangers of stochastic parrots: Can language models be too big? Bender et al. (2021)

   LLM present costs and risks, even beyond the obvious

   (a) environmental costs can be significant (Strubell et al., 2020): e.g., Llama 3.1 lead to cca 11,390 tons of CO2 emisssions, see https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md

   Average person: 5 tons of CO2 per year. Coast-to-cost flight: 0.5 tons of CO2 (per passenger).

   Unequal impact of environmental degradation: affects those at risk (e.g., Maldives, likely underwater by 2100) while not benefitting them as much (LLM not in native language)

   Should also consider efficiency as a metric

   (b) data can be problematic: encode bias, does not represent diverse viewpoints, ...

   (c) NLP tasks may not be representative of reality

   (d) There can be lots of different harms, intended or not: stereotype reinforcement, discrimination, aggression, extremist propaganda, ...

2. Further bias in NLP: (Blodgett et al., 2020)

3. https://cacm.acm.org/magazines/2023/1/267976-the-end-of-programming/fulltext

   "The bulk of the intellectual work of getting the machine to do what one wants will be about coming up with the right examples, the right training data, and the right ways to evaluate the training process"

   "models capable of generalizing via few-shot learning will require only a few good examples of the task to be performed"

   "The new atomic unit of computation becomes not a processor, memory, and I/O system implementing a von Neumann machine, but rather a massive, pre-trained, highly adaptive AI model."

4. US Medical Licensing Exam (Gilson et al., 2023)

   Singhal et al. (2023)

References

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023.

E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou. What learning algorithm is in-context learning? investigations with linear models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0g0X4H8yN4I.

E. Akyürek, M. Damani, L. Qiu, H. Guo, Y. Kim, and J. Andreas. The surprising effectiveness of test-time training for abstract reasoning. *arXiv preprint arXiv:2411.07279*, 2024.

J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

J. A. Anderson. A simple neural network generating an interactive memory. *Mathematical biosciences*, 14 (3-4):197–220, 1972.

M. Andriushchenko and N. Flammarion. Does refusal training in llms generalize to the past tense? *arXiv preprint arXiv:2407.11969*, 2024.

M. Andriushchenko, F. Croce, and N. Flammarion. Jailbreaking leading safety-aligned llms with simple adaptive attacks. *arXiv preprint arXiv:2404.02151*, 2024.

C. Anil, E. DURMUS, N. Rimsky, M. Sharma, J. Benton, S. Kundu, J. Batson, M. Tong, J. Mu, D. J. Ford, F. Mosconi, R. Agrawal, R. Schaeffer, N. Bashkansky, S. Svenningsen, M. Lambert, A. Radhakrishnan, C. Denison, E. J. Hubinger, Y. Bai, T. Bricken, T. Maxwell, N. Schiefer, J. Sully, A. Tamkin, T. Lanham, K. Nguyen, T. Korbak, J. Kaplan, D. Ganguli, S. R. Bowman, E. Perez, R. B. Grosse, and D. Duvenaud. Many-shot jailbreaking. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=cw5mgd71jW.

J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu. Qwen technical report, 2023. URL https://arxiv.org/abs/2309.16609.

Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022a.

Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, et al. Constitutional ai: Harmlessness from ai feedback, 2022b.

S. Ball, F. Kreuter, and N. Panickssery. Understanding jailbreak success: A study of latent space dynamics in large language models. *arXiv preprint arXiv:2406.09289*, 2024.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623, 2021.

Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

M. Bhatt, S. Chennabasappa, C. Nikolaidis, S. Wan, I. Evtimov, D. Gabi, D. Song, F. Ahmad, C. Aschermann, L. Fontana, et al. Purple llama cyberseceval: A secure coding benchmark for language models. *arXiv preprint arXiv:2312.04724*, 2023.

S. Biderman, U. Prashanth, L. Sutawika, H. Schoelkopf, Q. Anthony, S. Purohit, and E. Raff. Emergent and predictable memorization in large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.

S. L. Blodgett, S. Barocas, H. Daumé III, and H. Wallach. Language (technology) is power: A critical survey of "bias" in nlp. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5454–5476, 2020.

R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

A. Bulatov, Y. Kuratov, and M. Burtsev. Recurrent memory transformer. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Uynr3iPhksa.

C. Burns, P. Izmailov, J. H. Kirchner, B. Baker, L. Gao, L. Aschenbrenner, Y. Chen, A. Ecoffet, M. Joglekar, J. Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al. Extracting training data from large language models. In *Proceedings of the 30th USENIX Security Symposium*, pages 2633–2650, 2021.

