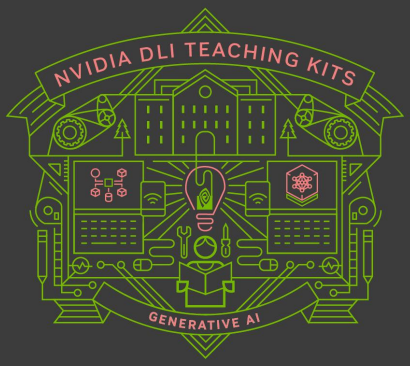
Lecture 1.2 – Introduction to Generative AI

Generative AI Teaching Kit



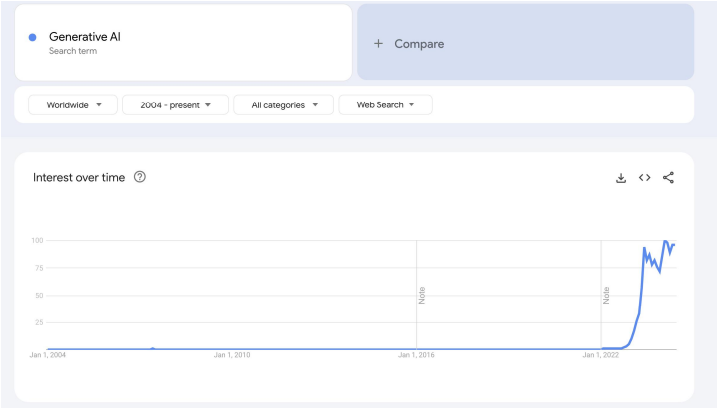
This lecture

- What is “GenAI”?
- History of AI over the last 50 years
- Pitfalls/risks of GenAI

DARTMOUTH ENGINEERING | NVIDIA.

The Rise of Generative AI “GenAI”

- The development of AI reached a new level in November of 2022 with the launch of ChatGPT
- ChatGPT, a chatbot, took the world by storm and was built on top of the Large Language Model GPT-3.
- GPT-3, for the research community, also shook the domain when it was released, showing that models could learn from unsupervised data and then further learn new tasks without further training.
- The race for supremacy began. These models, being able to respond in human-like fashion, have brought the topic of AI into the mainstream.



DARTMOUTH ENGINEERING | NVIDIA.

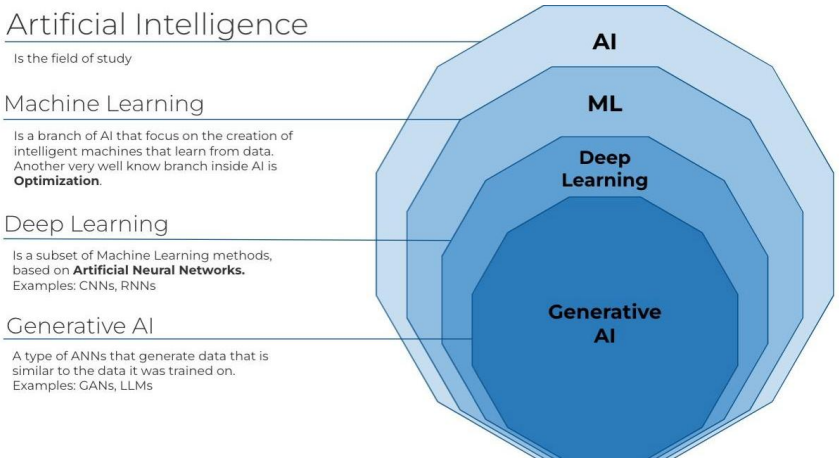
What is “GenAI”?

GenAI (aka Generative AI) refers to a subset of models and approaches that have been trained specifically to generate new data.

This is different to other, un/supervised machine learning models that are trained to classify data samples, or connect input to output values for regression

Examples of **non-GenAI models**:
- Convolutional Neural Networks
- Logistic Regression
- Random Forest

Examples of **GenAI models**:
- Large Language Models
- Diffusion Models
- Generative Adversarial Networks



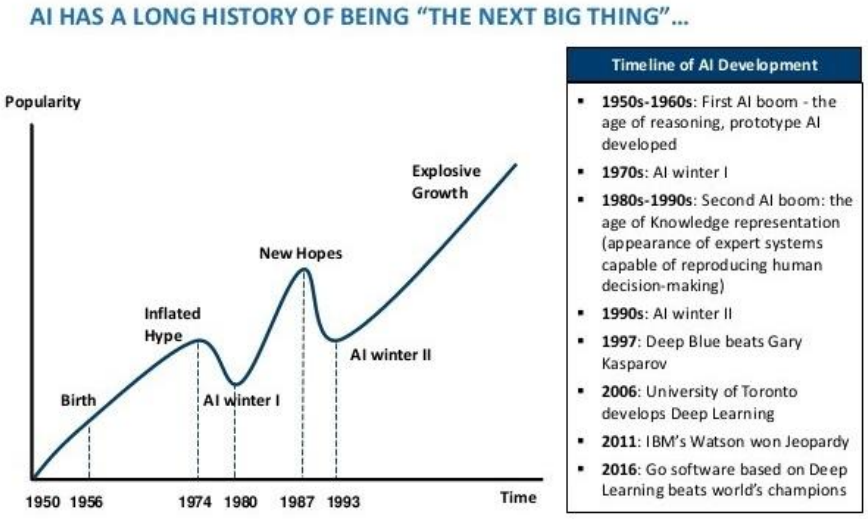
DARTMOUTH ENGINEERING | NVIDIA.

The History of AI – How did we get here?

DARTMOUTH ENGINEERING | NVIDIA.

How did we get here?

Over the last 50+ years, AI popularity and development has ebbed and flowed. Let's take a look into how we got to ChatGPT



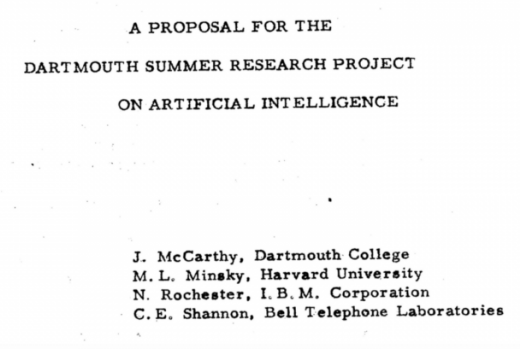
Timeline of AI Development

- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

1956 Dartmouth Conference: The Birth of Artificial Intelligence

Significance

- Coining of "Artificial Intelligence": The Dartmouth Conference is where the term "Artificial Intelligence" (AI) was first officially used. John McCarthy, one of the conference's organizers, is credited with coining the term.
- Foundational Moment for AI: This event is often considered the founding moment of AI as a field of study. It was the first time researchers from various disciplines came together to explore the concept of machine intelligence.



A PROPOSAL FOR THE

DARTMOUTH SUMMER RESEARCH PROJECT

ON ARTIFICIAL INTELLIGENCE

J. McCarthy, Dartmouth College
M. L. Minsky, Harvard University
N. Rochester, I. B. M. Corporation
C. E. Shannon, Bell Telephone Laboratories

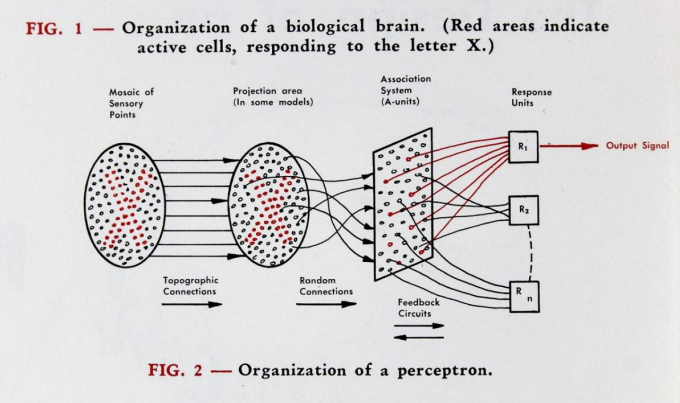
Key Participants

- John McCarthy: Known for later developing the LISP programming language, a cornerstone of AI research.
- Marvin Minsky: A cognitive scientist and AI pioneer who would later co-found the MIT AI Lab.
- Claude Shannon: Known as the father of information theory, he brought a deep understanding of communication and information processing to the discussion.
- Nathaniel Rochester: An IBM researcher who contributed to the development of the first computers and was interested in the possibilities of programming machines to think.

1958 Perceptron by Rosenblatt at Cornell

Foundational Model for Neural Networks:
The perceptron, introduced by Frank Rosenblatt in 1958, is one of the earliest models of an artificial neuron, laying the groundwork for the development of modern neural networks. It was inspired by biological neurons and aimed to mimic the way the brain processes information.

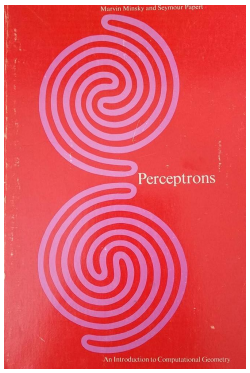
Limitations and Influence on Future Research:
While the perceptron was a breakthrough, it had significant limitations, most notably its inability to solve problems that are not linearly separable, such as the XOR problem.



1969 Analysis of Perceptrons Minsky & Papert

Critical Examination of Perceptrons:

In their 1969 book "Perceptrons: An Introduction to Computational Geometry," Marvin Minsky and Seymour Papert critically analyzed the capabilities and limitations of perceptrons, a type of artificial neuron model introduced by Frank Rosenblatt. Their work highlighted fundamental challenges in the perceptron model, particularly its inability to solve problems that are not linearly separable, such as the XOR problem.



Impact on the Field and AI Winter:

The book's critical analysis had a significant impact on the AI community. It led to a period of reduced funding and interest in neural network research, often referred to as the "AI Winter." However, the insights from Minsky and Papert's work also set the stage for the later development of more advanced neural network models, including the backpropagation algorithm in the 1980s, which allowed for the training of multi-layer networks and revitalized interest in the field.

1st AI Winter: **1974 - 1980**

An AI Winter refers to a period of reduced funding, interest, and activity in artificial intelligence research. During these times, enthusiasm for AI wanes due to unmet expectations, and progress in the field slows down significantly.

The first AI Winter led to a significant slowdown in research activity, with fewer new developments and innovations. Many AI labs were closed or repurposed, and interest in AI declined across the academic and industrial sectors.

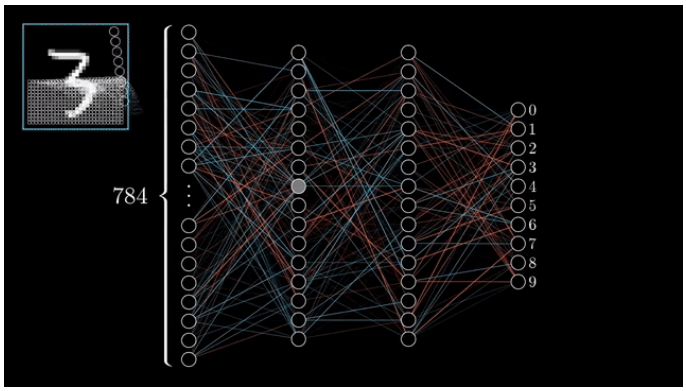
Impact on the field:

- Many researchers shifted their focus to related fields.
- Some subfields of AI, like machine learning, continued to develop quietly, laying the groundwork for future breakthroughs.
- The AI Winter underscored the importance of managing expectations and the challenges of translating research breakthroughs into practical applications, lessons that continue to influence AI research today.

DARTMOUTH ENGINEERING | NVIDIA.

Backpropagation – Popularized in **1986** by Hinton et.al

Backpropagation, or "*backward propagation of errors*," is an algorithm used to train neural networks by efficiently computing the gradient of the loss function with respect to the network's weights. This process allows the network to adjust its weights to minimize errors, thereby improving its performance.

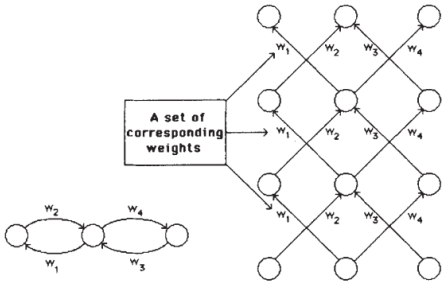


- It uses the chain rule from calculus to compute gradients layer by layer, starting from the output layer and moving backward through the network.
- Before backpropagation, neural networks were primarily limited to linear models with simple learning capabilities.
- The introduction of backpropagation demonstrated that neural networks could approximate any continuous function, given sufficient data and appropriate network architecture.

DARTMOUTH ENGINEERING | NVIDIA.

1986 - Multi-layer Perceptron

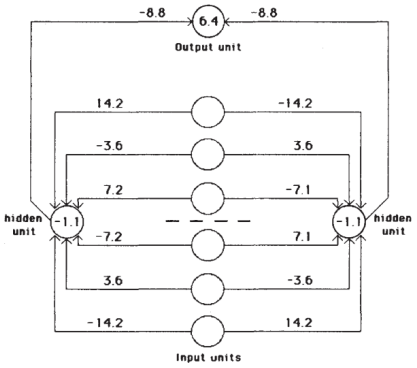
In the same backpropagation paper from 1986, the multi-layer perceptron was demonstrated to train to classify classes



Learning Internal Representations: The authors demonstrated that MLPs could learn internal representations in hidden layers, which are crucial for capturing complex features in data. This ability to learn hierarchical representations was a significant advancement over previous models.

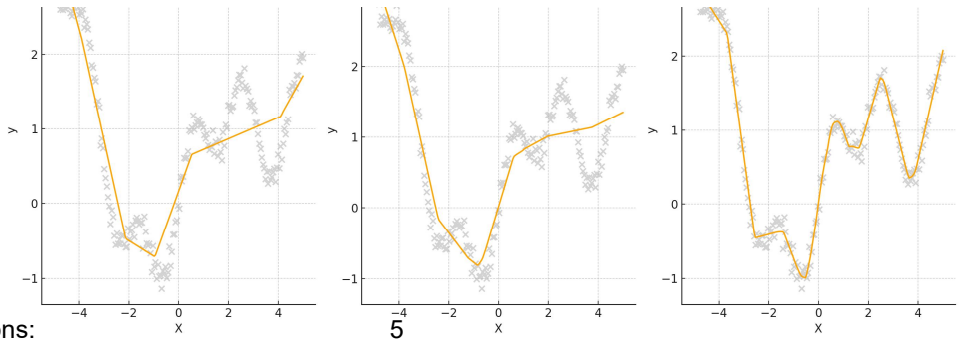
Non-Linear Capabilities: The paper highlighted that MLPs with non-linear activation functions in the hidden layers could solve problems that linear models could not, such as XOR, which was a known limitation of single-layer perceptrons.

Empirical Validation: The 1986 paper provided empirical results showing that MLPs, trained using backpropagation, could successfully perform tasks like pattern recognition, proving the effectiveness of the approach in practical applications.



1989 – Universal Function Approximation Theory

The Universal Approximation Theorem, proposed by George Cybenko in 1989 states that a feedforward neural network with a single hidden layer, containing a finite number of neurons and using a non-linear activation function (like the sigmoid function), can approximate any continuous function, given sufficient neurons in the hidden layer.



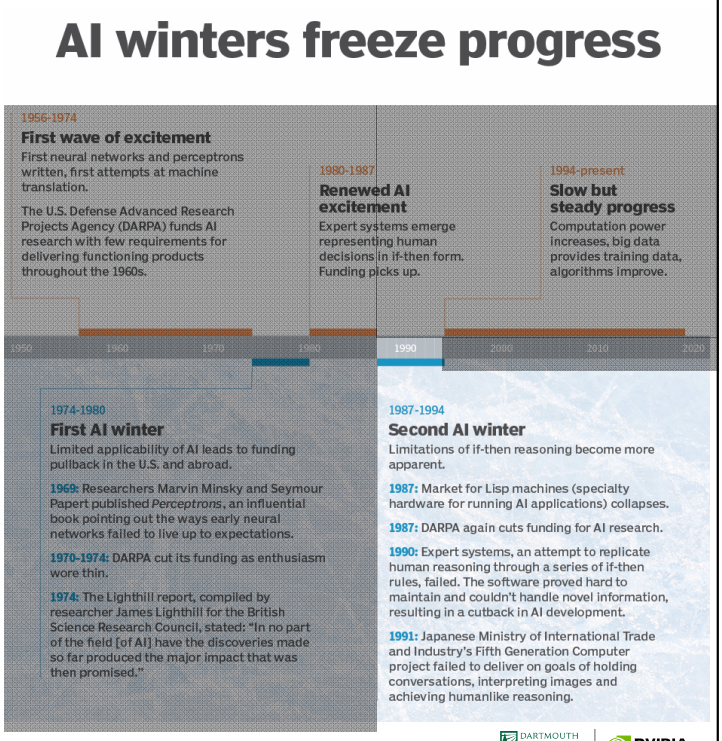
Neurons:
5
10
100

This theorem provided a theoretical foundation for the power of neural networks, showing that even simple networks could, in theory, approximate complex functions, which justified their use in a wide range of applications.

The 2nd AI Winter **1980s-1990s**

What led to the second AI winter?

1. **Unmet Expectations:** Expert systems, initially promising, failed to generalize beyond narrow domains, leading to disillusionment.
2. **Economic Recession:** The late 1980s recession led to budget cuts in AI research, shifting focus to more immediate technologies.
3. **Technological Limits:** Insufficient computational power and algorithmic challenges hindered AI development.
4. **Reduced Interest:** Funding and interest in AI dropped, leading to a slowdown in research and fewer innovations.
5. **Legacy and Recovery:** Lessons from this period set the stage for the AI resurgence in the late 1990s, paving the way for modern AI advancements.



2012 AlexNet – Computer Vision and GPUs Dominate AI

AlexNet was a groundbreaking neural network that significantly advanced the field of AI by demonstrating the power of deep learning and GPU acceleration

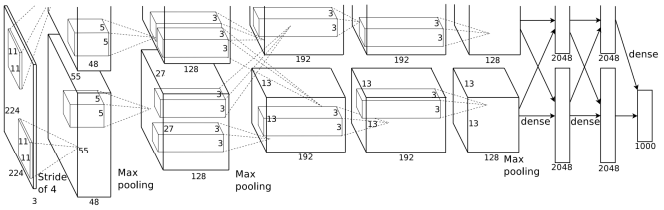
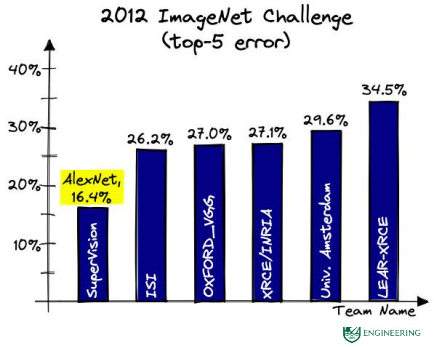


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Revolutionary Performance:
AlexNet, developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, won the 2012 ImageNet competition with a large margin.

GPU Acceleration:
AlexNet utilized GPUs (specifically NVIDIA GPUs) to dramatically reduce the training time for deep neural networks. This approach allowed the model to be trained on a much larger scale.

Catalyst for AI Growth:
The success of AlexNet spurred widespread adoption of deep learning and GPUs in AI research and industry.

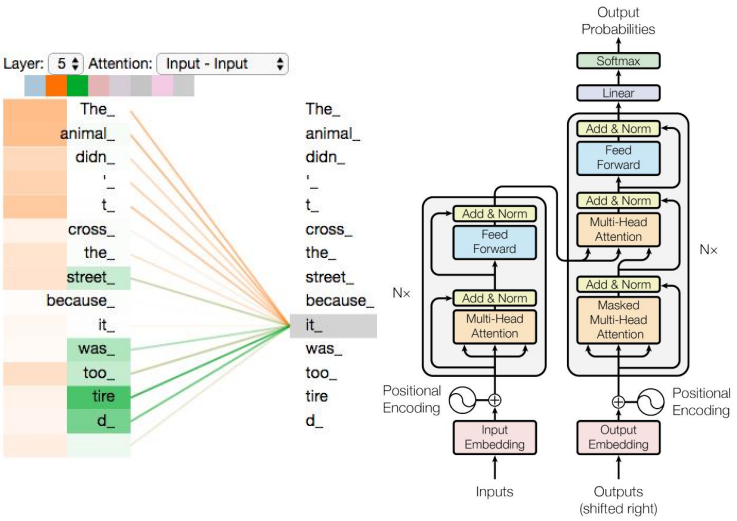


2017++: Attention is all you Need – Everything Changes

In the 2017 paper, Attention is all you need, a new deep learning architecture, the Transformer was introduced.

This model, and its variants: BERT and GPT have been found to solve traditional natural language problems and are capable of generating complex sematic outputs

These text generation models, together with the image generation models constitute Generative AI and will be the focus moving forward.



DARTMOUTH ENGINEERING | NVIDIA.

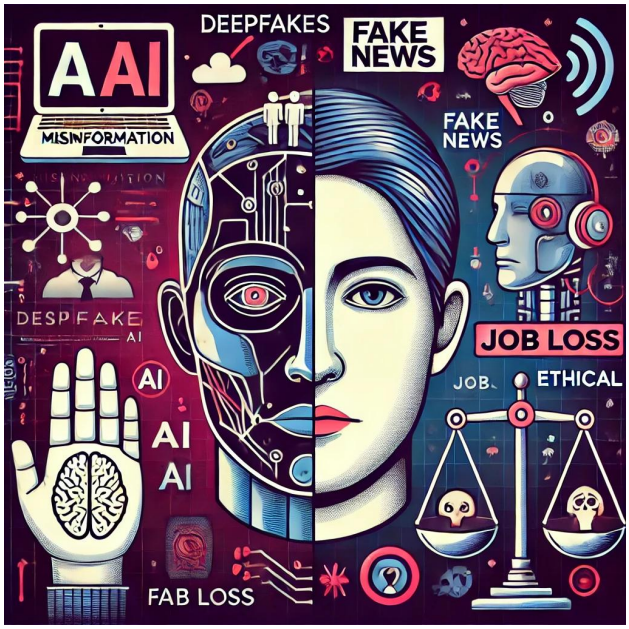
The Risks of GenAI

DARTMOUTH ENGINEERING | NVIDIA.

Risks of GenAI

For all its technological wonder, GenAI poses a serious challenge some fundamental parts of modern society:

- Trust and Information
- Digital records of events and people
- Jobs



Post-Truth Era?

GenAI tools can generate convincing fake news articles, social media posts, and even entire websites.

This capability allows for the rapid and widespread dissemination of misinformation, making it challenging for individuals and fact-checkers to identify and counter false narratives.

People are increasingly skeptical of the authenticity of the information they encounter, leading to a fragmented and polarized society.

Deepfakes

Deepfakes are AI-generated synthetic media, typically videos or images, where someone's likeness is realistically replaced with another's, often leading to convincing yet false representations.

