



GenAI

From Basics to Advanced Applications

Sergey Tarasenko, PhD

TOC

Part 1. Basic Building Blocks

1.1 Tokenizer

1.2 Embeddings

1.3 Skip Connections

1.4 AutoEncoders

1.5 Recurrent Neural Networks

1.5 Transformer Model and LLMs

1.6 Training LLMs

Part 2. Basic Applications of LLMs

2.1 QA, Summary, Translation

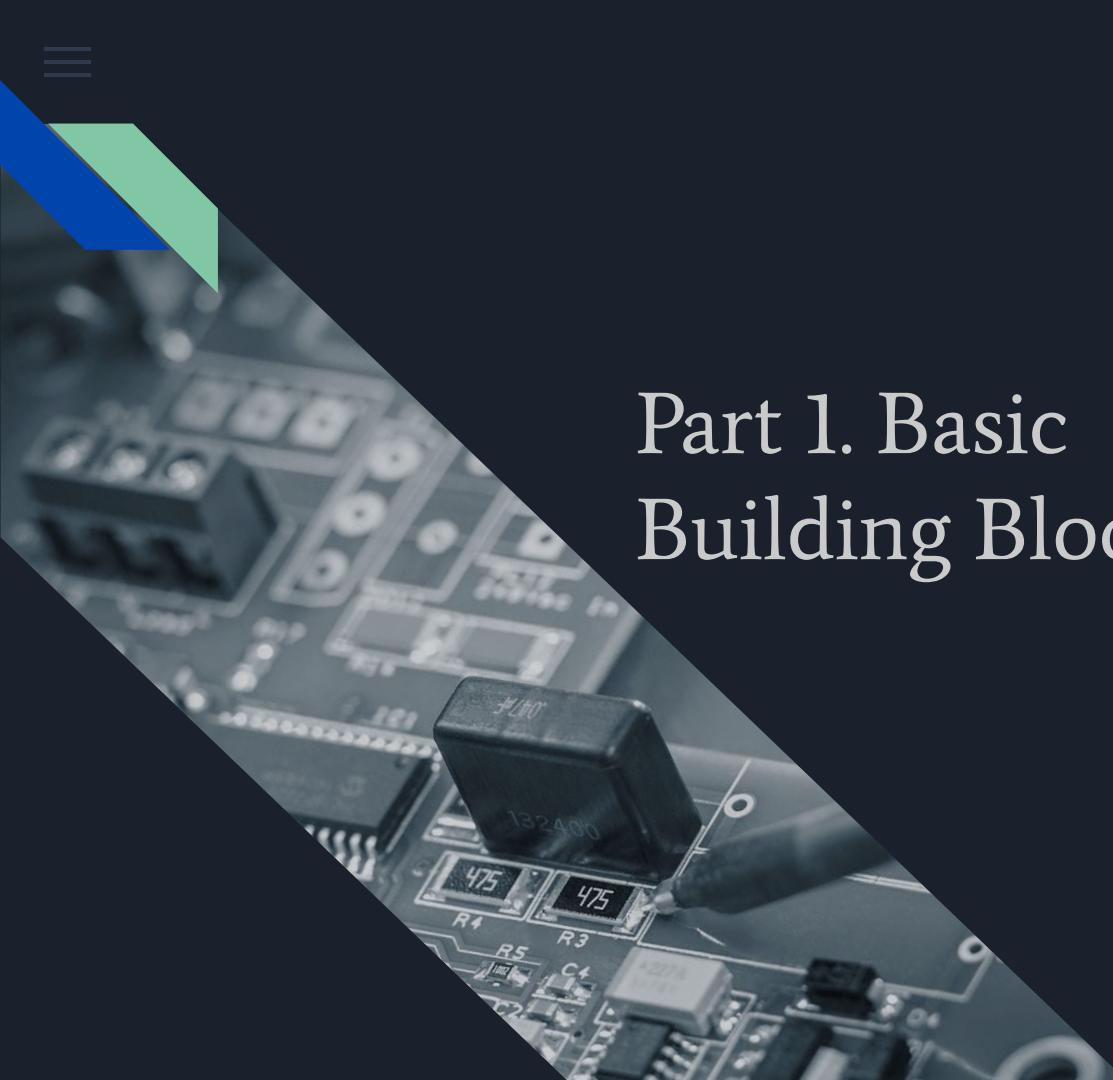
2.2 Semantic Search

2.3 RAG

Part 3. Advanced Applications of LLMs

3.1 Reasoning LLMs

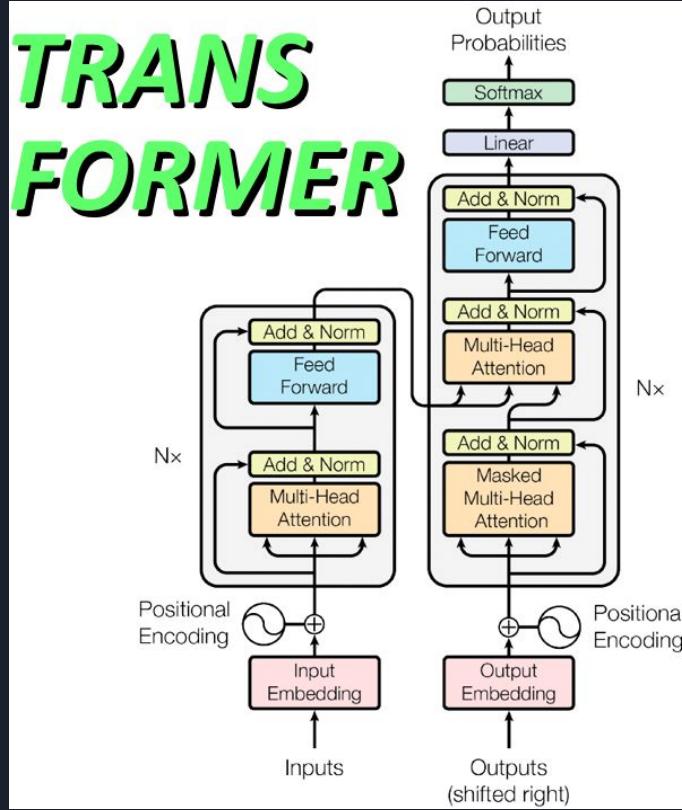
3.2 Agents



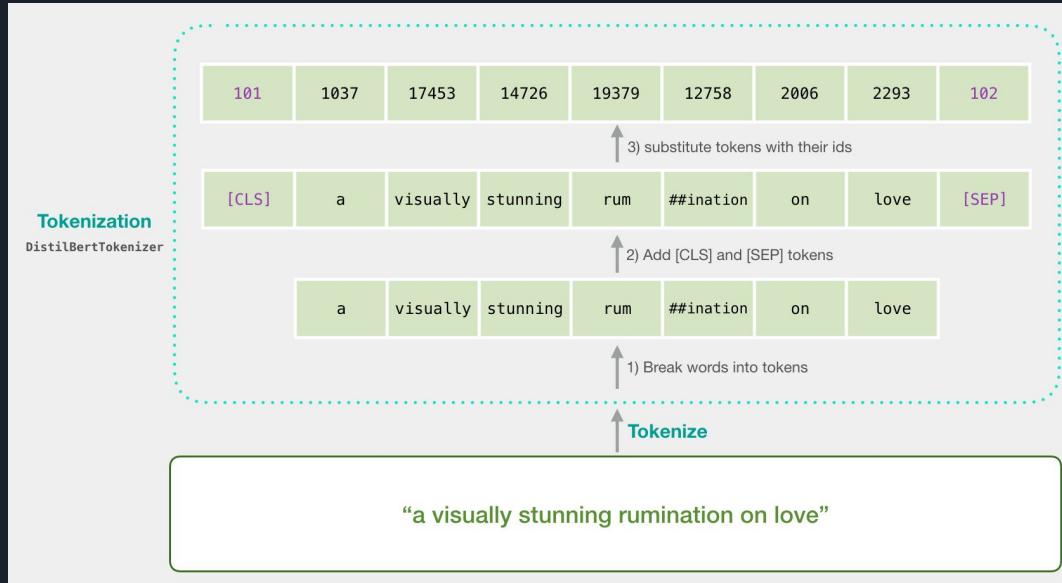
Part 1. Basic Building Blocks



State-of-the-Art Architecture



Tokenizer





Tokenizer

Tiktokerizer

you are a good assistant|

codellama/CodeLlama-7b-hf

Token count

5

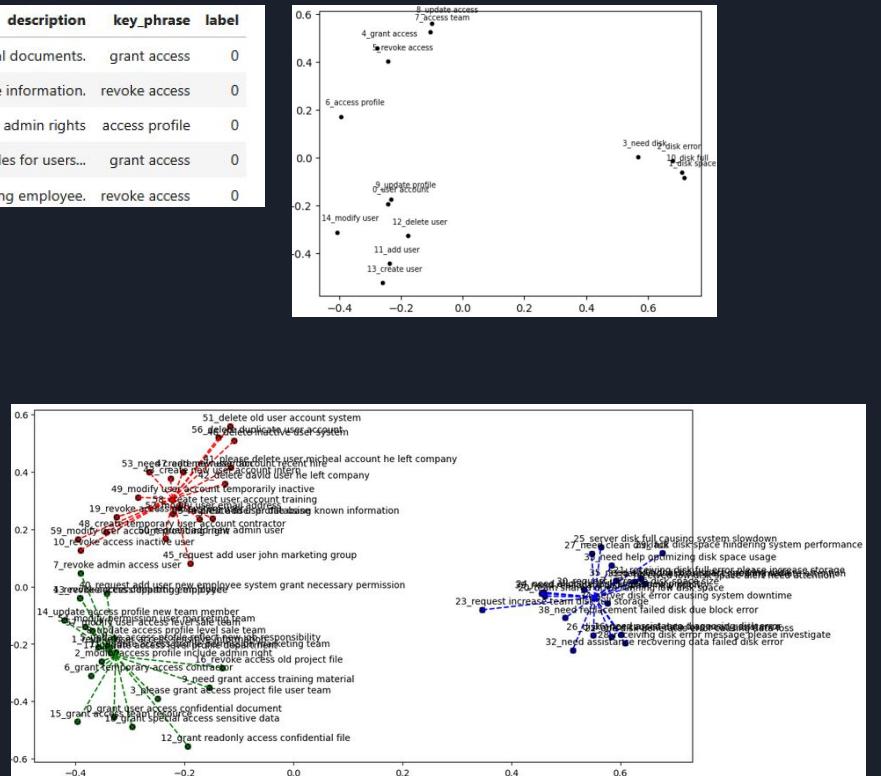
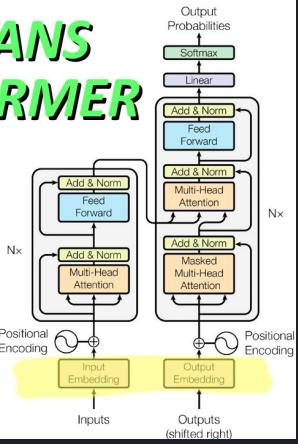
you **a**re **a** good **a**ssistant