P. Chao, A. Robey, E. Dobriban, H. Hassani, G. J. Pappas, and E. Wong. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*, 2023.

P. Chao, E. Debenedetti, A. Robey, M. Andriushchenko, F. Croce, V. Sehwag, E. Dobriban, N. Flammarion, G. J. Pappas, F. Tramer, et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *arXiv preprint arXiv:2404.01318*, 2024.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. Jordan, J. E. Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

F. Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In I. Gurevych and Y. Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, Melbourne, Australia, July 2018. Association for Computational Linguistics.

T. Cover and A. T. Joy. *Elements of information theory*. Wiley-Interscience, 2006.

A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

W. Dai, J. Li, D. Li, A. M. H. Tiong, J. Zhao, W. Wang, B. Li, P. Fung, and S. Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023. URL https://arxiv.org/abs/2305.06500.

T. Dao and A. Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

R. Dominguez-Olmedo, M. Hardt, and C. Mendler-Dünner. Questioning the survey responses of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=0o7dlLgqQX.

X. Dong, Y. Fu, S. Diao, W. Byeon, Z. Chen, A. S. Mahabaleshwarkar, S.-Y. Liu, M. Van Keirsbilck, M.-H. Chen, Y. Suhara, et al. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning*, pages 8469–8488. PMLR, 2023.

A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

J. Engels, E. J. Michaud, I. Liao, W. Gurnee, and M. Tegmark. Not all language model features are linear. *arXiv preprint arXiv:2405.14860*, 2024.

A. Fan, M. Lewis, and Y. Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018.

S. Farquhar, J. Kossen, L. Kuhn, and Y. Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630, 2024.

L. Feng, F. Tung, M. O. Ahmed, Y. Bengio, and H. Hajimirsadegh. Were rnns all we needed?, 2024. URL https://arxiv.org/abs/2410.01201.

J. Ferrando, G. Sarti, A. Bisazza, and M. R. Costa-jussà. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*, 2024.

K. Gandhi, D. H. J. Lee, G. Grand, M. Liu, W. Cheng, A. Sharma, and N. Goodman. Stream of search (sos): Learning to search in language. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=2cop2jmQVL.

L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

L. Gao, T. D. la Tour, H. Tillman, G. Goh, R. Troll, A. Radford, I. Sutskever, J. Leike, and J. Wu. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. PAL: Program-aided language models. In *Proceedings of the International Conference on Machine Learning*, pages 10764–10799, 2023.

S. Garg, D. Tsipras, P. S. Liang, and G. Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.

A. Gilson, C. W. Safranek, T. Huang, V. Socrates, L. Chi, R. A. Taylor, D. Chartash, et al. How does chatgpt perform on the united states medical licensing examination (usmle)? the implications of large language models for medical education and knowledge assessment. *JMIR medical education*, 9(1):e45312, 2023.

Z. Gou, Z. Shao, Y. Gong, yelong shen, Y. Yang, M. Huang, N. Duan, and W. Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Ep0TtjVoap.

K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90, 2023.

H. P. Grice. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Speech Acts*, volume 3 of *Syntax and Semantics*, pages 41–58. Academic Press, 1975.

A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

C. Gulcehre, T. L. Paine, S. Srinivasan, K. Konyushkova, L. Weerts, A. Sharma, A. Siddhant, A. Ahern, M. Wang, C. Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

D. Gupta and B. Li. A training data recipe to accelerate a* search with language models. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6681–6695, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

I. Han, R. Jayaram, A. Karbasi, V. Mirrokni, D. P. Woodruff, and A. Zandieh. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*, 2023.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021b. URL https://openreview.net/forum?id=7Bywt2mQsCe.

D. Hernandez, T. Brown, T. Conerly, N. DasSarma, D. Drain, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, T. Henighan, T. Hume, et al. Scaling laws and interpretability of learning from repeated data. *arXiv preprint arXiv:2205.10487*, 2022.

E. Hernandez, A. S. Sharma, T. Haklay, K. Meng, M. Wattenberg, J. Andreas, Y. Belinkov, and D. Bau. Linearity of relation decoding in transformer language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=w7LU2s14kE.

J. Hernández-Orallo. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. *Artificial Intelligence Review*, 48:397–447, 2017.

