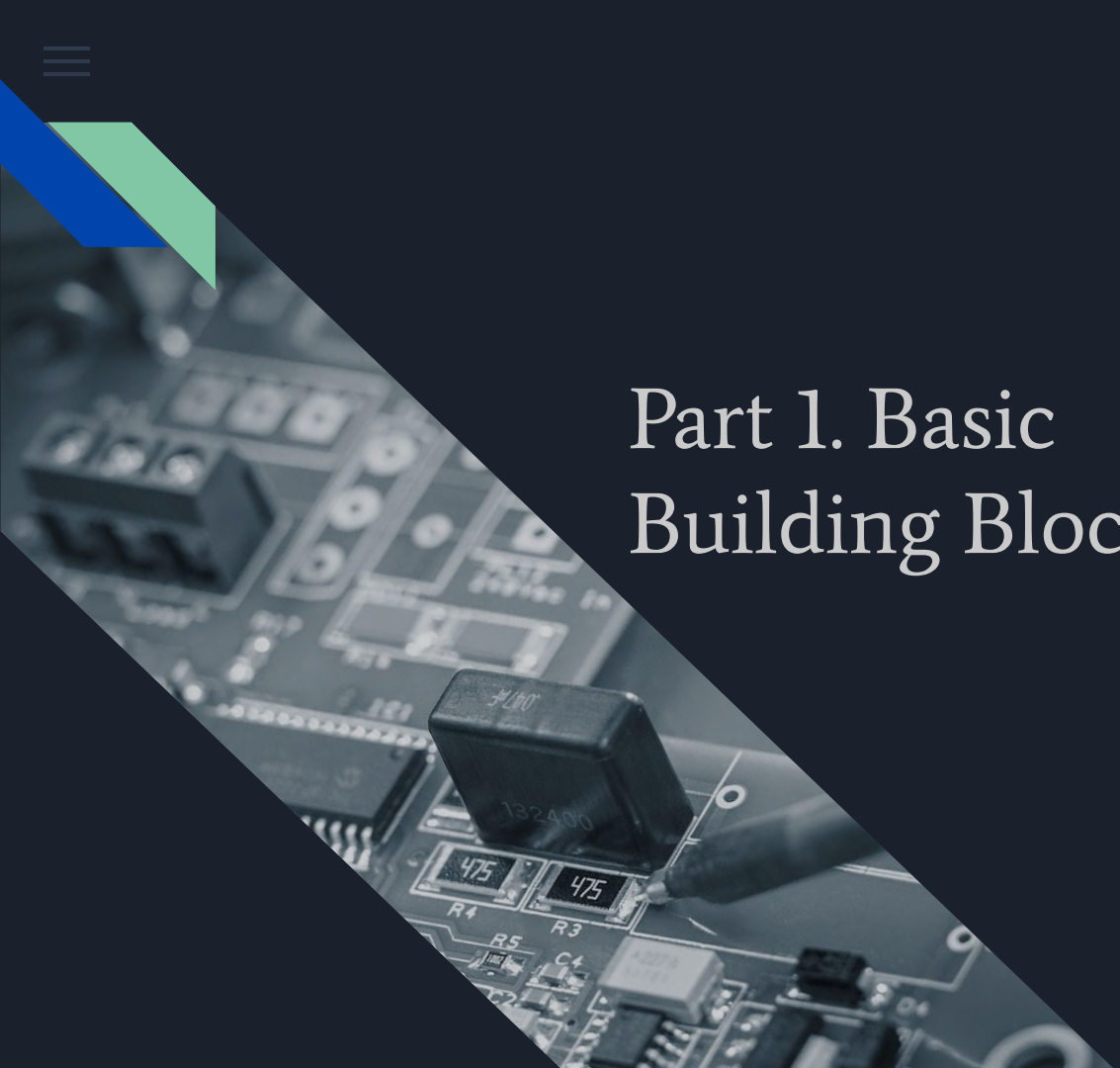
GenAI

From Basics to Advanced Applications

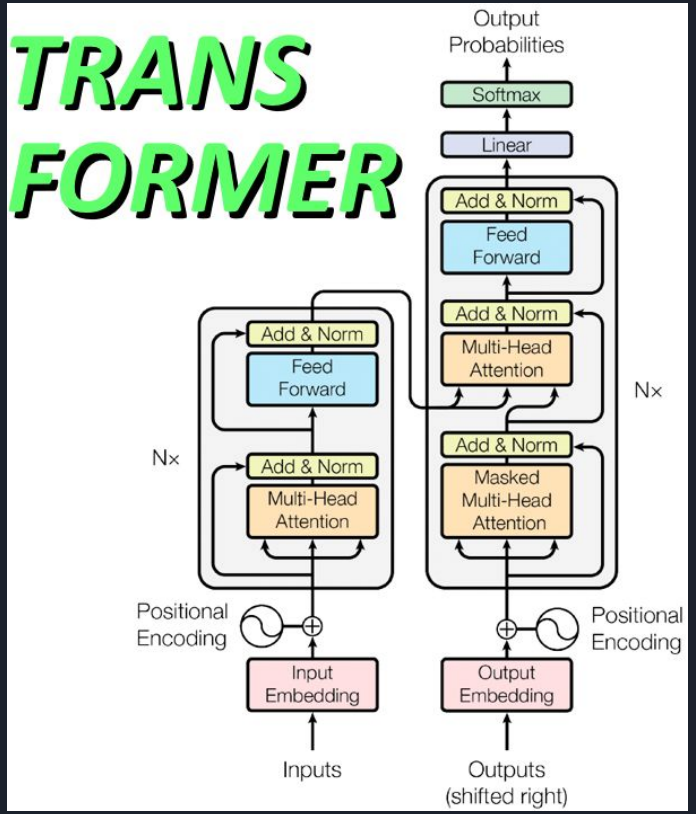
Sergey Tarasenko, PhD

TOC

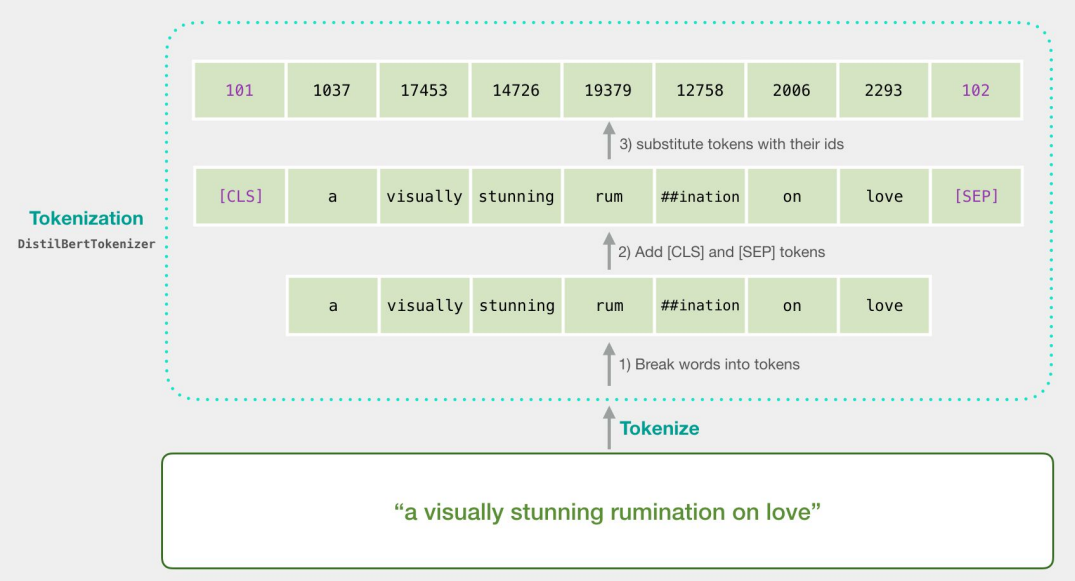


Part 1. Basic Building Blocks

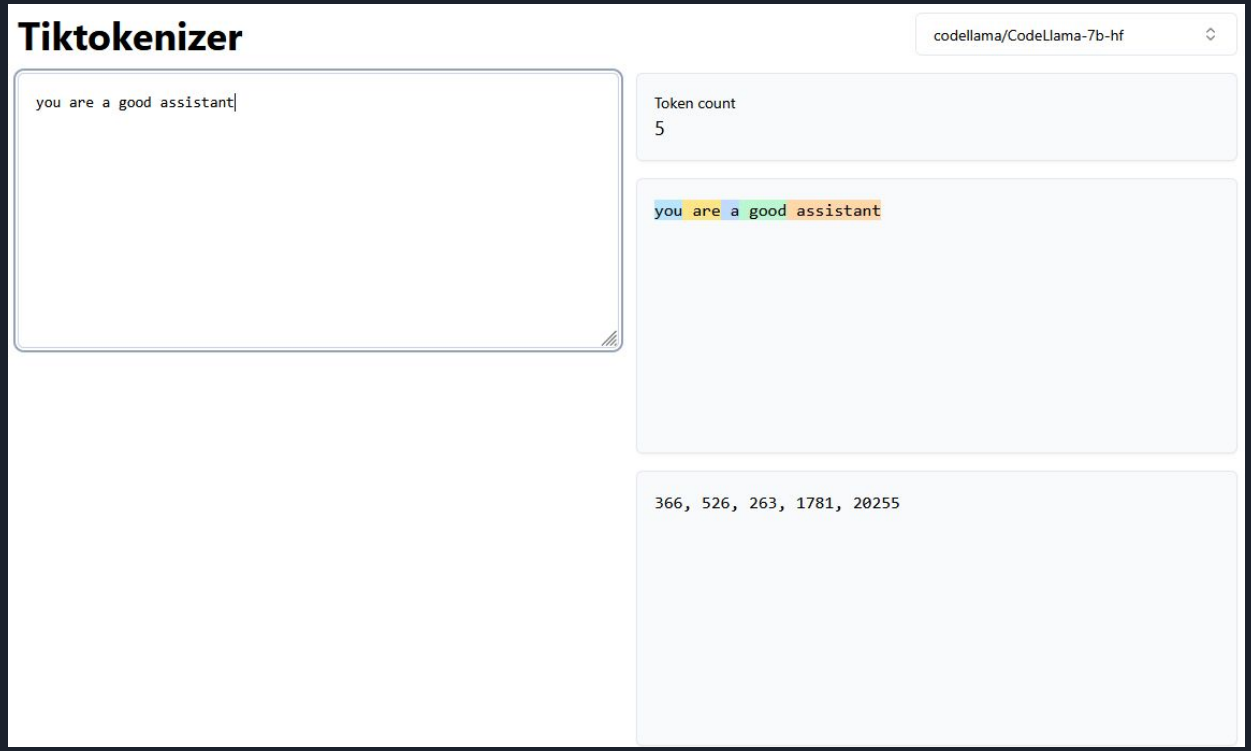