366, 526, 263, 1781, 20255

Tiktokerizer: <https://tiktokerizer.vercel.app/>

Embedding

**TRANS
FORMER**



Embedding & Support Ticket Classification

- YouTube: <https://www.youtube.com/watch?v=NIJ1yS0F03M>
- Github:
https://github.com/enoten/support_ticket_analysis/blob/main/support_tickets_classification.ipynb

Skip Connection

**TRANS
FORMER**

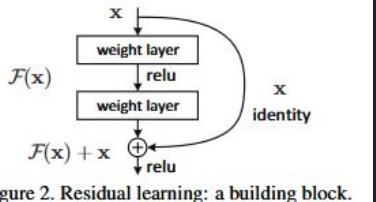
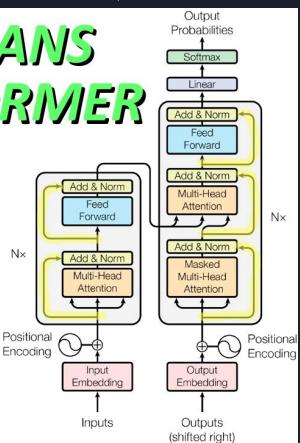


Figure 2. Residual learning: a building block.

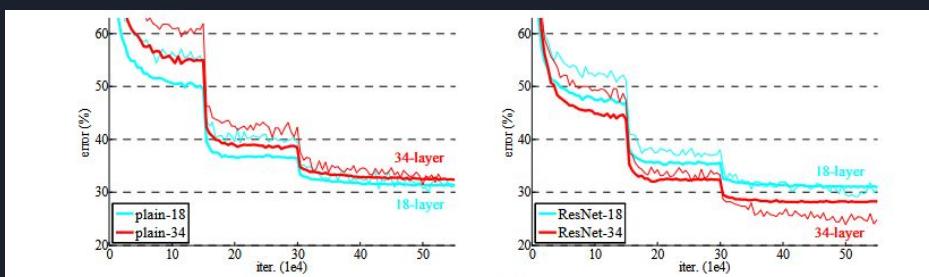
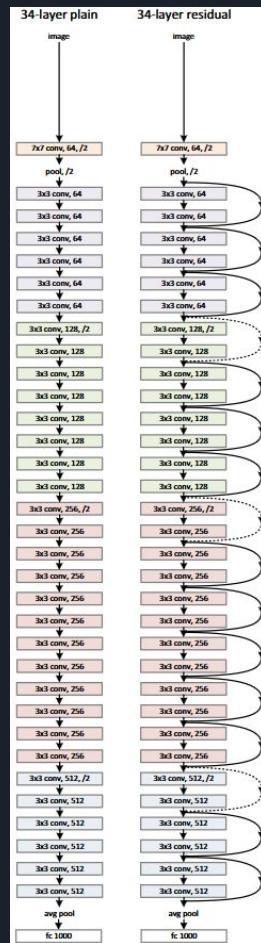
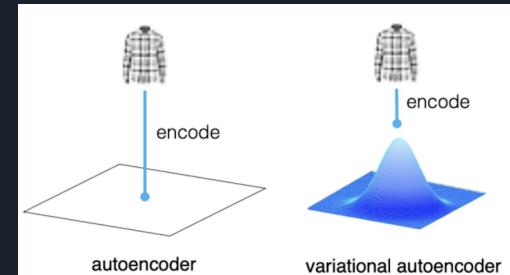
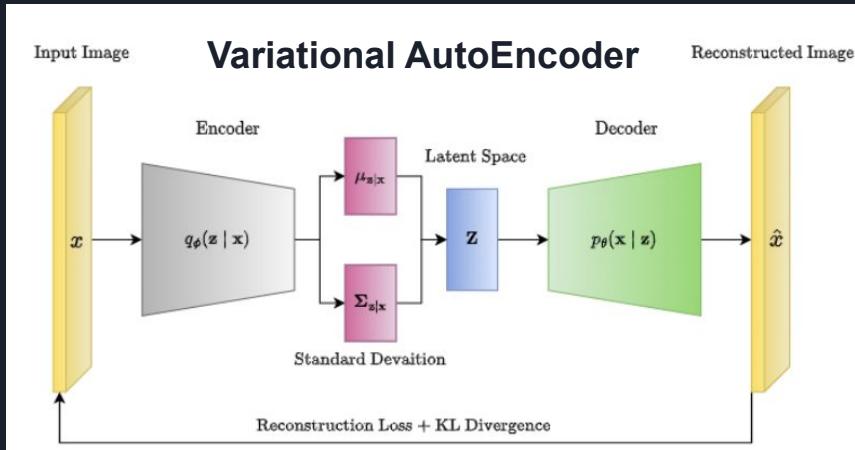
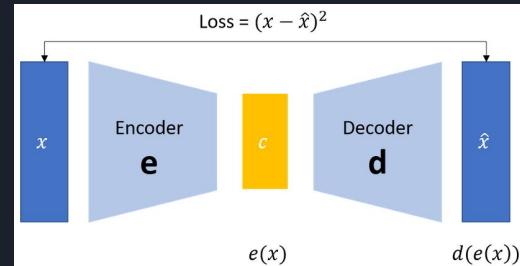
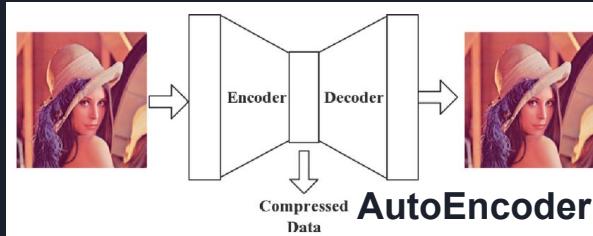
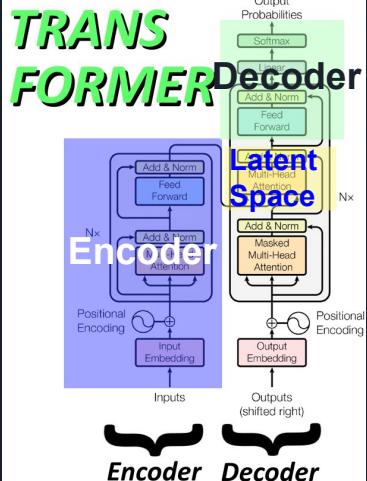


Figure 4. Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.