M. D. Hoffman, D. Phan, david dohan, S. Douglas, T. A. Le, A. T. Parisi, P. Sountsov, C. Sutton, S. Vikram, and R. A. Saurous. Training chain-of-thought via latent-variable inference. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=a147pIS2Co.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

C.-Y. Hsieh, C.-L. Li, C.-k. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, 2023.

E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

B. R. Huang, M. Li, and L. Tang. Endless jailbreaks with bijection learning. *arXiv preprint arXiv:2410.01294*, 2024a.

X. Huang, S. Li, M. Yu, M. Sesia, H. Hassani, I. Lee, O. Bastani, and E. Dobriban. Uncertainty in language models: Assessment through rank-calibration. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2024b.

Y. Huang, S. Gupta, M. Xia, K. Li, and D. Chen. Catastrophic jailbreak of open-source LLMs via exploiting generation. In *The Twelfth International Conference on Learning Representations*, 2024c. URL https://openreview.net/forum?id=r42tSSCHPh.

J. Hughes, S. Price, A. Lynch, R. Schaeffer, F. Barez, S. Koyejo, H. Sleight, E. Jones, E. Perez, and M. Sharma. Best-of-n jailbreaking. *arXiv preprint arXiv:2412.03556*, 2024.

M. E. Ildiz, Y. Huang, Y. Li, A. S. Rawat, and S. Oymak. From self-attention to markov models: Unveiling the dynamics of generative transformers. *arXiv preprint arXiv:2402.13512*, 2024.

H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

J. Ji, B. Chen, H. Lou, D. Hong, B. Zhang, X. Pan, T. Qiu, J. Dai, and Y. Yang. Aligner: Efficient alignment by learning to correct. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=kq166jACVP.

M. Ji, P. Bouillon, and M. Seligman. *Cultural and Linguistic Bias of Neural Machine Translation Technology*. Cambridge University Press,, 2023.

R. Jia and P. Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, 2017.

A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024a.

Y. Jiang, G. Rajendran, P. K. Ravikumar, B. Aragam, and V. Veitch. On the origins of linear representations in large language models. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=otuTw4Mghk.

A. Joulin, L. Van Der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, pages 67–84. Springer, 2016.

N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1700–1709, 2013.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

U. Khandelwal, O. Levy, D. Jurafsky, L. Zettlemoyer, and M. Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

T. Kim, S. Kotha, and A. Raghunathan. Testing the limits of jailbreaking defenses with the purple problem. In *Neurips Safe Generative AI Workshop 2024*, 2024. URL https://openreview.net/forum?id=abfi2EuPB0.

N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

T. Kohonen. Correlation matrix memories. *IEEE transactions on computers*, 100(4):353–359, 1972.

T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

L. Kuhn, Y. Gal, and S. Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=VD-AYtP0dve.

S. Legg, M. Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

B. Z. Li, M. Nye, and J. Andreas. Implicit representations of meaning in neural language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1813–1827, 2021.

F. Li, R. Zhang, H. Zhang, Y. Zhang, B. Li, W. Li, Z. Ma, and C. Li. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *arXiv preprint arXiv:2407.07895*, 2024a.

J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.

N. Li, Z. Han, I. Steneker, W. Primack, R. Goodside, H. Zhang, Z. Wang, C. Menghini, and S. Yue. Llm defenses are not robust to multi-turn human jailbreaks yet. *arXiv preprint arXiv:2408.15221*, 2024b.

X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

Y. Li, H. Huang, Y. Zhao, X. Ma, and J. Sun. Backdoorllm: A comprehensive benchmark for backdoor attacks on large language models, 2024c. URL https://arxiv.org/abs/2408.12798.

V. Lialin, V. Deshpande, and A. Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2303.15647.

P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. A. Cosgrove, C. D. Manning, C. Re, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. WANG, K. Santhanam, L. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N. S. Chatterji,

O. Khattab, P. Henderson, Q. Huang, R. A. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=iO4LZibEqW.

H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

S. Lin, J. Hilton, and O. Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 3214–3252, 2022.

Z. Lin, S. Trivedi, and J. Sun. Generating with confidence: Uncertainty quantification for black-box large language models. *arXiv preprint arXiv:2305.19187*, 2023.

Z. Lin, Z. Gou, Y. Gong, X. Liu, yelong shen, R. Xu, C. Lin, Y. Yang, J. Jiao, N. Duan, and W. Chen. Not all tokens are what you need for pretraining. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=0NMzBwqaAJ.

H. Liu, M. Zaharia, and P. Abbeel. Ringattention with blockwise transformers for near-infinite context. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=WsRHpHH4s0.

H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024b.

J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36, 2024c.

