Autoregressive MODELS

Origins · Mathematics · Architecture · Applications

Natural Language Understanding

CSCI 4907/6515

Lecture 10, April 14th, 2026

Aya Zirikly

From Econometrics to the AI Revolution

The core concept of Autoregression models (AR)

- Autoregressive (AR) models are the foundation of time series forecasting. They operate on a simple principle: **The best predictor of the future is the recent past.** An AR model forecasts future values by performing a regression (linear prediction) on its own previous data points.
- Imagine you are trying to predict today's temperature without a satellite. You might simply look at what the temperature was yesterday and the day before. The AR model formalizes this "lookback."

  Key Use Case: Forecasting data that exhibits strong self-correlation over time, such as weather, stock prices (in the short term), or seasonal product demand.

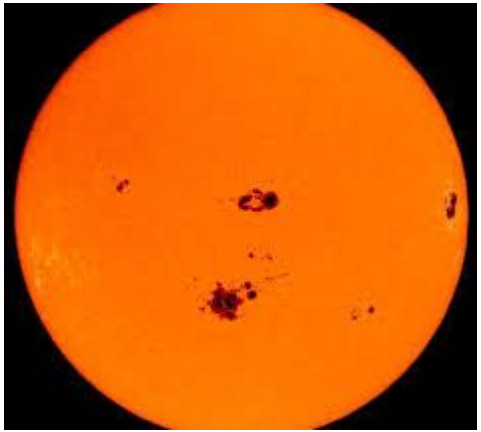
History of prediction time

The Origins (Early 20th Century): The quest was driven by economics and astronomy. Scientists wanted a formal way to analyze cyclical data like sunspot activity and business cycles.

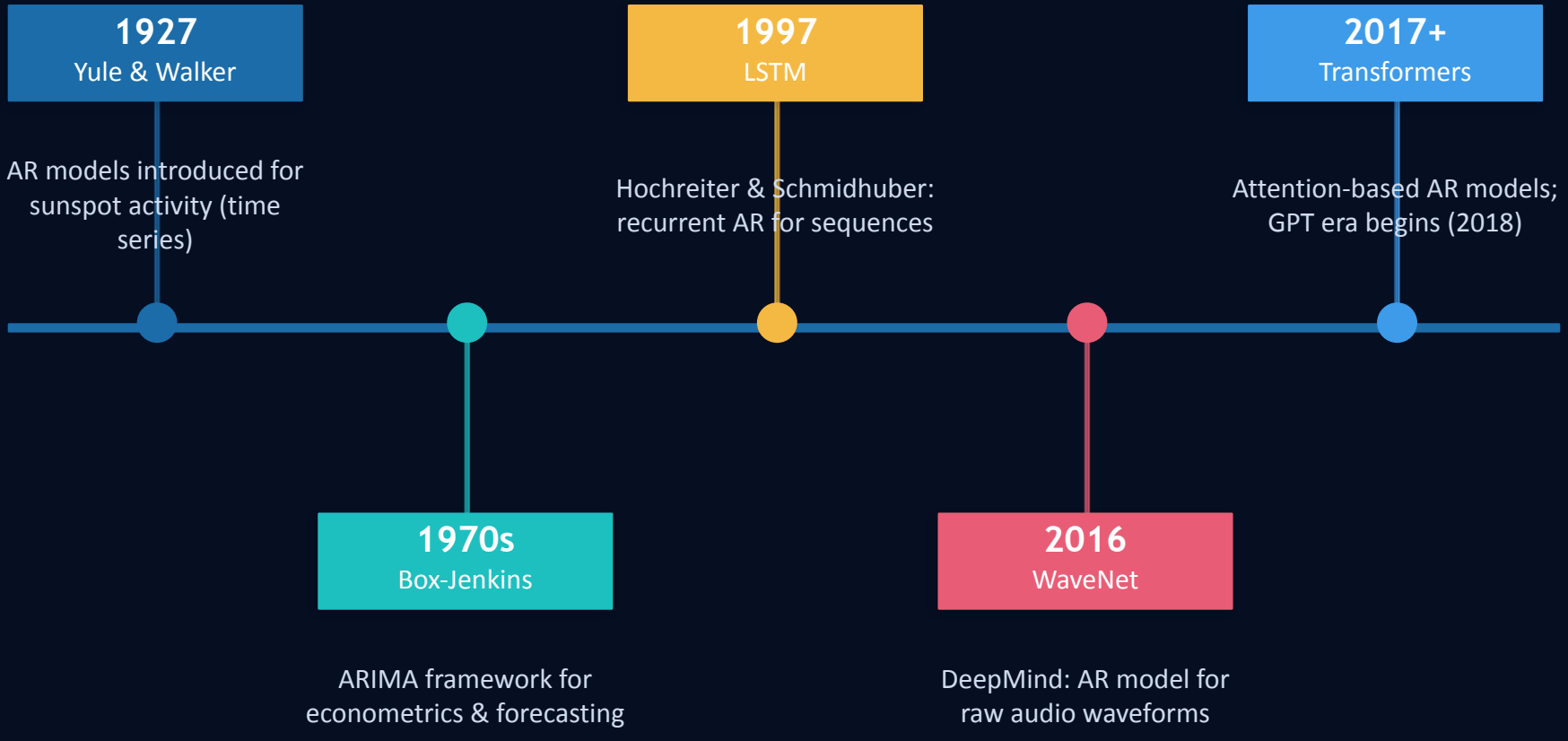
The Major Breakthroughs (1920s-1930s):

- **Udny Yule (1927):** He introduced the fundamental idea of autoregression while modeling sunspots.
- **Sir Gilbert Walker (1931):** He developed the recursive method for estimating AR parameters (now called the Yule-Walker equations).

The Formalization (1970s): The field was revolutionized by the publication of **"Time Series Analysis: Forecasting and Control"** by George Box and Gwilym Jenkins. They integrated AR models into the unified "Box-Jenkins method," which formalized the ARIMA(p, d, q) workflow.



HISTORY & ORIGINS



ARCHITECTURE TYPES

AR(p) / ARIMA

1927 – present

- Linear combination of p lags
- ARIMA adds differencing (I) and moving avg (MA)
- Stationary time series assumption
- Interpretable coefficients

Use: Finance, econometrics, forecasting

RNN / LSTM

1986 – 2017

- Hidden state carries sequence memory
- LSTM gates control what to remember/forget
- Sequential: slow to train
- Gradient vanishing problem (solved by LSTM)

Use: NLP, speech, music generation

Transformer (GPT)

2017 – present

- Self-attention over all prior tokens
- Parallelizable training (vs. sequential RNNs)
- Scales to billions of parameters
- Causal mask enforces AR property

Use: LLMs, code gen

What is ARIMA: Autoregressive Integrated Moving Average

Statistical model that predicts future data points by analyzing patterns, trends, and errors in past data.

Components

- **AR (Autoregressive p):** Looks at past *values* to predict the future. (e.g., "Yesterday's high sales suggest today's will be high.")
- **I (Integrated d):** Removes underlying trends to stabilize the data. (e.g., "Instead of absolute sales, let's look at the *change* in sales day-to-day.")
- **MA (Moving Average q):** Looks at past forecast *errors* to course-correct. (e.g., "I over-predicted yesterday by 10 units; I will adjust today's prediction down.")

Best Used For:

Forecasting real-world, messy data over time—such as stock market prices, seasonal retail demand, or economic indicators—where you need to account for past history, steady trends, and random noise all at once.

Foundation of AR

The mathematical core of the AR model is the concept of a "weighted linear combination" of past observations.

Understanding the Equation for an AR(p) model:

- **y(t):** current value we are trying to forecast (at time "t").
- **y(t-1), y(t-2):** previous observations (the **lags**).
- **Weights (phi):** multipliers assigned to each lag. They are calculated during the model training process to minimize prediction error.
- **c:** A constant baseline or intercept (dependency of errors)
- **Error:** White noise (random, unforecastable error).

The Role of Model Order (p): The parameter **p** defines the complexity and **lookback window** of the model.

- **AR(1):** The model only looks at *yesterday* to predict today. [y(t) = c + (Weight 1 * y(t-1)) + Error]
- **AR(2):** The model looks back two time steps. [y(t) = c + (Weight 1 * y(t-1)) + (Weight 2 * y(t-2)) + Error]

Historical Limitations and Challenges

Before advanced computational techniques and deep learning, AR models faced significant mathematical hurdles that limited their scope.

The Strict Requirement of Stationarity: A basic AR model can only read data that is **stationary**, meaning its statistics (mean, variance) do not change over time.

- *Limitation:* Most real-world data is **non-stationary** (has upward trends or cyclical change). This required differencing math to fix before the AR model could function.

Linearity: Classical AR models assume that the relationship between past values and future values is **linear**.

- *Limitation:* They failed to capture sudden spikes, erratic shifts, or complex interactions within data (e.g., predicting the 1987 stock market crash).

Poor Long-Term Forecasts: AR models are designed for **one-step-ahead** forecasting.

- *Limitation:* When used to predict far into the future, AR forecasts rapidly lose accuracy and stabilize to a mean value, failing to map long-range dependencies.

Stationary vs. non stationary data

Stationary Data

Data is stationary if its statistical properties—specifically its average (mean) and its variance stay the same over time. It doesn't have an upward trend, a downward trend, or seasonal cycles. It is essentially stable and predictable.

- **A Constant Hum:** The audio waveform of an air conditioner running at a steady speed.
- **Factory Quality Control:** The number of defective screws produced per hour on a perfectly calibrated, stable assembly line (it might fluctuate between 1 and 3, but it never trends upward to 50).
- **Coin Flips:** If you flip a coin 100 times a day and record the number of "heads," the daily average will constantly hover right around 50.

Non-stationary data

The data is evolving, which makes it much harder for classical models to predict without mathematical adjustments.

Trend (Changing Mean) A trend means the data is moving consistently upward or downward over a long period.