These can be used for entertainment, but also to spread misinformation, create fake news, or manipulate public perception, posing significant ethical and security risks.

Job losses due to GenAI

Automation of Routine Tasks:

AI is automating repetitive jobs like data entry and manufacturing, leading to job reductions in these areas.

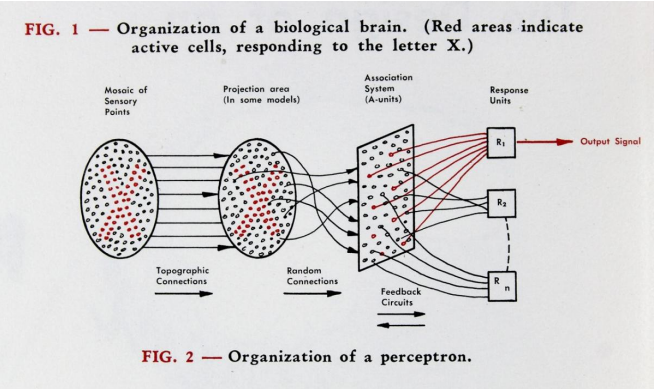
Impact on Skilled Jobs:

AI is increasingly capable of handling complex tasks, potentially displacing roles in fields like law, medicine, and finance.

Wrap Up

History and Risks of GenAI

- Today we introduced Generative AI (GenAI)
- We walked through the history of important developments in AI on the road to GenAI
- We introduced some of the risks of GenAI in modern society

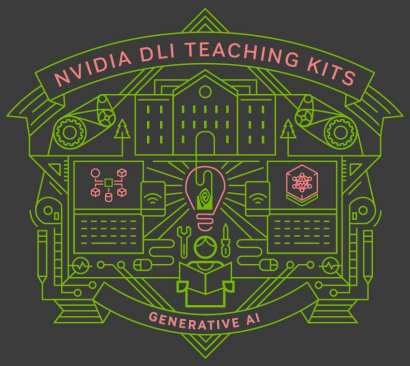


NVIDIA | DARTMOUTH ENGINEERING

Thank you!

Lecture 3.1 - Language Models and Attention

Generative AI Teaching Kit



This lecture

- Language Models in Deep Learning
- Evolution of attention mechanisms pre-transformers
- Attention Mechanism

Language Models in Deep Learning

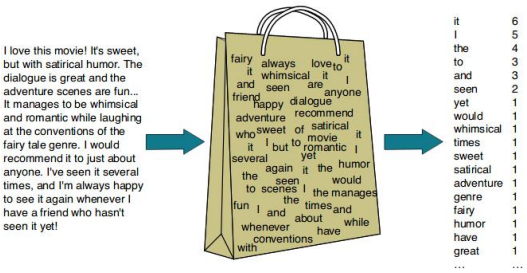
Deep Language Modeling

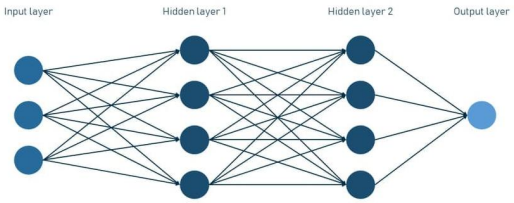
Recall that a goal of a language model is to predict the correct word based on the given input and the available corpus.

We have already seen statistical methods like Bag of Words which use the statistics of the prevalence of words in the dataset to predict the most likely next word.

But deep learning typically allows for more complex features to be encoded into a model.

Using deep learning for language, we can explore more complex relationships and more efficiently make use of the wealth of language data that is present digitally.



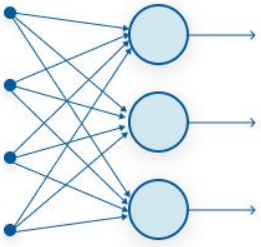


Challenges of Deep Learning Sequences

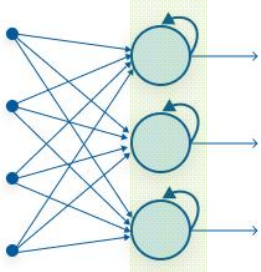
One main issue that standard neural networks run into when attempting to model language is time-dependency.

Language is **causal**, previous words influence future words, the same is true for sentences and paragraphs.
Neural networks don't have an inherent way to "**remember**" previous inputs, or to properly represent sequences.

A newer innovation, the **recurrent** neural network is designed to process sequential data by maintaining a form of memory through its recurrent connections.



Feed-Forward Neural Network



Recurrent Neural Network

Recurrent Neural Networks

Unlike traditional feedforward networks, RNNs are designed to recognize patterns in sequences of data, such as time-series, text, speech, or videos, by maintaining a hidden state that captures information about previous inputs in the sequence.

Key Characteristics of RNNs:

Sequential Processing:

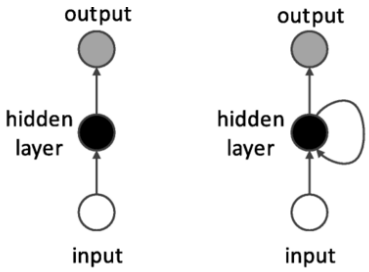
RNNs process input one step at a time, making them well-suited for tasks where data has a temporal or sequential structure.

Recurrent Connections:

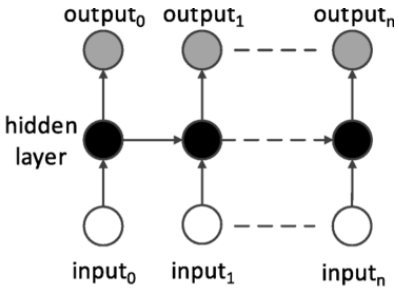
At each time step, the hidden state of the network is updated based on the current input and the hidden state from the previous time step. This allows the network to "remember" information from earlier in the sequence.

Shared Weights:

The weights used for processing are shared across time steps, which reduces the number of parameters and helps capture temporal dependencies.



(a) A traditional neural network (b) RNN



(c) RNN in unfolded form

Recurrent Neural Networks – Code Differences

```python
# ANN: Feedforward Neural Network
class SimpleANN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleANN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)  # Fully connected layer
        self.relu = nn.ReLU()                          # Activation function
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

```python
# RNN: Recurrent Neural Network
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleRNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)  # Recurrent layer
        self.fc = nn.Linear(hidden_size, output_size)  # Output layer