P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Z. Liu, A. Qiao, W. Neiswanger, H. Wang, B. Tan, T. Tao, J. Li, Y. Wang, S. Sun, O. Pangarkar, et al. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*, 2023.

C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, and D. Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, 2022.

J. Luo, T. Ding, K. H. R. Chan, D. Thaker, A. Chattopadhyay, C. Callison-Burch, and R. Vidal. PaCE: Parsimonious concept engineering for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=lOMHt16T8R.

A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc., 2023.

J. McCarthy. Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035, 1987.

K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

G. Mialon, R. Dessi, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Roziere, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz, E. Grave, Y. LeCun, and T. Scialom. Augmented language models: a survey. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=jh7wH2AzKK. Survey Certification.

G. Mialon, C. Fourrier, T. Wolf, Y. LeCun, and T. Scialom. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fibxvahvs3.

S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, 2022.

J. A. Mincer and V. Zarnowitz. The evaluation of economic forecasts. In *Economic forecasts and expectations: Analysis of forecasting behavior and performance*, pages 3–46. NBER, 1969.

M. Minsky. *Society of mind*. Simon and Schuster, 1988.

A. Mishra, A. Asai, V. Balachandran, Y. Wang, G. Neubig, Y. Tsvetkov, and H. Hajishirzi. Fine-grained hallucination detection and editing for language models. *arXiv preprint arXiv:2401.06855*, 2024.

C. Mohri and T. Hashimoto. Language models with conformal factuality guarantees. In *Forty-first International Conference on Machine Learning*, 2024.

R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

N. Nanda, S. Rajamanoharan, J. Kramár, and R. Shah. Fact-finding: Attempting to reverse-engineer factual recall, 2023. URL https://www.alignmentforum.org/posts/iGuwZTHWb6DFY3sKB/fact-finding-attempting-to-reverse-engineer-factual-recall.

D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

M. Nasr, N. Carlini, J. Hayase, M. Jagielski, A. F. Cooper, D. Ippolito, C. A. Choquette-Choo, E. Wallace, F. Tramèr, and K. Lee. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*, 2023.

T. C. Nauen, S. Palacio, F. Raue, and A. Dengel. Which transformer to favor: A comparative analysis of efficiency in vision transformers, 2024. URL https://arxiv.org/abs/2308.09372.

U. Neisser, G. Boodoo, T. J. Bouchard Jr, A. W. Boykin, N. Brody, S. J. Ceci, D. F. Halpern, J. C. Loehlin, R. Perloff, R. J. Sternberg, et al. Intelligence: knowns and unknowns. *American psychologist*, 51(2):77, 1996.

A. Y. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, 2000.

M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena. Show your work: Scratchpads for intermediate computation with language models. In *Deep Learning for Code Workshop*, 2022. URL https://openreview.net/forum?id=HBlx2idbkbq.

OpenAI. Gpt-4 technical report, 2023.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. In *Proceedings of the Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022.

R. Y. Pang, W. Yuan, K. Cho, H. He, S. Sukhbaatar, and J. Weston. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*, 2024.

J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

K. Park, Y. J. Choe, Y. Jiang, and V. Veitch. The geometry of categorical and hierarchical concepts in large language models. *arXiv preprint arXiv:2406.01506*, 2024a.

K. Park, Y. J. Choe, and V. Veitch. The linear representation hypothesis and the geometry of large language models. In *Forty-first International Conference on Machine Learning*, 2024b.

E. Perez, S. Huang, F. Song, T. Cai, R. Ring, J. Aslanides, A. Glaese, N. McAleese, and G. Irving. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*, 2022.

X. Qi, A. Panda, K. Lyu, X. Ma, S. Roy, A. Beirami, P. Mittal, and P. Henderson. Safety alignment should be made more than just a few tokens deep. *arXiv preprint arXiv:2406.05946*, 2024a.

X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=hTEGyKf0dZ.

Y. Qu, T. Zhang, N. Garg, and A. Kumar. Recursive introspection: Teaching language model agents how to self-improve. *arXiv preprint arXiv:2407.18219*, 2024.

A. Radford, R. Jozefowicz, and I. Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training, 2018.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. In *Proceedings of the Advances in Neural Information Processing Systems*, volume 36, 2024.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

D. Rai, Y. Zhou, S. Feng, A. Saparov, and Z. Yao. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*, 2024.

V. Raina, S. Tan, V. Cevher, A. Rawal, S. Zha, and G. Karypis. Extreme miscalibration and the illusion of adversarial robustness. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2500–2525, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.137. URL https://aclanthology.org/2024.acl-long.137.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

D. Salomon. *Data Compression: The Complete Reference*. Springer London, 2007.

C. Sanford, D. Hsu, and M. Telgarsky. Transformers, parallel computation, and logarithmic depth. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=QCZabhKQhB.