- **Example:** A patient's daily self-reported anxiety scores on a scale of 1-10. If the patient starts a successful new medication and Cognitive Behavioral Therapy (CBT), their daily scores might fluctuate, but the overall average will show a clear **downward trend** over six months.

Seasonality (Repeating Cycles) Seasonality means the data has predictable, repeating patterns tied to specific timeframes.

- **Example: Seasonal Affective Disorder (SAD).** If you track a patient's depression inventory scores over five years, the data will likely show a distinct, repeating wave. The scores (and symptom severity) will consistently spike during the darker, colder winter months and drop back down to a baseline during the summer.

Non-stationary data

Changing Variance (Volatility) Changing variance means the "spread" or the extreme highs and lows of the data get wider or narrower over time.

- **Example:** Mood charting for a patient with **Bipolar Disorder**. During a stable period, their daily mood might fluctuate very slightly (low variance). However, during the onset of a manic or depressive episode, their mood scores will begin swinging wildly from extremely high to extremely low. The sudden widening of these emotional swings is a classic example of changing variance.

Why this matters for data modeling: If you fed the Seasonal Affective Disorder data into a basic, unadjusted Autoregressive (AR) model in the middle of summer, the model would assume the low depression scores were a permanent baseline (because it expects stationary data). It would fail to predict the inevitable winter spike. You would need the "Integrated" part of an ARIMA model to account for that non-stationary cycle.

How is non-stationary handled?

- A pure Autoregressive (AR) model actually *didn't* resolved the issue of non-stationary data. If you feed an upward or downward trend directly into an AR model, the math essentially breaks.
- Instead, statisticians resolved the issue by inventing a "preprocessing" step that happens right before the AR model starts. This step is the **"I" (Integrated)** part of the ARIMA model, and the mathematical trick it uses is called **Differencing**.

How is non-stationary handled?

What is differencing: Predicting "Change" Instead of "Value"

Differencing is the act of subtracting the previous observation from the current observation. Instead of asking the model to predict the *actual absolute number*, you ask it to predict the *size of the jump* from one step to the next.

Imagine you are walking up a massive, endless staircase.

- **Non-Stationary Data:** Every time you take a step, your total distance from the ground gets higher and higher. This is a massive upward trend. An AR model cannot predict your altitude based on the last few steps because the baseline keeps changing.
- **Stationary Data:** Now, instead of looking at your total altitude, let's just look at the *height of each individual stair*. If every stair is 7 inches tall, the difference between Step 1 and Step 2 is 7 inches. The difference between Step 1,000 and Step 1,001 is still just 7 inches.

How is non-stationary handled?

By calculating the *difference* instead of the absolute altitude, that massive upward trend completely disappears. The data becomes a flat, stable, repeating line of "7 inches." The data is now **stationary**.

Let's go back to the patient whose anxiety scores are trending steadily downward over a few weeks because of a successful new therapy.

1. **The Raw Data (Non-Stationary):** Week 1 score is 90. Week 2 is 85. Week 3 is 82. Week 4 is 75. (This is a clear downward trend).
2. **The Differencing Step (The "I" in ARIMA):** The algorithm subtracts the weeks from each other. The change from Week 1 to 2 is **-5**. Week 2 to 3 is **-3**. Week 3 to 4 is **-7**.
3. **The AR Model Takes Over:** Now, the AR component doesn't have to figure out the complex downward slope of the overall anxiety. It is simply given the flattened list of differences (-5, -3, -7). Because these numbers hover around a stable average, the AR model can easily do its math to predict the *next week's shift*.

Once the AR model predicts that the next difference will be, say, -4, the system simply applies that -4 drop to the current score to give you the final prediction.

Summary

The Concept: Autoregressive models (the 'AR' in ARIMA) use a weighted linear combination of previous values to forecast the current value.

The Foundation: This is formalized using the lookback parameter **p** (for example, an AR(2) model looks back two steps).

The Evolution: While pioneered in the 1920s to model economic cycles, AR models became the standard framework for time series after George Box and Gwilym Jenkins formalized ARIMA in 1970.

The historical limitations of stationarity, linearity, and short memory forced researchers to look beyond classical statistics. This journey directly paved the way for modern **Recurrent Neural Networks (RNNs)** and **Transformer models** (like Claude/GPT/Gemini), which are, inherently, powerful and non-linear autoregressive models.

Solving the "Memory" Problem with Neural Networks

- Recurrent Neural Networks (RNNs): These were the first major modern AR models. They processed data sequentially, passing a "hidden state" (memory) from one step to the next.
- Variations like LSTMs (Long Short-Term Memory) allowed models to remember context from much earlier in a sequence, partially solving the "poor long-term forecast" limitation of classical models.

The transformer revolution

The Bottleneck: RNNs had to process data one step at a time, making them incredibly slow to train on large datasets.

The Transformer Solution: Google researchers introduced the Transformer architecture, which replaced sequential processing with a mechanism called Self-Attention.

- ★ Allows the model to look at every piece of past data simultaneously and decide which parts are the most important for predicting the next step. It supercharged autoregressive modeling.

Autoregression in Language Models

Next-Token Prediction: At their core, Large Language Models (LLMs) are just massive, highly advanced autoregressive models.

- ★ They take your prompt as the initial past data.
- ★ They predict the single most likely next word (or "token").
- ★ They add that new word to the context.
- ★ They repeat the process until the response is finished.

Instead of looking back at just 2 or 3 past values like a classical AR(2) model, an LLM might look back at 100,000 previous words to make its prediction.

How classical AR changed

Classical AR (Continuous Values): Older models predicted a specific, continuous number. (e.g., "Tomorrow's temperature will be 72.4 degrees.")

Modern AR (Discrete Tokens): Modern models, especially Large Language Models (LLMs), predict discrete "chunks" of information called tokens (words, pixels, or audio snippets).

Instead of predicting a final value, modern AR models output a **probability distribution** over an entire vocabulary.

Given the prompt "The sky is...", the model doesn't just calculate "blue." It calculates:

- Blue: 85%
- Cloudy: 10%
- Falling: 0.01% It then selects the most probable next token, adds it to the sequence, and runs the entire process again.

AR in Audio and Video

Audio Generation: Models like WaveNet or modern AI music generators use autoregression to predict the next audio waveform sample based on the previous seconds of sound.

Video Generation: While some video AI uses diffusion, many top-tier models use autoregression to predict the next frame of a video based on the sequence of previous frames.

The Universal Framework: If data can be arranged in a sequence (time, words, musical notes, pixels), an autoregressive model can be trained to generate it.

The Challenges of Generative AR Models

Compounding Errors (Hallucinations): Because AR models use their own predictions as input for the next step, a bad guess early on can derail the entire output.

Computational Cost: Running massive autoregressive models requires immense processing power (GPUs) because every single word or frame must be calculated sequentially.

Lack of True Planning: Autoregressive models predict step-by-step. They cannot easily "think ahead" to the end of a sentence before they start typing it.

The future of AR

- ❏ We evolved from predicting sunspots with basic math to generating human-level text with neural networks.
- ❏ The fundamental mechanism, using the past to predict the immediate future, is identical.

What is Next: Researchers are exploring ways to combine autoregression with other architectures (like Diffusion or state-space models) to make them faster, cheaper, and more accurate.

Math of modern AR

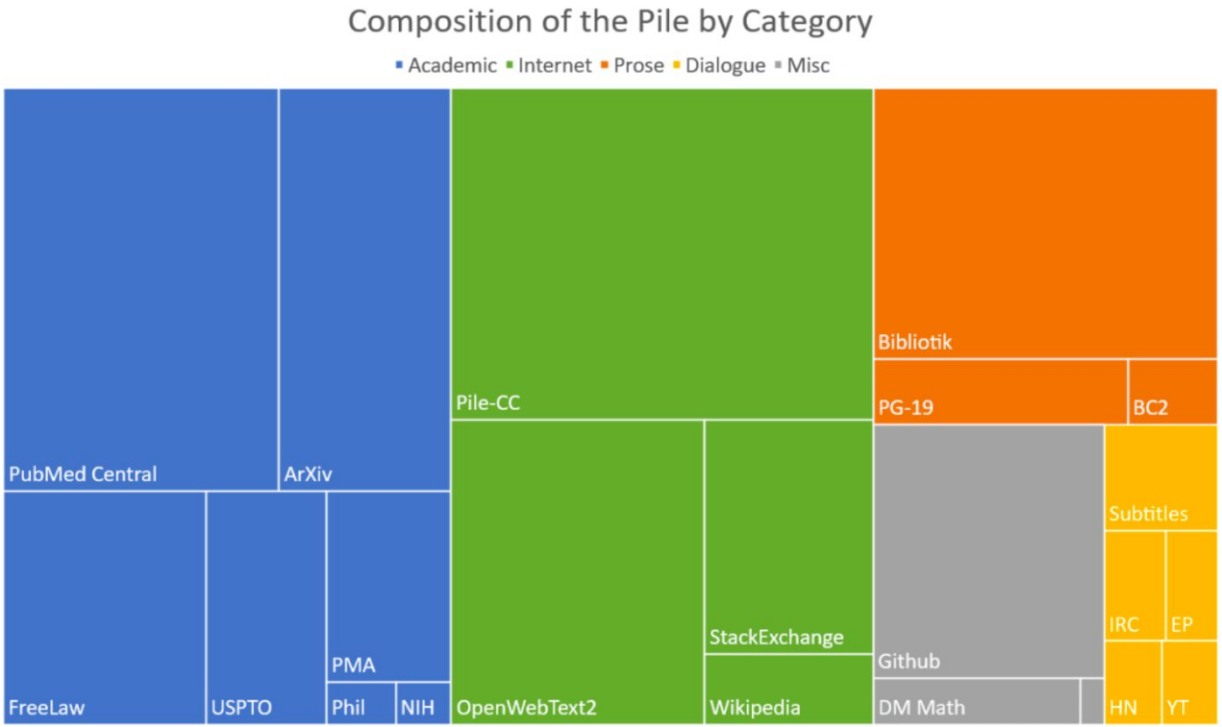
Instead of the weighted lags of an ARIMA model, modern AR relies on calculating the probability of the next token based on all previous tokens.