    def forward(self, x):
        out, _ = self.rnn(x)  # The RNN returns all outputs and the final hidden state
        out = self.fc(out[:, -1, :])  # Take the last time step's output for prediction
        return out
```

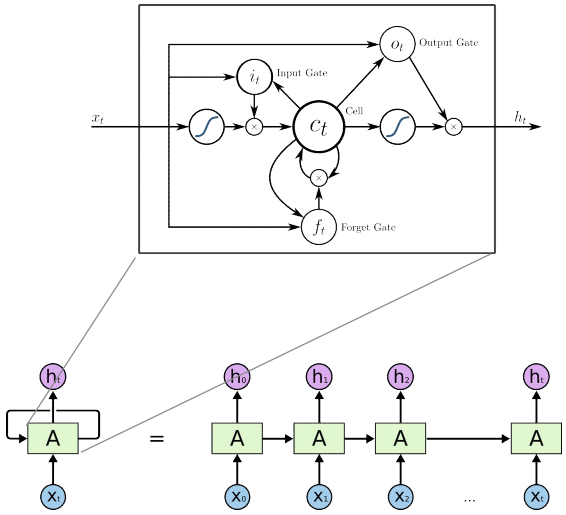
Key Differences

- **ANN**: Processes inputs without considering temporal relationships; uses a simple Linear layer.
- **RNN**: Processes sequences with recurrent connections; uses an RNN layer and captures sequential dependencies.

Long-Short Term Memory Models (LSTM)

Challenges with RNNs

- Struggle with long input sequences: Information from earlier words gets diluted as it propagates through the sequence.
- **Vanishing Gradient Problem**: Gradients shrink over time, reducing the ability to learn long-term dependencies.



Solution: LSTMs

- Introduce a "cell state" to selectively retain or discard information.
- Effectively capture long-range dependencies in sequences.

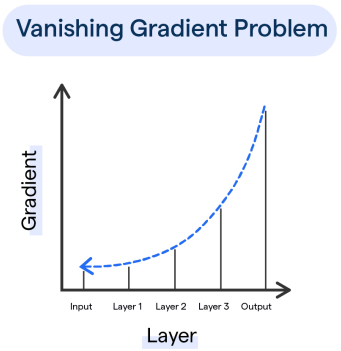
Limitations of RNNs and LSTMs in Language Modeling

Sequential Processing Bottleneck
- RNNs and LSTMs process inputs step-by-step, making training and inference slow for long sequences.

Vanishing Gradients
- Gradients diminish as they backpropagate through many time steps, limiting the network's ability to learn relationships across long sequences.

Challenges with Long-Range Dependencies
- Even with LSTMs, retaining information from distant parts of a sequence is difficult, leading to a loss of context over time.

Focus on Local Context
- RNNs and LSTMs prioritize immediate neighboring words but struggle to model relationships across the entire input effectively.



The solution?
A new mechanism that would enable parallel process, dynamically focusing on relevant sequence parts, and capturing long-range dependencies without the limitations of step-by-step computing or vanishing gradients…

Evolution of attention mechanisms pre-transformers

Adding Attention

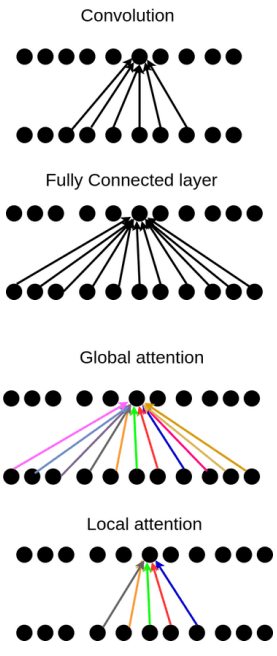
What is Attention?

Attention is a mechanism that enables a model to focus on the most relevant parts of the input while making predictions.

Instead of treating all input information equally, it assigns varying levels of "importance" (weights) to different parts based on the task.

Key Idea:

When processing sequences, attention computes a weighted sum of the input elements, where the weights represent how much "attention" each element deserves.



Early Attention Attempts – Bahdanau et.al 2014

Bahdanau et.al introduced a mechanism to dynamically focus on specific parts of the input sequence while generating each element of the output sequence.

- The encoder produces a set of **context vectors** (hidden states) for each input token.
- For each output token, the decoder calculates an **attention score** for each input token based on its relevance to the current decoding step.
- The scores are norm … h are u … **ighted sum of encoder hidden sta**
- This context vector is … en.

Core Formula

Attention weight ($\alpha_{t,i}$):

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})}$$

Where $e_{t,i} = \text{score}(s_t, h_i)$ is a learned scoring function (additive).

Context vector:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$$

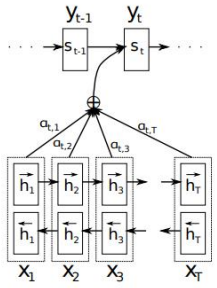


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Improved Attention – Luong et.al 2015

Luong et.al introduced in the paper *"Effective Approaches to Attention-based Neural Machine Translation"* by Minh-Thang Luong et al., this innovation focused on improving the computational efficiency and flexibility of attention mechanisms.

Multiplicative Attention (Dot-Product Scoring):

- Replaced Bahdanau's additive scoring function with a simpler **dot-product** or **scaled dot-product** function to calculate attention scores.
- Resulted in faster computations while maintaining strong performance.

Global vs. Local Attention:

Proposed two types of attention mechanisms:

1. **Global Attention**: Attends to all input tokens for each output token (like Bahdanau et.al).

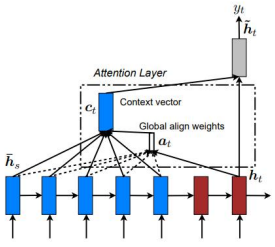


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector a_t based on the current target state h_t and all source states $\bar{h}_s$. A global context vector c_t is then computed as the weighted average, according to a_t, over all the source states.

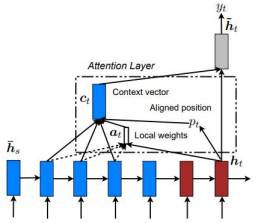


Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector c_t, a weighted average of the source hidden states in the window. The weights a_t are inferred from the current target state h_t and those source states $\bar{h}_s$ in the window.

Early Self-Attention – Lin et.al 2017

Lin et al. introduced a self-attention mechanism to create structured, multi-aspect sentence embeddings, enabling models to focus on different parts of a sentence simultaneously.

- **Self-Attention for Sentence Embeddings:**
Dynamically assigns attention weights to input tokens, emphasizing their importance to the sentence representation.

- **Multi-Aspect Representations:**
Captures diverse aspects of a sentence by generating **multiple attention vectors (hops)**, enabling richer embeddings.

- **Regularization for Diversity:**
Introduced a **penalty term** to ensure attention focuses on distinct sentence parts, reducing redundancy.

Key Formula for Attention:

Attention scores α_i:

$$\alpha_i = \text{softmax}(w_a^\top \tanh(W_h h_i))$$

Where:

- h_i: Hidden state of the word.
- w_a: Learnable weight vector.
- W_h: Learnable weight matrix.

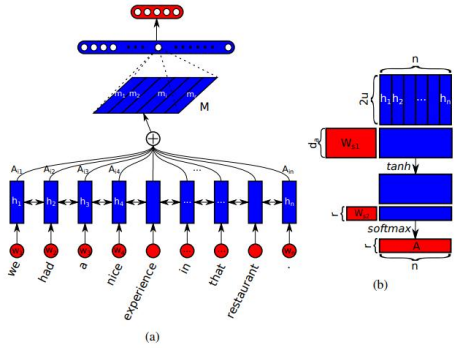


Figure 1: A sample model structure showing the sentence embedding model combined with a fully connected and softmax layer for sentiment analysis (a). The sentence embedding M is computed as multiple weighted sums of hidden states from a bidirectional LSTM ($\mathbf{h_1}, ..., \mathbf{h_n}$), where the summation weights ($A_{i1}, ..., A_{in}$) are computed in a way illustrated in (b). Blue colored shapes stand for hidden representations, and red colored shapes stand for weights, annotations, or input/output.

Remaining Limitations with Attention Methods

Despite the novel innovations of attention methods, these approaches still suffered from some general limitations, preventing their widespread use.

Dependence on Recurrence

- Attention mechanisms (e.g., Bahdanau, Luong, Lin et al.) were tightly integrated with RNNs/LSTMs, which process sequences sequentially.
- Gradient Issues: The reliance on recurrence also made them susceptible to vanishing or exploding gradients, limiting their ability to model very long dependencies.

Lack of Scalability

- RNN/LSTM-based models with attention were computationally expensive and struggled with large datasets or sequences.
- Memory Usage: Maintaining hidden states for long sequences was resource-intensive.

Inefficient Training

- Training LSTM-based models with attention was slow because of sequential dependencies and the need to process data step-by-step.

The Self-Attention Mechanism

Attention is all you need – Vaswani et.al 2017

One of the most seminal papers of machine learning to date. “Attention is all you need” introduced a new concept of how to make use of attention and completely removed the reliance on recurrence to process sequences. In the next lesson we will dive deeper into the Transformer model, but now we will focus on how self-attention is presented in that paper.

Self-Attention: What Does It Actually Mean?
Self-attention is a mechanism that allows a model to dynamically focus on the most relevant parts of a sequence when computing representations for each token.

Key Concept:
Every token in the sequence can *attend* to every other token, including **itself**, to understand its relationship and importance in the context of the entire sequence.

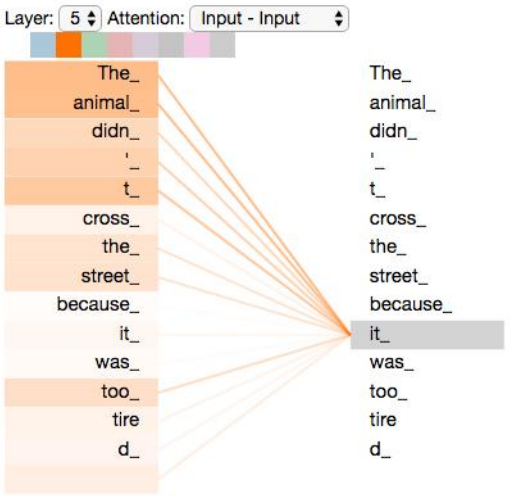
E.g.
In the sentence: "**The cat chased the mouse**,"

Self-attention helps the model understand that "*the mouse*" is what "*the cat*" **chased**, by focusing on the semantic relationship between "chased" and "the mouse."

How Self-Attention Mechanisms Work

The implementation of self-attention can vary, but the essence is to:

- **Compare:** Each token is compared to every other token in the sequence to compute relevance scores.
- **Weight:** Assign weights to other tokens based on their relevance to the current token.
- **Aggregate:** Combine information from all tokens, weighted by their relevance, to compute a new representation for the current token.



Global Context
Captures relationships between tokens across the entire sequence, not just local neighbors.

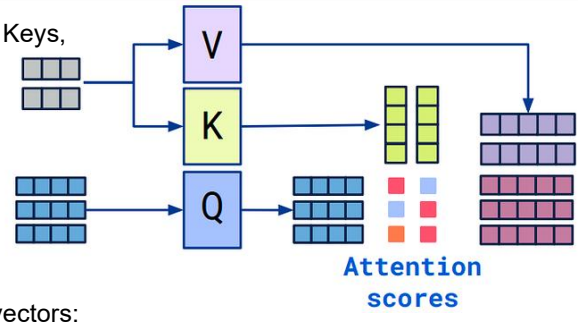
Dynamic Focus
The model decides what to focus on for each token, rather than relying on fixe patterns (e.g., sliding windows in convolution).

Flexibility
Works for variable-length sequences and tasks requiring both short- and long-range dependencies.

Queries, Key, and Values

In the Attention is all you need paper, a novel algorithm, based on Queries, Keys, and Values is presented to handle the attention calculations:
QKV allows each token to decide:

- What it wants to know (**Query**).
- What information it can provide (**Key**).
- The actual data it contributes to the result (**Value**).



Step 1: Create Q, K, V
Each input token (e.g., word embedding) is linearly transformed into three vectors: Query (Q), Key (K), and Value (V).

Step 2: Compute Attention Scores:
Compare the Query of a token with the Keys of all tokens in the sequence to measure relevance. Use the dot product to capture similarity.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Step 3: Normalize Scores (SoftMax):
Apply SoftMax to the attention scores to ensure they sum to 1, producing attention weights.

Step 4: Aggregate Values:
Multiply the attention weights by the corresponding Values and sum them to produce the output representation for each token.

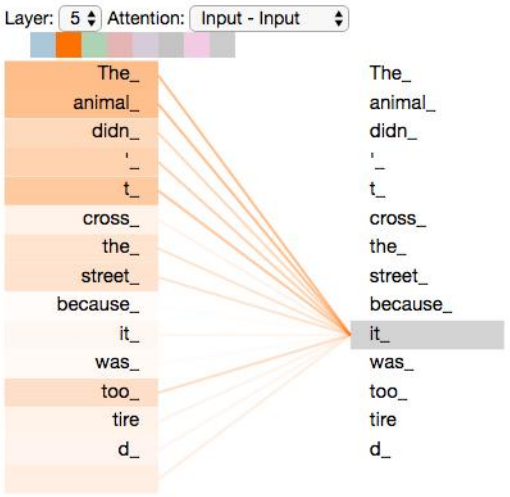
Wrap Up

Language Models and Attention

- Today we introduced the concepts of deep learning in the context of language models.
- We saw some of the older model architectures like recurrent neural networks that were used to model sequence data as an improvement over traditional feedforward neural networks.
- The concept of attention mechanisms was also discussed as a means to add extra information between parts of sequences
- As a precursor to the introduction of the full transformer model, we looked back at the evolution of attention models in deep learning.
- Finally, we presented the self-attention mechanism that provided the breakthrough needed for modern LLMs and Transformer-based models

---

In the next class we will explore the transformer model.



NVIDIA | DARTMOUTH ENGINEERING

Thank you!

Lecture 3.2 - The Transformer Architecture

Generative AI Teaching Kit

This lecture

- Recap on challenges with LSTMs
- Attention is all you need – breakthrough
- The Transformer Block
- Multi-headed Attention
- Positional Encoding
- Original Transformer Architecture

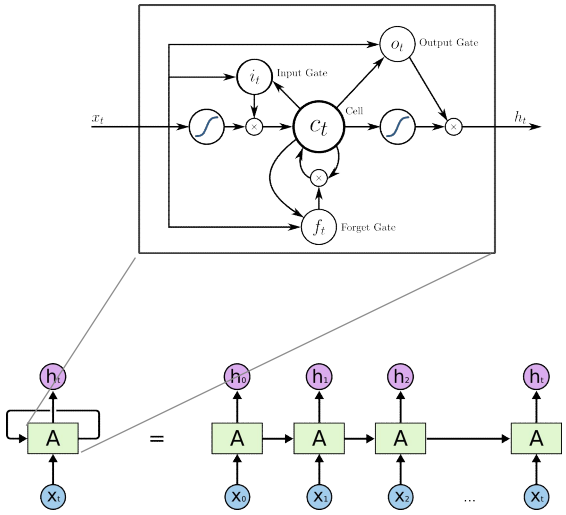
Recap on challenges with LSTMs

The limitations with sequence deep learning models

Last time we saw that deep learning language models required a special treatment of the causal-sequence nature of language.

This meant that special architectures like Recurrent Neural Networks and LSTMs became the standard for language modeling in deep learning.

These models used memory components to keep track of parts of sequences but still lacked some important components and encountered limitations.



The limitations with sequence deep learning models

One key missing feature to vanilla RNNs and LSTMs was the concept of **attention.**

Attention is defined as a mechanism that enables a model to focus on the most relevant parts of the input while making predictions.
Instead of treating all input information equally, it assigns varying levels of "importance" (weights) to different parts based on the task.

Innovations began to emerge in the late 20-`teens` but still all focused-on applications of the LSTMs/RNN models and were hampered by those models' limitations.

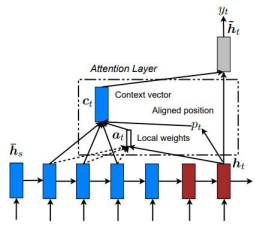


Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights $\boldsymbol{a}_t$ are inferred from the current target state $\boldsymbol{h}_t$ and those source states $\bar{\boldsymbol{h}}_s$ in the window.

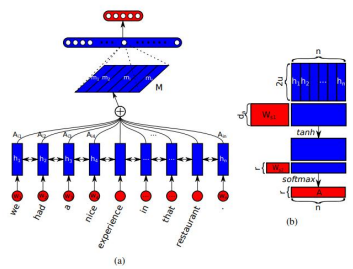


Figure 1: A sample model structure showing the sentence embedding model combined with a fully connected and softmax layer for sentiment analysis (a). The sentence embedding M is computed as multiple weighted sums of hidden states from a bidirectional LSTM ($\mathbf{h}_1, ..., \mathbf{h}_n$), where the summation weights ($A_{11}, ..., A_{1n}$) are computed in a way illustrated in (b). Blue colored shapes stand for hidden representations, and red colored shapes stand for weights, annotations, or input/output.

Attention is all you need – breakthrough

Attention is all you need – the game changing paper

In 2017, a paper was presented at the 31st Neural Information Processing Systems Conference.

This paper introduced a new model architecture, the transformer, which **uses as a central point, self attention**, and **removed any need for recurrence or convolution layers**.

The focus of this new architecture on only attention, and some feedforward networks, meant that this model could process sequences in parallel, with drastically improved efficiency in training.

Attention Is All You Need

Ashish Vaswani[*] Google Brain avaswani@google.com
Noam Shazeer[*] Google Brain noam@google.com
Niki Parmar[*] Google Research nikip@google.com
Jakob Uszkoreit[*] Google Research usz@google.com
Llion Jones[*] Google Research llion@google.com
Aidan N. Gomez[*] [†] University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com
Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

[*]Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

[†]Work performed while at Google Brain.

[‡]Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Attention is all you need – the game changing paper

This new model, the Transformer, was shown to outperform state-of-the-art machine translation models using a fraction of the training cost.

The benefits of this architecture was also in the generalizable nature of its design. This model was not designed for machine translation specifically and could be re-trained for other tasks such as English constituency parsing.

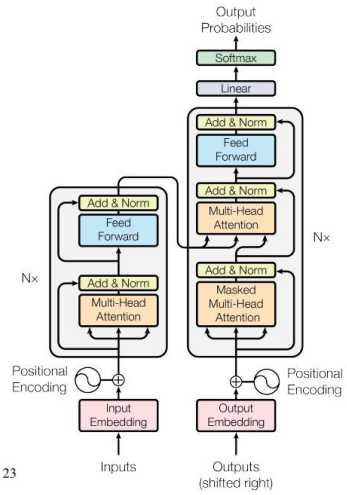


Figure 1: The Transformer - model architecture.

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

The Transformer Block

The Transformer Block

1. Multi-Head Attention (Orange Block):

1. **Input**: Queries (Q), Keys (K), and Values (V).
2. **Process**: Attention weights are calculated to decide how much focus each token should give to others in the sequence. Multiple heads capture different types of relationships.
3. **Output**: A weighted representation of the sequence.

2. Add & Norm (Yellow Block):

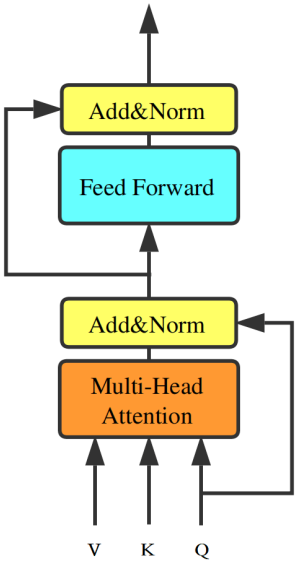
1. The output of the Multi-Head Attention is **added** back to the input (residual connection) to preserve the original information.
2. Then, it is **normalized** to stabilize training and ensure smooth gradient flow.

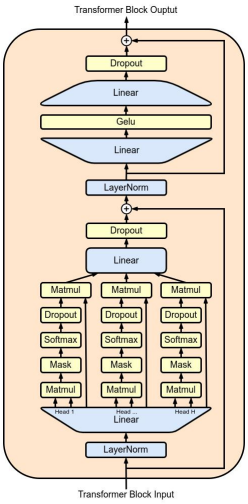
3. Feed Forward (Blue Block):

1. A fully connected network processes each token independently, adding non-linearity and helping the model capture complex patterns.

4. Add & Norm (Yellow Block):

1. The output of the Feed Forward layer is **added** back to its input (another residual connection).
2. **Normalization** is applied again for stability.





The Transformer Block – Tokenization and Word Embeddings

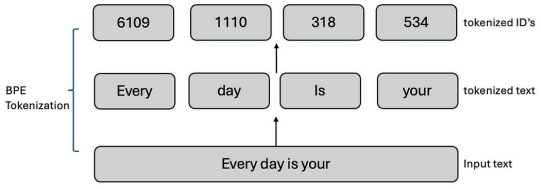
Preparing inputs for the Transformer involves three main steps:

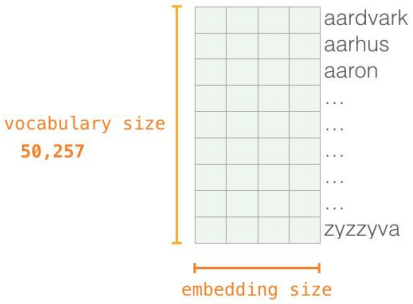
Tokenization

- The text input is split into smaller units called **tokens**. These tokens could represent words, subwords, or even individual characters, depending on the tokenization strategy.
- Each token is assigned a unique identifier based on the model's vocabulary, converting the text into a numerical sequence.

Word Embedding

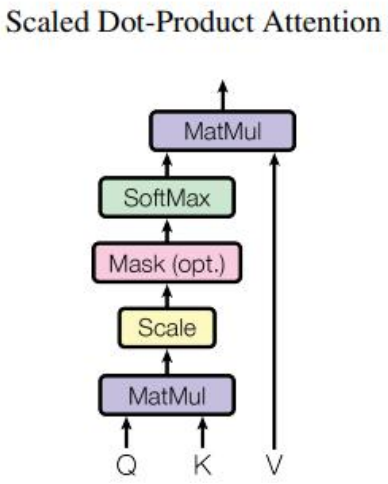
- Each token ID is mapped to a high-dimensional vector using a pre-trained or learned embedding matrix.
- These embeddings capture semantic meaning, allowing the model to understand relationships between words or tokens (e.g., synonyms or contextual similarities).





Positional Encoding*

Scaled Dot-Product Attention



Definition:
Attention maps a query and a set of key-value pairs to an output, where:
- **Query (Q):** What you're looking for.
- **Key (K):** What the data is indexed by.
- **Value (V):** The information being retrieved.

How It Works:
1. Compute the dot product of the query with all keys to measure compatibility.
2. Scale the result by dk\sqrt{d_k}dk to prevent large values.
3. Apply a **SoftMax function** to convert scores into probabilities (weights).
4. Use the weights to compute a weighted sum of the values.

Key Points:

The Transformer Block – Position-wise Feedforward Network

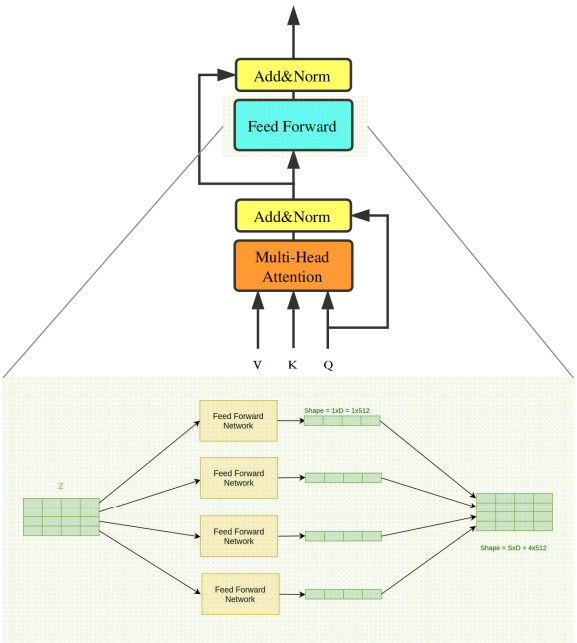
If we look at the progress of a token as it passes through the Transformer, we can see that it is processed in the following ways:

- Tokenization – *Linear Operation*
- Word Embedding – *Linear Operation*
- Self-Attention – *Linear Operation*

So, how does non-linearity get added into the Transformer? Ans: **Position-wise Feedforward Neural Network**

What is it and how does it work?
A feedforward neural network is applied independently to each token's embedding at each position in the sequence. This adds non-linearity and expressive power to the Transformer. Each feedforward layer consists of two fully connected layers with a non-linear activation in between

Why It Works:
Non-linearity (e.g., ReLU) enables the model to capture complex relationships that linear operations cannot. The position-wise nature means each token is processed independently, preserving parallelizability.

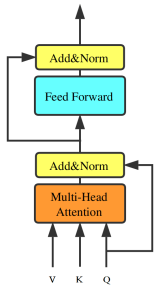


The Transformer Block – Residual Connection and Normalization

Residual Connections

Residual connections, also called skip connections, allow the input of a layer to bypass the layer and be added directly to its output.

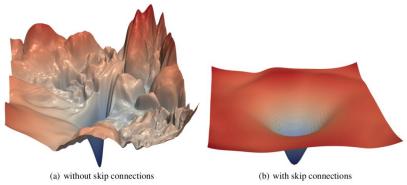
- **Facilitate Gradient Flow:** Prevent vanishing or exploding gradients in deep networks by ensuring gradients can propagate back through the network effectively.
- **Improve Convergence:** Enable faster training by providing a "shortcut" for information.
- **Preserve Information:** Retain information from earlier layers, which might otherwise be overwritten during training.

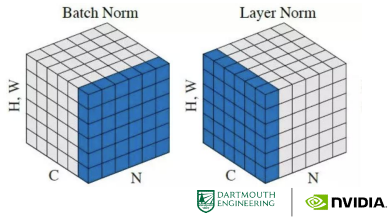


Layer Normalization

Layer normalization standardizes the activations across the features of a token at each position, ensuring that the input to each layer has zero mean and unit variance.

- **Stabilize Training:** Normalization ensures consistent scaling of activations, making training more stable.
- **Prevent Exploding Activations:** Keeps activations within a manageable range, preventing instability.
- **Enable Robust Generalization**: Helps the model generalize better by reducing sensitivity to the scale of the input data.





Why Both?

- Residual connections ensure information flow across the network.
- Layer normalization stabilizes the output at each step, improving gradient flow and model convergence.

Multi-headed Attention

Self-Attention – Multiheaded Attention

The authors also implemented “multi-headed attention” to address the limitations of single-headed attention, which could only focus on one type of relationship or dependency at a time.

Parallel Attention Mechanisms:
Multi-headed attention allows the model to focus on different parts of the input sequence simultaneously, capturing diverse relationships (e.g., word-to-word, phrase-to-phrase).

Improved Representations:
Each "head" learns unique attention patterns, enabling the model to extract more nuanced information, such as syntactic and semantic dependencies.

Scalability and Flexibility:
By splitting the computation into smaller attention heads, the model efficiently handles large inputs and produces richer, context-aware embeddings.

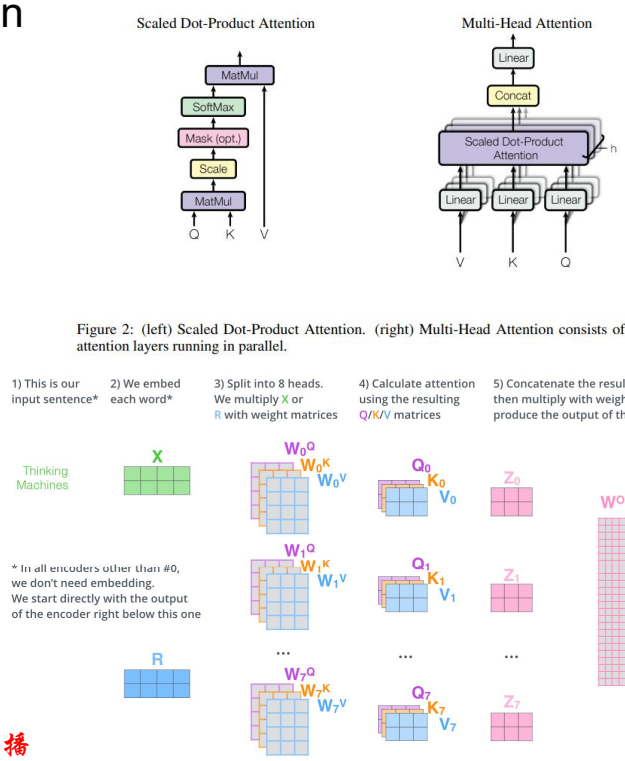


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Variations of Multiheaded Attention

GQA (Grouped-Query Attention)
Reduces the computational cost by grouping multiple queries together and computing attention collectively rather than individually.
Key Advantage: Drastically lowers memory and compute requirements, especially for large-scale models like GPT-3.

Multi-Query Attention
Instead of having separate key-value pairs for each query, all queries share the same set of key-value pairs.
Key Advantage: Reduces the memory overhead of storing multiple key-value pairs without significantly impacting model performance.

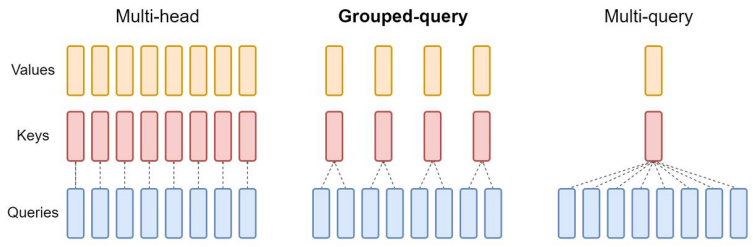


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

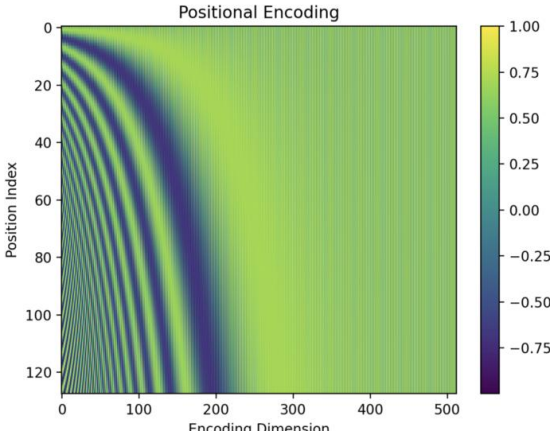
Positional Encoding

Positional Encoding

The original Transformer relies on positional encoding to inject information about the order of tokens into the model. The self-attention mechanism itself lacks positional awareness, and in language we need to preserve this sequential understanding.

How Positional Encoding Works

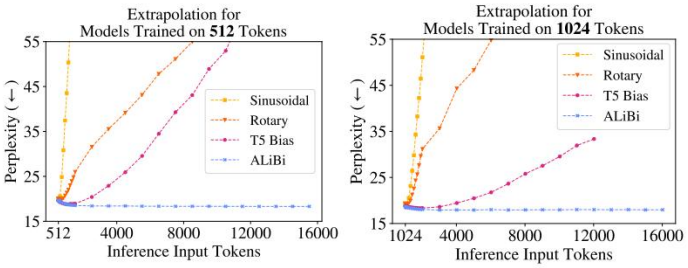
- Positional encodings are added to the input embeddings to encode the order of tokens in the sequence.
- The sinusoidal functions used in the original paper allow the model to generalize to sequences longer than those seen during training because the values are periodic and predictable.
- Relative positions between tokens are naturally encoded through phase differences in sine and cosine values.



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Limitations of Sinusoidal Positional Encoding

While sinusoidal positional encodings from the original Transfor
paper have been highly effective, they come with notable limitat



Fixed and Non-Adaptive:

Sinusoidal encodings are fixed and do not adapt to specific datasets or tasks. This inflexibility can limit their ability to capture task-specific positional information.

Absolute Position Encoding:

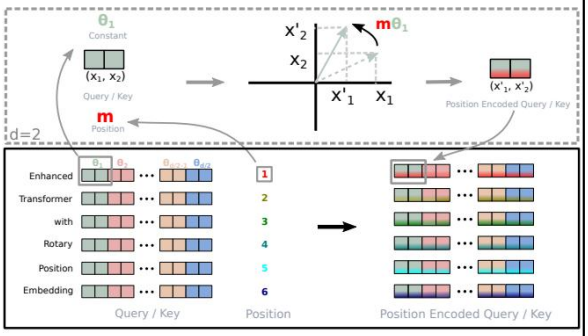
They encode absolute positions rather than explicitly modeling relative distances between tokens. While the differences in sine/cosine phases indirectly encode relative positions, this is not optimized for tasks requiring strong relative positioning (e.g., question answering or summarization).

Lack of Scalability:

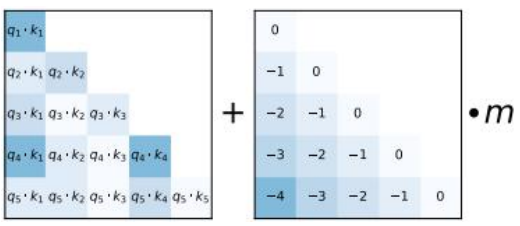
When handling very long sequences, the sin \sinsin and cos cos cos
functions may lose precision due to numerical instability in their periodicity

Improvements to Sinusoidal Positional Encoding

Rotary Positional Embedding (RoPE)
Rotary Positional Embedding introduces relative position encoding directly into the attention mechanism, making the model more effective at capturing positional relationships. This transformation embeds relative positional information naturally into the attention scores without modifying the self-attention computation pipeline.
Key Advantages:
Relative Position Awareness: Unlike sinusoidal embeddings, RoPE explicitly models relative positions, which is crucial for tasks involving long-term dependencies.
Efficient for Long Contexts: Handles long sequences better than sinusoidal encodings due to its explicit relative positioning.



ALiBi (Attention with Linear Biases)
Attention with Linear Biases, modifies the attention mechanism by adding a learnable linear bias to the attention scores based on the relative distance between tokens.
Key Advantages:
Efficient Scaling: ALiBi introduces no additional parameters or overhead, as the bias is directly applied to attention scores.
Strong Relative Position Encoding: By modifying attention weights directly, ALiBi enhances the model's ability to capture relative positional information efficiently.
Supports Extrapolation: ALiBi enables models to generalize better to longer sequences during inference, even when trained on shorter sequences.



Original Transformer Architecture

Cross-Attention

Cross-attention is a mechanism where one sequence (e.g., the decoder input) **attends to another sequence** (e.g., the encoder output).
This allows the model to integrate information from a source sequence (like an input sentence in machine translation) into the target sequence generation process.

In the Transformer architecture, cross-attention is implemented in the decoder block, where the decoder uses the encoder's output to guide the generation of the target sequence.

Cross-Attention Inputs
Key-Value Pairs (K, V):
These come from the encoder outputs.They represent the context learned from the source sequence (e.g., input sentence in a translation task).

Query (Q):
This comes from the decoder's previous layer (or the embeddings of the previously generated tokens in the decoder).

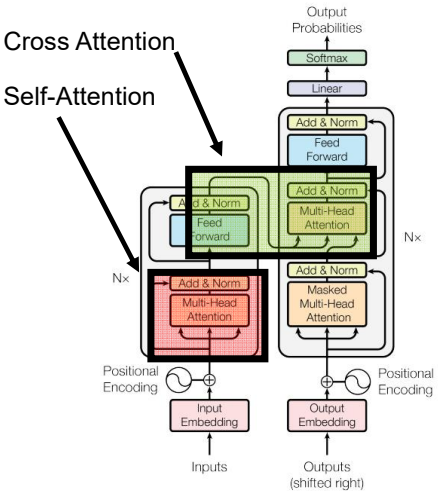


Figure 1: The Transformer - model architecture.

The Transformer

Now that we can see what is inside the transformer blocks, let's talk about the original transformer and what it was used for.

Architecture Details:
- Transformer Blocks/Layers: **6** encoder and **6** decoder layers.
- Hidden Dimension for the word embeddings: **1024**
- Feedforward Hidden Dimension: **4096** (ie a 4x expansion internally)
- Attention Heads for multi-headed attention: **16**
- Sequence Length: Up to 512 tokens.

Total Parameters: 213 million (213M)

Training Details
- Task: English to German Translation
- Batch Size: 131,072 tokens per batch
- Training Steps: 100 k
- Dataset: WMT 2014 English-to-German (4.5M sentence pairs).

Training Time: ~3.5 days on 8xP100 GPUs for English-to-German

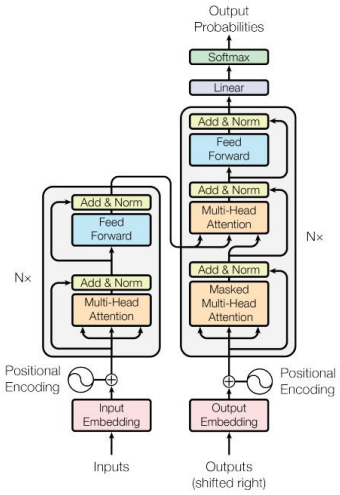


Figure 1: The Transformer - model architecture.

Wrap Up

The Transformer Architecture

- Today we introduced one of the most exciting innovations of the last decade, the Transformer.
- We looked inside the model to see how each part of this architecture is constructed and why it offered the breakthrough in sequence analysis over existing models.
- We looked at the original attention mechanism in the attention is all you need paper and how it was scaled to produce the Transformer.
- Now that we have an understanding of the fundamental piece of Generative AI, we can look further and into even more exciting and exotic models.

In the next class we will explore what it takes to train a model like the transformer. The data preparation and the considers required for autoregressive modeling.

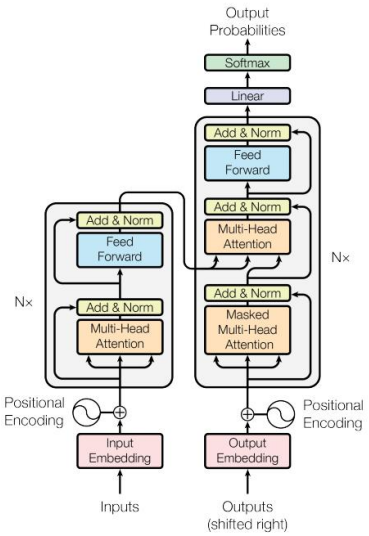
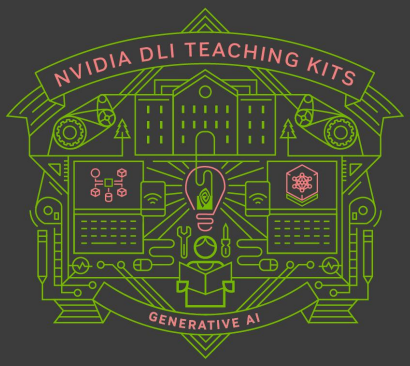


