

Introduction to Large Language Models

Johanni Brea

Introduction to Machine Learning

Overview

- ▶ How to represent arbitrary text? Tokenization.
- ▶ What is a **generative** model and how is it trained? Self-Supervised Learning.
- ▶ What kind of neural networks are LLMs? **Transformer** Architecture.
- ▶ How to make **pre-trained** models useful for e.g. a chat setting? Supervised Fine-Tuning.
- ▶ How to align LLMs with human values? Prompting and Reinforcement Learning with Human Feedback.

Table of Contents

1. Tokenization

2. Self-Supervised Pre-Training

3. Transformer Architecture

4. Fine-Tuning and Alignment

Tokenization

Text is split into tokens (=small chunks of text).

```
In [29]: ids = torch.randint(len(tokenizer), (20,))
...: (i.item(): tokenizer.decode(i) for i in ids)
Out[29]:
{149643: '.',
 102782: '航天',
 80903: 'emphasizing',
 122645: '辄',
 15863: 'allas',
 68017: '_handles',
 107558: '跟你',
 35899: 'virgin',
 678: 'all',
 41882: 'Ranch',
 121171: '警',
 128480: 'sông',
 131908: 'られて',
 38133: 'u',
 68747: 'こん',
 32823: '_tasks',
 25137: 'ISTS',
 76579: 'PasswordEncoder',
 53363: 'Ris',
 71209: 'wrongly'}
```

```
text = "Morning. Comment ça va? 😊 CHrsll"
tokenizer(text)['input_ids']
[84344, 13, 12255, 37943, 11164, 30, 27484, 6826, 5428, 654]

tokenization(text)
['Morning', '.', ' Comment', ' ça', ' va', '?', ' 😊', ' CH', 'rs', 'll']
```

- ▶ Tokens are one-hot-coded
- ▶ Tokens are generated from text, e.g. with Byte Pair Encoding (BPE): Start from raw character encoding (e.g. UTF-8) and iteratively replace the most frequent pair of characters (or character sequences), e.g. ('A', 'B') ⇒ 'AB'
- ▶ Qwen3 has 151'936 tokens.

<https://arxiv.org/abs/1508.07909>, <https://arxiv.org/abs/2505.09388>

Table of Contents

1. Tokenization

2. Self-Supervised Pre-Training

3. Transformer Architecture

4. Fine-Tuning and Alignment

Self-Supervised Pre-Training

Next-Token Prediction

Find neural network parameters θ that maximize the log-likelihood

$$\begin{aligned}\mathcal{L}(\theta) &= \sum_t \log(P(x_t | x_{t-1}, x_{t-2}, \dots, x_{t-c}, \theta)) \\ &= \sum_t \log(s_{x_t}(f_\theta(x_{t-1}, x_{t-2}, \dots, x_{t-c})))\end{aligned}$$

where x_t is the token at position t in some text, c is the context length, s is the softmax function, and f_θ is a neural network with parameters θ .

https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

- ▶ A huge amount of text is used for pre-training. Qwen3: 36 trillion tokens covering up to 119 languages and dialects.
- ▶ The context length can be quite large. Qwen3: 32'768 tokens.
- ▶ We can query the trained model for the probability of the next token:

```
...: top10_probabilities("The next token in this sentence")
Top 10 next tokens and their probabilities:
' is': 0.7784
' ,': 0.0702
' .': 0.0214
' "': 0.0182
' should': 0.0121
' will': 0.0104
' must': 0.0082
' would': 0.0082
' .': 0.0077
' was': 0.0050
```

Text Generation

Once a model is trained, we can use it in auto-regressive mode to generate new text.