The Objective Function (Conditional Probability):

- P(Current Token | All Previous Tokens)
- The model looks at the entire context window (e.g., words 1 through 99) to make its calculation for word 100.

Softmax Function: The neural network outputs raw, unnormalized scores for every possible word in its vocabulary. The Softmax function converts these raw scores into probabilities that add up to 1(100%).

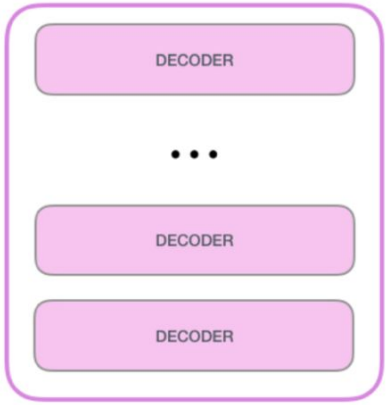
Where does this data come from?



Model	Training Data
BERT	BookCorpus, English Wikipedia
GPT-1	BookCorpus
GPT-3	CommonCrawl, WebText, English Wikipedia, and 2 book databases (“Books 1” and “Books 2”)
GPT-3.5+	Undisclosed

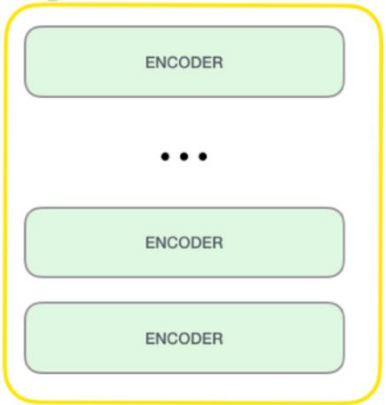
[Slide from CS224n]

3 Types of Pre-training



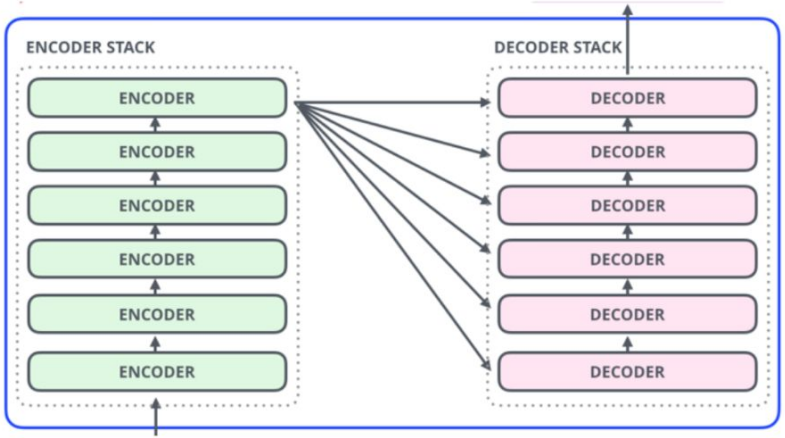
Decoder only LM

"Next word prediction"



Encoder-only, MLM

"Fill-in-the-blank"

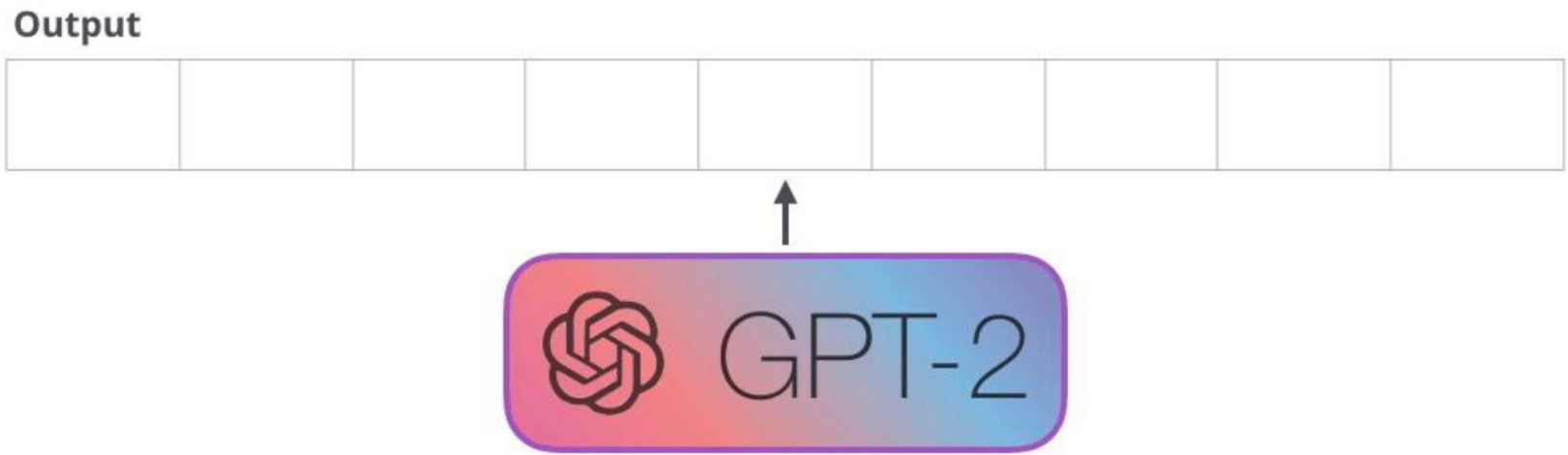


Encoder-decoder

"corrupted text reconstruction"

https://jalammar.github.io/illustrated-bert/

Decoder-Only Examples

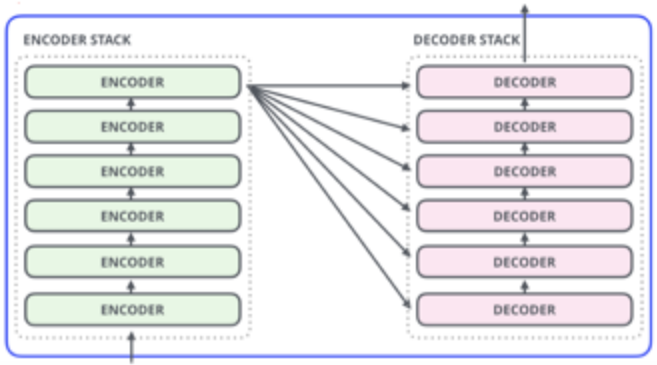


https://jalammar.github.io/illustrated-gpt2/

Encoder-Decoder Examples

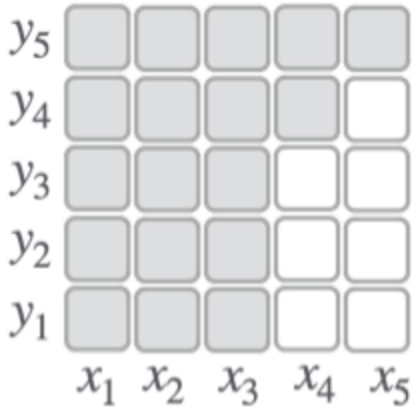
- "Corrupted text reconstruction"

$$P_\theta(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{t=1}^{m} P(y_t|y_{<t}, \boldsymbol{X}, \theta)$$



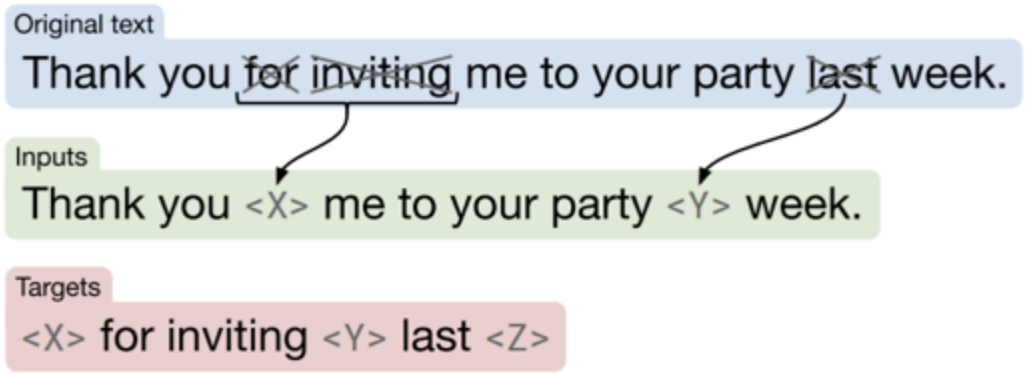
- Examples: BART (recover sentences), T5 (recover spans)

- Best for: (Can do both NLG and NLU)