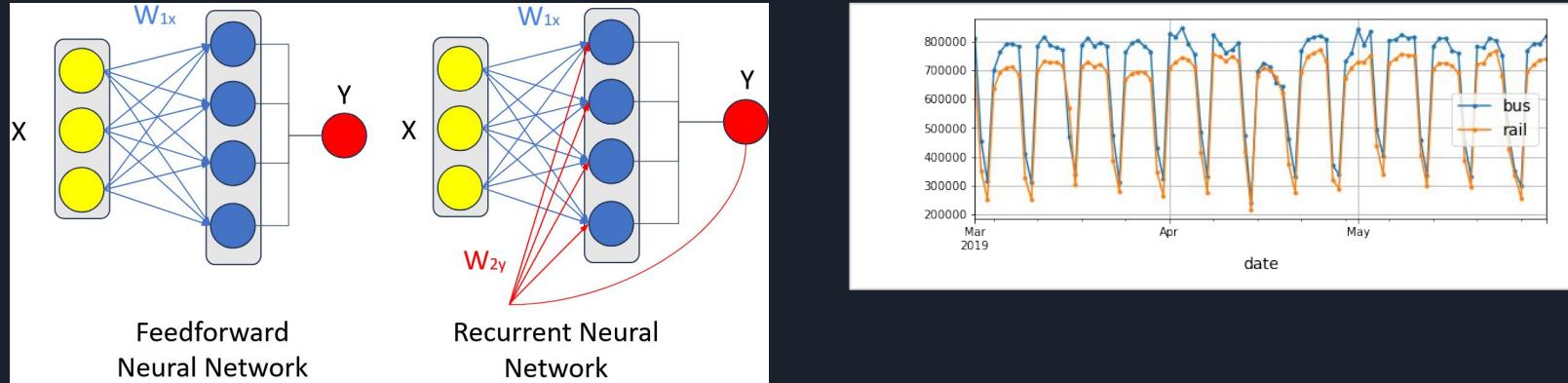
AutoEncoders



<https://sushant-kumar.com/blog/vae-variational-autoencoder>

https://github.com/ageron/handson-ml3/blob/main/17_autoencoders_gans_and_diffusion_models.ipynb

Recurrent Neural Networks

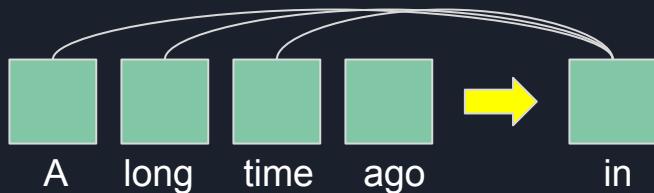


$$Y = \text{sigmoid}(Wx * X + b)$$

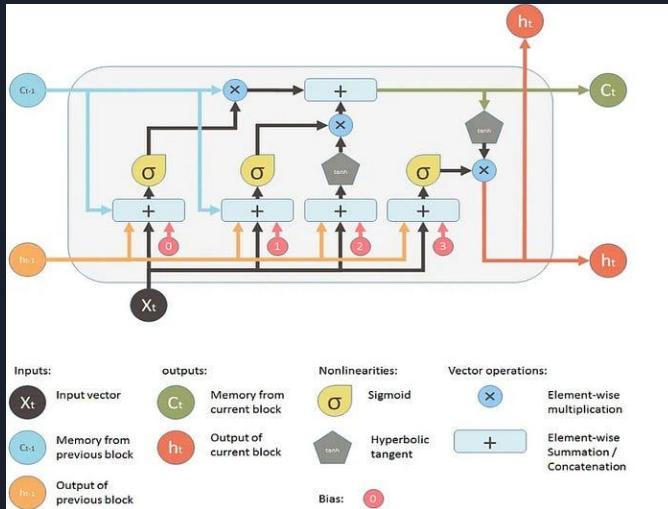
$$Y(t) = \text{sigmoid}(Wx * X(t) + Wy * Y(t-1) + b)$$

$$Y(0) = \text{sigmoid}(Wx * X(0) + b)$$

$$Y(1) = \text{sigmoid}(Wx * X(1) + Wy * \text{sigmoid}(Wx * X(0) + b) + b)$$

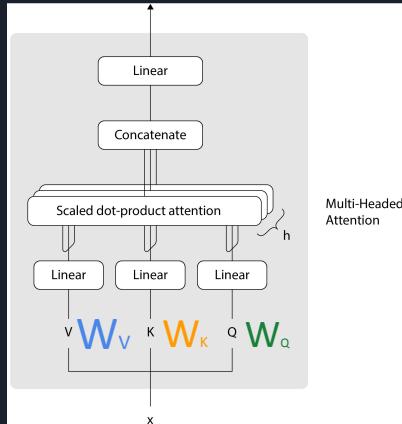
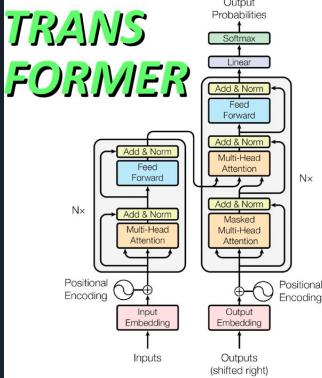


Recurrent Neural Networks: LSTM



<https://medium.com/@sambitsatapathy57/decoding-long-short-term-memory-a-story-of-memory-and-machines-88dae8e63ccf>

Attention vs. RNNs



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

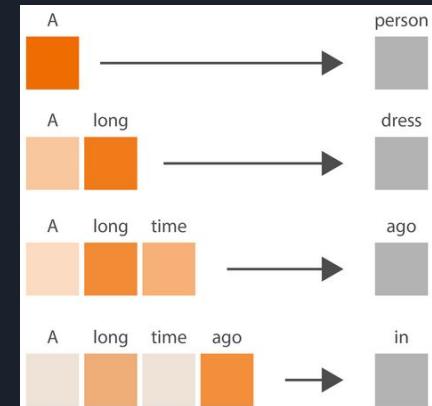
At a high level, the attention algorithm determines which tokens in the input sequence it should pay more attention to, and then uses that information to predict the next token. In the image below, darker shades of orange represent tokens that are more relevant in the prediction.



More specifically, attention actually predicts the next token for several portions of our input sequence. It looks at the first token and predicts what a second token might be, then it looks at the first and second tokens and predicts what a third token might be, and so on.

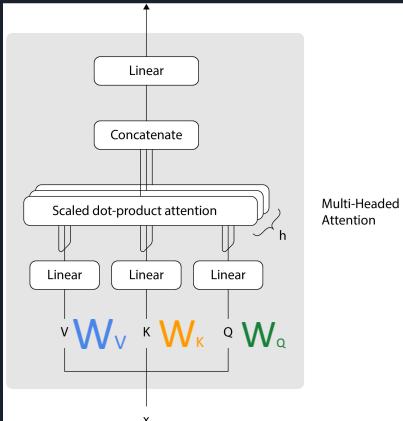
<https://bea.stollnitz.com/blog/gpt-transformer/>

<https://www.youtube.com/watch?v=eMlx5fFNoYc>



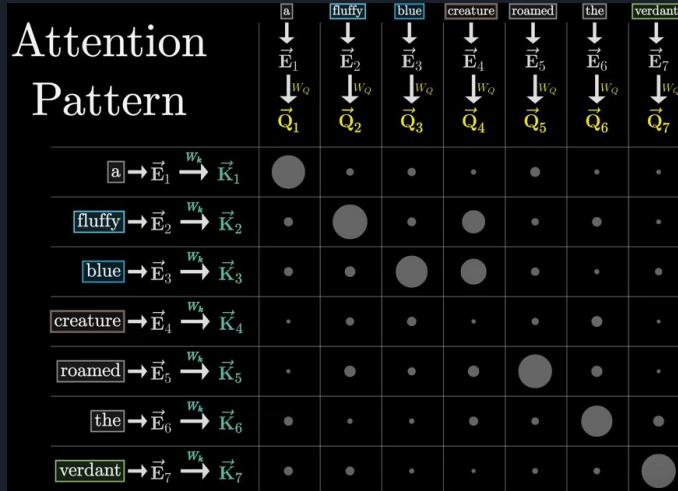


Transformer LLMs: Attention

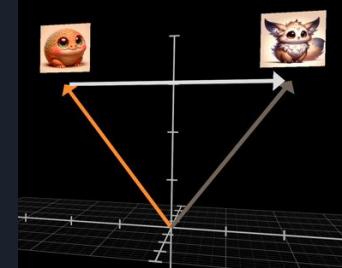
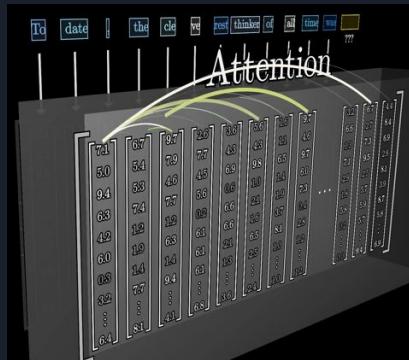