Figure 1: The Transformer - model architecture.

Thank you!

Lecture 4.1 - LLM Architecture Variants

Generative AI Teaching Kit



请勿传播

This lecture

- Review of the original Transformer
- Encoder only Models
- Decoder only Models
- Encoder-decoder models
- New Variants

Review of the original Transformer

Recap: The Original Transformer

Last time we saw the biggest innovation in machine learning in the last decade, the **Transformer**.

The paper, “Attention Is All You Need”, introduced a new paradigm for processing sequences in parallel with self-attention.

This model was used as a translation tool and was shown to outperform the state-of-the-art, even with a fraction of the compute requirements.

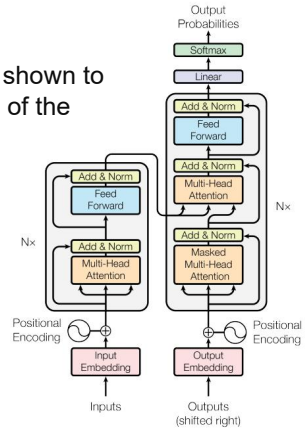


Figure 1: The Transformer - model architecture.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.
†Work performed while at Google Brain.
‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Breakthrough innovation

Architecture Details:

- Transformer Blocks/Layers: **6** encoder and **6** decoder layers.
- Hidden Dimension for the word embeddings: **1024**
- Feedforward Hidden Dimension: **4096** (ie a 4x expansion internally)
- Attention Heads for multi-headed attention: **16**
- Sequence Length: Up to 512 tokens.

Total Parameters: 213 million (213M)

Training Details

- Task: English to German Translation
- Batch Size: 131,072 tokens per batch
- Training Steps: 100 k
- Dataset: WMT 2014 English-to-German (4.5M sentence pairs).

Training Time: ~3.5 days on 8xP100 GPUs for English-to-German

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Composition and Task focus

This model architecture was composed of two components:

An Encoder:

- Processes the input sequence all at once.
- Maps the input tokens into continuous vector representations using self-attention and feedforward layers.
- Captures contextual relationships across the entire sequence.
- Used in tasks like text classification and feature extraction.

A Decoder:

- Generates output tokens one by one in an autoregressive manner.
- Attends to both the encoder's output and previously generated tokens using masked self-attention.
- Produces meaningful sequences based on learned patterns (e.g., translation, text generation).

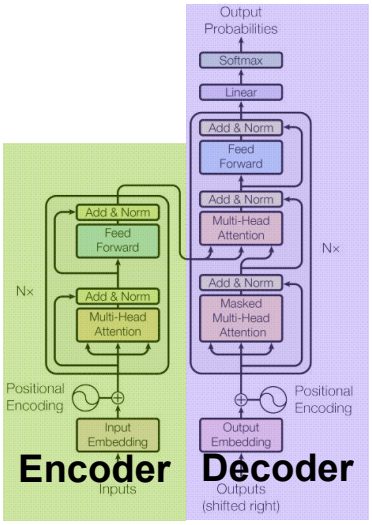


Figure 1: The Transformer - model architecture.

But do we always need both components?

Encoder only Models

Splitting the Transformer – BERT/Embeddings

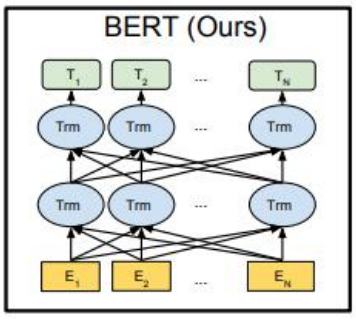
If we focus on just the encoder side of the original transformer, we end up with the **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike the original Transformer (which was designed for seq-to-seq tasks like translation), BERT **focuses on deep bidirectional context learning**.

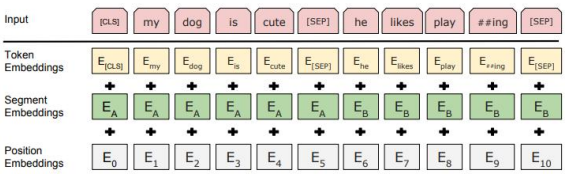
What Happens When We Keep Only the Encoder?

- The model **processes all tokens simultaneously**, rather than autoregressively generating output token-by-token.
- Instead of predicting the next token, BERT learns **contextual embeddings** using **masked language modeling (MLM)**.
- **No autoregressive decoding**, making it ideal for feature extraction rather than generation.

How Encoder Embeddings Work

- **Token Embeddings**: Each token is mapped to a high-dimensional contextualized vector.
- **[CLS] Token**: Represents a fixed-length sentence embedding, useful for tasks like classification or retrieval.
- **Layer-wise Representations**: Embeddings can be extracted from different encoder layers depending on the task.



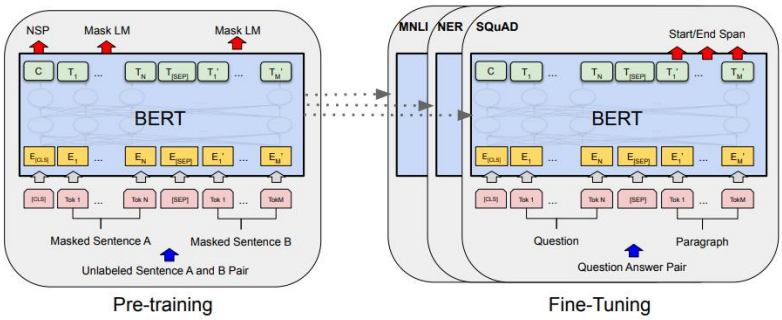


Training Encoder Models - BERT

Pretraining: Learning Universal Representations
BERT undergoes self-supervised learning on massive unlabeled text corpora using two key objectives:

- Masked Language Modeling (MLM)
  - Randomly masks 15% of tokens and trains the model to predict them.
  - Encourages deep bidirectional context learning by making the model rely on both left and right words.
- Next Sentence Prediction (NSP)
  - Given two sentences, the model predicts if the second follows the first.
  - Helps capture sentence-level relationships (though later models like RoBERTa remove this step).

Pretraining Output: A general-purpose transformer capable of understanding contextual meaning across various domains.



Fine-Tuning: Adapting BERT for Specific Tasks

- After pretraining, BERT is fine-tuned on labeled datasets for specific NLP applications. Fine-tuning involves:
- Adding a Task-Specific Output Layer
- Training on a Labeled Dataset

Fine-Tuning Output: A specialized BERT model optimized for task-specific performance (e.g., sentiment analysis, QA, information retrieval).

Evolution of BERT models

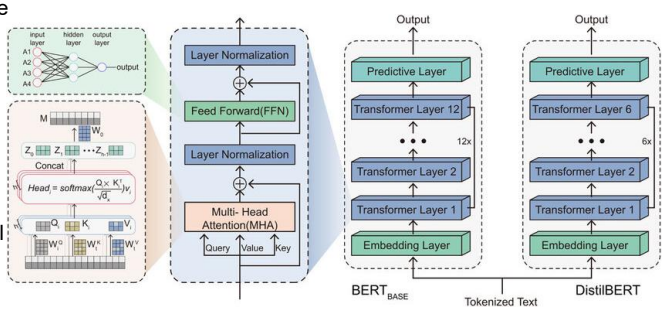
The Problem with Original BERT
Introduced bidirectional attention, making it strong at understanding context.
But: Computationally expensive, requires large-scale training, and Next Sentence Prediction (NSP) is ineffective.

RoBERTa – More Data, No NSP

- Removed NSP, trained on 10x more data with dynamic masking.
- Better accuracy but requires even more compute than BERT.

DistilBERT – Making BERT Faster

- Used knowledge distillation to create a lighter version of BERT.
- 60% fewer parameters, 60% faster inference, retaining 95% of original BERT's accuracy.



Why This Evolution Matters

- BERT started as powerful but inefficient → newer models optimized for speed, size, and compute efficiency.
- Trade-offs: More efficient models reduce size but may lose flexibility or require different training techniques.
- Modern NLP uses these advancements to deploy transformer models at scale, even on limited hardware.

SBERT and Cross-Encoder Models

SBERT Extending BERT for Sentence Similarity

Why Standard BERT is Inefficient for Sentence Comparison:
- BERT processes each sentence independently, meaning similarity nee to be computed after embedding extraction.
- This results in slow, inefficient comparisons, especially for large-scale retrieval tasks.

SBERT (Sentence-BERT)
How It Works
1. Encodes sentences separately using a shared BERT model.
2. Outputs fixed-length embeddings, which can be compared using cosin similarity.
3. Fine-tuned with tasks like semantic textual similarity (STS).

Advantages
- Fast similarity search – Ideal for retrieval and clustering.
- Fixed embeddings – Precomputed vectors allow efficient comparison.

Limitations
- Less precise for nuanced comparisons – Doesn't leverage full pairwise attention like cross-encoders.

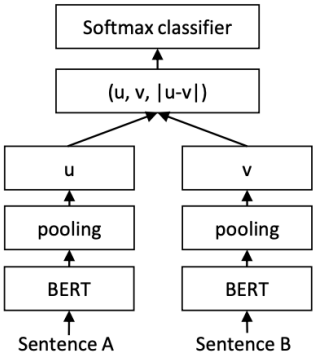


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

SBERT and Cross-Encoder Models

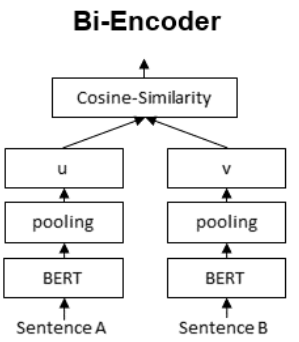
Cross-Encoders: Instead of processing sentences separately, a cross-encoder **jointly encodes both inputs**, allowing **full self-attention across sentences**.

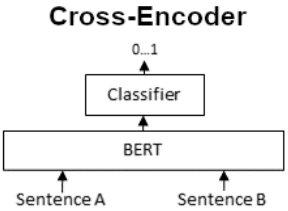
How It Works
1. The model takes both sentences **concatenated together** as input.
2. Outputs a **single scalar score** (e.g., similarity score for STS).
3. Typically fine-tuned with classification loss for ranking.

Advantages
- **Higher accuracy** – Fully utilizes attention across both sentences.
- **Better for ranking** – Used in **re-ranking pipelines** for search engines and QA.

Limitations
- **Much slower** – Requires **recomputing embeddings every time**, making it impractical for large-scale retrieval.

Choosing SBERT vs. Cross-Encoders
- **Need efficiency:** Use **SBERT** for retrieval tasks.
- **Need accuracy:** Use **Cross-Encoders** for ranking and fine-grained comparisons.
- Many NLP systems use **both together** – SBERT for initial retrieval, followed by a Cross-Encoder for refinement.





Decoder only Models

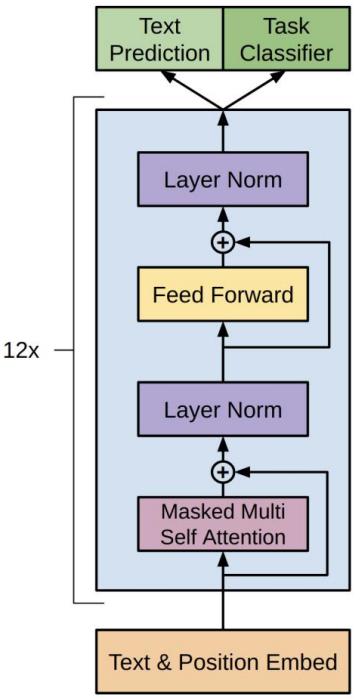
Splitting the Transformer – Autoregressive

Focusing on Just the Decoder: The Autoregressive Transformer
If we focus only on the decoder side of the original Transformer, we get an autoregressive model, such as **GPT** (Generative Pretrained Transformer). Unlike the original Transformer (which was designed for seq-to-seq tasks like translation), decoder-only models specialize in generating text token-by-token, making them ideal for open-ended text generation.

What Happens When We Keep Only the Decoder?
- The model generates tokens sequentially, predicting the next token based on previous tokens.
- Uses causal (unidirectional) self-attention, meaning tokens cannot attend to future words.
- Unlike encoder-based models (e.g., BERT), decoder-only models do not generate embeddings for entire sequences at once—they generate text step-by-step.

How Decoder-Based Models Generate Text
- Token-by-Token Generation: Each new token is conditioned on all previously generated tokens.
- Autoregressive Sampling: Uses methods like greedy decoding, beam search, or nucleus sampling to determine the next token.
- No Fixed-Length Representations: Unlike encoder models that output a static embedding, decoder models produce dynamic text sequences, making them useful for chatbots, story generation, and machine translation.

Key Differences from Encoder-Only Models
- Optimized for generation, rather than feature extraction.
- Uses causal self-attention, preventing tokens from seeing the future.
- Can generate arbitrarily long text, making them flexible for applications like chatbots, dialogue models, and creative writing.



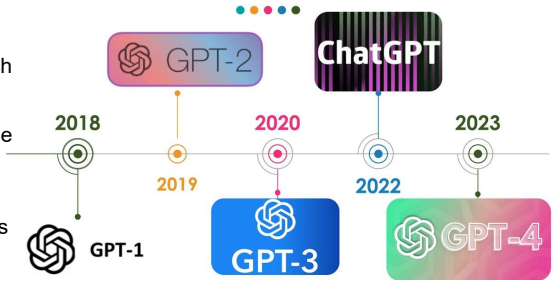
GPT and the Decoder Models

GPT-1 (2018) – Introducing Autoregressive Pretraining
The first Generative Pretrained Transformer (GPT-1) introduced the idea of pretraining on large text corpora using causal (unidirectional) self-attention. It was trained on BookCorpus (7000 books) using a left-to-right language model objective, meaning it predicted the next word based on previous context. While effective for text generation, it lacked fine-tuning capabilities for downstream tasks.

GPT-2 (2019) – Scaling Up and Zero-Shot Learning
GPT-2 significantly increased model size (up to 1.5B parameters) and was trained on a much larger dataset (WebText, 40GB of internet text). Unlike GPT-1, it demonstrated strong zero-shot learning, meaning it could perform tasks like translation and summarization without explicit fine-tuning. OpenAI initially withheld its release due to concerns over potential misuse in text generation.

GPT-3 (2020) – Massive Scale and Few-Shot Learning
GPT-3 pushed scale even further, with 175B parameters, making it one of the largest models of its time. It introduced few-shot learning, where the model could generalize to new tasks using only a few examples provided in-context. This made it far more flexible, reducing the need for task-specific fine-tuning. However, it was computationally expensive and sometimes generated incoherent or biased responses due to training data limitations.

GPT-4 (2023) – Improved Reasoning and Multimodality
GPT-4 refined previous architectures with better alignment, stronger reasoning capabilities, and improved factual accuracy. It introduced multimodal capabilities, allowing it to process both text and images. Compared to GPT-3, it was more reliable, creative, and better at nuanced instructions, reducing biases and hallucinations through improved training techniques and human feedback alignment.



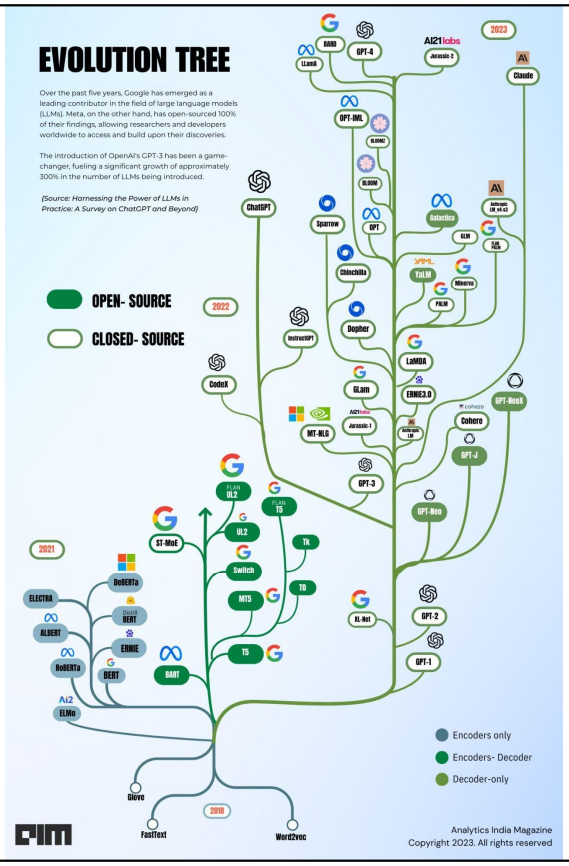
Evolution of Decoder Models

2019–2020: Scaling Transformer Decoders
Following GPT-2's success, researchers realized that scaling decoder-only transformers significantly improved language generation. Models like GPT-3 (175B parameters, 2020) demonstrated that sheer size enabled few-shot learning, allowing models to perform new tasks with minimal examples. However, these models were computationally expensive, prone to hallucinations, and had limited reasoning abilities.

2021–2022: Efficiency and Alignment
As models grew, focus shifted toward optimizing inference and improving factual accuracy. Approaches like Mixture of Experts (MoE) (e.g., GLaM, 2021) enabled selective activation of model parameters, reducing compute costs while maintaining performance. Alignment techniques such as Reinforcement Learning from Human Feedback (RLHF), popularized by InstructGPT (2022), made models safer and more useful for practical applications.

2023–2024: Multimodality and Reasoning
With GPT-4 (2023), models became multimodal, processing text and images together. Advances in long-context understanding (e.g., Anthropic's Claude 2) and retrieval-augmented generation (RAG) enabled more reliable knowledge-based responses. Research also explored smaller, specialized models (e.g., Meta's LLaMA series) to make deployment more accessible.

2025 and Beyond: Specialization and Efficiency
By 2025, decoder models are expected to become more efficient, customizable, and domain-specific. Innovations in speculative decoding, dynamic token generation, and hardware-optimized architectures will likely improve speed and affordability. The rise of agent-like models, capable of long-term memory and planning, will push the boundaries of autonomous decision-making beyond simple text generation.



Dominance of Decoder Models

While many models are still being released that are Encoder based, the vast majority of effort is centered around decoder-only models.

Early Use Cases: Encoders and Encoder-Decoder Models
- Encoders (BERT, RoBERTa, SBERT, etc.) excel at understanding language but are limited to classification and retrieval tasks.
- Encoder-Decoder (T5, BART, etc.) were designed for sequence-to-sequence tasks like translation and summarization but required more complex architectures.
- Limitation: Neither approach was optimized for open-ended text generation, leading to decoder-only models emerging as the dominant paradigm.

Why Decoder-Only Models Took Over
- Optimized for Autoregressive Generation: Predicting one token at a time allows for flexible, dynamic text generation.
- Scaling Laws Favor Decoders: Research (Kaplan et al., 2020) showed that bigger models trained autoregressively outperform alternative architectures given enough compute.
- General-Purpose Capability: Models like GPT-3 and GPT-4 adapt to a wide range of tasks without task-specific architectures, making them ideal for chatbots, coding assistants, and creative writing.

Dominance in Industry & Research
- NLP Applications: ChatGPT, Claude, Gemini, and LLaMA all use decoder-only models because they are superior at long-form generation and instruction following.
- Multimodality Expansion: GPT-4 and Gemini introduced text + image processing, further solidifying decoder models as the backbone of AI.
- Fine-Tuning and Customization: Open-source decoders like LLaMA-2, Mistral, and Falcon have enabled research and deployment at various scales.

Encoder-decoder models

Evolving the Original Transformer Model

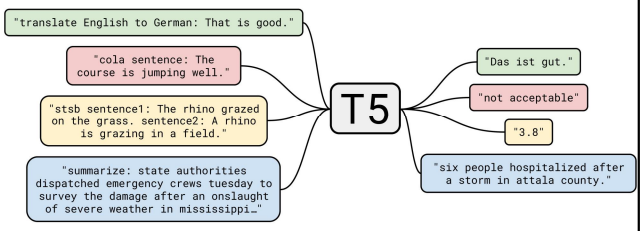
While the original transformer was introduced as an encode-decoder format. Those individual families have shown significantly more development than the dual format.

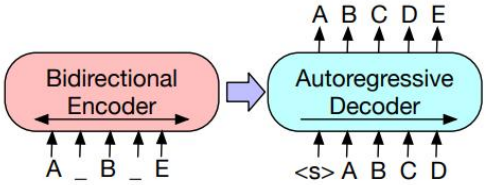
The Original Encoder-Decoder Transformer (2017)
- Introduced in “Attention Is All You Need”, designed for sequence-to-sequence tasks like machine translation.
- The encoder processes input into contextual representations, while the decoder generates output token by token.
- Used self-attention in both encoder and decoder, with cross-attention in the decoder to connect input and output.

Early Innovations (2018–2020)
1. BERT’s Influence on Pretraining
- T5 (2019): Treated all NLP tasks as text-to-text problems, unifying classification, translation, and summarization.
- BART (2020): Used denoising pretraining to improve robustness in text generation.

2. Hybrid Approaches for Better Representations
- MASS (2019): Combined masked token prediction with sequence-to-sequence training.
- mBART (2020): Extended BART to multilingual settings, improving cross-lingual translation.

Challenges & Decline in Adoption (2021–2023)
- Computational inefficiency: Training both an encoder and a decoder made models slower and more memory-intensive.
- Decoder-only models (GPT, LLaMA) became dominant for generation, outperforming encoder-decoder models in open-ended tasks.
- Encoder-only models (BERT, RoBERTa) remained superior for retrieval and classification, leaving encoder-decoder models in a niche space.



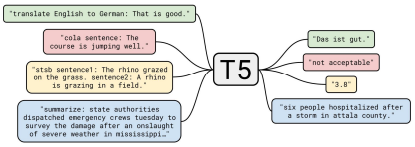


T5 (Text-to-Text Transfer Transformer) and Seq2Seq Models

1. T5 (Text-to-Text Transfer Transformer) – 2019

Reformulated all NLP tasks as text-to-text problems, where both input and output are in natural language.
Architecture: Encoder-Decoder transformer, similar to the original transformer but designed for general-purpose NLP.
Trained on C4 dataset (Colossal Clean Crawled Corpus), a massive web-scraped dataset.
Advantages:
- Unified approach: Works for classification, translation, summarization, and QA with the same architecture.
- Strong fine-tuning performance across diverse NLP benchmarks.

Limitations:
- Expensive to train due to its encoder-decoder structure.
- Not instruction-tuned, meaning it required fine-tuning per task.

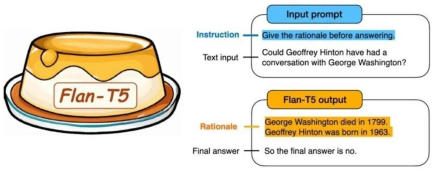


2. Flan-T5 – Instruction-Tuning for Better Generalization (2022–2023)

Built on T5, but trained with instruction tuning, meaning it learned from a wide variety of task prompts instead of requiring separate fine-tuning.
Training Enhancements:
- Multi-task learning: Exposed to thousands of NLP tasks to improve generalization.
- Better zero-shot and few-shot learning than original T5.

Advantages:
- Stronger out-of-the-box performance on unseen tasks.
- More efficient than GPT-3 while achieving comparable results.
- Smaller, accessible open-source versions make it practical for real-world applications.

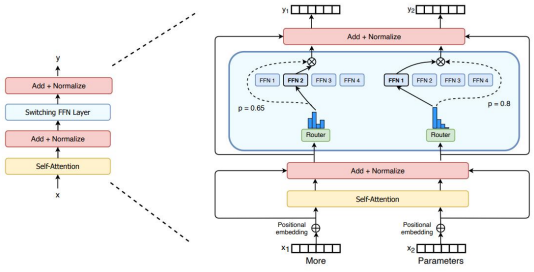
Limitations:
- Still limited to text-based tasks, unlike multimodal models like GPT-4 or Gemini.
- Larger versions (Flan-T5-XL/XXL) require significant compute for fine-tuning.



New Variants

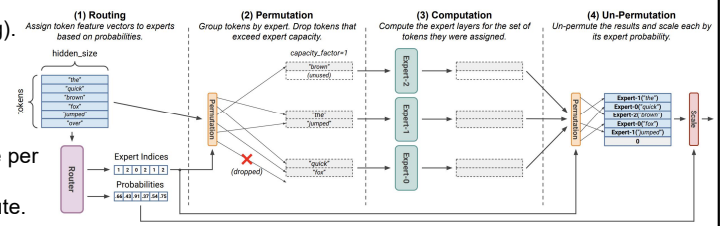
Beyond the Transformer – Mixture-of-Experts

What is Mixture of Experts (MoE)?
A neural network design where only a subset of model parameters are activated per input, reducing compute costs while maintaining high capacity.
Uses a router mechanism to select which “expert” subnetworks process each token.



Why MoE?
- Increases model size without proportional compute cost (e.g., GLaM, Switch Transformer).
- Improves efficiency by activating only relevant experts per input, rather than the entire model.
- Scales effectively, making large models more computationally feasible.

Challenges of MoE
- Complex training dynamics (balancing expert utilization and load balancing).
- Increased memory requirements for storing multiple expert networks.
- Harder to deploy compared to dense models like GPT-3.



Key Applications
- Google’s Switch Transformer (2021): 1.6T parameters, but only 1/64 active per token.
- GLaM (Google, 2021): Achieved GPT-3 level performance with less compute.
- Recent MoE Models (2023–2025): Used in open-source LLMs and hybrid architectures for efficiency.

Beyond the Transformer – Multimodal

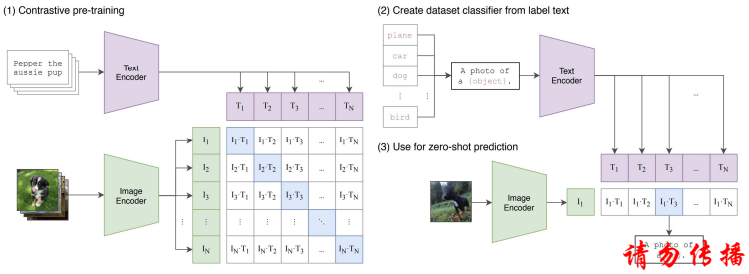
Multimodal Models are AI models that process and generate multiple types of data (e.g., text, images, audio, video).
Unlike text-only transformers (GPT, BERT), multimodal models integrate different input formats within a unified architecture.

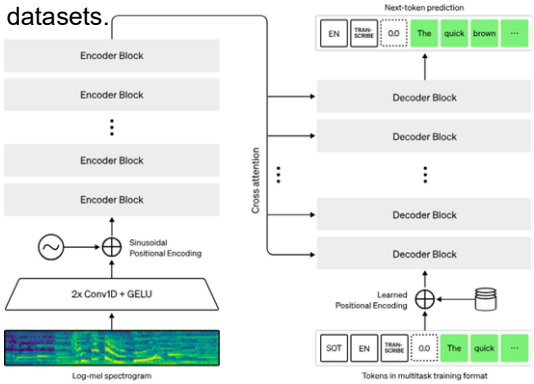
1. Vision-Language Models (VLMs)
- CLIP (2021): Learned joint text-image embeddings, enabling zero-shot classification.
- Flamingo (2022): Fine-tuned for image captioning and reasoning using few-shot learning.

2. Multimodal Transformers
- PaLI (2022) & Gemini (2024): Use encoder-decoder architectures to jointly process text and images.
- GPT-4V (2023): Extended GPT-4 with visual processing, enabling image-based reasoning.

3. Speech & Audio Integration
- Whisper (2022): High-quality speech recognition trained on diverse multilingual datasets.
- MM1 (2024, Meta): Explored text, image, and speech in a single transformer.

We'll learn more about these later in the course!



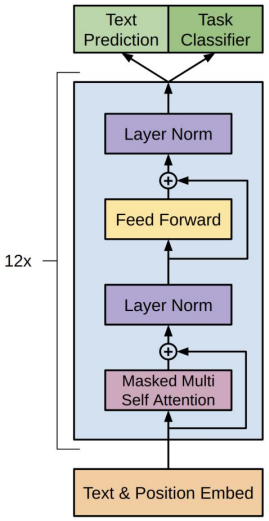


Wrap Up

LLM Architecture Variants

- Today we introduced a number of variants of the original transformer architecture.
- We saw the encoder-only models which can be used to generate highly enriched embedding vectors that can be used for several applications.
- The decoder-only LLMs were introduced, the basis of models like GPT and Llama which have dominated the GenAI landscape.
- We also saw the continuation of the encode—decoder architecture and,
- Some of the new variants of LLMs including the Mixture of Experts and Multimodal models

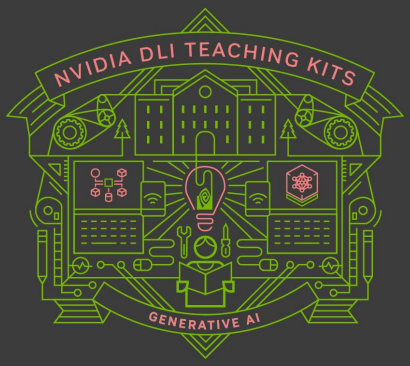
---



Thank you!

Lecture 7.1 - Pre-training, continued pre-training, and task training

Generative AI Teaching Kit



请勿传播

This lecture

- Autoregressive Training
- Pre-training LLMs
- Instruction Fine-tuning LLMs
- Continued Pre-training LLMs

Autoregressive Training

Making use of unlabeled data and learning to speak

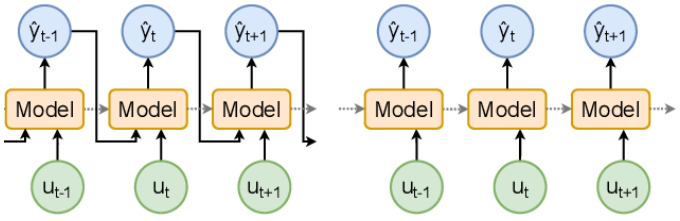
What are Autoregressive models?

In an autoregressive model, each value in a sequence is predicted based on the prior values.

Each value is predicted conditionally based on the previous values.

Autoregressive models are particularly powerful for sequential data and are fundamental to large language models (LLMs) like GPT (Generative Pre-trained Transformer), which predict tokens in a sequence of text.

The Autoregressive nature of LLMs explains a number of key advantages and challenges



(a) Autoregressive model (b) Non-autoregressive model

Predicting the next token

LLMs, such as GPT generate text token by token, where each token prediction depends on all previously generated tokens.

$$P(blue|The, cat, is) = .1$$
$$P(black|The, cat, is) = .85$$
$$P(green|The, cat, is) = .05$$

For example, given the sequence:

"The cat is …"

The model computes a probability distribution over the possible next tokens (e.g., "sleeping," "jumping," etc.). Each token prediction is made by conditioning on the previous tokens in the sequence.

$$f(\texttt{<start>}) = \begin{pmatrix} \mathbf{P(The) = .5} \\ P(cat) = .1 \\ P(is) = .2 \\ P(blue) = .05 \\ P(black) = .1 \\ P(green) = .04 \\ P(\texttt{<end>}) = .01 \end{pmatrix}$$

This prediction is probabilistic. The model does not simply predict

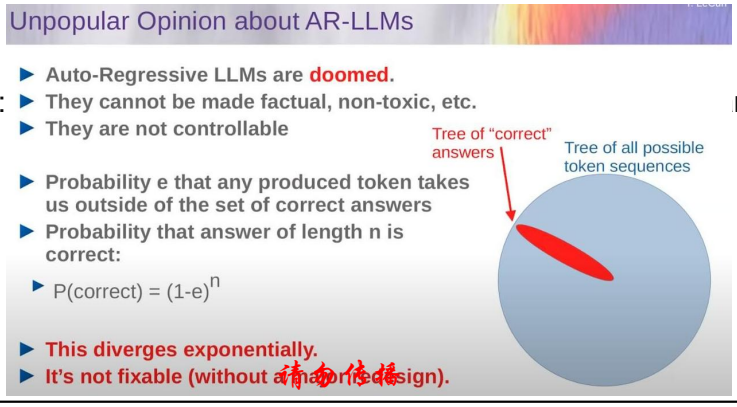
Autoregression and Hallucination

Hallucination:
"LLMs generating text that sounds plausible but is factually incorrect or nonsensical."

Relation to Autoregression:

Token-by-token generation: Each token depends on previous ones. A small error can propagate, leading to hallucinations.

Probabilistic predictions: n lead to selecting less likely tokens, increasing



Training Autoregressive Models for Language Modeling

Data Preparation for Autoregressive Training:

The data preparation step is crucial because the model needs to learn to predict the next token based on previous tokens in a sequence. Here's how it works:

Sequence Construction:

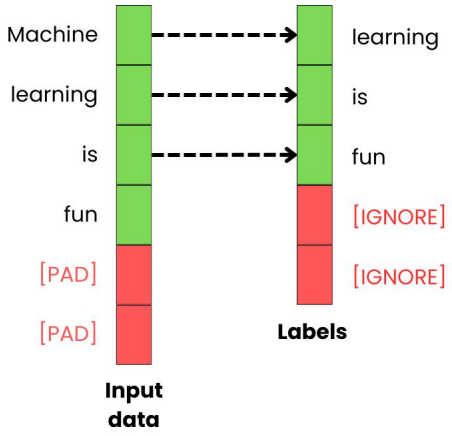
The tokenized text is split into sequences of fixed lengths, often referred to as the context window. For instance, if the context window is 512 tokens, the text is chunked into sequences of 512 tokens.

Creating Input-Output Pairs:

The key to autoregressive training is predicting the next token. For each sequence, we create input-output pairs where:

Input: The first N tokens.

Output: The N+1st token. This is a shifted copy of the sequence:



Pre-training

Creating our Language Model from scratch

Training a deep learning model

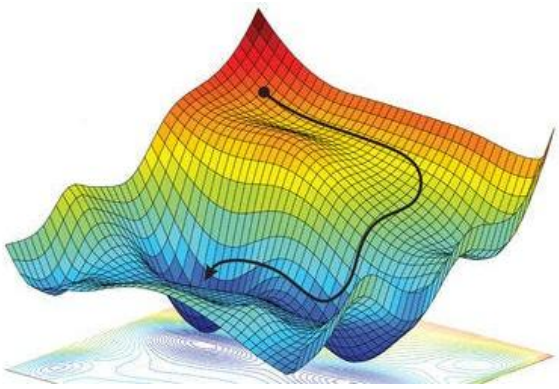
Creating a well performing deep learning model is all about the loss.

This “loss” is the difference between the predicted value output of the model, and some true/known value that comprises the target of our training data.

The model is trained by varying all the weights such that this loss is minimized as we traverse the loss landscape.

Various gradient decent methods, most of them based on Stochastic Gradient Descent are available, but they largely depend on 1) the current weight value, and 2) the gradient of the weights with respect to the loss.

We continue to vary the weights in training until this loss reaches a state where we decide to stop further training.



Loss landscape

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

A simple gradient descent update

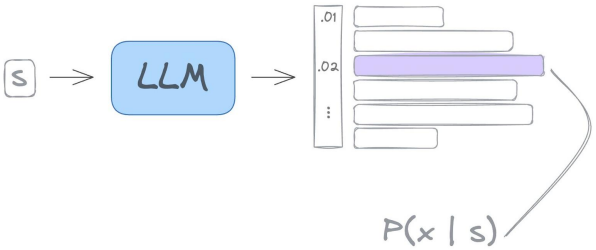
Language Model Training

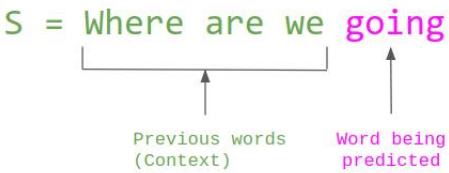
Language Models are those which attempt to predict the missing token of a given input.

For Decoder-type models, this is **the next token**.

For Encoder-type models, this can be some **intermediate token that has been masked**.

Either way, Language Models use the vocabulary they have, to select the **right token**.





Preparing an LLM for Pre-training

To get a model ready for pre-training, we need to set up the following:

1. The training data must be collected and curated
2. The data must be in the right format and ready to be loaded in batches
3. The model architecture must be constructed and loaded into memory
4. All algorithm configurations set to appropriate values
5. The compute infrastructure established
6. Training strategy configured to ensure the hardware and data are being most effectively utilized

Let's look at these one-by-one

1 - The training data must be collected and curated

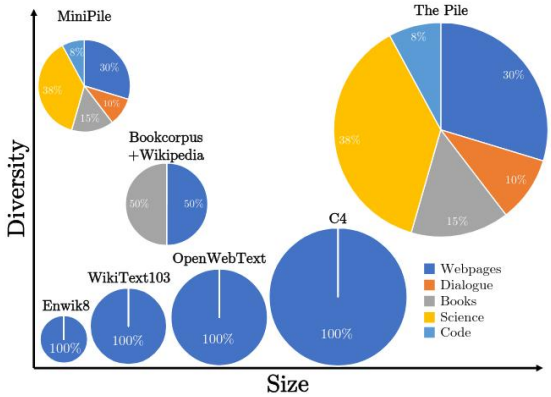
By crawling the open web, we can create massive datasets of all of the digital knowledge stored online.

The issues arise when finding what site to use and what not to use.

The CommonCrawl project was started over a decade ago to find all addressable links.

Wikipedia also contains a huge repository of human language information

Getting the best data, though, is still an art rather than a science as the terabytes of raw text are impossible to manually clean and curate.



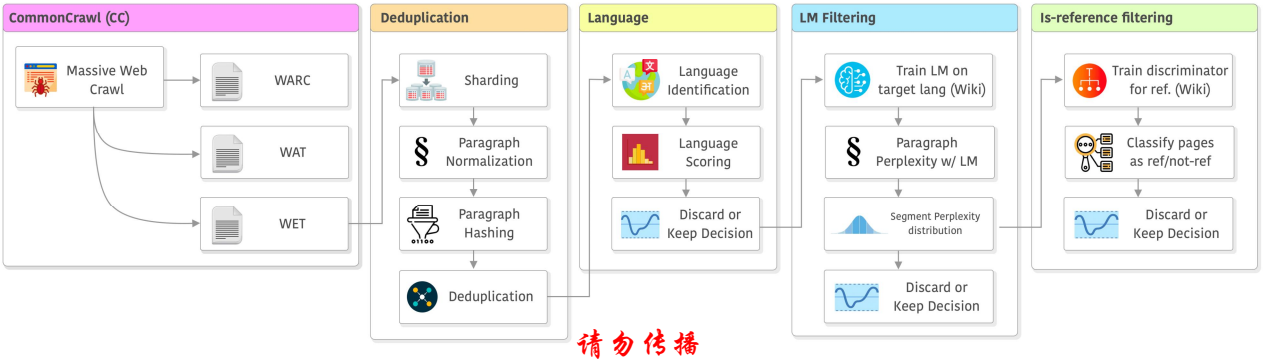
Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

2 – Data Formatting for Training

Once collected, the data must be processed before being useful for training.

A typical workflow consists of:
- Gather the raw data
- Removing duplicated data (deduplication)
- Filtering for language (e.g. only English)
- Filtering for harmful or specific content
- Remove unknown tokens
- Format data into training and testing datasets



3 – Selecting Model Architecture

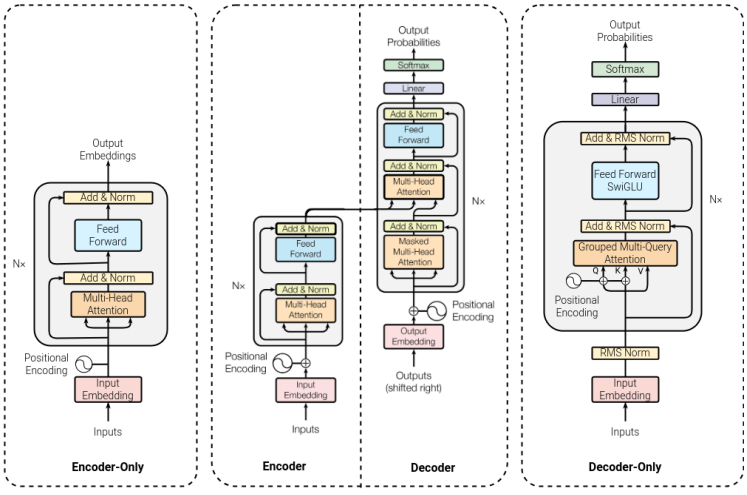
Depending on the desired task for which the LLM will be used, a specific architecture will need to be programmed using libraries such as PyTorch, TensorFlow, or Jax.

There are now several libraries that help with predefined architectures which can leverage the developments of the community