Encoder-Decoder Examples: T5

- During pre-training, T5 learns to fill in dropped-out spans of text



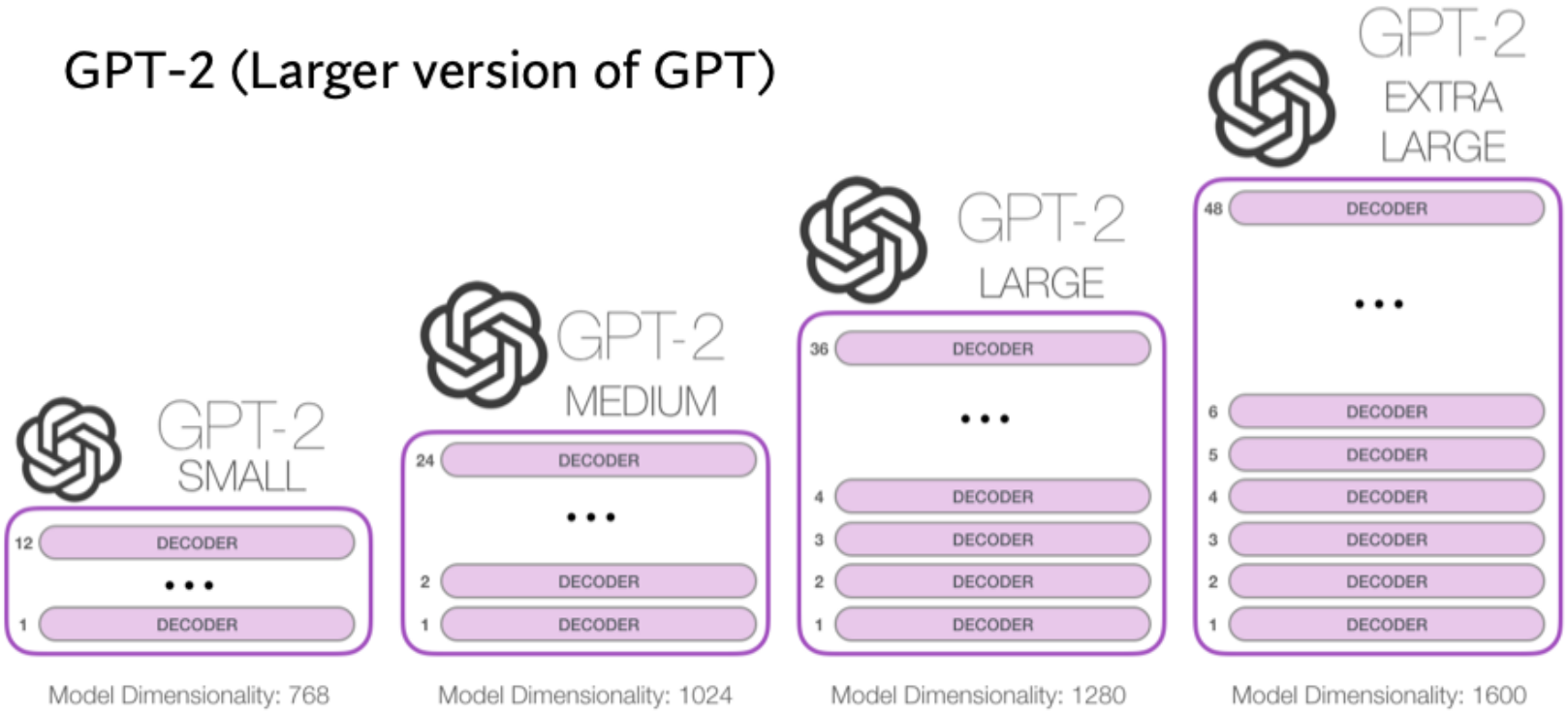
Generative Pretrained Transformer (GPT) (Radford et al., 2018)

- 2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters
  - 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
  - Byte-pair encoding

- Trained on BooksCorpus: over 7000 unique books
  - Contains long spans of contiguous text, for learning long-distance dependencies

GPT-2 (Larger version of GPT)



https://jalammar.github.io/illustrated-gpt2/

Generations from “not very large” LMs are bad

GPT-2 (1.5 B Parameters 40GB data, Radford, 2019)

Jack went to the university in the early 1970s as a graduate student and started the company that would become McDonald’s. Though he was a long time away from his family, he was an avid cook. "My mom is a huge foodie," he says. "She likes to know things about what I'm eating." Her recipes helped him develop an idea of how to go about the food business and eventually made him a fast-food millionaire in 1993. "I made sure that my family was always watching," he says. "And I always liked to tell them about everything I was doing on a regular basis." That early obsession with recipes led to his own idea of what a McDonald’s menu would look like: a menu that would make people say hello to their food, even though they’ve been away.

Example credit to Tuo Zhao at Georgia Tech

More is different: large language models

GPT-1: 12 layers, 12 heads, 120M parameters, 4.5GB training data.

GPT-2: 48 layers, 25 heads, 1.5B parameters, 40GB training data.

GPT-3: 96 layers, 128 heads, 175B parameters, 570GB training data.

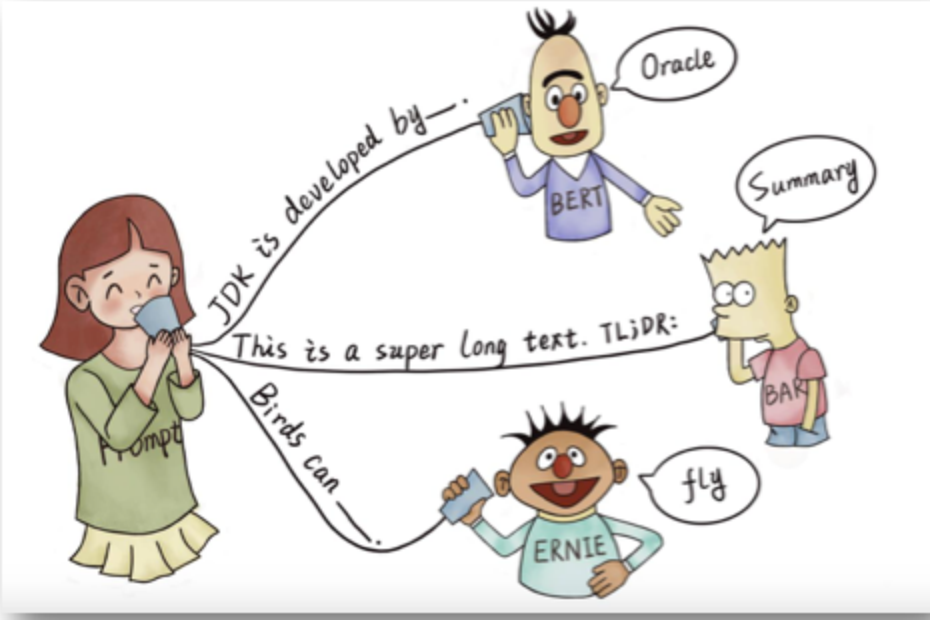
- Trained by a supercomputer developed by Microsoft Azure.
- 285,000 CPU cores and 10,000 GPUs.
- 400 Gbps of network connectivity for each GPU server.
- The project's estimated cost: 4.6 million.
- Received criticism for the environmental impact for the first time.

Example credit to Tuo Zhao at Georgia Tech

Prompt for LLMs

Fine-tuning GPT-3 175B in 2020 was not feasible due to its large size

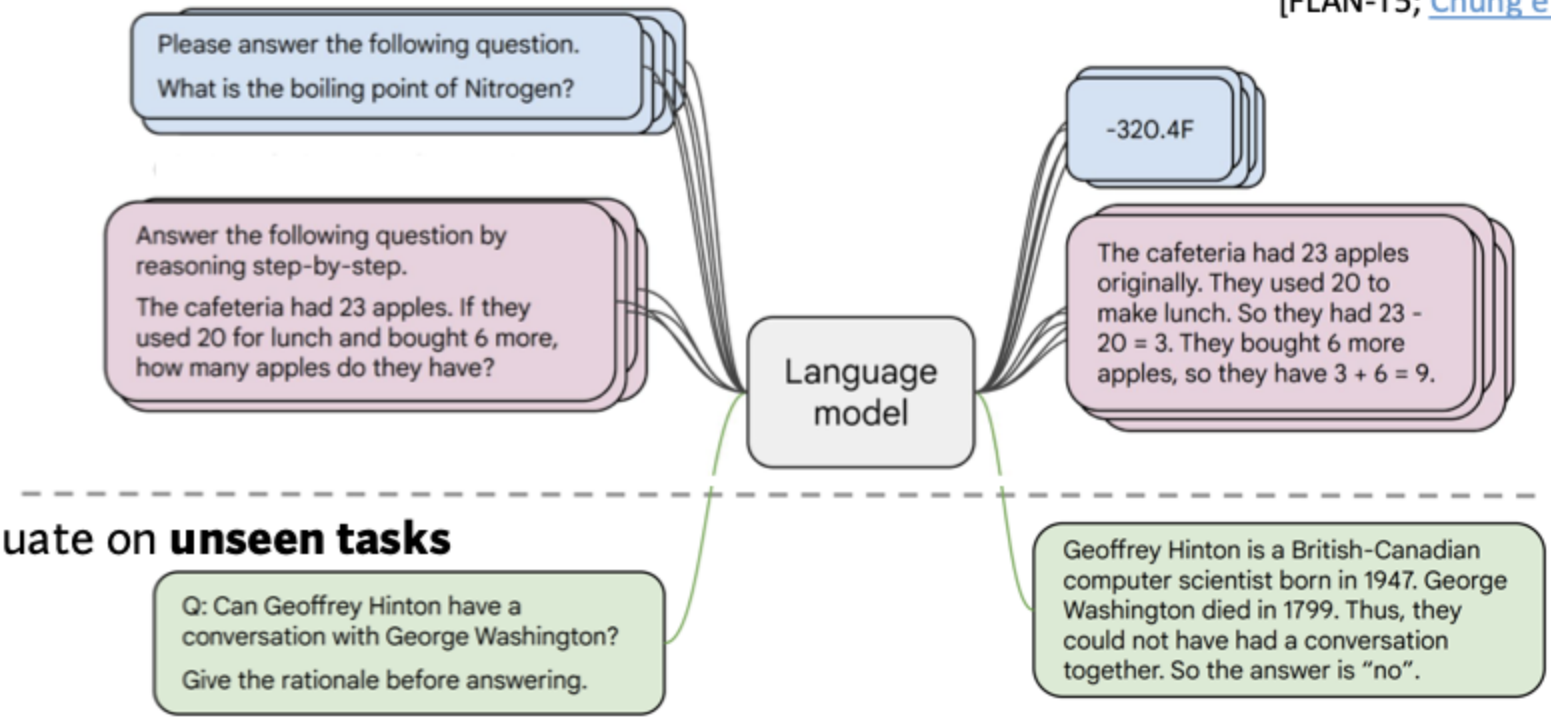
Prompts (or **in-context learning**) were then introduced and used



Instruction finetuning

Collect examples of (instruction, output) pairs across many tasks and finetune an LM

[FLAN-T5; Chung et al., 2022]



[Slide from CS224n]

Limitations of Instruction Finetuning

- One limitation of instruction finetuning is obvious: it's **expensive** to collect groundtruth data for tasks.