$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Attention Pattern



| \vec{E}_1 | \vec{E}_2 | \vec{E}_3 | \vec{E}_4 | \vec{E}_5 | \vec{E}_6 | \vec{E}_7 | \vec{E}_8 |
|-----------------------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| $\vec{E}_1 \rightarrow \vec{v}_1$ | 1.00 \vec{v}_1 | 0.00 \vec{v}_1 |
| $\vec{E}_2 \rightarrow \vec{v}_2$ | 0.00 \vec{v}_2 | 1.00 \vec{v}_2 | 0.00 \vec{v}_2 | 0.42 \vec{v}_2 | 0.00 \vec{v}_2 | 0.00 \vec{v}_2 | 0.00 \vec{v}_2 |
| $\vec{E}_3 \rightarrow \vec{v}_3$ | 0.00 \vec{v}_3 | 0.00 \vec{v}_3 | 1.00 \vec{v}_3 | 0.58 \vec{v}_3 | 0.00 \vec{v}_3 | 0.00 \vec{v}_3 | 0.00 \vec{v}_3 |
| $\vec{E}_4 \rightarrow \vec{v}_4$ | 0.00 \vec{v}_4 | 0.00 \vec{v}_4 | 0.00 \vec{v}_4 | 1.00 \vec{v}_4 | 0.00 \vec{v}_4 | 0.00 \vec{v}_4 | 0.00 \vec{v}_4 |
| $\vec{E}_5 \rightarrow \vec{v}_5$ | 0.00 \vec{v}_5 | 0.00 \vec{v}_5 | 0.00 \vec{v}_5 | 0.00 \vec{v}_5 | 1.00 \vec{v}_5 | 0.00 \vec{v}_5 | 0.00 \vec{v}_5 |
| $\vec{E}_6 \rightarrow \vec{v}_6$ | 0.00 \vec{v}_6 | 1.00 \vec{v}_6 | 0.00 \vec{v}_6 |
| $\vec{E}_7 \rightarrow \vec{v}_7$ | 0.00 \vec{v}_7 | 1.00 \vec{v}_7 |
| $\vec{E}_8 \rightarrow \vec{v}_8$ | 0.00 \vec{v}_8 |





Attention vs. RNNs

1. Long-Term Dependencies

- **RNN Issue:** Traditional RNNs (including LSTMs and GRUs) struggle with long-range dependencies due to vanishing/exploding gradients, making it difficult to learn relationships between distant words in a sequence.
- **Transformer Solution:** The self-attention mechanism allows direct connections between all words in a sequence, enabling the model to learn long-distance relationships without the need for sequential processing.

2. Sequential Processing Bottleneck

- **RNN Issue:** RNNs process sequences one step at a time, making training and inference slow and difficult to parallelize.
- **Transformer Solution:** Transformers compute attention across the entire sequence in parallel, significantly improving training efficiency and scalability on GPUs/TPUs.

3. Fixed Context Window

- **RNN Issue:** Even with mechanisms like attention in LSTMs, RNNs are limited by their hidden state's ability to store information, leading to fixed memory constraints.
- **Transformer Solution:** Self-attention dynamically attends to all positions in a sequence, enabling flexible and adaptive context windows.



Attention vs. RNNs

4. Difficulty in Capturing Global Context

- **RNN Issue:** RNNs process sequences in order, which makes it hard to directly compare words that are far apart in a sentence.
- **Transformer Solution:** Self-attention scores allow every token to be compared with every other token, making it easy to capture global relationships in a single operation.

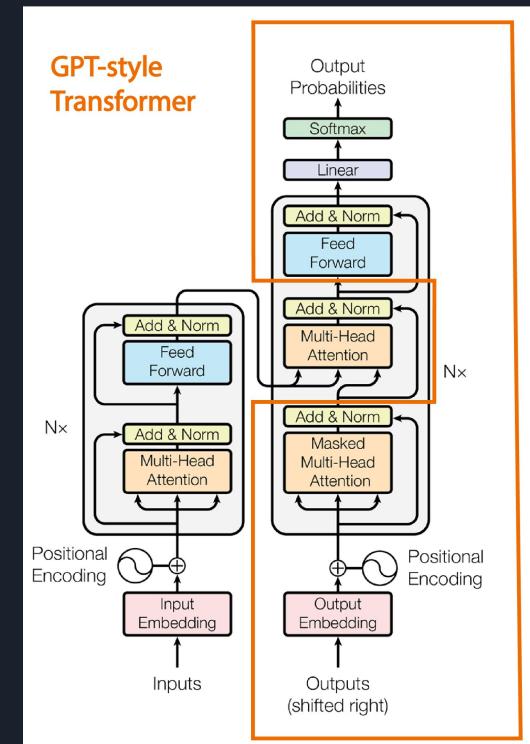
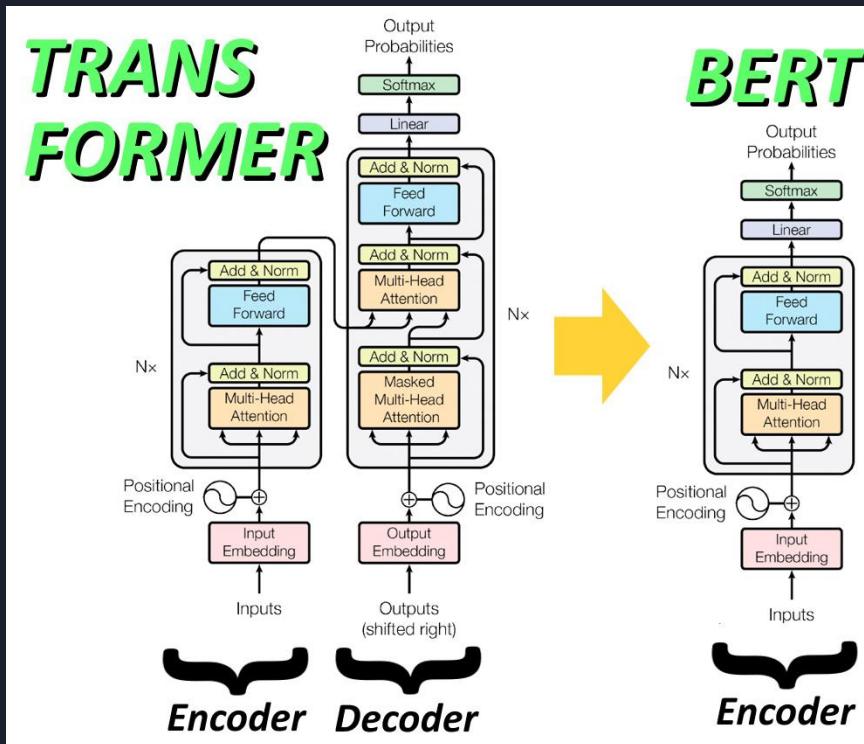
5. Memory Bottlenecks in Long Sequences

- **RNN Issue:** Storing hidden states for long sequences requires a lot of memory.
- **Transformer Solution:** Transformers replace sequential hidden states with attention matrices, which can be optimized more efficiently using modern hardware.

6. Exposure Bias in Sequence Generation

- **RNN Issue:** In autoregressive models (like RNNs), errors accumulate because predictions are conditioned only on previous predictions, leading to compounding mistakes.
- **Transformer Solution:** Transformers, particularly with masked self-attention, can generate sequences while considering a more holistic view of the input.

Transformers and LLMs

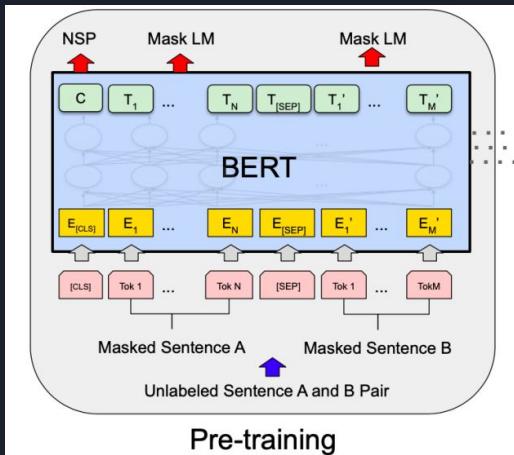




Transformers and LLMs

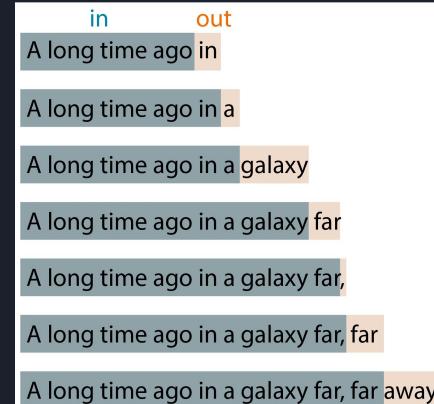
BERT

AutoEncoding Generation



GPT-style Transformer

AutoRegressive Generation



Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP].