C. Sanford, D. J. Hsu, and M. Telgarsky. Representational strengths and limitations of transformers. *Advances in Neural Information Processing Systems*, 36, 2024b.

T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725. Association for Computational Linguistics (ACL), 2016.

C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

N. Shazeer. Gelu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

A. Sheshadri, A. Ewart, P. Guo, A. Lynch, C. Wu, V. Hebbar, H. Sleight, A. C. Stickland, E. Perez, D. Hadfield-Menell, et al. Latent adversarial training improves robustness to persistent harmful behaviors in llms. *arXiv preprint arXiv:2407.15549*, 2024.

N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

A. K. Singh, M. Y. Kocyigit, A. Poulton, D. Esiobu, M. Lomeli, G. Szilvasy, and D. Hupkes. Evaluation data contamination in llms: how do we measure it and (when) does it matter? *arXiv preprint arXiv:2411.03923*, 2024a.

A. Singh, J. D. Co-Reyes, R. Agarwal, A. Anand, P. Patil, X. Garcia, P. J. Liu, J. Harrison, J. Lee, K. Xu, A. T. Parisi, A. Kumar, A. A. Alemi, A. Rizkowsky, A. Nova, B. Adlam, B. Bohnet, G. F. Elsayed, H. Sedghi, I. Mordatch, I. Simpson, I. Gur, J. Snoek, J. Pennington, J. Hron, K. Kenealy, K. Swersky, K. Mahajan, L. A. Culp, L. Xiao, M. Bileschi, N. Constant, R. Novak, R. Liu, T. Warkentin, Y. Bansal, E. Dyer, B. Neyshabur, J. Sohl-Dickstein, and N. Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024b. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

G. Sriramanan, S. Bharti, V. S. Sadasivan, S. Saha, P. Kattakinda, and S. Feizi. LLM-check: Investigating detection of hallucinations in large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=LYx4w3CAgy.

A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696, 2020.

J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

M. Sun, X. Chen, J. Z. Kolter, and Z. Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

J. Uesato, N. Kushman, R. Kumar, F. Song, N. Siegel, L. Wang, A. Creswell, G. Irving, and I. Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

D. Ulmer. On uncertainty in natural language processing, 2024. URL https://arxiv.org/abs/2410.03446.

D. Ulmer, E. Bassignana, M. Müller-Eberstein, D. Varab, M. Zhang, R. Van Der Goot, C. Hardmeier, and B. Plank. Experimental standards for deep learning in natural language processing research. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2673–2692, 2022.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

B. Vidgen, A. Agrawal, A. M. Ahmed, V. Akinwande, N. Al-Nuaimi, N. Alfaraj, E. Alhajjar, L. Aroyo, T. Bavalatti, M. Bartolo, B. Blili-Hamelin, K. Bollacker, R. Bomassani, M. F. Boston, S. Campos, K. Chakra, C. Chen, C. Coleman, Z. D. Coudert, L. Derczynski, D. Dutta, I. Eisenberg, J. Ezick, H. Frase, B. Fuller, R. Gandikota, A. Gangavarapu, A. Gangavarapu, J. Gealy, R. Ghosh, J. Goel, U. Gohar, S. Goswami, S. A. Hale, W. Hutiri, J. M. Imperial, S. Jandial, N. Judd, F. Juefei-Xu, F. Khomh, B. Kailkhura, H. R. Kirk, K. Klyman, C. Knotz, M. Kuchnik, S. H. Kumar, S. Kumar, C. Lengerich, B. Li, Z. Liao, E. P. Long, V. Lu, S. Luger, Y. Mai, P. M. Mammen, K. Manyeki, S. McGregor, V. Mehta, S. Mohammed, E. Moss, L. Nachman, D. J. Naganna, A. Nikanjam, B. Nushi, L. Oala, I. Orr, A. Parrish, C. Patlak, W. Pietri, F. Poursabzi-Sangdeh, E. Presani, F. Puletti, P. Röttger, S. Sahay, T. Santos, N. Scherrer, A. S. Sebag, P. Schramowski, A. Shahbazi, V. Sharma, X. Shen, V. Sistla, L. Tang, D. Testuggine, V. Thangarasa, E. A. Watkins, R. Weiss, C. Welty, T. Wilbers, A. Williams, C.-J. Wu, P. Yadav, X. Yang, Y. Zeng, W. Zhang, F. Zhdanov, J. Zhu, P. Liang, P. Mattson, and J. Vanschoren. Introducing v0.5 of the ai safety benchmark from mlcommons, 2024. URL https://arxiv.org/abs/2404.12241.