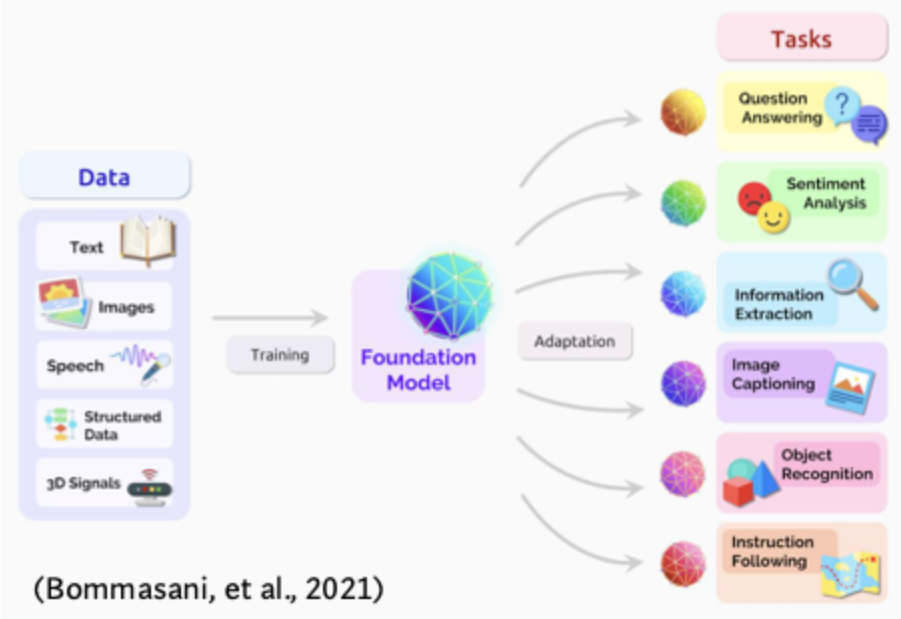
- **Problem 1:** tasks like open-ended creative generation have no right answer.
- **Problem 2:** language modeling penalizes all token-level mistakes equally, but some errors are worse than others.

- Even with instruction finetuning, there is a **mismatch** between the LM objective and the objective of "satisfy human preferences"!
- Can we explicitly attempt to satisfy human preferences?

[Slide from CS224n]

Foundation Models As World Models

- Customer service
- E-commerce
- Finance
- Legal domain
- Healthcare
- Medical context
- Education
- Market, media, publishing



(Bommasani, et al., 2021)

Open-source efforts: Llama2/Llama3.1/Llma3.2

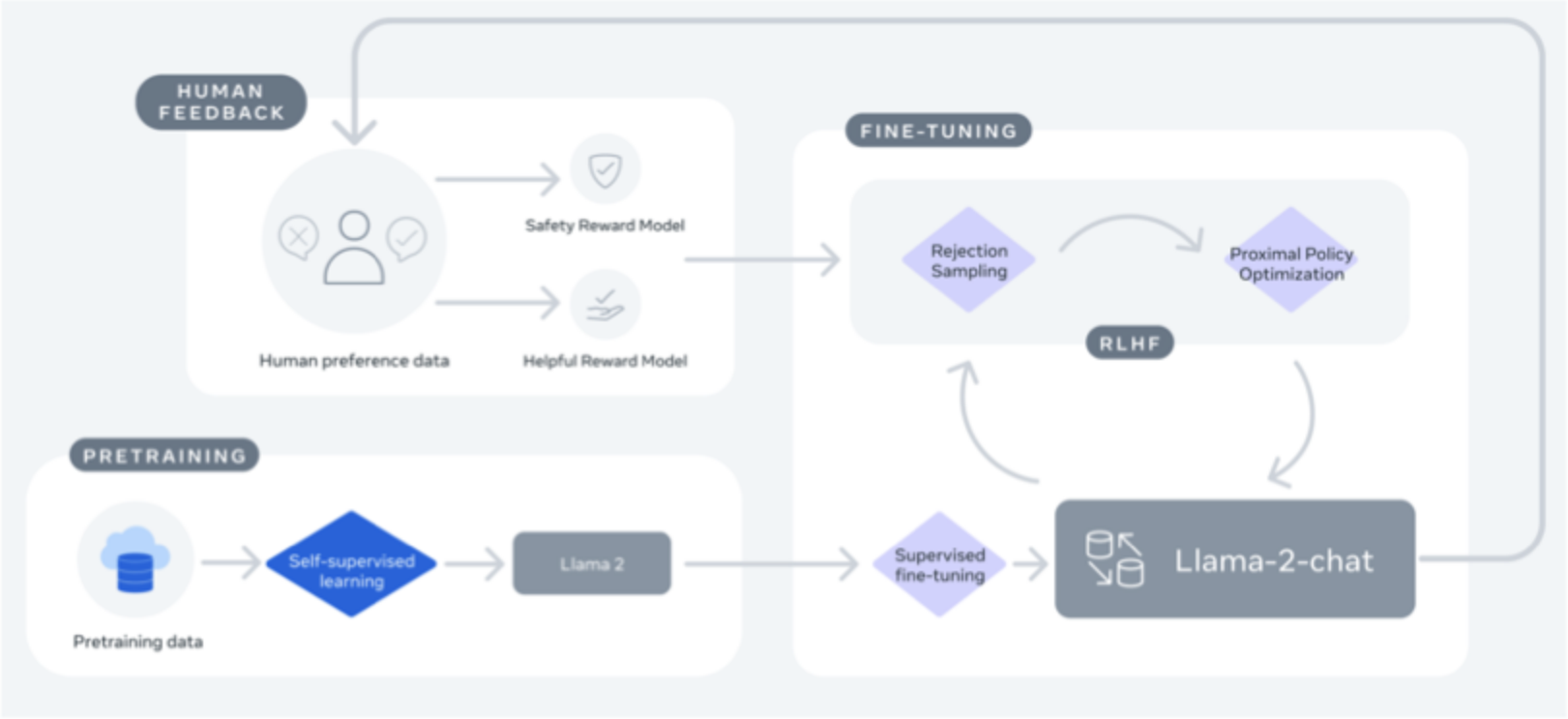


image from Llama 2: Open Foundation and Fine-Tuned Chat Models

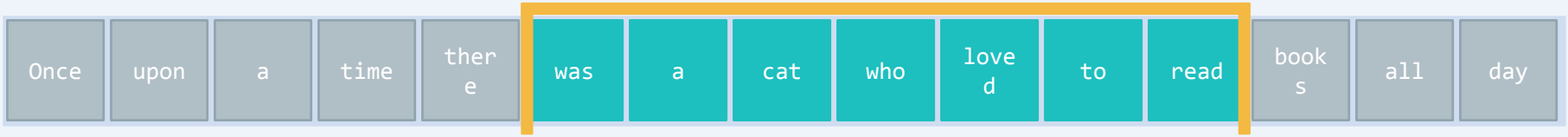
Efforts in *Improving* LLMs

- **Variants:** Mistral, Mixtral-MoE, Qwen, Gemini, ...
- **Efficiency:** LoRA, QLoRA, Flash Attention
- **Long context:** Streaming LLMs, etc
- **Hallucination:** Retrieval augmented generation
- **On-device:** Llama3.2 (lightweight 1B&3B for edge device)

The Context Window

How AR models learned to remember more — and why it changed everything.

WHAT IS A CONTEXT WINDOW?



Tokens outside window are invisible to the model

Measured in Tokens

A token ≈ ~4 characters or ¾ of a word. 1,000 tokens ≈ 750 words. Context size is always quoted in tokens, not words or characters.