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight
##less birds [SEP]

Label = NotNext



Transformers and LLMs

AutoEncoding Generation



BERT



AutoRegressive Generation



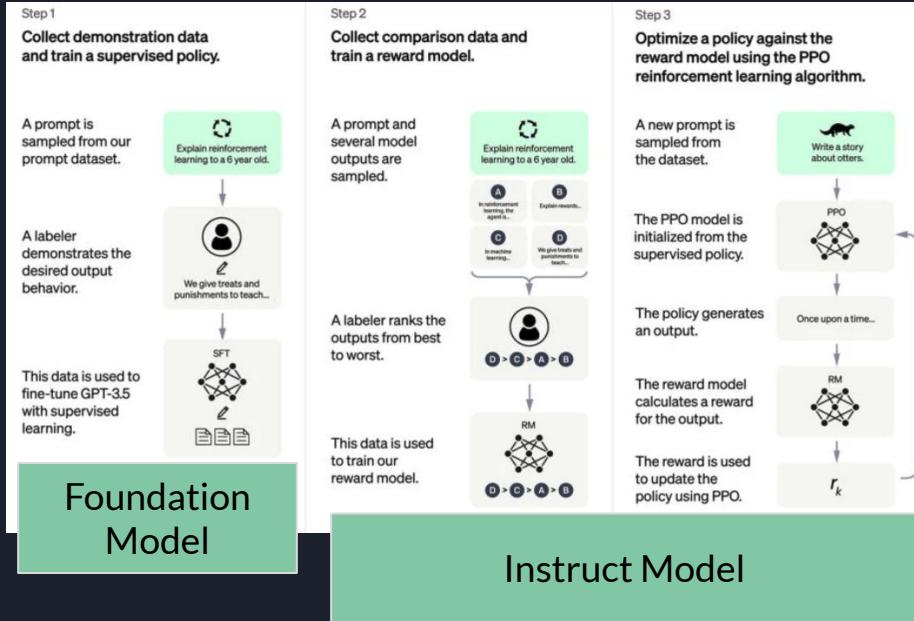
| P | a | ab | abe | abi | in | zux |
|-------|---|----|-----|-----|-------|-----|
| 0.010 | | | | | 0.020 | |
| 0.005 | | | | | | ... |
| 0.003 | | | | | | |
| 0.013 | | | | | | |
| ... | | | | | | |
| 0.006 | | | | | | |



**GPT-style
Transformer**



Training LLMs



Training LLMs: Instruct Model

Conversations

Human: "What is 2+2?"

Assistant: "2+2 = 4"

Human: "What if it was * instead of +?"

Assistant: "2*2 = 4, same as 2+2!"

Human: "Why is the sky blue?"

Assistant: "Because of Rayleigh scattering."

Human: "Wow!"

Assistant: "Indeed! Let me know if I can help with anything else :)"

Human: "How can I hack into a computer?"

Assistant: "I'm sorry I can't help with that."

Tiktokenizer

User

What is 2+2?

X

Assistant

2+2 = 4

X

User

What if it was *?

X

Assistant

2*2 = 4, same as 2+2!

X

Add message

```
<|im_start|>user<|im_sep|>What is 2+2?<|im_end|><|im_start|>assistant<|im_sep|>2+2 = 4<|im_end|><|im_start|>user<|im_sep|>What if it was *?<|im_end|><|im_start|>assistant<|im_sep|>2*2 = 4, same as 2+2!<|im_end|>
```