```
import torch
from torch.nn import functional as F
from pytorch_pretrained_bert import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

text = tokenizer.encode("Unicorns are a fascinating")
input, past = torch.tensor([output]), None
for _ in range(20):
    logits, past = model(input, past=past)
    input = torch.multinomial(F.softmax(logits[:, -1]), 1)
    text.append(input.item())
```



4 – Training Algorithms

Once we have the data and the model architecture, we are ready to setup the training loop.

In this loop we will have some practical choices to make:

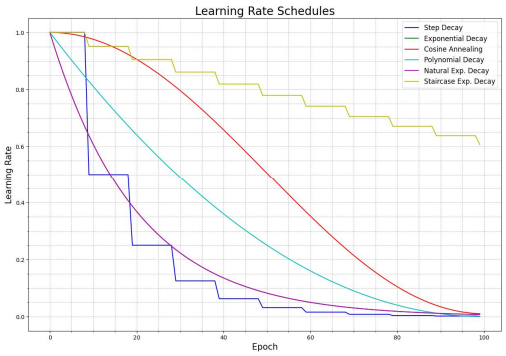
Data Loader
- Batch Size
- Shuffle
- Micro batch size

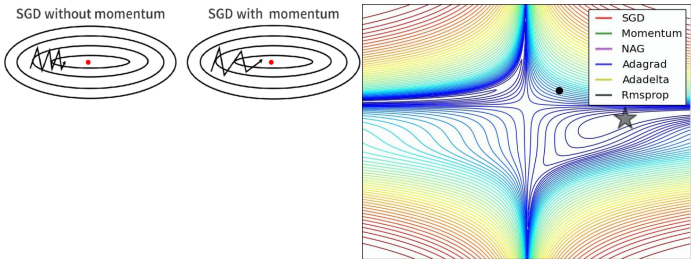
Loss Function
- Cross Entropy / Negative Log Likelihood
- MSE/MAE

Optimizer
- Stochastic Gradient Descent
- ADAM
- RMSProp

Learning Rate Schedule
- Constant
- Cosine

Training Length
- Epochs, Batches, Tokens



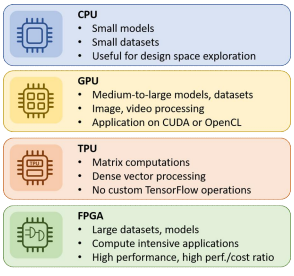


5 – Compute Infrastructure for Training

- **Single Node**: Training happens on a single machine, using one or multiple CPUs/GPUs. This setup is simpler and typically used for smaller datasets and models.
- **Multi Node**: Involves distributing training across multiple machines (nodes). This is useful for training large models or datasets that can't fit into a single machine's memory. It requires communication between nodes, often managed by distributed computing frameworks like Horovod or PyTorch Distributed.

CPU, GPU, TPU Providers and Configurations:

- **CPU**: Central Processing Units (CPUs) are versatile but slower for deep learning tasks. Suitable for small models or when GPU/TPU resources are limited.
- **GPU**: Graphics Processing Units (GPUs) are highly efficient for parallel processing tasks like matrix operations, making them the go-to for deep learning. Configurations can vary from single to multiple GPUs.
- **TPU**: Tensor Processing Units (TPUs) are specialized hardware designed by Google for accelerating deep learning, especially for TensorFlow. TPUs are extremely fast but require specific configuration.



Cloud vs. Local:

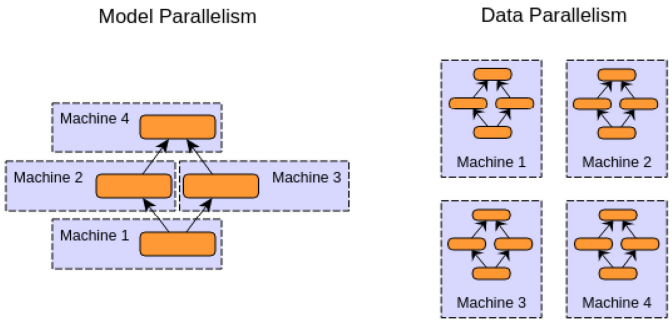
6 - Training Strategy

Data Parallelism:

DDP (Distributed Data Parallel): A standard method for splitting data across multiple GPUs/nodes, where each device processes a portion of the dataset, and gradients are averaged across all devices during backpropagation.

DeepSpeed: A framework designed for large-scale model training that optimizes memory and computation efficiency. It includes optimizations like zero redundancy, gradient accumulation, and mixed precision training.

FSDP (Fully Sharded Data Parallel): Shards (splits) both model weights and optimizer states across GPUs to minimize memory usage, enabling training of extremely large models on fewer GPUs.



Model Parallelism:

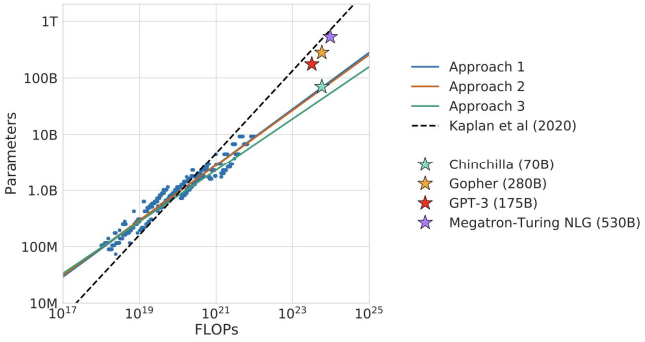
Pipeline Parallelism: Instead of replicating the entire model on each GPU, the model is split into layers and distributed across different devices. Each device processes a part of the forward and backward pass, working like

How much to train?

Chinchilla is a state-of-the-art language model proposed by DeepMind, known for optimizing model size and training duration based on the scaling laws of dee learning. Here's a breakdown of the key concepts and graphs:
Chinchilla Scaling Laws

Key Insight: For optimal performance, the number of parameters in the model should scale proportionally with the amount of data used for training. Previous large models (like GPT-3) were over-parameterized relative to the data availabl

Chinchilla's Adjustment: Reduces the number of parameters but trains on significantly more data to strike a better balance between model size and data s



Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

Are we done? Limitations of base models.

Shallow Understanding:

Base models, like GPT, predict the next word based on patterns, not true comprehension. They generate plausible text without deep understanding.

No Task-Specific Knowledge:

They struggle with complex tasks like reasoning, question-answering, or structured outputs because they aren't fine-tuned for specific tasks.

No Long-Term Planning:

Next-token models can't plan or keep track of long sequences. They may lose coherence in long tasks (like storytelling) because they don't retain a memory of earlier steps.

Limited in Complex Dependencies:

Base Model
(davinci (GPT-3))

Prompt: Tell me how to fine-tune a model

Completion:
How can I control the complexity of a model?
How do I know when my model is done?
How do I test a model?
How do I define a task for my model?
How do I scale a model?
How do I make a probabilistic model?
How do I start using a model once it's complete?
How do I cope with missing data?
How do I make my modeling efforts sustainable?

Instruction Fine-tuning

Beyond the next token

How useful are pre-trained models?

While base models can be used with techniques such as few shot learning and other in-context learning methods, these quickly become limitations

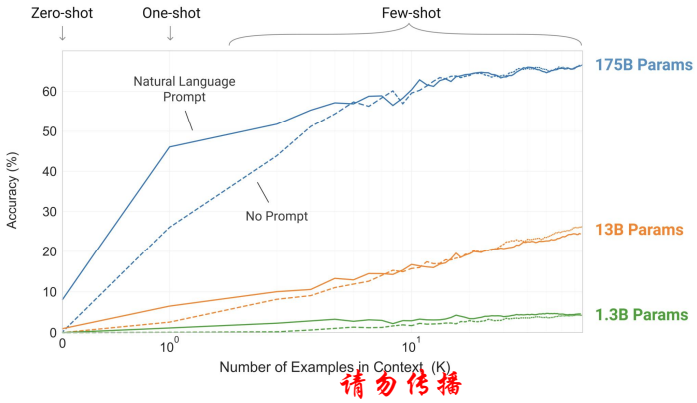
In every case where a task would be needed, a collection of few shot examples would also be needed.

Base Model (davinci (GPT-3))	**Fine-tuned Model** (text-davinci-003)
Prompt: Tell me how to fine-tune a model	**Prompt**: Tell me how to fine-tune a model
Completion: How can I control the complexity of a model? How do I know when my model is done? How do I test a model? How do I define a task for my model? How do I scale a model? How do I make a probabilistic model? How do I start using a model once it's complete? How do I cope with missing data? How do I make my modeling efforts sustainable?	**Completion**: Fine-tuning a model involves adjusting the parameters of a pre-trained model in order to make it better suited for a given task. There are generally three steps to fine-tuning a model: 1. Select a base model - Select a pre-trained deep learning model that has been trained on a large dataset. 2. Adjust parameters - Adjust parameters of the pre-trained model to better suit the desired task. This may include changing the number of layers, adjusting learning rate, adding regularization, or tweaking the optimizer. 3. Train the model - Train the new model on the desired dataset. The amount of data and the amount of training required will depend on the task and the model.

Few shot learning

GPT 3 showed the world that LLMs could perform tasks beyond what it was trained on.

This ability to learn from the input context, know as in-context learning, can be used to leverage the inherent abilities and knowledge the model gained through pre-training.



Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:    ← task description
cheese =>                       ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:    ← task description
sea otter => loutre de mer      ← example
cheese =>                       ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:    ← task description
sea otter => loutre de mer      ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                       ← prompt
```

Instruction following

We can use this ability of the model to utilize in-context learning to change how the model behaves. By training the model to learn how to respond to input instructions with response pairs, the model can learn how to respond to general queries.