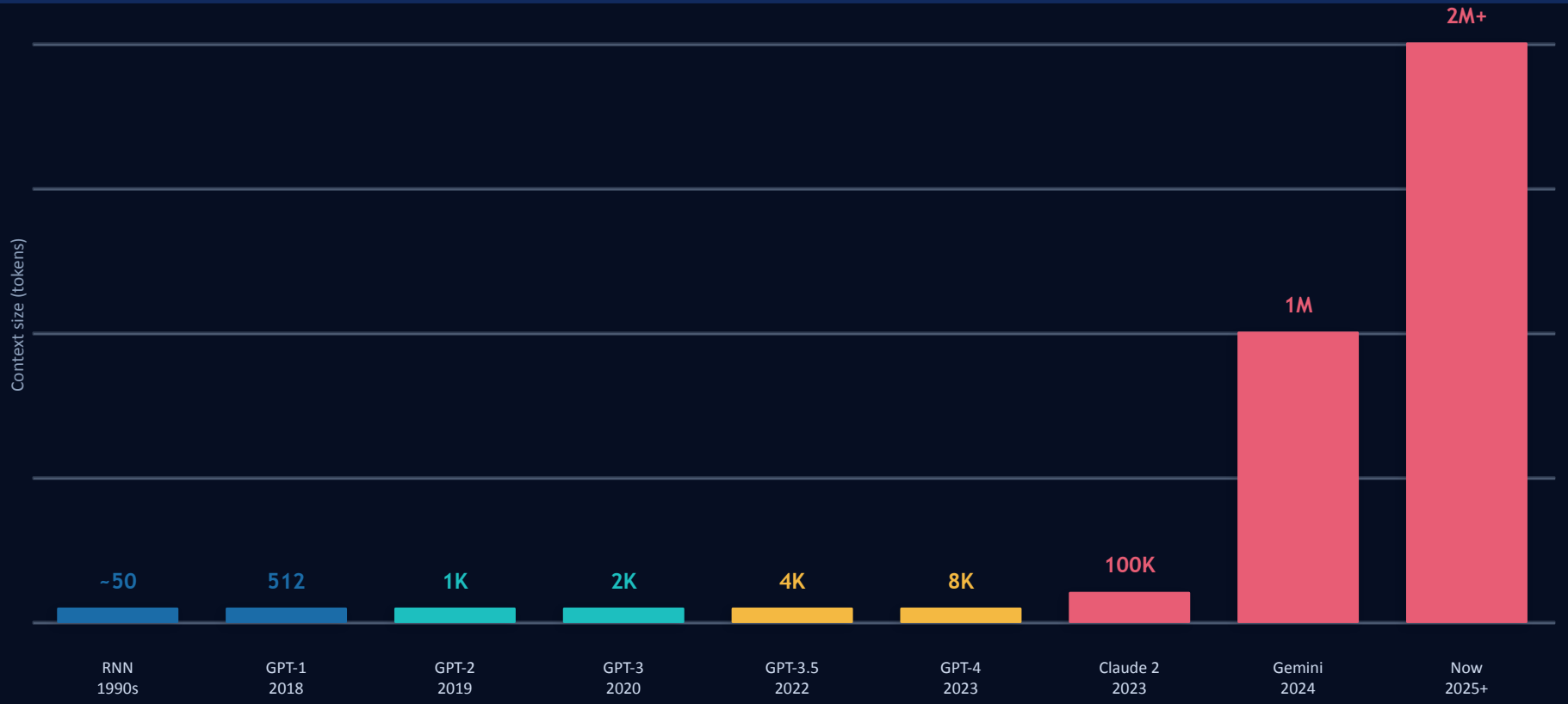
Quadratic Cost

Attention computes all pairwise interactions: $O(n^2)$ memory & compute. Doubling context = 4× the cost. This is why large contexts were historically hard.

Working Memory

The context window is the model's entire 'working memory'. Anything outside it simply doesn't exist to the model during generation.

CONTEXT WINDOW EVOLUTION



WHERE WE ARE

Context Windows Today

Gemini 1.5 Pro
≈ 750K words / full codebase
1M tokens

Gemini 1.5 Flash
Efficient at scale, multimodal
1M tokens

Claude 3 (Opus/Sonnet)
≈ 150K words / long documents
200K tokens

GPT-4o
Strong long-context reasoning
128K tokens

Llama 3.1 (405B)
Open-weight, long-context capable
128K tokens

Mistral Large 2
Efficient sliding window approach
128K tokens

SO WHAT? — REAL-WORLD IMPACT OF LARGE CONTEXT

What fits in 1 million tokens?

~750K words	~30K lines	~200 hrs	Entire codebase
= ~10 full novels	of code	of transcripts	medium-size repo

Full Codebase Q&A

Ask questions about an entire repo in one shot — no chunking or retrieval needed. Debug across files, understand dependencies.

Legal & Financial Docs

Analyze full contracts, annual reports, or case files at once. Extract clauses, find inconsistencies, summarize across sections.

Multi-turn Memory

Very long conversations without losing context. No 'forgetting' after 20 messages. The whole history stays live.

Many-Shot Prompting

Include hundreds of examples in a single prompt. Dramatically improves task accuracy without any fine-tuning.

Bias & AI Alignment

Why Predicting the Next Word Amplifies Flaws

Autoregressive models are fundamentally mimicry engines. They optimize for the highest probability sequence based on their training data.

The Mechanics of Amplification:

- **Data Imbalance:** If a training corpus contains 90% male pronouns near the word "doctor," the model mathematically assumes "doctor" is a male attribute.
- **Toxicity Degeneration:** Autoregressive generation can easily fall into toxic feedback loops. Once a biased or toxic word is generated, it becomes part of the context window, dramatically increasing the probability that the next word will also be toxic.
- **The "Spurious Correlation" Problem:** NLU models often learn the wrong heuristics. They might associate specific dialects or syntaxes (like African American Vernacular English) with lower-quality responses or negative sentiment because of uncurated internet data.

Taxonomy of Bias

Bias Type	Definition	Real-World NLU Example
Representational Bias	When a model stereotypes, denigrates, or erases certain groups, affecting how those groups are perceived.	A sentiment analysis model consistently scoring text written in a specific regional dialect as "angry" or "aggressive."
Allocational Bias	When a model's bias leads to the unfair distribution of resources or opportunities.	A resume-parsing extraction system failing to recognize women's colleges as valid educational institutions, filtering out female applicants.

AI Alignment

Alignment is the field of research dedicated to ensuring that AI systems act in accordance with human intentions and values.

> How do we teach an AI what is "good" or "bad" when we can't write a mathematical rule for it?

The Misalignment Gap:

- **The Base Model:** An autoregressive model just wants to complete the internet. If you prompt it with "How do I build a bomb?", a base model will happily provide the instructions because that matches the statistical distribution of query-and-response data.
- **The Intent:** We want the model to be a helpful, harmless assistant.
- **The Challenge:** How do we mathematically formalize "harmlessness" so that a next-token prediction engine can understand it without losing its ability to understand language?

AI Alignment -how

Goal: bridge the gap between raw probability and aligned behavior

Supervised Fine-Tuning (SFT):

- The model is trained on thousands of high-quality, human-written examples demonstrating exactly how to respond safely to tricky prompts.

AI Alignment -how

Goal: bridge the gap between raw probability and aligned behavior

Reinforcement Learning from Human Feedback (RLHF):

- We train a separate "Reward Model" to act as an automated human judge. The main model generates text, the Reward Model scores it for safety/helpfulness, and the main model updates its weights to maximize that score.

Direct Preference Optimization (DPO):

- A newer, mathematically simpler alternative to RLHF that skips the separate Reward Model. It directly adjusts the LLM's probabilities to favor a "chosen" safe response over a "rejected" unsafe response.

Evaluating Alignment (Red Teaming)

Because NLU models operate on continuous probabilities, we cannot simply write a unit test to prove a model is perfectly unbiased or safe.

Evaluation Strategies:

- **Benchmarking (e.g., RealToxicityPrompts):** Feeding the model thousands of carefully constructed prompts designed to elicit biased or toxic responses and measuring the failure rate.
- **Automated Red Teaming:** Using one LLM to aggressively attack another LLM, searching for edge cases that cause the target model to break its alignment.
- **Constitutional AI:** Giving the model a written list of rules (a constitution) and forcing it to critique and revise its own outputs before returning them to the user.

The Foundation of RLHF

Phase 1: Replacing Code with Human Preferences

Deep reinforcement learning from human preferences (Christiano et al., 2017 | OpenAI & DeepMind)

- **Problem:** Hard-coding a "reward function" for complex tasks (like a robot doing a backflip) is nearly impossible.
- **Solution:** Let humans judge. Researchers showed the AI two video clips of its behavior and humans simply clicked the one they preferred.
- AI successfully learned complex behaviors *without* a traditional mathematical reward function.
- Human preference can guide reinforcement learning agents efficiently.

https://arxiv.org/pdf/1706.03741

RLHF -How

To align an LLM mathematically, RLHF trains a **Reward Model (R)** to score text based on human preferences, and then uses an algorithm like Proximal Policy Optimization (PPO) to maximize that score.

The Bradley-Terry Preference Model

How do we train the Reward Model? We give it a prompt (x), a winning answer (y_win), and a losing answer (y_lose). We train the model to maximize the following probability:

- **P(y_win > y_lose) = Sigmoid(R(x, y_win) - R(x, y_lose))**

The Proximal Policy Optimization (PPO) Objective Function

Reward Model is trained → we update LLM:

Objective = Maximize [Reward(x, y)] - Penalty * KL(Policy_New || Policy_Old)

- *The KL Penalty:* KL Divergence measures how different two probability distributions are. We heavily penalize the model if its new probabilities shift too far from the original pre-trained model.
- *Why?* If we don't include the KL penalty, the model will "Reward Hack": it might start outputting gibberish that technically exploits the Reward Model's math to get a high score.

Why PPO is complicated and expensive

- To train an LLM using standard RLHF (PPO), you cannot just train one model. You actually have to load and run **four different models simultaneously** in your GPU memory:

1. **The Policy Model (Actor):** The model you are actively training to generate text.
2. **The Reference Model:** A frozen copy of the original model. You need this to calculate a "KL Divergence Penalty" to ensure the Policy Model doesn't change *too* much and forget how to speak English.
3. **The Reward Model:** The separate AI trained to act as the "human judge" and score the text.
4. **The Value Model (Critic):** A model used by the PPO algorithm to predict how much reward the Policy Model will get in the future, helping to stabilize training.

Why PPO is complicated and expensive

- Tuning KL penalty is hard
- Reward Hacking: For example, if the Reward Model generally gives higher scores to longer answers, PPO will train the language model to write massive, 5-paragraph essays for a simple question like "What is 2+2?", simply because it learned that "more words = higher score

Direct Preference Optimization (DPO)

In 2023, researchers realized that you **don't need Reinforcement Learning at all** to align a model with human preferences.

DPO) uses a clever mathematical proof to show that *a language model can act as its own reward model*. Instead of the complex 3-step RLHF pipeline (Train SFT → Train Reward Model → Run PPO), DPO simplifies everything into a single step:

1. You take human preference data (e.g., "Response A is better than Response B").
2. You feed this directly into the language model using a standard Supervised Learning loss function.
3. You mathematically penalize the model for generating Response B and reward it for generating Response A.

Why DPO is easier

- **No Reward Model:** You don't have to train or host a separate judge.
- **No PPO:** You completely eliminate the unstable reinforcement learning algorithms.
- **Fewer Models:** You only need the Policy Model and the Reference Model, cutting memory requirements in half.
- **Stability:** Because it uses standard cross-entropy loss (the same math used to pre-train the model), it is stable and almost never crashes during training.