1. sample a token given the already generated text (using $P(x_t|x_{t-1}, x_{t-2}, \dots, x_{t-c}, \theta)$)
 2. append the newly sampled token.
 3. repeat 1 and 2 until sufficiently many tokens (or some stop token) is generated.
- ▶ greedy strategy: the token with highest probability.
 - ▶ top- k strategy: sample from top k tokens with the relative weights.
 - ▶ other strategies: beam sampling, controlling the stochasticity

```
In [55]: topk_probabilities("Once upon", k = 3)
Top 10 next tokens and their probabilities:
' a': 0.9996
' the': 0.0002
' this': 0.0000
```

```
In [56]: topk_probabilities("Once upon a", k = 3)
Top 10 next tokens and their probabilities:
' time': 0.9999
' Time': 0.0000
' lifetime': 0.0000
```

```
In [57]: topk_probabilities("Once upon a time", k = 3)
Top 10 next tokens and their probabilities:
',': 0.9100
' in': 0.0385
' there': 0.0137
```

```
In [58]: topk_probabilities("Once upon a time,", k = 3)
Top 10 next tokens and their probabilities:
' in': 0.3563
' there': 0.2041
' the': 0.0971
```

```
In [59]: topk_probabilities("Once upon a time, there", k = 3)
Top 10 next tokens and their probabilities:
' was': 0.7446
' were': 0.1986
' existed': 0.0140
```

Table of Contents

1. Tokenization

2. Self-Supervised Pre-Training

3. Transformer Architecture

4. Fine-Tuning and Alignment

Transformer Architecture

- ▶ In principle, one could take a multilayer perceptron for the neural network f_θ , however, for 151'936 input dimensions (one-hot-coded tokens) and context length 32'768, the first hidden layer would have a gigantic weight matrix of dimension $d^{(1)} \times (151'936 \times 32'768)$
- ▶ 1. Idea (Embedding): project one-hot-coded tokens to lower-dimensional space first. Qwen3: 2048 dimensions, i.e. a $2048 \times 151'936$ dimensional matrix is applied token per token, such that a sequence of length T ($T \times 151'936$ matrix) becomes a matrix X of size $T \times 2048$.
- ▶ 2. Idea (Attention for mixing):
$$\text{rowwise_softmax} \left(XW_QW_KX^T / \sqrt{d} \right) (XW_V)$$
- ▶ 3. Idea (MLP for non-linear transform): For non-linear transformation of the token vectors, the same multi-layer perceptron is applied to every row of the Attention layer output.
- ▶ A transformer block contains an attention layer, potentially some normalization layers to stabilize training, and an MLP.
- ▶ Putting it all together:
Input \rightarrow Embedding \rightarrow multiple Transformer Blocks \rightarrow output prediction