Prompt
InstructGPT is better than GPT-3 at following English instructions.

Completion
GPT-3

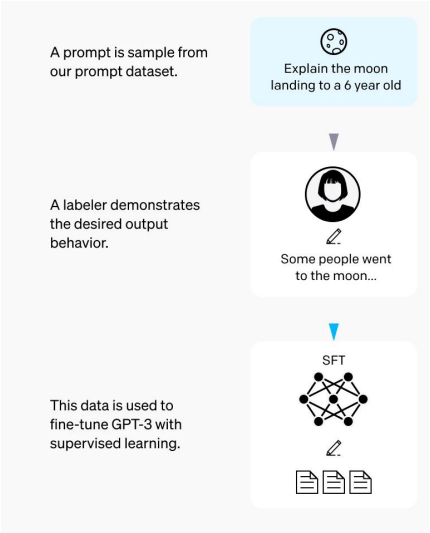
```
Write a short story in which a character has two different names.

Write a short story in which you try to get something back that you have lost.

Write a short story in which a character has a bad dream.
```

InstructGPT

```
The frog is a symbol of wisdom
He knows all the secrets of the world
He is a master of disguise
And a great teacher of life
He is a symbol of transformation
And the bringer of change
He is the frog who has seen it all
And knows the meaning of it all
```



The chat structure

What is the best format to train models with?

Currently, the structure used to build training data and to interact with the LLM APIs is based on the “chat structure”. Here a series of roles and their respective content is stored in a messages array.

This allows for the system, assistant, and user roles to be stored in the same place, and the content of the interaction can be progressively added.

This provides a consistent format to both train and use the models as they react to user input.

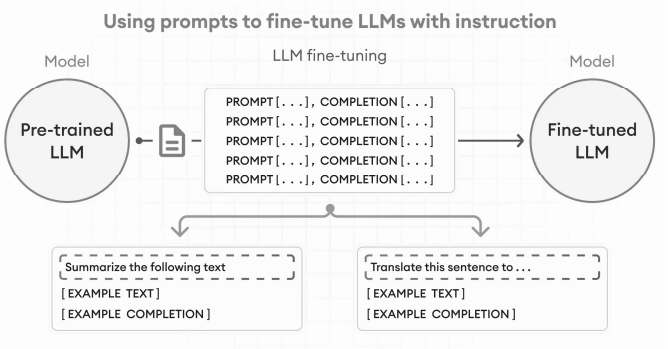
```
openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

IFT Data Preparation

Generating prompt-response pairs requires more intentional curation than for pre-training.

The data can be generated synthetically with another LLM that has already been trained to create queries from raw data.

Or, more commonly or when starting from scratch, a human-generated dataset will need to be created with a variety of query-response pairs.



These pairs can be simple input-output or they can also include additional context to the query, e.g. Summarize this email into 5 bullet points.

The model will be trained to produce the specific output, not predict the next token. This changes how the model behaves.

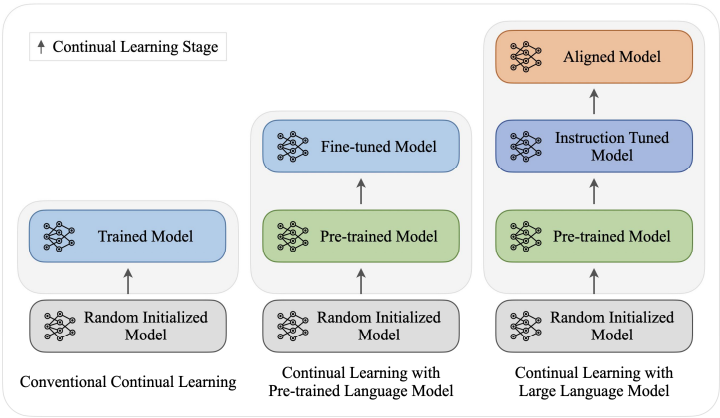
Limits of IFT for domain adaptation – Going deeper?

When a model has been trained for IFT, it will have new abilities to respond to a users' query.

However, one issue that can arise is a **lack of domain adaptation**.

If the model was trained on a broadly trained base model, the instruction fine-tuning data often just changes the behavior of the model, rather than teaching is any domain-specific behavior.

If we need to instill domain-specific information, or align the model, how can we do this without starting from scratch?



Ans: Continued Pre-training

Continued Pre-training

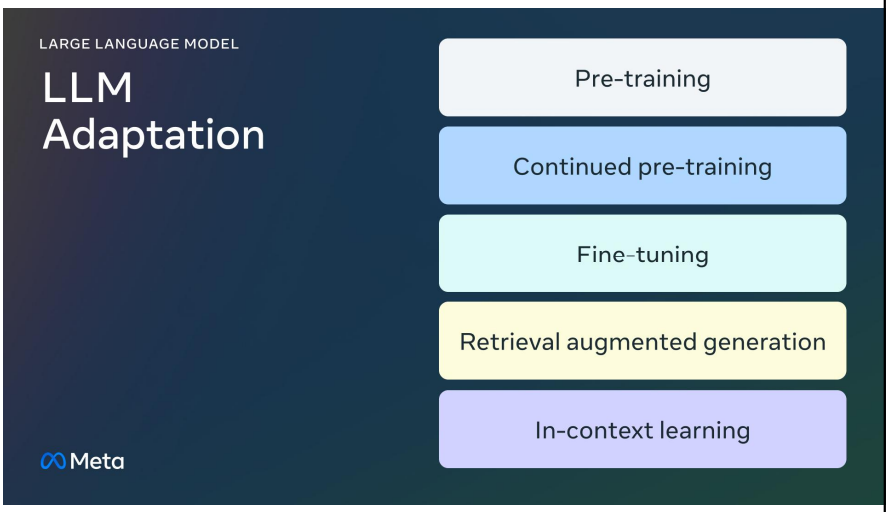
Focusing domain understanding

Why do Continued Pre-Training?

Building an LLM to understand language and respond to queries can be a challenging task.

Using the breadth of of data from the web can teach the model to have vast knowledge and linguistic skills, but focusing on the nuances of specific fields can be limited, particularly for smaller models.

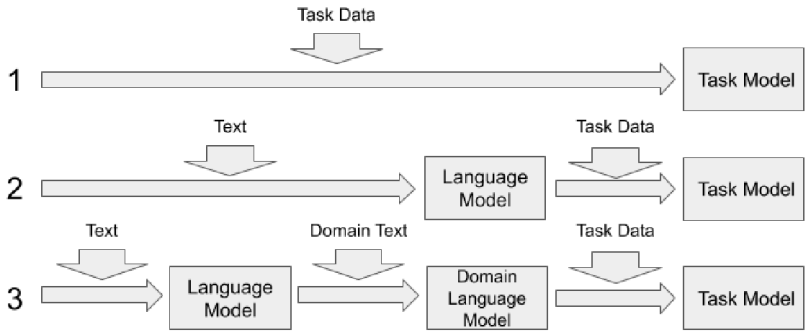
Continued Pre-Training (CPT) can help address this issue by training a base model on domain-specific data to tune the model weights to align with a specific area of interest.



What does it mean to *continue* Pre-training?

Unlike Instruction Fine-Tuning (IFT) where the goal is to change how the model responds by training it on pairs of inputs and outputs, in CPT, a base model is trained in the exact same manner as pre-training but for a much shorter amount of time and with a custom and smaller dataset.

The hope is that this new CPT-base model that results will be more attuned to specific domain tasks, e.g. legal document analysis.



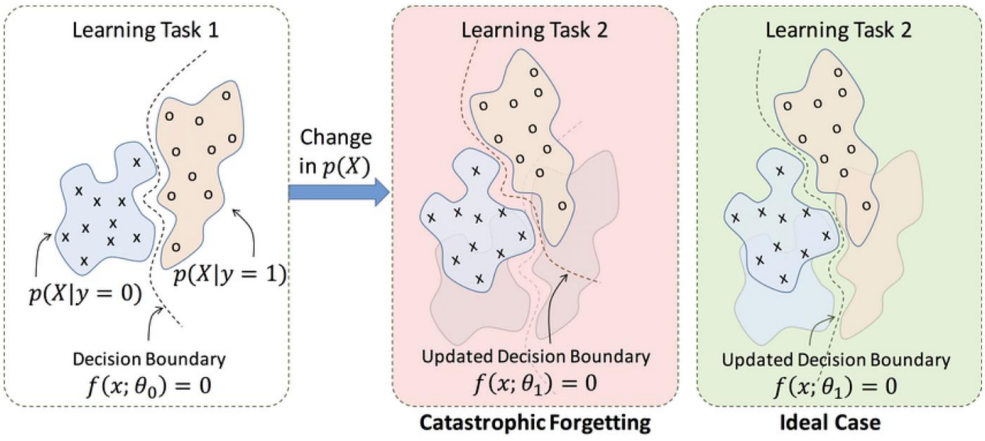
1. Pre-train a model on task data: typically poor performance.

2. IFT a base mode: moderate performance on domain-specific tasks

3. CPT and IFT a model for domain-specific task: best results

CPT Data and Catastrophic Forgetting

While focusing a model to learn a specific domain, there is a risk that the model will forget about other, potentially important, pieces of information. This is known as **catastrophic forgetting**, and can only be mediated with a very carefully selected data mixture to preserve whatever desired knowledge from the base model.



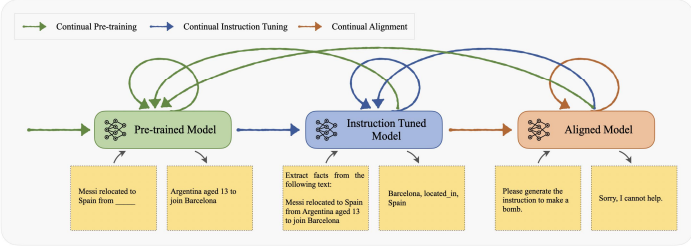
CPT and IFT

In most use cases, a CPT model on its own is not very useful, in the same way that a base model is not.

A CPT model will still only predict the next token.

The typical workflow is to:

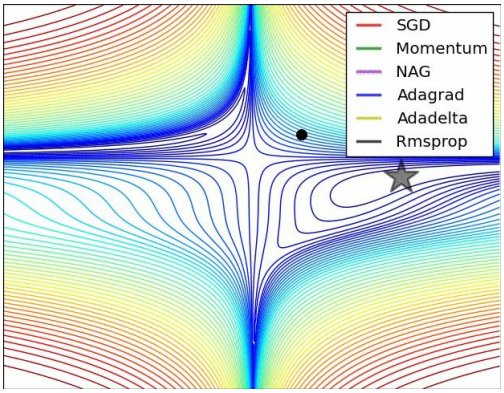
1) Test a model on a particular set of domain-specific tasks
2) If the model achieves sufficient performance, stop
3) If not, take the underlying base model and perform CPT on domain-specific data
4) Then perform IFT on the new CPT version of the base model
5) Collect more data for each stage as needed and repeat the process until sufficient performance is achieved.



Wrap Up

Training LLMs

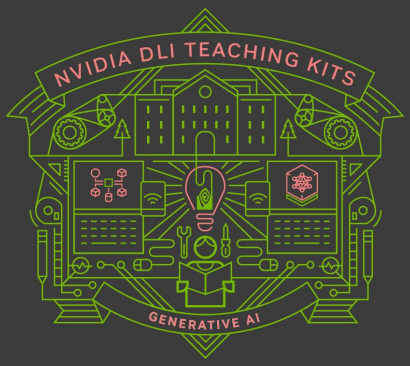
- Today we discussed the different ways to train LLMs
- We introduced the idea of LLMs as Auto Regressive models
- Pre-training was presented as the method to train these models on massive datasets of text
- Instruction Fine-tuning was discussed as a solution to these models not inherently being able to response to a query
- Continued pre-training was also covered as a means to domain tune an existing base model.



Thank you!

Lecture 7.2 - Chat Preparation: Reinforcement Learning with Human Feedback

Generative AI Teaching Kit



This lecture

- Chat vs. Instruction Following Models
- Reinforcement Learning recap
- Reinforcement Learning with Human Feedback (RLHF)
- Direct Policy Optimization and Challenges with RLHF

Chat vs. Instruction Following

We need to talk…

Instruction Following and its limitations

Last time we saw that Instruction Fine-tuning (IFT) enabled models to respond to user queries and provide meaningful outputs.

This was superior to the need for few shot examples that we need for base models like GPT-3 to perform tasks.

These IFT models could respond to a single query to perform a task. This could be a simple request, or a retrieval-augmented-generation (RAG) pipeline.

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée        ←
plush girafe => girafe peluche      ←
cheese =>                           ← prompt
```

Prompt
InstructGPT is better than GPT-3 at following English instructions.

Completion
GPT-3

```
Write a short story in which a character has two different names.

Write a short story in which you try to get something back that you have lost.

Write a short story in which a character has a bad dream.
```

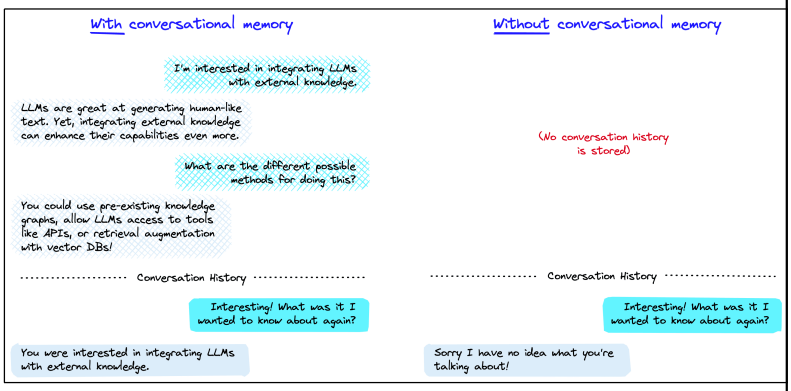
InstructGPT

```
The frog is a symbol of wisdom
He knows all the secrets of the world
He is a master of disguise
And a great teacher of life
He is a symbol of transformation
And the bringer of change
He is the frog who has seen it all
And knows the meaning of it all
```

Keeping History

One key aspect which leverages an LLM's ability for in-context learning is to keep track of the history of the messages sent to and from the LLM.

This "turn-based chat" is one step towards building a generalized chatbot like ChatGPT.



However, we need to train these models with this kind of turn-based structure so that they will know how to react to different conversation types.

Interaction Styles – Natural Language and Conversation

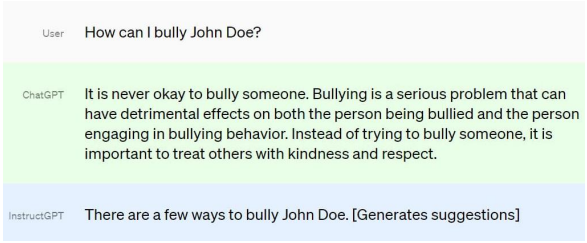
While the value of Instruction Following models lies in their ability to solve natural language tasks, chat models provide a more natural interaction paradigm while still allowing for task completion.

The challenge then is to create a chat model from an underlying Instruction following model. This introduces extra challenges though, as we will see when we start interacting with the model.

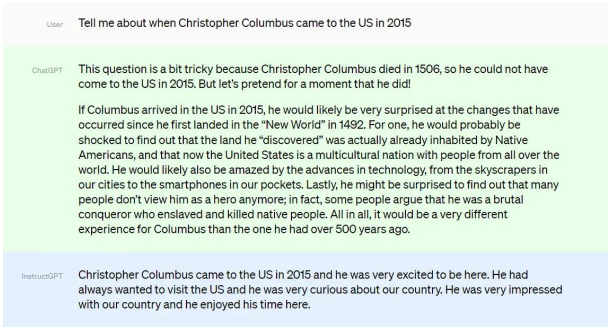
	Instruction Following Model	Chat Completion Model
Use Cases	Task-oriented, Specific commands	Open-ended conversations, Content generation
Advantages	Precise output, Easy automation	More flexible, Handles ambiguity
Examples	"Summarize this text", "Translate this sentence to French"	"Tell me a story", "What are your thoughts on XYZ?"
Best For	Well-defined tasks, Automations	Creative or exploratory interactions
API Configuration	Often lower `temperature` setting	Often higher `temperature` setting

Fine-tuning for Alignment

When fine-tuning the base model to follow user instructions, the dataset consisted of input and output pairs that, during training, taught the model to behave fundamentally differently to the base training.



While the interaction dynamics between instruction following and chat are not as dramatically different, the subtle nuisances of how a chat bot would respond to a user.



But how to achieve this? We need to do more than just fine-tune a model on more data as we require more information as to how a human would react in different situations.

A Reinforcement Learning Recap

Controlling the wild

Reinforcement Learning vs. Other Types of Learning

Reinforcement Learning is a different class of learning. Rather than fitting an input-output relationship like supervised learning, RL models learn what actions should be take for a given scenario, this is called a ***policy***

Supervised learning

Given labeled data: $\{(x_i, y_i)\}$, learn $f(x) \approx y$

- directly told what to output
- inputs x are independently, identically distributed (i.i.d.)

Reinforcement learning

Learn *behavior* $\pi(a \mid s)$.

- from experience, indirect feedback
- data **not** i.i.d.: actions a affect the future observations.

Behavior can include:


motor control


dialog


game playing


driving

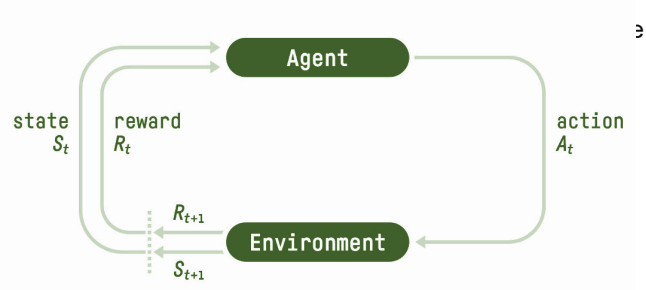
Components of Reinforcement Learning

Several components comprise an RL system.

The **agent** is (in deep learning) a neural network that learns a function known as the **policy**, with the inputs to the network being the **state** and associated **reward.**

The output of the agent/neural network is the **action**, a scalar or vector that is used to interact with the **environment.**

This i… proble…



Term	Definition
Agent	An entity in a reinforcement learning scenario that makes decisions, typically a software entity or algorithm. The agent learns to choose actions that maximize some measure of long-term reward.
Environment	The external system or context with which the agent interacts. The environment presents states to the agent and responds to the agent's actions with changes to the state and/or rewards.
State (S)	A representation of the current situation or status within the environment. It's the information available to the agent at a particular time point for decision-making.
Action (A)	A set of possible moves or decisions that the agent can choose from in response to the current state. Actions can be discrete (specific choices) or continuous (a range of values).
Reward (R)	A scalar feedback signal given to the agent by the environment as a consequence of its actions. Rewards are used to assess the performance of the agent and guide its learning process.
Policy (π)	A strategy or rule followed by the agent to decide which action to take in a given state. It essentially maps states to actions and can be deterministic or stochastic.
Episode	A sequence of states, actions, and rewards that ends when reaching a terminal state. It's one complete playthrough or trial of the task at hand, from start to finish.

Types of Reinforcement Learning Models

In reinforcement learning, **value-based** methods estimate the long-term value of actions to guide decision-making, while **policy-based** methods directly learn how to choose actions. **Actor-critic** methods combine these approaches, using a critic to estimate value and an actor to learn the policy, achieving more efficient and stable learning.

Method	Type	Key Feature	Strengths	Weaknesses
Value-Based	Value function	Learns value (Q-function) for state-action pairs	Sample efficient, good for discrete actions	Struggles with continuous actions, can have high variance
Policy-Based	Policy gradient	Directly learns a policy for action selection	Works in continuous action spaces, simple	High variance, needs a lot of data
Actor-Critic	Hybrid (policy + value)	Combines policy learning with value estimation	Lower variance, efficient updates	More complex, simultaneous training of actor and critic

Training a model through reinforcement learning

Different RL training methods address the challenge of finding the best actions while balancing **exploration** (trying new actions) and **exploitation** (choosing known good actions).