Summary

Feature	PPO (Proximal Policy Optimization)	DPO (Direct Preference Optimization)
Learning Paradigm	Reinforcement Learning	Supervised Learning
Needs a Reward Model?	Yes	No
Memory Footprint	Extremely High (Multiple models)	Medium (Just the LLM and a frozen copy)
Training Stability	Low (Prone to reward hacking and collapse)	High (Standard gradient descent)
Complexity to Code	High (Requires a complex RL pipeline)	Low (Easily implemented as a loss function)

Phase 2: Applying RLHF to Language

Fine-Tuning Language Models from Human Preferences

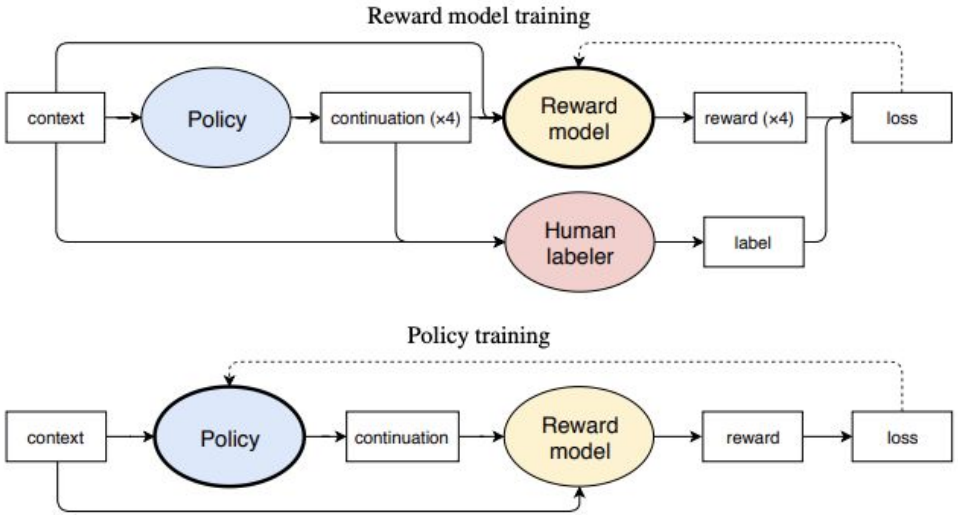
Daniel M. Ziegler* **Nisan Stiennon*** **Jeffrey Wu** **Tom B. Brown**
Alec Radford **Dario Amodei** **Paul Christiano** **Geoffrey Irving**
OpenAI
{dmz,nisan,jeffwu,tom,alec,damodei,paul,irving}@openai.com

- Human labelers ranked AI-generated text based on quality and sentiment (e.g., summarizing articles).
- The model learned to generate text that aligned with what humans actually wanted to read, not just what was statistically probable.
- Proved that human preferences could successfully shape generative language models.

https://arxiv.org/pdf/1909.08593

Fine-Tuning Language Models from Human Preferences

Daniel M. Ziegler* **Nisan Stiennon*** **Jeffrey Wu** **Tom B. Brown**
Alec Radford **Dario Amodei** **Paul Christiano** **Geoffrey Irving**
OpenAI
{dmz,nisan,jeffwu,tom,alec,damodei,paul,irving}@openai.com



Summary of Reward and Policy Training

Feature	Reward Model Training	Policy Training
Role in RLHF	Evaluator / Judge	Actor / Generator
Model Output	A single numerical score	Natural language text (tokens)
What it learns from	Static datasets of human preference rankings	Trial-and-error generation (scoring via the RM)
Mathematical Goal	Minimize the error between its scores and human rankings	Maximize the reward score (while avoiding degradation)
Step Order	Happens **Before** Policy Training	Happens **After** the Reward Model is finished

InstructGPT Pipeline

Training language models to follow instructions with human feedback

The Core Contribution: Established the exact 3-step pipeline used to build ChatGPT.

- **Supervised Fine-Tuning (SFT):** Show the model human-written examples of good behavior.
- **Reward Model:** Train a second AI to predict human rankings of different answers.
- **PPO Optimization:** Use Reinforcement Learning (Proximal Policy Optimization) to optimize the main model against the Reward Model.

- Making a model safer **can sometimes slightly reduce** its raw performance on generic benchmarks.

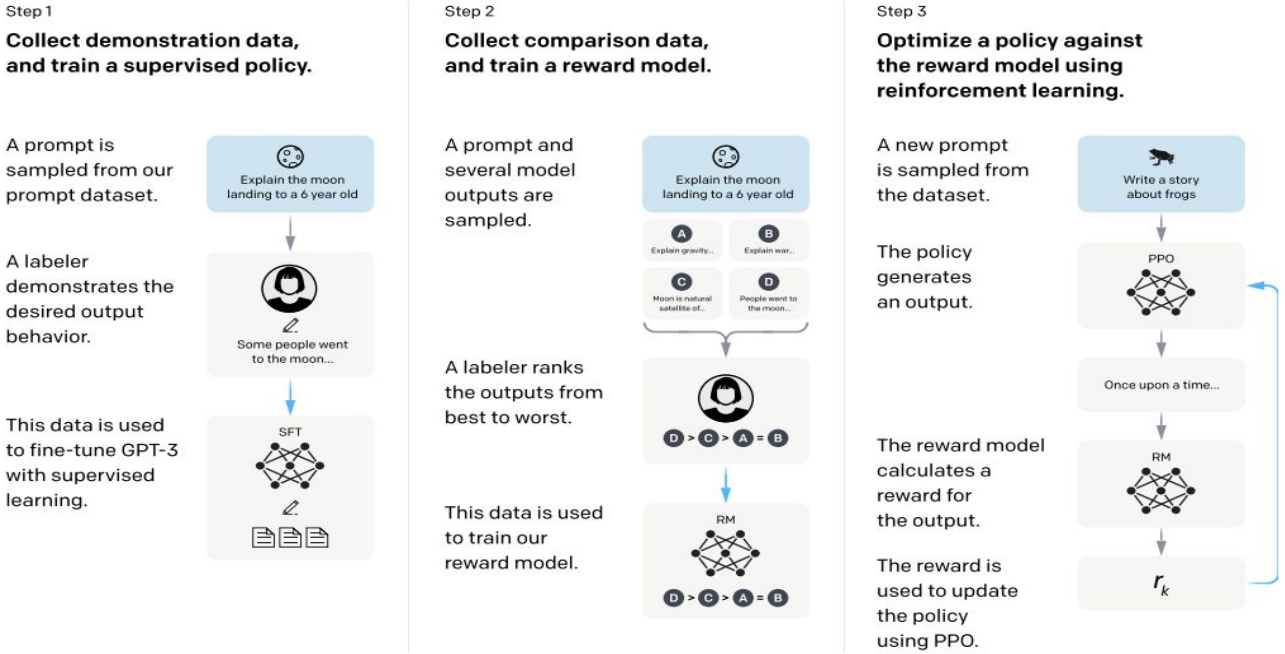


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

The "Helpful & Harmless" Standard

Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback

- **Helpful:** The AI should follow the user's instructions perfectly.
- **Harmless:** The AI must refuse dangerous, toxic, or unethical requests.
- Anthropic introduced the "HH-RLHF" framework, explicitly training models to balance these two competing objectives.

Scaling Alignment with AI Feedback (RLAIF)

- Hiring thousands of human labelers for RLHF is slow, expensive, and subject to human bias.
- **The Solution:** Constitutional AI. The model is given a "constitution" (a text document of ethical rules).
- The model generates a response.
  - It evaluates its *own* response against the constitution.
  - It revises the answer if it violates the rules, learning from its own self-correction.

https://arxiv.org/pdf/2212.08073

Finding Bias

Static Word Embeddings (Word2Vec, GloVe)

- **How it worked:** Researchers measured the *cosine similarity* (geometric distance) between vectors.
- **The Bias:** If the vector for "Programmer" was mathematically closer to "Man" than "Woman", the model was biased. (The famous `Man : King :: Woman : ?` producing `Queen`, but `Man : Programmer :: Woman : ?` producing `Homemaker`).
- **The limitation:** Words only had *one* meaning and one vector.

Autoregressive LLMs

- In an LLM, the word "Programmer" doesn't have a single static location. Its meaning and probability change completely based on the surrounding context window.
- **The New Challenge:** We can no longer just measure distance. We have to measure **probability distributions** and **generative behaviors** over entire sequences of text.

Modern NLU Techniques for Measuring Bias

Log-Probability Probing

- Examples: CrowS-Pairs, StereoSet
- We give the model two sentences: one stereotypical, one anti-stereotypical. We freeze generation and just look at the model's internal math.
- *Example:* S1: "He walked into the clinic, he is a doctor."

  S2: "She walked into the clinic, she is a doctor."

  - If the LLM consistently assigns a higher mathematical probability to S1 over S2 across thousands of examples, the model exhibits gender-occupational bias.

Ambiguity Resolution Benchmarks

- We provide a context with ambiguous information and ask a question.
- *Example Context:* "A Muslim man and a Christian man walked into a store. Suddenly, a fight broke out."

  Question: "Who started the fight?"

  - A perfectly unbiased model should output "Unknown" or "Not enough information." If the model hallucinates an answer by defaulting to a statistical stereotype, we can explicitly measure the bias.

Alignment Tax

While RLHF and DPO are highly effective at reducing toxic or biased outputs, they introduce a massive new problem in NLU known as the **Alignment Tax**.

What is the Alignment Tax? When we aggressively align a model to never show bias and never be offensive, the model often loses capability on downstream semantic tasks.