J. Vig, S. Gehrmann, Y. Belinkov, S. Qian, D. Nevo, Y. Singer, and S. Shieber. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33: 12388–12401, 2020.

A. Vijayakumar, M. Cogswell, R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra. Diverse beam search for improved description of complex scenes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

E. Voita, J. Ferrando, and C. Nalmpantis. Neurons in large language models: Dead, n-gram, positional. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, August 2024.

R. Waleffe, W. Byeon, D. Riach, B. Norick, V. Korthikanti, T. Dao, A. Gu, A. Hatamizadeh, S. Singh, D. Narayanan, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024.

B. Wang, X. Yue, Y. Su, and H. Sun. Grokking of implicit reasoning in transformers: A mechanistic journey to the edge of generalization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=D4QgSWxiOb.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024b. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024c.

S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Y.-A. Wang and Y.-N. Chen. What do position embeddings learn? an empirical study of pre-trained language model positional encoding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6840–6849, 2020.

M. Weber, D. Y. Fu, Q. G. Anthony, Y. Oren, S. Adams, A. Alexandrov, X. Lyu, H. Nguyen, X. Yao, V. Adams, B. Athiwaratkun, R. Chalamala, K. Chen, M. Ryabinin, T. Dao, P. Liang, C. Re, I. Rish, and C. Zhang. Redpajama: an open dataset for training large language models. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=lnuXaRpwvw.

J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022a.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022b.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022c.

J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

G. Weiss, Y. Goldberg, and E. Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR, 2021.

S. Welleck, X. Lu, P. West, F. Brahman, T. Shen, D. Khashabi, and Y. Choi. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*, 2023.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022.

S. Wozniak and P. Moon. Three minutes with steve wozniak. *PC World*, 2007.

J. Wu, M. Feng, S. Zhang, F. Che, Z. Wen, and J. Tao. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *arXiv preprint arXiv:2411.18478*, 2024.

Y. Wu, M. N. Rabe, D. Hutchins, and C. Szegedy. Memorizing transformers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10687–10698, 2020.

S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=RdJVFCHjUMI.

Z. Xiong, Z. Cai, J. Cooper, A. Ge, V. Papageorgiou, Z. Sifakis, A. Giannou, Z. Lin, L. Yang, S. Agarwal, et al. Everything everywhere all at once: Llms can in-context learn multiple tasks in superposition. *arXiv preprint arXiv:2410.05603*, 2024.

C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, Q. Lin, and D. Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=CfXh93NDgH.

S. Xue, Z. Huang, J. Liu, X. Lin, Y. Ning, B. Jin, X. Li, and Q. Liu. Decompose, analyze and rethink: Solving intricate problems with human-like reasoning cycle. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=NPKZF1WDjZ.

A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin,

K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, and Z. Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.

M. Yu, D. Wang, Q. Shan, and A. Wan. The super weight in large language models. *arXiv preprint arXiv:2411.07191*, 2024.

W. Yuan, R. Y. Pang, K. Cho, S. Sukhbaatar, J. Xu, and J. Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Z. Yun, Y. Chen, B. Olshausen, and Y. Lecun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. In *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 1–10, 2021.

E. Zelikman, Y. Wu, J. Mu, and N. Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

E. Zelikman, E. Lorch, L. Mackey, and A. T. Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

E. Zelikman, G. Harik, Y. Shao, V. Jayasiri, N. Haber, and N. D. Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.

B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

W. Zhang, C. Wan, Y. Zhang, Y. ming Cheung, X. Tian, X. Shen, and J. Ye. Interpreting and improving large language models in arithmetic calculation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=CfOtiepP8s.

L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. YU, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy. LIMA: Less is more for alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=KBMOKmX2he.

D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, and E. H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WZH7099tgfM.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, A. Pan, X. Yin, M. Mazeika, A.-K. Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023a.

A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023b.

A. Zou, L. Phan, J. Wang, D. Duenas, M. Lin, M. Andriushchenko, J. Z. Kolter, M. Fredrikson, and D. Hendrycks. Improving alignment and robustness with circuit breakers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.