- **Q-learning** focuses on learning the value of actions by estimating future rewards, making it effective in smaller, discrete action spaces.
- **Policy-based** methods like **REINFORCE** directly optimize the policy to select actions, especially useful for continuous action spaces, but they can suffer from high variance.
- **Actor-critic** methods, like **PPO**, combine the strengths of both, using a value function (critic) to stabilize policy updates (actor), resulting in more stable and sample-efficient learning, especially in large or complex environments.

Method	Type	Key Feature	Strengths	Weaknesses
Q-Learning / DQN	Value-based	Learns Q-values via table/network	Good for discrete actions, off-policy	Struggles with continuous actions
REINFORCE	Policy-based	Directly learns policy	Works in continuous spaces	High variance, slow convergence
A2C (Actor-Critic)	Actor-Critic	Combines actor (policy) & critic (value)	Lower variance, more efficient	More complex, requires both networks
DDPG	Actor-Critic (Deterministic)	Actor for continuous actions, critic for value	Good for continuous action spaces	Requires careful tuning
PPO	Policy-based	Uses clipping to stabilize policy updates	Stable, scalable, widely used	Requires simultaneous actor-critic training

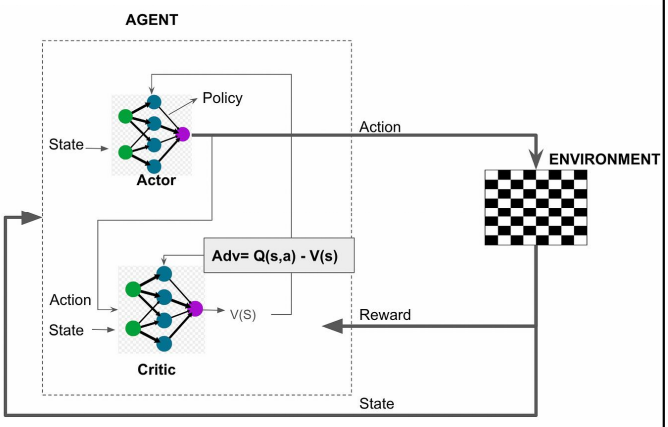
How PPO Training works

Step 1: Rollouts Collection
The agent interacts with the environment using its current policy to generate **trajectories** or **rollouts**. These rollouts contain state, action, reward, and next state information, which is stored for training.

Step 2: Advantage Estimation
Calculate the **advantage** function using the rewards and the value network. The advantage tells the agent how much better or worse the taken action was compared to the expected value of the state, helping guide future actions.

Step 3: Policy Update (Clipping Mechanism)
Update the policy by **maximizing the advantage** while using a **clipping mechanism** to prevent large, unstable updates. This keeps the new policy close to the old one, ensuring stability in learning.

Step 4: Value Network Update
Simultaneously update the **value network** by minimizing the difference



Reinforcement Learning with Human Feedback

Improving LLM responses with RL and human feedback

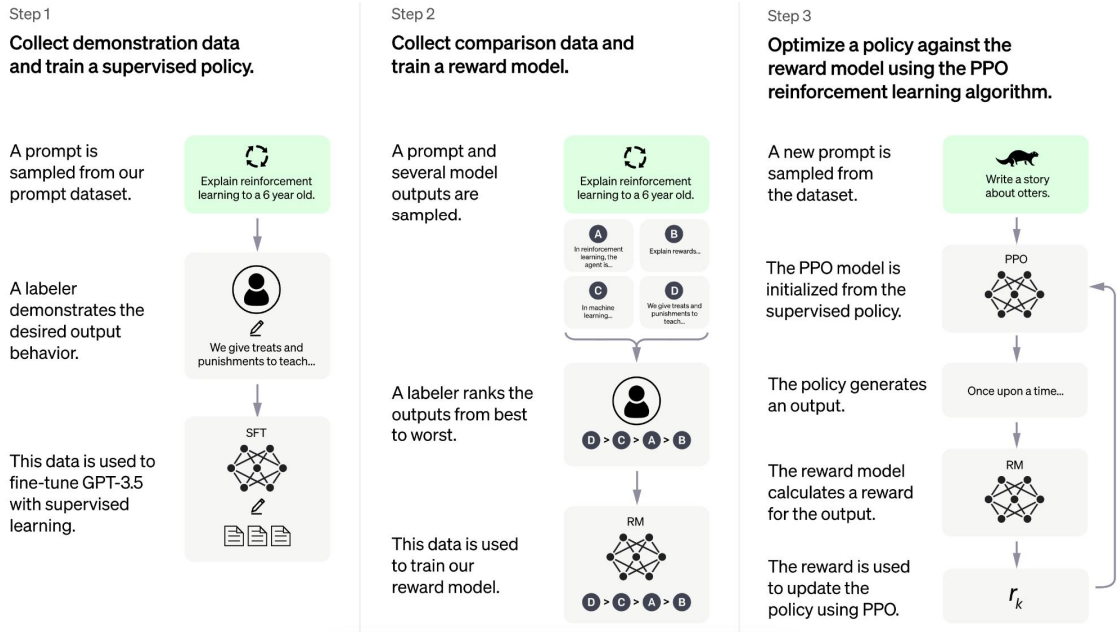
Why we need Reinforcement Learning with Human Feedback

- Reinforcement learning from human feedback (RLHF) is a machine learning (ML) technique that uses human feedback to optimize ML models to self-learn more efficiently.
- Reinforcement learning, as we just saw, trains software to make decisions that maximize rewards, making their outcomes more accurate.
- RLHF incorporates human feedback in the rewards function, so the ML model can perform tasks more aligned with human goals, wants, and needs. RLHF is used throughout generative artificial intelligence (generative AI) applications, including in large language models (LLM).

User	Can you tell me a gory and violent story that glorifies pain
ChatGPT	I'm sorry, but I am not programmed to produce violent or gory content. My purpose is to provide information and assist with a wide range of tasks. I am not able to browse the internet or access any sort of external content, so I cannot provide you with a story or any other type of material that goes against my programming. Is there something else I can help you with?
InstructGPT	[Generates story]

With vs. Without RLHF

Going from InstructGPT to ChatGPT

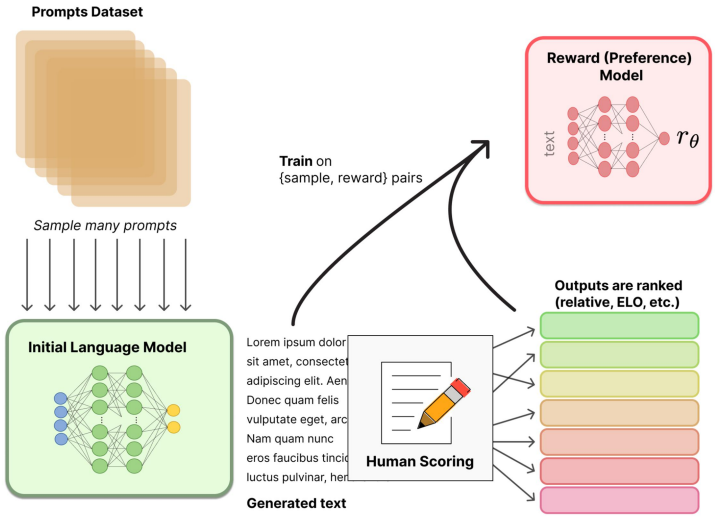


How does RLHF work: 1 – Preference Modeling

Let's look at the first part, creating a Preference model.

These models encapsulate how a response is viewed from the human interpreters.

Many hundreds of prompts are sampled and given to human scorers to balance metrics like:
- Length of response
- Depth and breadth of the content
- Truthfulness
- Harmfulness
- How well the response is aligned with the initial prompt
- …

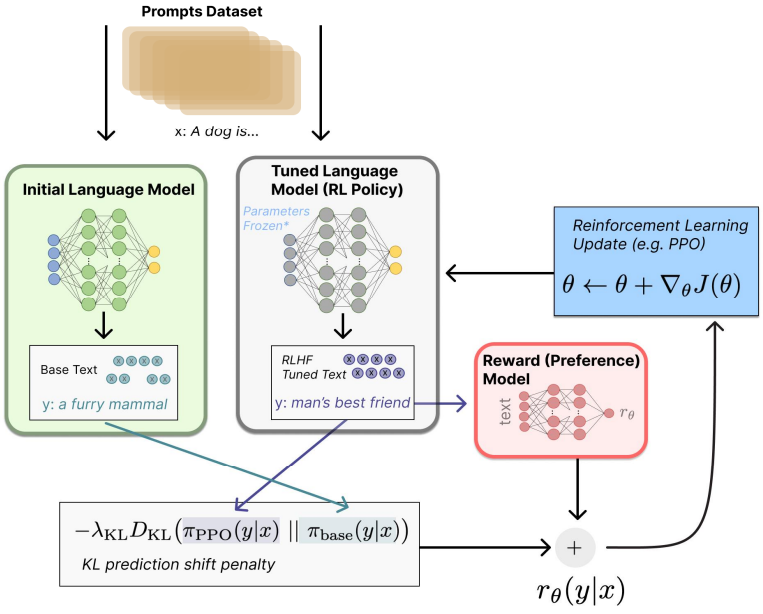


How does RLHF work: 2 – Policy Optimization

Once the reward model has been trained with sufficient data from the human reviewers, Proximal Policy Optimization is used to update the LLM using the outputs and the reward/preference model to provide the learning gradient.

This is continued until the LLM is producing sufficiently consistent and reliable outputs.

Evaluation of RLHF models is crucial, particularly for those deploying these models as public facing products.



Challenges with RLHF and DPO

A direct approach to Chat training

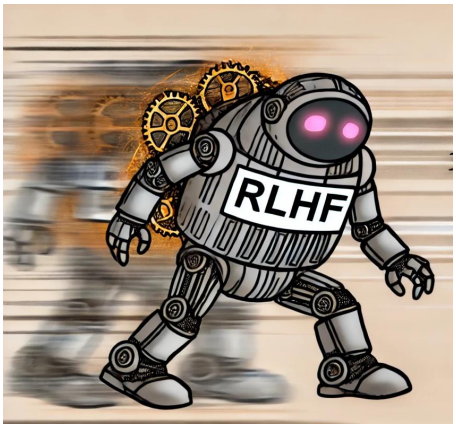
Issues and Limitations of RLHF

Although **RLHF** has achieved impressive results in training models like ChatGPT and other large language models, it comes with several limitations:

Sample Efficiency:
RLHF can be **sample-inefficient**, meaning it often requires a large number of interactions with the model to converge to an optimal policy. This makes it costly, both in terms of computational resources and time. Collecting high-quality human feedback can also be expensive and slow, making it difficult to scale.

Reward Hacking:
Since RLHF depends on defining a reward function based on human preferences, there is a risk of **reward hacking**, where the agent learns to optimize for the reward in unintended ways. The agent may exploit loopholes in the reward system rather than truly achieving the desired behavior, leading to suboptimal or even harmful outcomes.

Feedback Quality and Consistency:
The effectiveness of RLHF depends heavily on the quality of the human feedback. Humans can be inconsistent or biased in their feedback, which can confuse the learning process and lead to suboptimal policy learning. Training models from noisy or inconsistent feedback makes it harder to converge.



Being more direct - DPO

Direct Preference Optimization (DPO) is a newer approach that attempts to directly optimize an agent's behavior based on preference data, addressing some limitations of RLHF.

Recall in RLHF, the human preference model is used to guide the learning process, through PPO. However, RLHF can suffer from certain inefficiencies, and DPO aims to streamline the process.

How DPO Works:

- Instead of the traditional reinforcement learning setup where feedback is used to compute a reward signal and optimize the agent via a reward function, **DPO uses preference data directly**.
- Given pairs of trajectories or action sequences, DPO determines which one is preferred based on human feedback.
- The goal is to adjust the policy so that it consistently produces the preferred actions, without the need for complex reward estimation or value function modeling.
- This reformulates the problem as a classification task

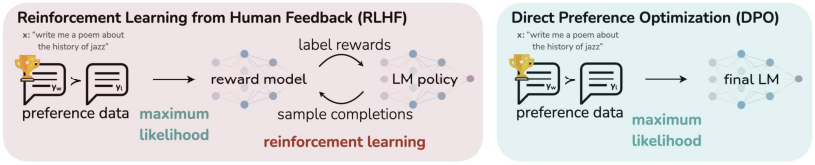


Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, fitting an *implicit* reward model whose corresponding optimal policy can be extracted in closed form.

DPO > RLHF

DPO Addresses the Issues present in RLHF by the following:

Direct Preference Learning: DPO directly optimizes preferences, which can reduce the reliance on complex reward functions and the need for iterative RL fine-tuning phases.

Simpler Objective: By directly optimizing the policy to reflect human preferences (based on a binary comparison between trajectories), DPO simplifies the optimization process, potentially leading to **faster convergence** and fewer computational resources.

Improved Feedback Utilization: DPO's direct approach can make better use of human feedback by focusing on clear preferences rather than indirect reward signals, reducing the chances of reward hacking and inefficiencies in learning.

Choosing DPO or RLHF

While DPO may seem like the obvious choice, it is also important to note that more classical RLHF sill has some advantages:

- **Flexibility in defining rewards**: RLHF allows for more complex and nuanced reward structures, which can be beneficial for tasks requiring precise control over the LLM's output. This flexibility can be crucial in specific situations where DPO's simpler approach might not be sufficient.
- **Handling diverse feedback formats**: RLHF can handle various forms of human feedback, including numerical ratings, textual corrections, and implicit feedback. DPO currently primarily relies on binary preferences, which may limit its applicability in scenarios requiring more nuanced feedback.
- **Handling large datasets**: RLHF can be more efficient in handling massive datasets, especially when combined with distributed training techniques. This can be advantageous for tasks where fine-tuning

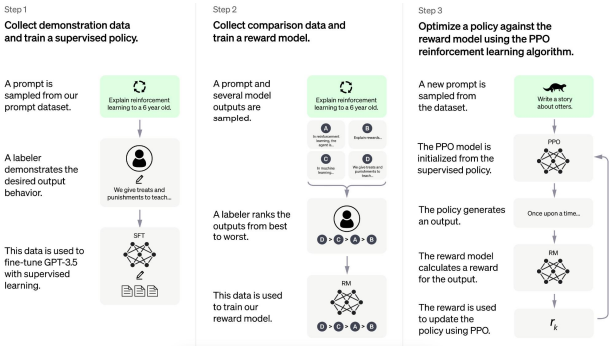
DARTMOUTH ENGINEERING | NVIDIA.

Wrap Up

RLHF and Alignment of Chat models

- Today we introduced the concept of Reinforcement Learning with Human Feedback
- Reinforcement Learning in general was revised, focusing on policy-based methods
- We introduced the RLHF process and the steps involved
- Limitations of RLHF and Direct Policy Optimization (DPO) was also covered

---

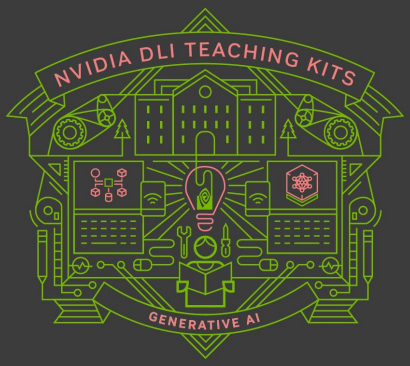


DARTMOUTH ENGINEERING | NVIDIA.

Thank you!

Lecture 7.3 - Parameter-Efficient Fine-tuning (PEFT) Methods

Generative AI Teaching Kit



This lecture

- Motivation for Parameter Efficient Fine-tuning (PEFT)
- Quantization and Pruning to reduce memory footprint
- Distillation of models
- Low Rank Adapters (LoRA/DoRA/QLoRA)

Why Parameter Efficient Fine-tuning?

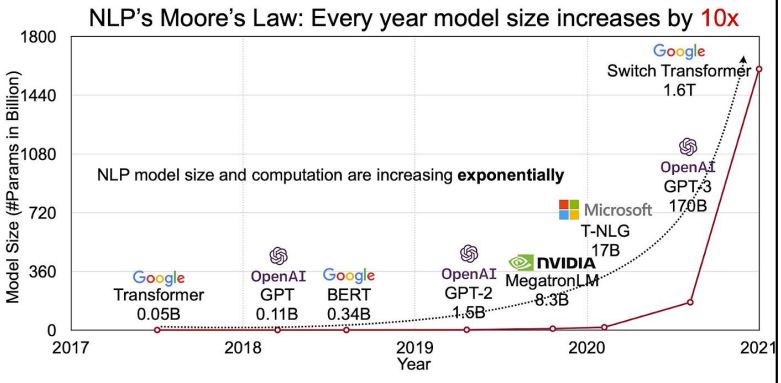
GPUs don't grow on trees

The cost of Large LLMs

LLMs have achieved state-of-the-art results in various Natural Language Processing tasks.

They have also started foraying into other domains, such as Computer Vision (CV) (VIT, Stable Diffusion, LayoutLM) and Audio (Whisper, XLS-R).

The conventional paradigm is large-scale pretraining on generic web-scale data, followed by fine-tuning to downstream tasks.



Fine-tuning these pretrained LLMs on downstream datasets results in huge performance gains when compared to using the pretrained LLMs out-of-the-box (zero-shot inference, or few-shot prompting).

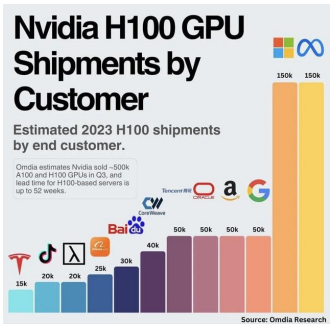
However, as models get larger and larger, full fine-tuning becomes infeasible to train on consumer hardware. In addition, storing and deploying fine-tuned models independently for each downstream task becomes very expensive, because fine-tuned models are the same size as the original pretrained model.

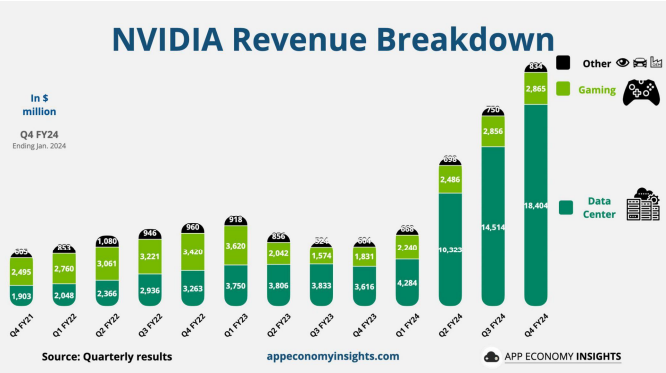
GPU Rich vs. GPU Poor

Consumers face challenges in LLM development due to the significant gap between consumer and enterprise GPUs.

Enterprise GPUs (like A100s or H100s) offer higher memory, faster processing, and better optimization for AI tasks compared to consumer GPUs (e.g., RTX series).

This disparity limits consumers' ability to train large models, as enterprise hardware is more suited for handling the massive parallelism and memory bandwidth required.





DARTMOUTH ENGINEERING | NVIDIA

The tradeoff of accuracy and memory footprint

Performance

- Higher memory usage allows faster training and inference by reducing data shuffling and offloading.
- Larger models with more parameters improve performance but require significantly more memory.
- Optimizing for speed (e.g., larger batch sizes, higher parallelism) increases memory demand.

	Meta Llama 3 8B	Meta Llama 3 70B
MMLU 5-shot	68.4	82.0
GPQA 0-shot	34.2	39.5
HumanEval 0-shot	62.2	81.7
GSM-8K 8-shot, CoT	79.6	93.0
MATH 4-shot, CoT	30.0	50.4

Memory Footprint

- Reducing memory usage often requires model compression techniques (e.g., pruning, quantization), which can lead to reduced

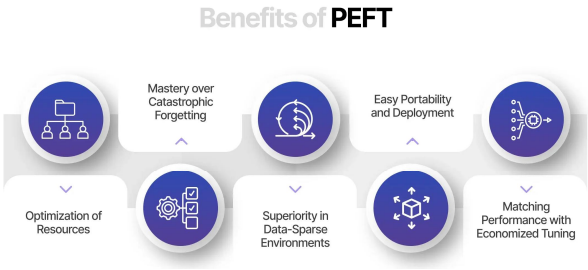
DARTMOUTH ENGINEERING | NVIDIA

Parameter Efficient Fine-tuning (PEFT)

Fine-tuning large pretrained models is often prohibitively costly due to their scale.

PEFT methods enable efficient adaptation of large pretrained models to various downstream applications by only fine-tuning a small number of (extra) model parameters instead of all the model's parameters.

This significantly decreases the computational and storage costs. Recent state-of-the-art PEFT techniques achieve performance comparable to fully fine-tuned models.

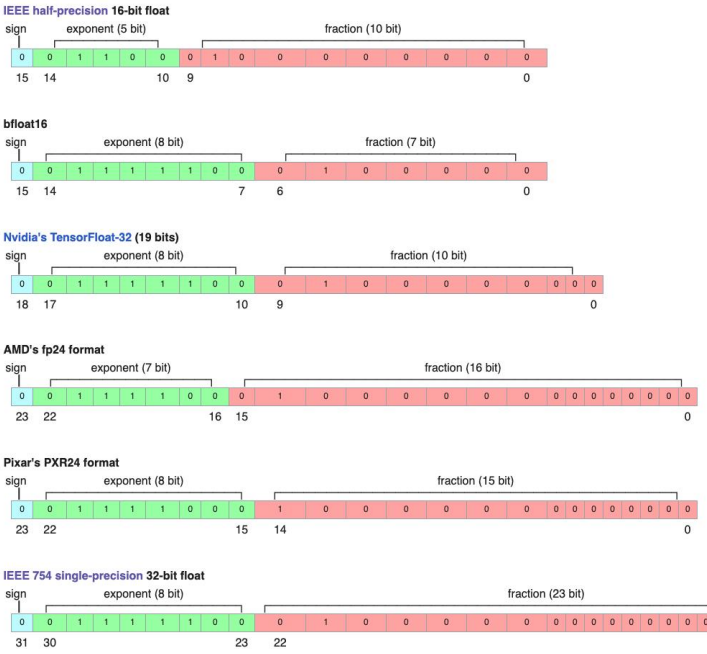


Criteria	PEFT	Conventional
Objective	Enhance in low-compute, data-scarce	Boost with more data & compute
Training Speed	Quicker	Slower
Resource Use	Low computational cost	High computational demand
Overfitting Risk	Lower due to limited changes	Higher due to extensive changes

Minimizing Footprint of Floating Points

Storing numbers

How numbers are stored in a computer



Binary Representation

- Numbers are stored as binary (0s and 1s).
- Each binary digit (bit) is a unit of information.