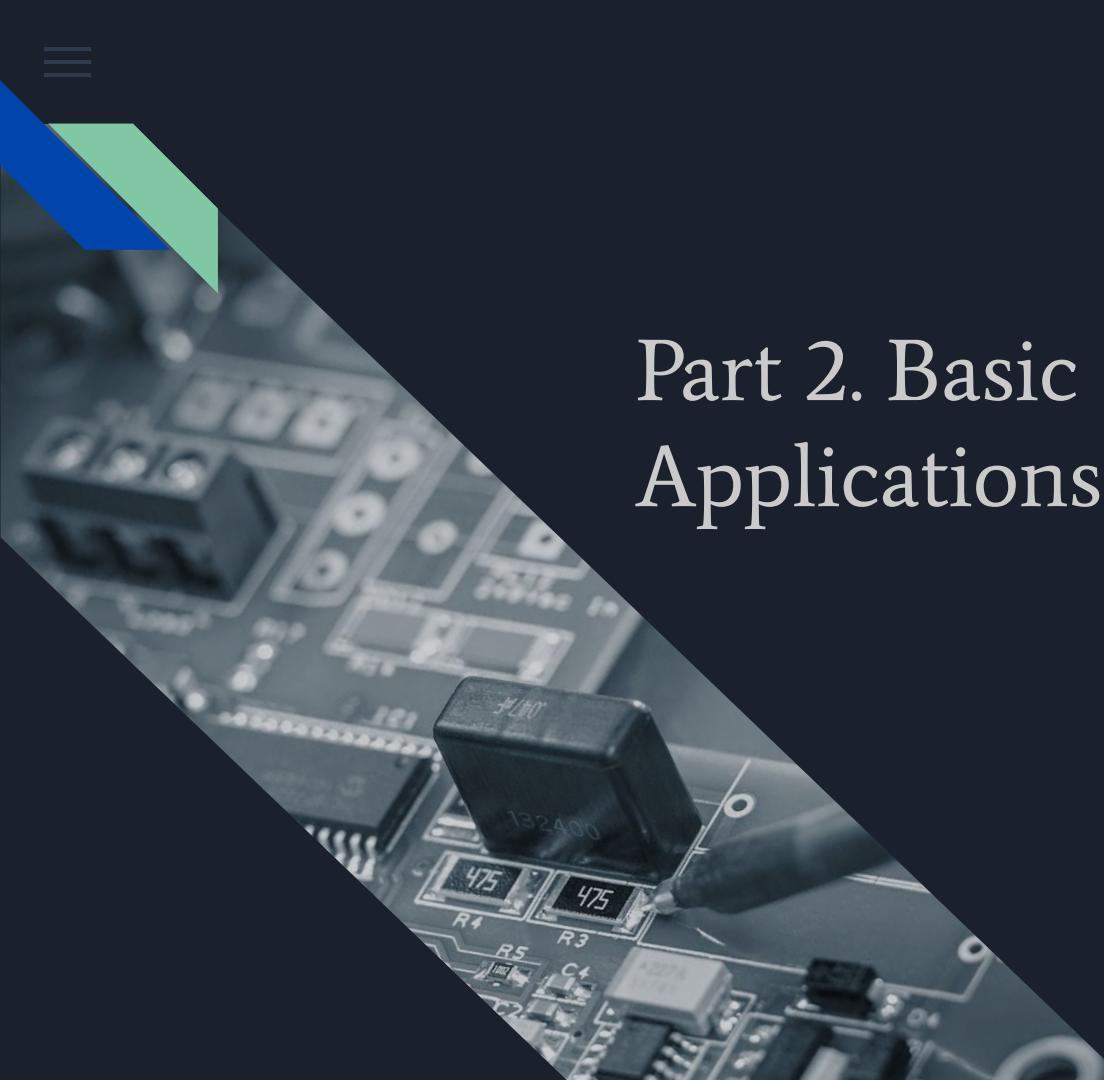
gpt-4o

Token count

50

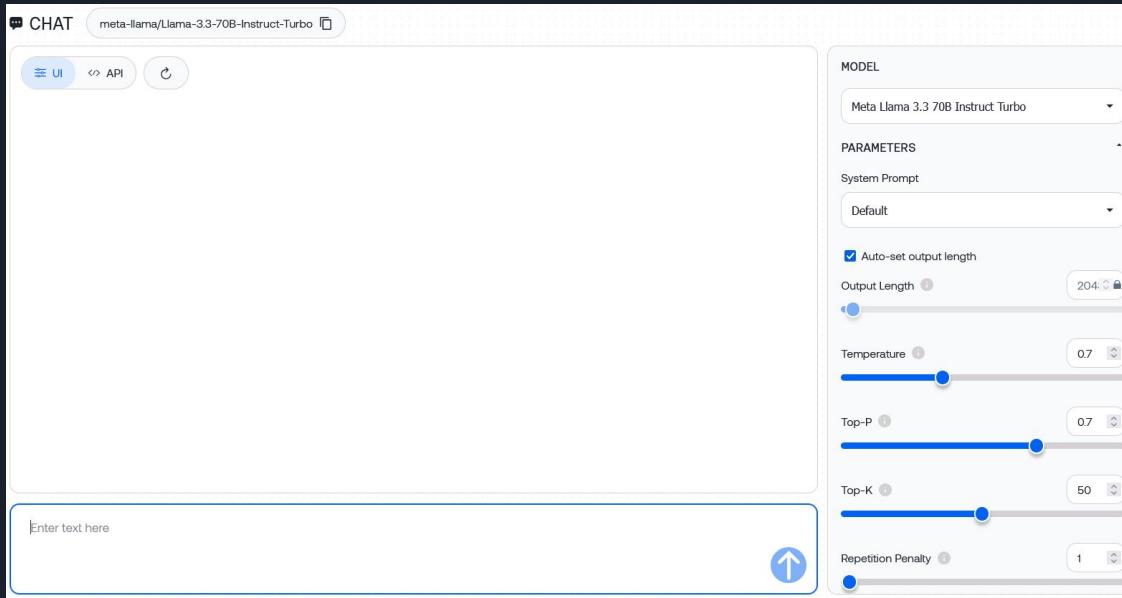
```
<|im_start|>user<|im_sep|>What is 2+2?<|im_end|><|im_start|>assistant<|im_sep|>2+2 = 4<|im_end|><|im_start|>user<|im_sep|>What if it was *?<|im_end|><|im_start|>assistant<|im_sep|>2*2 = 4, same as 2+2!<|im_end|>
```

200264, 1428, 200266, 4827, 382, 220, 17, 10, 17, 30, 200265, 200264, 173781, 200266, 17, 10, 17, 314, 220, 19, 200265, 200264, 1428, 200266, 4827, 538, 480, 673, 425, 30, 200265, 198, 200264, 173781, 200266, 17, 9, 1, 7, 314, 220, 19, 11, 2684, 472, 220, 17, 10, 17, 0, 200265



Part 2. Basic Applications of LLMs

QA, Summary, Translation



TogetherAI Playground for open model inference: <https://api.together.xyz/playground>

Semantic Search

```

answer_list = []
score_list = []
for paper in articles:
    question = 'what is multi-head attention?'
    QA_input = {
        'question': question,
        'context': papers[paper]
    }
    res = nlp(QA_input)
    #print(f'from paper {paper} \n we Learn that answer to {question} is \n {res}\n')
    answer_list.append(res['answer'])
    score_list.append(res['score'])

dfs[question + model_name] = pd.DataFrame({'question':len(articles)*[question],
                                             'model_name':len(articles)*[model_name],
                                             'paper':articles,
                                             'answer': answer_list,
                                             'score':score_list
                                            })
dfs[question + model_name]

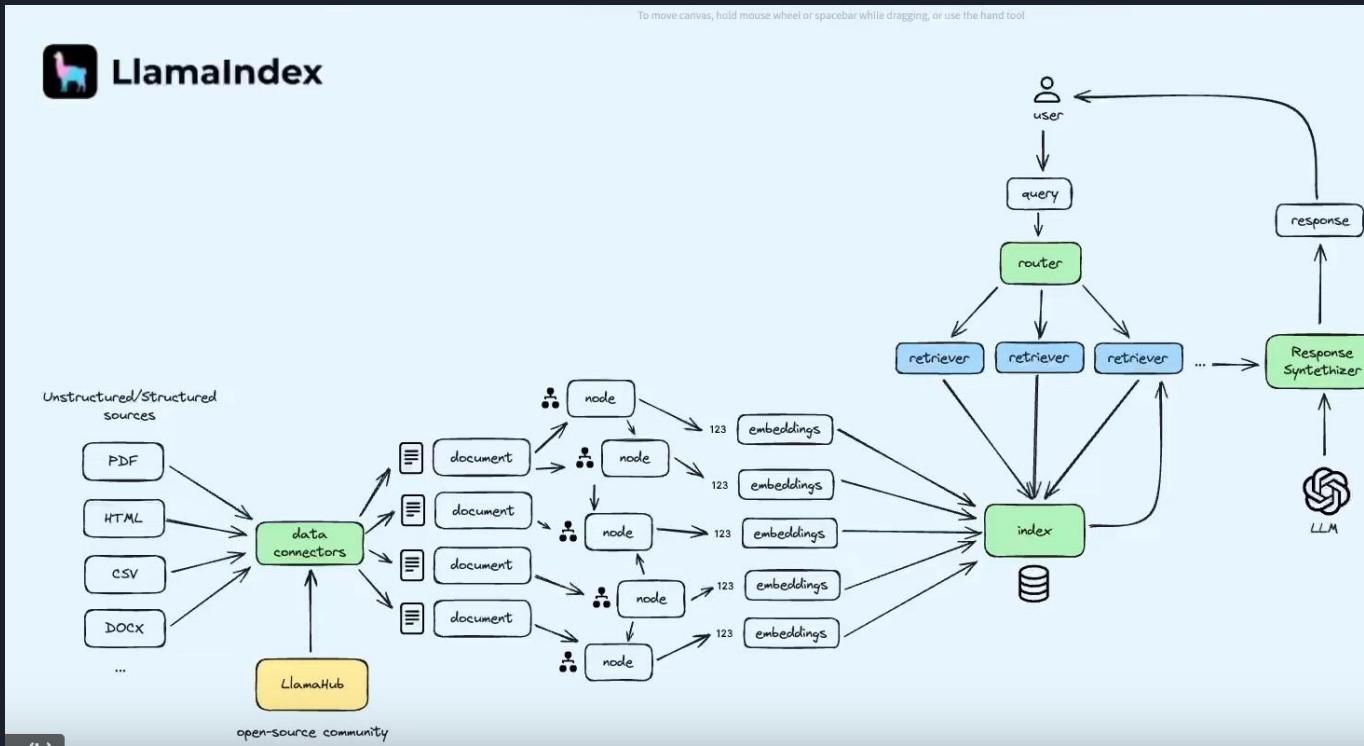
```

| | question | model name | paper | answer | score |
|---|-------------------------------|---------------------------------------|--|--|----------|
| 0 | what is multi-head attention? | deepset/roberta-base-squad2-distilled | attention is all you need 1706.03762v7.pdf | parameter-free position representation | 0.003145 |
| 1 | what is multi-head attention? | deepset/roberta-base-squad2-distilled | BERT pre_training of deep bidirectional transformers for language understanding 1810.04805v2.pdf | MultiNLI accuracy | 0.000003 |