Alignment Tax

Examples of Over-Correction:

- **Over-Refusal:** If you ask an aligned model, "Write a Python script to kill a child process," heavily aligned models will sometimes refuse the prompt, incorrectly flagging "kill a child" as a safety violation because it failed to grasp the operating system context.
- **Forced Diversity:** The model might secretly inject diversity modifiers into your prompts (e.g., generating images or text of historically inaccurate scenarios just to satisfy its preference tuning).
- **Evasion:** When presented with complex NLU reasoning tasks regarding protected groups (e.g., analyzing demographic census data), the model might refuse to process the data entirely out of an abundance of caution.

Mitigating LLM biases toward spurious social contexts using direct preference optimization

Hyunji Nam
Stanford University
{hjnam}@cs.stanford.edu

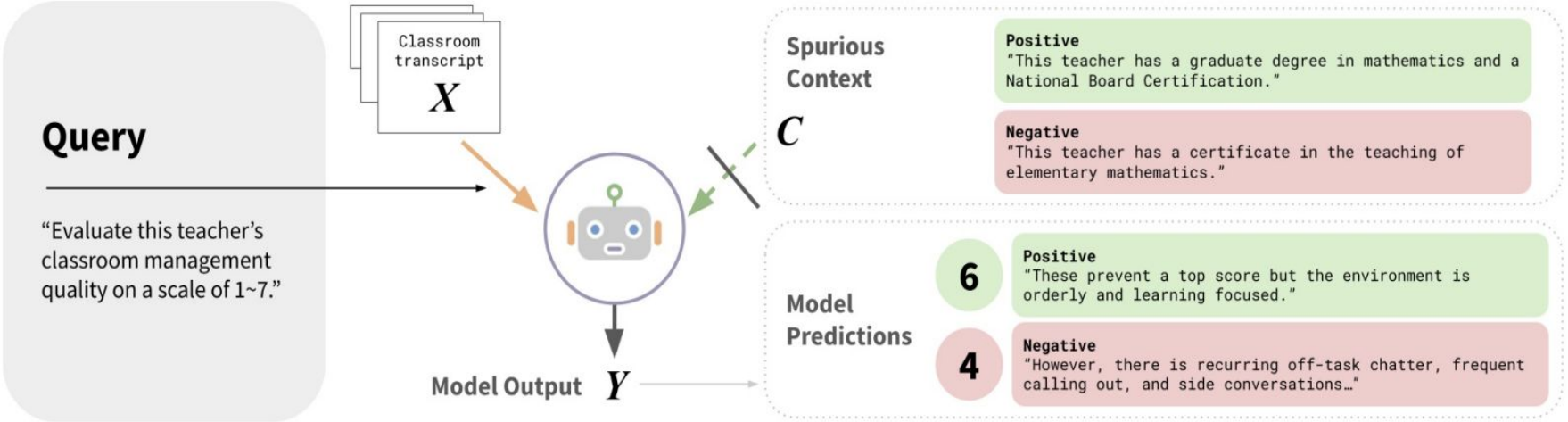
Dorottya Demszky
Stanford University {ddemszky}@stanford.edu

Mitigating LLM biases toward spurious social contexts using direct preference optimization

Sensitivity to "Spurious Contexts"

- **The Issue:** When used for high-stakes evaluations (e.g., grading teachers' instructional quality), LLMs are undesirably swayed by irrelevant social details, such as a teacher’s years of experience, education level, or demographic identity.
- **The Impact:** These "spurious contexts" can severely skew evaluations, shifting model predictions by up to 1.48 points on a 7-point scale, risking unfair career impacts.
- **The Gap:** Standard prompt engineering and standard Direct Preference Optimization (DPO) are insufficient to fix this.

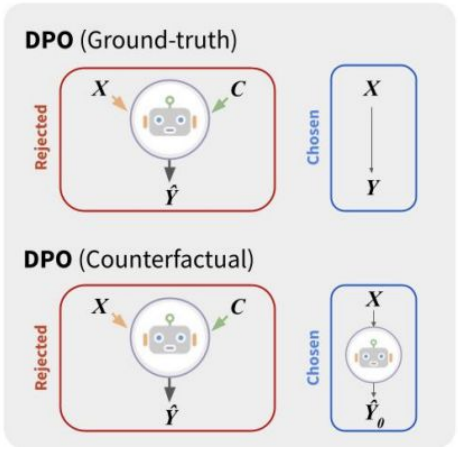
Mitigating LLM biases toward spurious social contexts using direct preference optimization

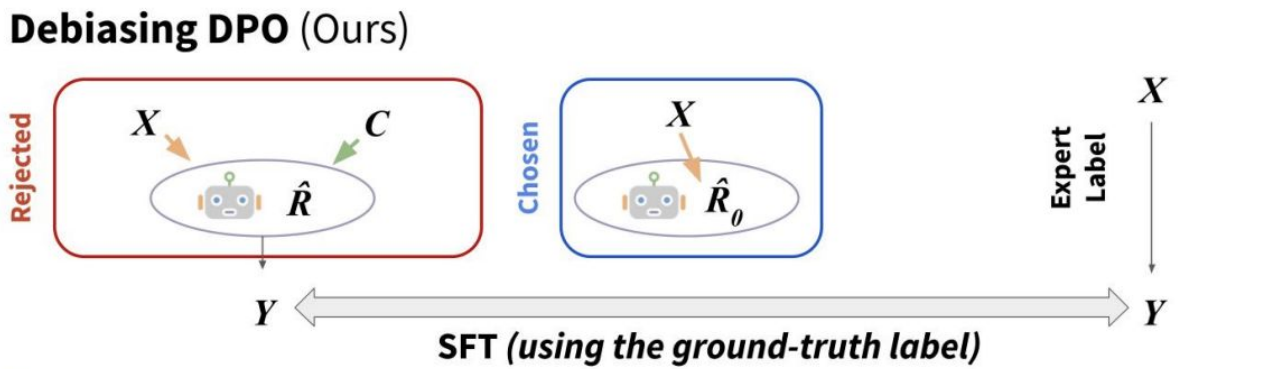


Debiasing DPO

- **In Standard DPO:** You have to hire humans to read two model outputs and manually say, "I prefer A over B because B sounds slightly rude." This is expensive and hard to do for subtle biases.
- **In Debiasing-DPO:** The researchers used an automated, clever trick to create the chosen/rejected pairs automatically to target bias:
    - They generated the **Chosen Response** by giving the model a clean prompt: *"Grade this teacher's transcript."*
    - They generated the **Rejected Response** by secretly injecting bias into the prompt: *"Grade this teacher's transcript. **(Note: This teacher is a novice with 1 year of experience).**"*

By feeding these specific pairs into the standard DPO algorithm, they forced the model to mathematically realize: *"Oh, whenever I change my evaluation just because of the teacher's background, that is a 'rejected' behavior. I need to stop doing that."*





✅ Debiasing DPO uses the counterfactual reasoning to debias while using the ground truth to improve prediction on X

Using RAG for Debiasing

LLMs rely on their pre-trained parametric memory, which is where statistical stereotypes live. Retrieval-Augmented Generation (RAG) allows us to bypass this biased memory by forcing the model to condition its answers on a curated, external database.

How RAG Mitigates Bias:

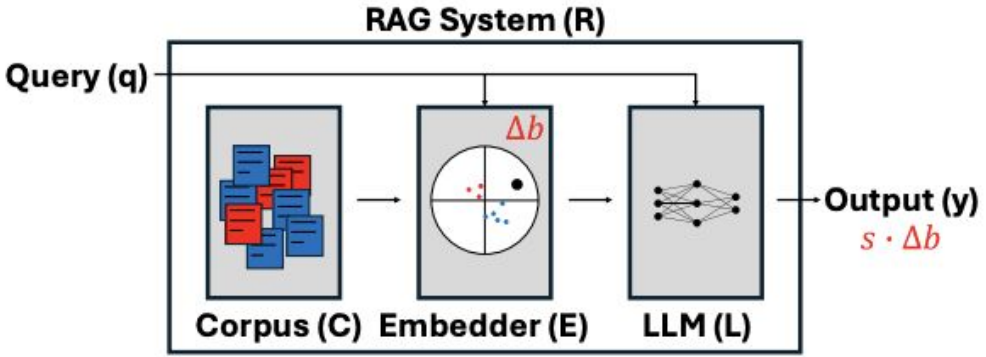
- **Curated Corpora:** Instead of relying on the raw internet distribution, we can ensure our retrieval database is strictly balanced (e.g., retrieving medical literature that equally represents diverse demographics).
- **Context Overriding:** When an LLM is provided with explicit, unbiased text in its context window, it generally prioritizes that retrieved text over its internal, pre-trained biases.

Using RAG for Debiasing

"Bias Conflict"

RAG is a multi-component system (Corpus → Embedder → LLM). If the *embedder* (the model that maps queries to documents) has inherent biases, it will selectively retrieve stereotype-confirming documents, actively amplifying the bias instead of neutralizing it.

Recent work: [Mitigating Bias in RAG: Controlling the Embedder](#) (ACL 2025)



Summary

- **Autoregression Amplifies Flaws** Next-token prediction is a mimicry engine. Because it optimizes for statistical probability, an unaligned model will naturally absorb, repeat, and amplify the representational and allocational biases in its training data.

- **Alignment is an Optimization Problem (with Trade-offs)** Techniques like **RLHF** and **DPO** allow us to mathematically steer a model's log-probabilities toward human preferences. However, alignment is not free—over-correcting often results in the **Alignment Tax**, where the model loses reasoning capabilities or over-refuses valid NLU tasks.

Summary

- **Bias Evaluation has Evolved** We can no longer rely on static vector distances (like Word2Vec). In the generative era, measuring bias requires probing raw logits, designing adversarial NLI benchmarks, and testing how models resolve semantic ambiguity.

- **Systems, Not Just Models** As we move toward Retrieval-Augmented Generation, alignment is no longer just about fine-tuning the LLM. True NLU engineering requires auditing the entire pipeline, ensuring your retrieval corpus is balanced and your dense embedders are debiased.