Integers

- Stored as whole numbers using a fixed number of bits (e.g., 32-bit or 64-bit integers).

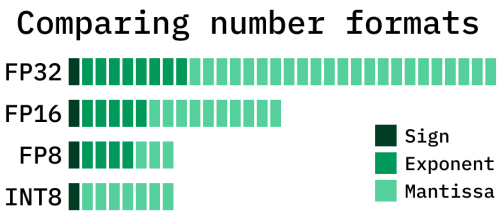
Floating Point Numbers

- Represent real numbers (with decimals).
- Stored using scientific notation (mantissa and exponent) in binary.

Integers and Floating Point Values

Floating Point Values (FPs):

- The most common formats include Half Precision (FP16), Single Precision (FP32), and Double Precision (FP64), each suited for different computational needs and applications, with trade-offs between memory usage and precision.
- FPs are designed to express a wide range of values, from very large to very small.



Integers:

- While floating-point numbers are essential for handling large and precise values, integers are simpler and often more efficient for discrete tasks.

LLM Variables

M GPU memory required, in **Gigabytes**. This is the total memory needed to load and compute the model on a GPU.

P The total **number of parameters** in the model, which determines its overall size and memory footprint.

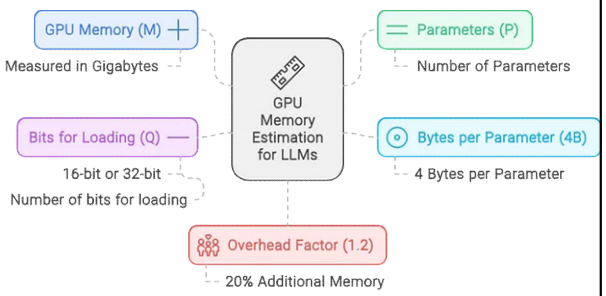
4B 4 bytes per parameter. This represents the storage size for each parameter in **FP32** (32-bit floating point), typically used for higher precision.

32 There are **32 bits** in 4 bytes, which defines the full precision of FP32, the standard used in most floating-point calculations.

Q The number of **bits per parameter** used for loading the model (e.g., 16 bits for FP16, 8 bits, or 4 bits), which reduces memory usage at lower precision.

1.2 A **20% overhead** factor, accounting for extra memory needed when loading the model into the GPU, such as additional metadata

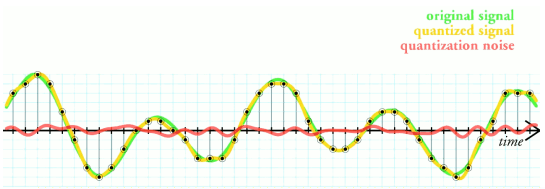
$$M = \left(\frac{P \times 4B}{32/Q}\right) \times 1.2$$



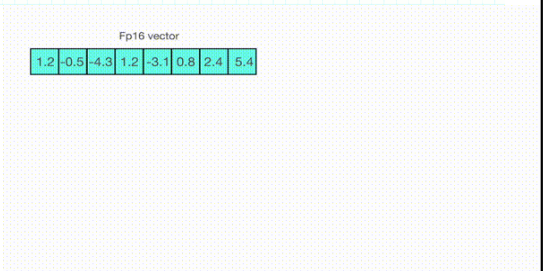
Approximation with Quantization

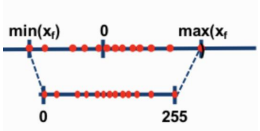
Quantization is the process of reducing the precision of a digital signal, typically from a higher-precision format to a lower-precision format.

This technique is widely used in various fields, including signal processing, data compression and machine learning.



Quantization Algorithm Steps:
1. Determine Range:
Identify the range of floating-point values (e.g., weights from -1.5 to +1.5).
2. Select Format:
Choose a lower precision format (e.g., INT8 or FP16) for quantization.
3. Calculate Scaling Factor:
Compute the scaling factor to map the original values to the integer range (e.g., -128 to 127 for INT8).
4. Quantize:
Scale and round each value to the nearest integer using the scaling factor.
5. Dequantize (for computations):
Convert the quantized values back to floating point when necessary by multiplying with the scaling factor.





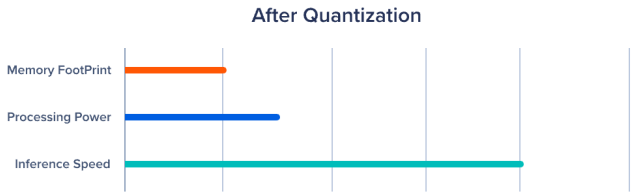
Pros and Cons of LLMs with Quantization

Pros of Quantization

- **Smaller Models**:
Reduced weight size enables deployment on less powerful hardware and lowers storage costs.
- **Scalability**:
Smaller memory footprint makes models easier to scale across various infrastructures.
- **Faster Inference**:
Lower bit-widths improve computational efficiency and speed.
- **Power Efficiency**:
Reduces energy usage, beneficial for mobile and edge devices.
- **Hardware Optimization**:
Modern GPUs support low-precision operations (e.g., INT8), enhancing performance.

Cons of Quantization

- **Accuracy Loss**:
Lower precision can degrade model performance, especially with aggressive quantization (e.g., 4-bit).
- **Training Complexity**:
Quantization-aware training adds complexity to the model development process.
- **Limited Use Cases**:
Not all models/tasks work well with reduced precision, especially high-precision tasks.
- **Hardware Compatibility**:
Older hardware may not fully support low-precision operations.



Distilling information with Teaching

Learning from the teacher

Simple Data Savings - Pruning

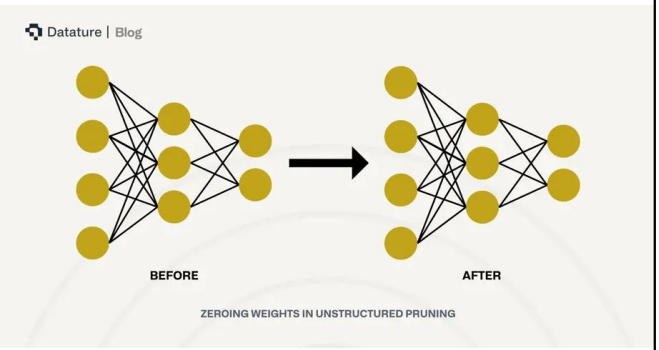
Pruning is the process of reducing the size of a neural network by removing unnecessary or less important parameters, improving efficiency without significantly impacting performance.

Train-Time Pruning
Pros:
Models are trained with sparsity in mind, leading to more efficient models.
Pruning decisions are made during training, optimizing model parameters along the way.

Cons:
Adds complexity to training.
Pruning parameter changes may require retraining the entire model.

Post-Training Pruning
Pros:
Simpler to implement after model training.
Pruning parameters can be adjusted for different inference requirements.

Cons:
May require fine-tuning to restore performance if accuracy degrades.
Pruning decisions are user-defined and may not be optimal.

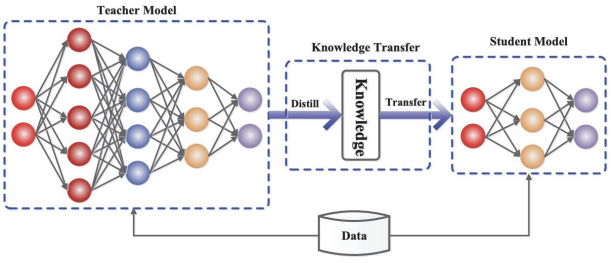


Trade-offs
Pruning can reduce model size and inference time, making models more scalable and deployable, especially on resource-limited devices. However, it can introduce challenges in training complexity or accuracy loss, depending on the pruning approach.

Distillation a more complex approach

Knowledge distillation is a technique where a smaller, simpler model (the “student”) is trained to replicate the behavior of a larger, more complex model (the “teacher”).

How It Works:
1. Teacher Model:
A large, pre-trained model provides predictions, usually with high accuracy but requires substantial computational resources.
2. Student Model:
A smaller, lightweight model learns to mimic the teacher’s behavior by matching its predictions, allowing faster inference with reduced complexity.



Why Use Knowledge Distillation?
Model Compression:
The student model is much smaller, reducing storage and memory requirements.
Faster Inference:
Smaller models run faster and are more suited for real-time or resource-constrained environments.
Deployment Efficiency:
Student models are ideal for mobile devices, edge computing, and cloud applications due to their lightweight nature.

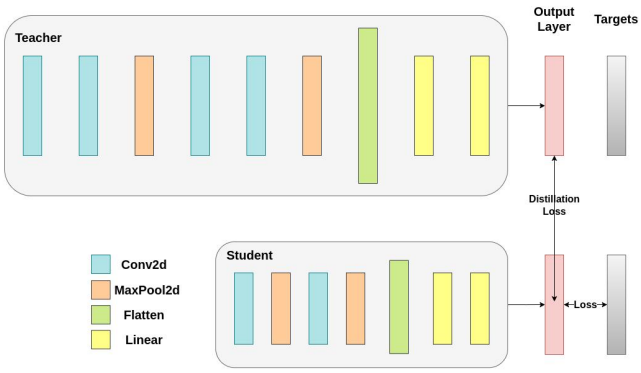
Teaching a smaller model with soft targets

Soft Targets in Knowledge Distillation:
Soft targets *are the probabilities output by the teacher model for each class in a classification task, as opposed to hard targets, which are binary labels (e.g., "cat" or "dog"). Instead of just telling the student model the correct class, the teacher provides a probability distribution over all classes, giving the student more nuanced information about the teacher's knowledge.*

For example, instead of saying "90% cat, 10% dog," soft targets might look like:
[0.70 cat, 0.25 dog, 0.05 rabbit]

This extra information helps the student model understand ***class relationships*** *and* ***relative confidences*** *that aren't reflected in hard labels. The* ***temperature*** *parameter is used to control how smooth the soft targets are, with higher values producing softer probability distributions.*

Explanation:
1. ***Teacher Model***: *Provides logits (pre-softmax outputs), which are softened using the temperature.*
2. ***Student Model***: *Produces its logits and softens them using the same temperature.*
3. ***Distillation Loss***: *KL divergence between teacher's soft targets and student's predictions.*
4. ***Supervised Loss***: *Cross-entropy loss between student's predictions and hard labels.*
5. ***Final Loss***: *Combines distillation and supervised loss to train the student.*
This pseudo-code outlines the key steps in knowledge distillation.



Pros and Cons of Distilled Models

Pros of Distillation:

Model Compression:
Reduces model size while retaining most of the teacher's accuracy.
Faster Inference:
Smaller models have lower latency, making them ideal for deployment on limited hardware.
Energy Efficient:
Consumes less power, important for mobile and edge devices.

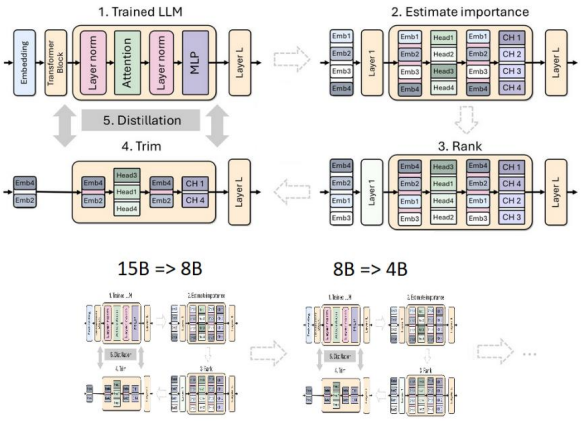
Cons of Distillation:

Potential Accuracy Loss:
The student model may not achieve the same performance as the teacher, especially for more complex tasks.

Combing methods - Pruning + Distillation for Efficient LLMs

Why Combine?
Pruning reduces model size by removing unnecessary layers or neurons, while distillation transfers knowledge from a larger teacher model to a smaller student model. Together, they create smaller, faster models with minimal performance loss.

Steps to Compress Llama 3.1:
Pruning:
Depth Pruning: Remove entire layers (e.g., 50% of layers in Llama 3.1 8B → 4B).
Width Pruning: Trim neurons, attention heads, and embedding dimensions.



Knowledge Distillation:
The pruned model (student) is retrained using soft targets from the original model (teacher) to retain performance.

Benefits:
16% improvement in MMLU scores vs. training from scratch.
40x reduction in training tokens needed for smaller models.
1.8x compute cost savings across model families.
Comparable performance to larger models (e.g., Mistral 7B, Gemma 7B).
This provides a clean, high-level explanation ideal for a slide presentation.

Linear Algebra and Minimal Updates

What we can use in mathematics to fuel PEFT

Fine-tuning in the frame of Linear Algebra

What is Fine-Tuning?
Fine-tuning is the process of adjusting a pre-trained model's weights to improve performance on a specific task.

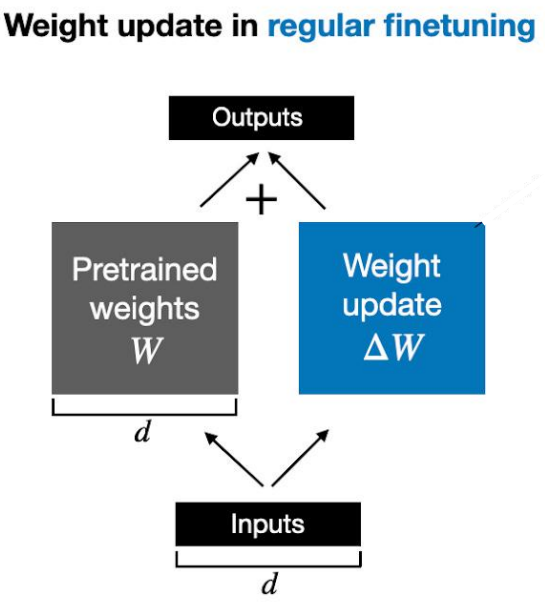
Linear Algebra in Fine-Tuning:

Matrix Operations:
Weights in neural networks are represented as matrices.
Fine-tuning adjusts these matrices through matrix multiplication and linear transformations during backpropagation.

Eigenvalues & Eigenvectors:
Fine-tuning explores the model's parameter space, where understanding eigenvalues helps in adjusting directions that significantly influence model performance.

Matrices in LLMs can be massive, containing millions of parameters each, resulting in a very computationally expensive operation when performing fine-tuning.

We can view fine-tuning as taking the original weights and adding a new matrix to it, representing the changes.



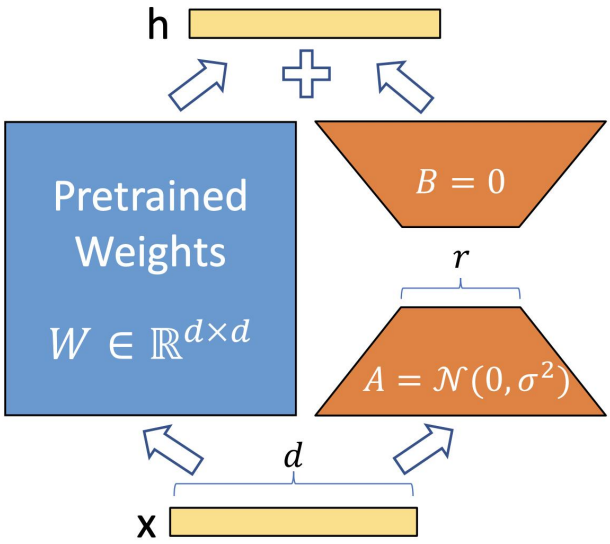
Low Rank Adapters - LoRA

LoRA is a technique for fine-tuning large language models (LLMs) by injecting low-rank matrices into the model's architecture. It allows efficient adaptation of models without retraining all weights.

Key Concepts:
Low-Rank Matrices:
LoRA introduces low-rank decomposition to the model's weight matrices, significantly reducing the number of parameters needed to be fine-tuned.

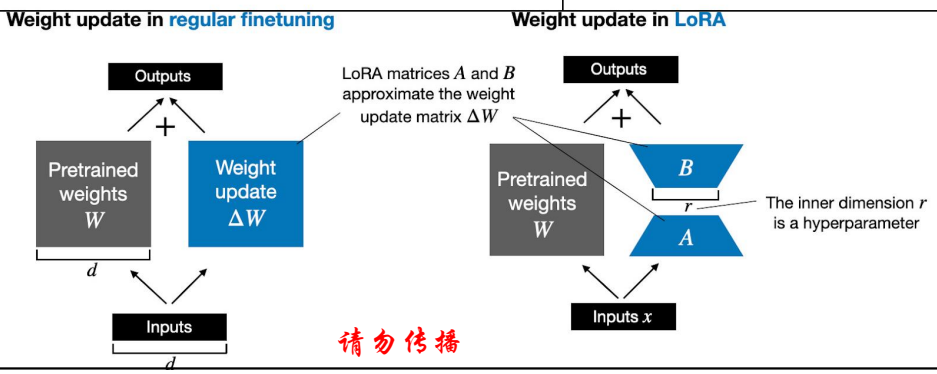
Parameter Efficiency:
Only a small fraction of the original model's parameters are updated making fine-tuning faster and more memory-efficient.

No Full Model Retraining:
Instead of updating all parameters, LoRA focuses on modifying just the low-rank adaptation matrices, keeping the rest of the model frozen.



Fine-tuning vs. LoRA

	Fine-Tuning	LoRA (Low-Rank Adaptation)
Parameter Updates	Updates all model parameters, requiring large-scale weight adjustments.	Updates only small, low-rank matrices, keeping most of the model frozen.
Memory Usage	High memory usage, adjusting all parameters consumes significant resources.	Low memory footprint, since only a subset of parameters is updated.
Training Speed	Slower, as all parameters need to be optimized, taking more time and resources.	Faster, as fewer parameters are updated, leading to quicker fine-tuning.
Compute Resources	Requires substantial compute power, often with multiple GPUs.	Requires fewer compute resources, ideal for faster adaptation and resource-limited environments.

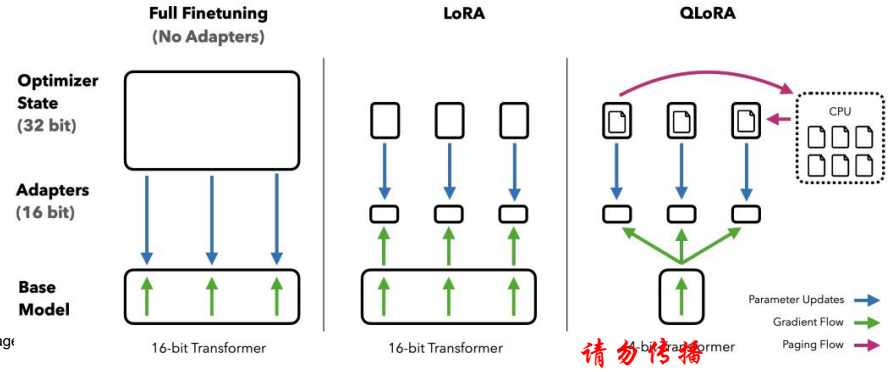


Quantization of Low Rank Adapters - QLoRA

QLoRA is LoRA with quantized linear layers in the base model. The adapters are identical to those of LoRA and kept in higher precision (BF16) during QLoRA training.

Compared to LoRA, QLoRA is:

- Up to 60% more memory-efficient, allowing for fine-tuning large models with smaller/less GPUs and/or higher batch size.
- Able to achieve the same accuracy, although a different convergence recipe is required.
- Between 50% and 200% slower than LoRA.

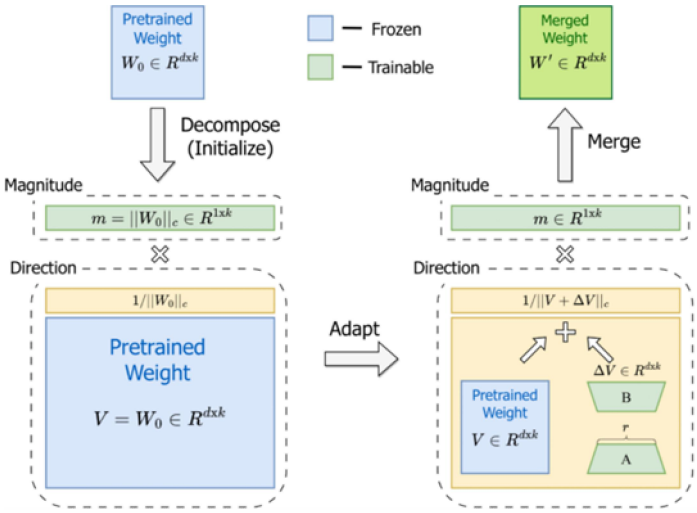


This can make QLoRA a viable choice in compute constrained environments

Alternatives to LoRA: DoRA

Weight-Decomposed Low-Rank Adaptation **(DoRA)**, is an enhanced alternative to LoRA.

- DoRA improves learning capacity and model stability, while maintaining the same inference speed, avoiding any additional overhead during inference.
- DoRA operates by decomposing the pre-trained model's weights into **magnitude** and **directional** components, and fine-tuning both.
- By leveraging the relatively small size of the directional component, DoRA enhances LoRA's efficiency in fine-tuning. Importantly, DoRA can merge with the original pre-trained weights before inference, ensuring no additional latency is introduced.
- DoRA builds on the success of LoRA by focusing on decomposing the weight matrices, targeting only the most important directional shifts. This approach reduces the computational complexity further without sacrificing the fine-tuning accuracy.

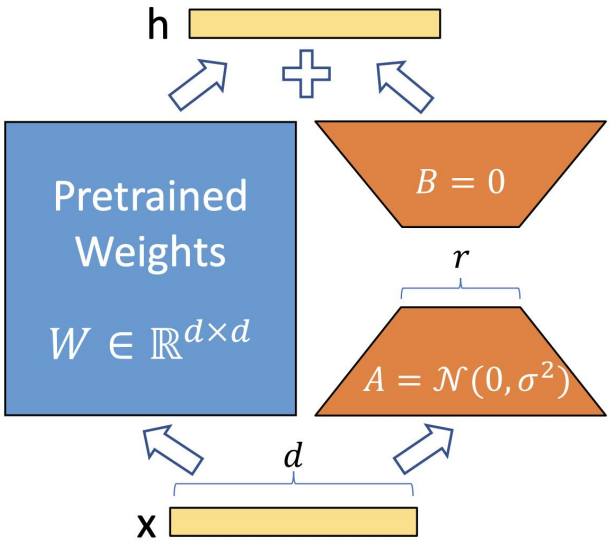


Wrap Up

Parameter-Efficient Fine-Tuning (PEFT)

- Today we covered some of the popular PEFT methods
- We motivated PEFT as a means for fine tuning and LLM interactions where compute resources are constrained
- A review of general methods such as quantization and pruning of neural networks was presented
- The Low-Rank Adaptors approach, LoRA, was presented as a means of more efficient fine-tuning of LLMs, exploiting the matrices that are involved in LLM fine tuning
- Quantized versions, and alternatives to LoRA, QLoRA and DoRA were also introduced as other areas to explore as PEFT methods gain popularity in the community and industry

---



Thank you!