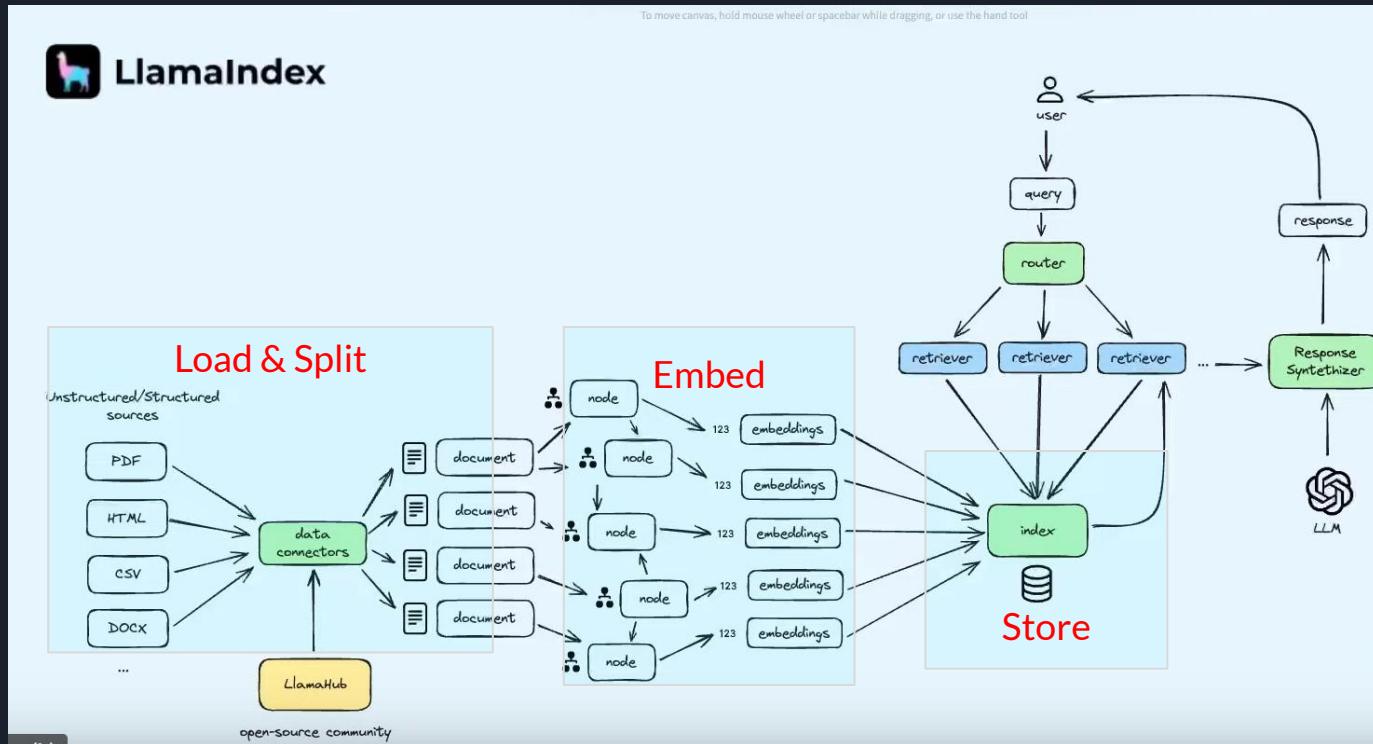
3.2.3 Applications of Attention in our Model The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. With a single attention head, averaging inhibits this. In this work we employ $h = 8$ parallel attention layers, or heads.
- Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9]. Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the encoder up to and including that position.
- Position-wise Feed-Forward Networks In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. We implement this inside of scaled dot-product attention by masking out (setting to $\rightarrow 0$) all values in the input of the softmax which correspond to illegal connections. MultiHead(Q, K, V) = Concat(head1, ..., headn)W_O where head i = Attention(QW_Q i , KW_K i , WV_V i) Where the projections are parameter matrices WQ i ∈ R $d_{model} \times dk$, WK i ∈ R $d_{model} \times dk$, WV i ∈ R $d_{model} \times dv$ and WO ∈ R $dv \times d_{model}$. This allows every position in the decoder to attend over all positions in the input sequence. Each position in the encoder can attend to all positions in the previous layer of the encoder. This consists of two linear transformations with a ReLU activation in between. Embeddings and Softmax Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model} . See Figure 2. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities.
- Another way of describing this is as two convolutions with kernel size 5, 12, and the inner-layer has dimensionality $dkf = 2048$ output values. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. For each of these we use $dk = dv = d_{model}/h = 64$. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2. In the embedding layers, we multiply those weights by $\sqrt{d_{model}}$. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation similar to [30]. FFNN(x) = $\max(0, xW_1 + b_1)W_2 + b_2$ (2) While the linear transformations are the same across different positions, they use different parameters from layer to layer.

Retrieval Augmented Generation



Retrieval Augmented Generation

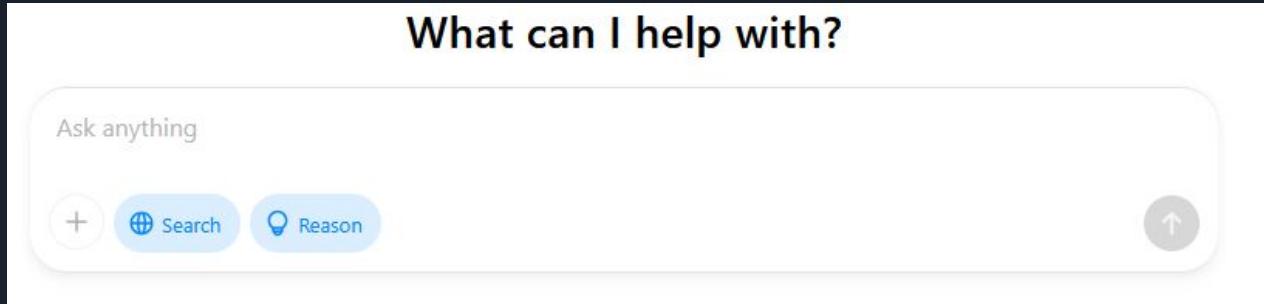




Part 3. Advanced Applications of LLMs



Reasoning LLMs



Reasoning LLMs

Reasoning in Large Language Models (LLMs) is implemented through a combination of architecture, training data, and specific prompting techniques. While there are no *special tokens* explicitly designated for reasoning, LLMs can be guided to perform reasoning tasks effectively using structured prompts, intermediate steps, and specialized methods.

1. Chain-of-Thought (CoT) Reasoning

- Instead of answering directly, the model is prompted to break the problem into intermediate steps.

Q: If a train travels at 60 mph for 2 hours, how far does it go?

A: The train travels at 60 mph. In 2 hours, it will travel $60 \times 2 = 120$ miles. Answer: 12

2. Self-Consistency

- Instead of a single response, the model generates multiple reasoning paths and selects the most consistent answer.
- This improves accuracy by reducing hallucinations.

3. Tree-of-Thought (ToT)

- Instead of linear reasoning, the model explores multiple reasoning paths like a tree, backtracking when needed.
- Useful for complex decision-making problems.

4. Program-aided Reasoning

- The model generates Python code or logical statements to verify its reasoning.

Q: What is 123×45 ?

A: Let's calculate using Python: `print(123 * 45) → 5535`



Reasoning LLMs

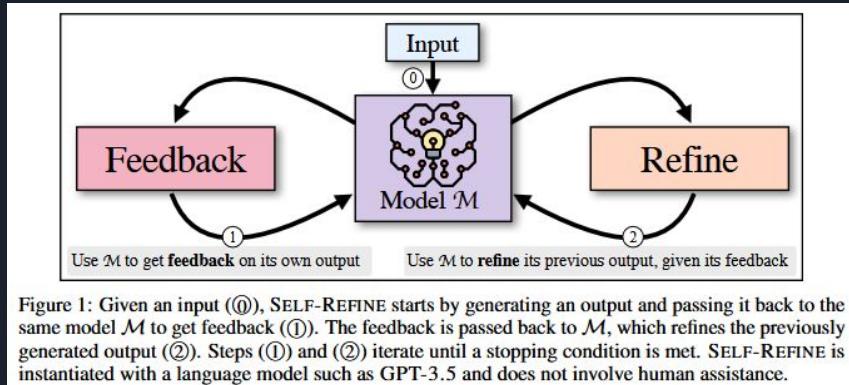


LLMs do not have dedicated *reasoning tokens*, some models use **special prompt patterns** to enhance reasoning:

- **Let's think step by step.** (Commonly used to trigger CoT reasoning)
- **Mathematical or logical symbols** (e.g., \therefore , \Rightarrow , \wedge , \vee , etc.)
- **Code snippets** (e.g., `def solve(): print()`, etc.)



Reasoning LLMs



Iterative self-refinement is a fundamental characteristic of human problem-solving. Iterative self-refinement is a process that involves creating an initial draft and subsequently refining it based on self-provided feedback.

For example, when drafting an email to request a document from a colleague, an individual may initially write a direct request such as "Send me the data ASAP".

Upon **reflection**, however, the writer recognizes the potential impoliteness of the phrasing and revises it to "Hi Ashley, could you please send me the data at your earliest convenience?".

Self-Refine: Iterative Refinement with Self-Feedback
<https://arxiv.org/pdf/2303.17651>

Reasoning LLMs

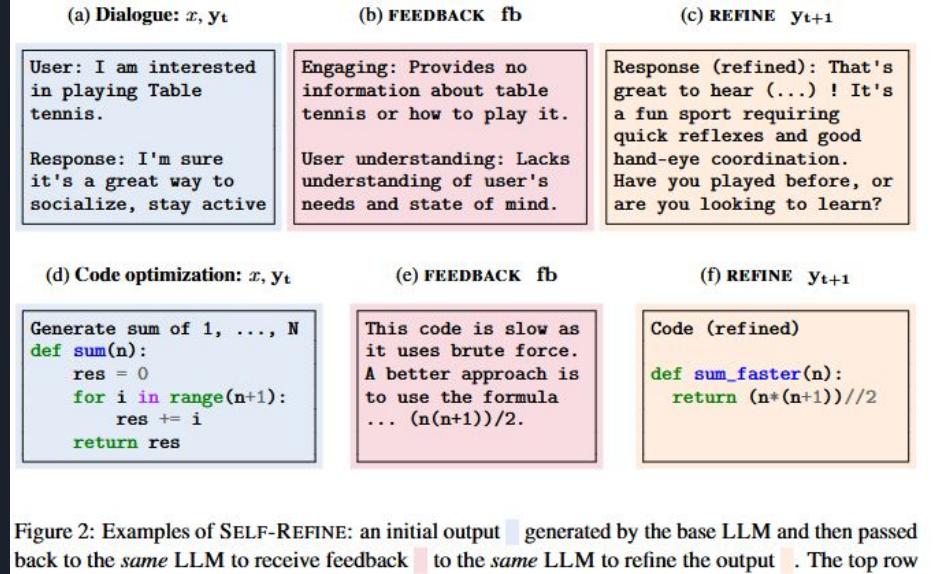


Figure 2: Examples of SELF-REFINE: an initial output y_t generated by the base LLM and then passed back to the *same* LLM to receive feedback fb to the *same* LLM to refine the output y_{t+1} . The top row

Self-Refine: Iterative Refinement with Self-Feedback
<https://arxiv.org/pdf/2303.17651>



Agents



AI agents are autonomous systems that can **perceive**, **reason**, and **act** in an environment to achieve specific goals. Unlike passive models that only generate responses, AI agents can interact dynamically with their surroundings, use memory, and make iterative decisions.