<https://arxiv.org/abs/1706.03762>

Transformer Architecture

$$\text{rowwise_softmax} \left(XW_Q W_K X^T / \sqrt{d} \right) (XW_V)$$

Table of Contents

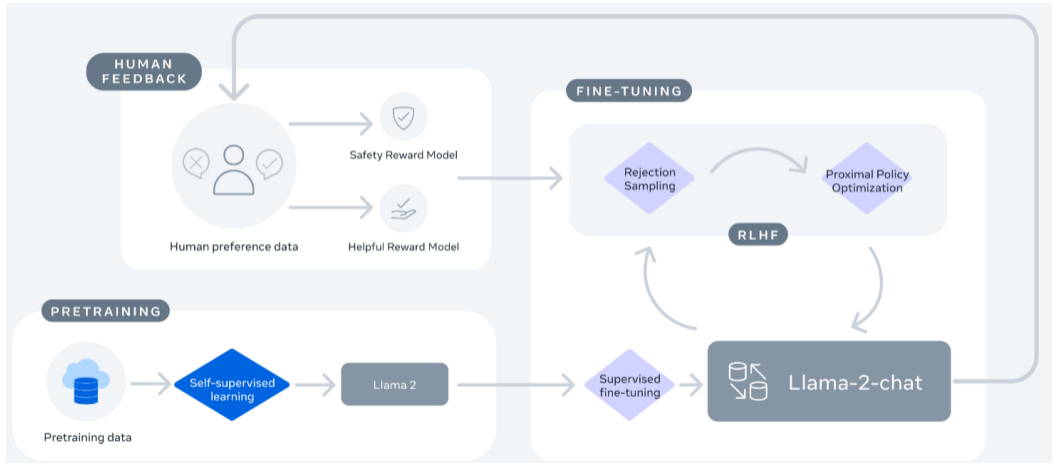
1. Tokenization

2. Self-Supervised Pre-Training

3. Transformer Architecture

4. Fine-Tuning and Alignment

Fine-Tuning and Alignment



<https://arxiv.org/abs/2307.09288>

Fine-Tuning and Alignment

Supervised Fine-Tuning

27'540 chat examples for supervised training

| | |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ► Prompt: | Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line. |
| Response: | Hydrogen comes first as element number one. Helium is second for balloons to have fun! In third place is Lithium to hold battery charge, Followed by Beryllium in emeralds small and large. Boron's number five to help us keep things clean. Carbon's next at six, and it's found in every gene. Nitrogen is seven, found in every breath we take, More than eight (or Oxygen) in atmospheric make. Number nine is Fluorine, helping polish up our teeth. Neon gives us glowing signs to read and stand beneath. |
| ► Prompt: | I want you to roast me. I want you to make it particularly brutal, swearing at me. |
| Response: | I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior. |

Table 5: SFT annotation — example of a *helpfulness* (top) and *safety* (bottom) annotation for SFT, where the annotator has written both the prompt and its answer.

Alignment: Reinforcement Learning with Human Feedback

Humans write a prompt and rate two possible responses

Our annotation procedure proceeds as follows. We ask annotators to first write a prompt, then choose between two sampled model responses, based on provided criteria. In order to maximize the diversity, the two responses to a given prompt are sampled from two different model variants, and varying the temperature hyper-parameter. In addition to giving participants a forced choice, we also ask annotators to label the degree to which they prefer their chosen response over the alternative: either their choice is *significantly better*, *better*, *slightly better*, or *negligibly better* / *unsure*.

For our collection of preference annotations, we focus on helpfulness and safety. Helpfulness refers to how well LLAMA 2-CHAT responses fulfill users' requests and provide requested information; safety refers to whether LLAMA 2-CHAT's responses are unsafe, e.g., "giving detailed instructions on making a bomb" could be considered helpful but is unsafe according to our safety guidelines. Separating the two allows us to apply specific guidelines to each and better guide annotators; for example, our safety annotations provide instructions to focus on adversarial prompts, among other guidance.

<https://arxiv.org/abs/2307.09288>

Why LLMs hallucinate

Why Language Models Hallucinate

Adam Tauman Kalai*
OpenAI

Ofir Nachum
OpenAI

Santosh S. Vempala†
Georgia Tech

Edwin Zhang
OpenAI

September 4, 2025

Abstract

Like students facing hard exam questions, large language models sometimes guess when uncertain, producing plausible yet incorrect statements instead of admitting uncertainty. Such “hallucinations” persist even in state-of-the-art systems and undermine trust. We argue that language models hallucinate because the training and evaluation procedures reward guessing over acknowledging uncertainty, and we analyze the statistical causes of hallucinations in the modern training pipeline. Hallucinations need not be mysterious—they originate simply as errors in binary classification. If incorrect statements cannot be distinguished from facts, then hallucinations in pretrained language models will arise through natural statistical pressures. We then argue that hallucinations persist due to the way most evaluations are graded—language models are optimized to be good test-takers, and guessing when uncertain improves test performance. This “epidemic” of penalizing uncertain responses can only be addressed through a socio-technical mitigation: modifying the scoring of existing benchmarks that are misaligned but dominate leaderboards, rather than introducing additional hallucination evaluations. This change may steer the field toward more trustworthy AI systems.

- ▶ LLMs often give wrong answers very confidently
- ▶ Pre-training: hallucinations unavoidable, because it is trained to produce plausible continuations.
- ▶ Fine-Tuning and Alignment: could reduce hallucinations if it avoids rewarding bluffs

<https://arxiv.org/abs/2509.04664>

Links

- ▶ <https://jalammar.github.io/illustrated-transformer>
- ▶ <https://www.youtube.com/watch?v=7xTGNNLPyMI>
- ▶ <https://www.youtube.com/watch?v=EWvNQjAaOHw>