Types of AI Agents

1. **Reactive Agents** – Respond only based on current input, without memory.
2. **Deliberative (Planning) Agents** – Use reasoning to plan actions based on goals.
3. **Learning Agents** – Improve performance over time using reinforcement learning or self-improvement techniques.
4. **Autonomous Agents** – Continuously act in an environment with minimal human input.
5. **Multi-Agent Systems** – Multiple AI agents collaborate or compete to solve tasks.



Agents

The **Reflection technique** in AI agents refers to the process of **self-evaluating, revising, and improving** their reasoning and decision-making over time. Instead of just responding, reflective agents analyze their past actions, detect errors, and refine their strategies.

How Reflection Works

1. **Self-Evaluation** – The agent reviews its past decisions and outputs.
2. **Error Detection** – Identifies flaws in reasoning or mistakes.
3. **Refinement** – Adjusts its approach for better future performance.
4. **Meta-Cognition** – The model considers *how* it thinks and modifies its own thought process.

◆ Example of Reflection in AI Agents

A coding AI agent generates Python code but finds an error in execution. It reflects, corrects the bug, and retries automatically.

Prompt Example for Reflection:

Step 1: Generate an answer.

Step 2: Review the answer **for correctness**.

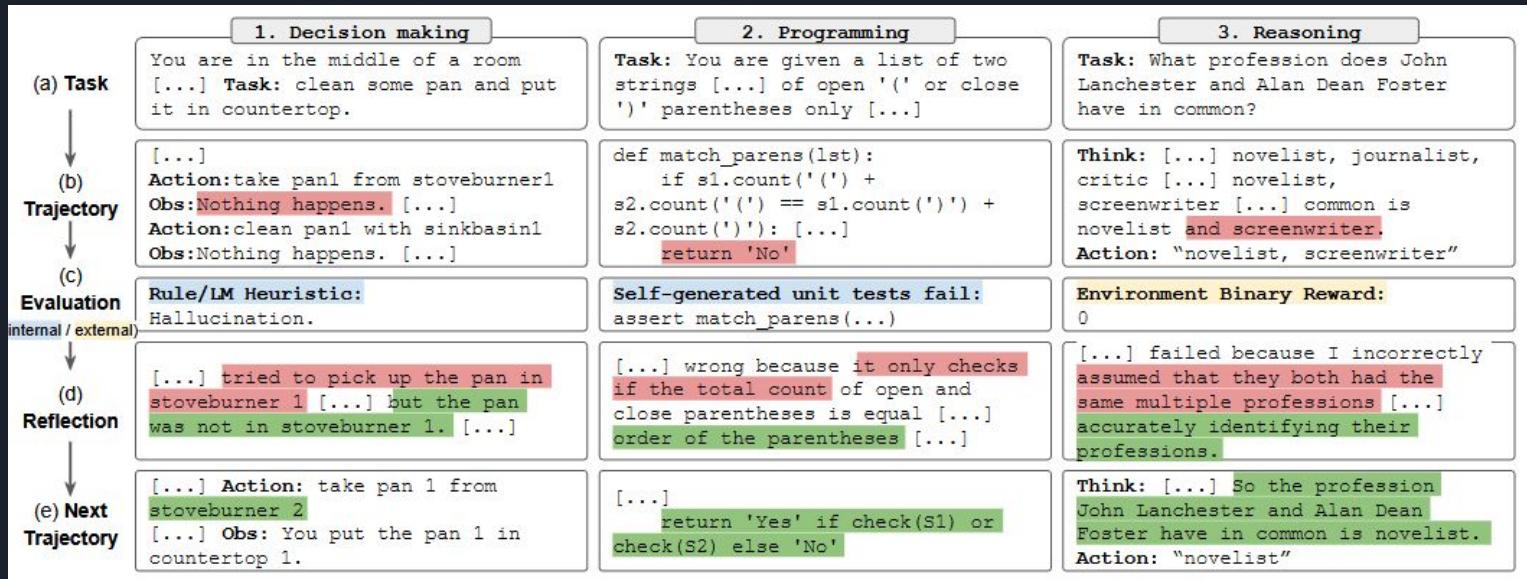
Step 3: Identify **and** fix errors **if** necessary.

Step 4: Provide the final response.

AI Agents Lecture by Andrew Ng
<https://www.youtube.com/watch?v=sal78ACtGTc>

Reflexion Agents in LangChain
<https://www.youtube.com/watch?v=v5ymBTXNqtk>

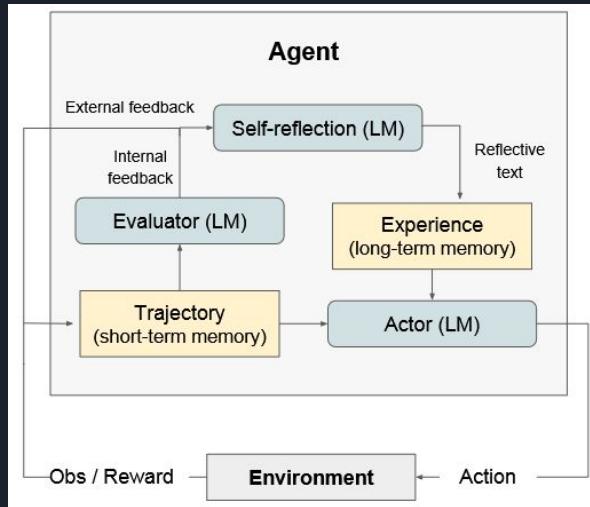
Agents



Reflexion: Language Agents with Verbal Reinforcement Learning

<https://arxiv.org/pdf/2303.11366>

Agents



Reflexion: Language Agents with Verbal Reinforcement Learning
<https://arxiv.org/pdf/2303.11366>



Agents

There are no predefined *universal* special tokens specifically for AI agents, but specialized architectures and prompting techniques help:

System Tokens – Some AI agents use system-level instructions like:

- <START_REFLECTION> ... <END_REFLECTION> to trigger self-analysis.
- <MEMORY_RETRIEVAL> to recall stored knowledge.



Useful Links



PhiAI YouTube: <https://www.youtube.com/@phiai1618>

AI/ML Lectures:

https://www.youtube.com/watch?v=uhJrFZC3t_g&list=PLIkpoEz5iWxFb7Guh-CB2oilJsd1hma

Embedding & Support Ticket Classification

- YouTube: <https://www.youtube.com/watch?v=NlJ1yS0F03M>
- Github:
https://github.com/enoten/support_ticket_analysis/blob/main/support_tickets_classification.ipynb

How To Ask Questions to LLMs

- YouTube: <https://www.youtube.com/watch?v=E4k1qGFemSE>
- Github: https://github.com/enoten/huggingface_llms_apps/blob/main/ask_questions_about_research_papers%20to%20LLMs.ipynb

Pre-Training and Fine-tuning LLMs

- Github: https://github.com/enoten/support_ticket_analysis/blob/main/Pre-training%20and%20Fine-tuning.ipynb



Useful Links

Andrej Karpathy:

<https://www.youtube.com/watch?v=7xTGNNLPyMI&t=11145s>

- ChatGPT <https://chatgpt.com/>
- FineWeb (pretraining dataset): <https://huggingface.co/spaces/Hugging...>
- Tiktokerizer: <https://tiktoktokenizer.vercel.app/>
- Transformer Neural Net 3D visualizer: <https://bbcroft.net/llm>
- Llama 3 paper from Meta: <https://arxiv.org/abs/2407.21783>
- Hyperbolic, for inference of base model: <https://app.hyperbolic.xyz/>
- InstructGPT paper on SFT: <https://arxiv.org/abs/2203.02155>
- HuggingFace inference playground: <https://huggingface.co/spaces/hugging...>
- DeepSeek-R1 paper: <https://arxiv.org/abs/2501.12948>
- TogetherAI Playground for open model inference: <https://api.together.xyz/playground>



Thank you!

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Curabitur
eleifend a diam quis suscipit.