State-of-the-Art Architecture



Tokenizer

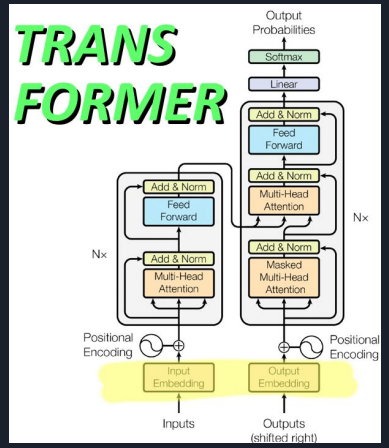

Tokenization
DistilBertTokenizer
101 1037 17453 14726 19379 12758 2006 2293 102
3) substitute tokens with their ids
[CLS] a visually stunning rum ##ination on love [SEP]
2) Add [CLS] and [SEP] tokens
a visually stunning rum ##ination on love
1) Break words into tokens
Tokenize
“a visually stunning rumination on love”

Tokenizer

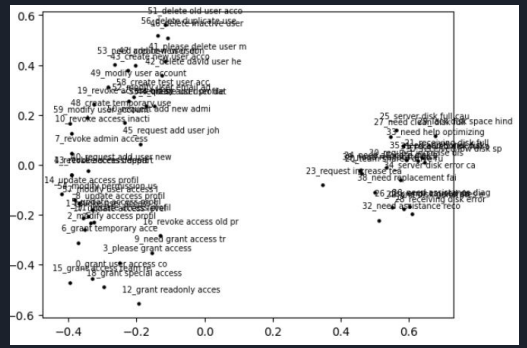


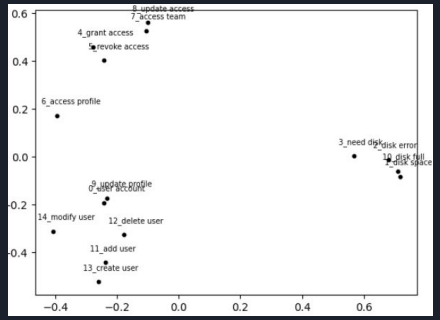
Tiktokenizer: https://tiktokenizer.vercel.app/

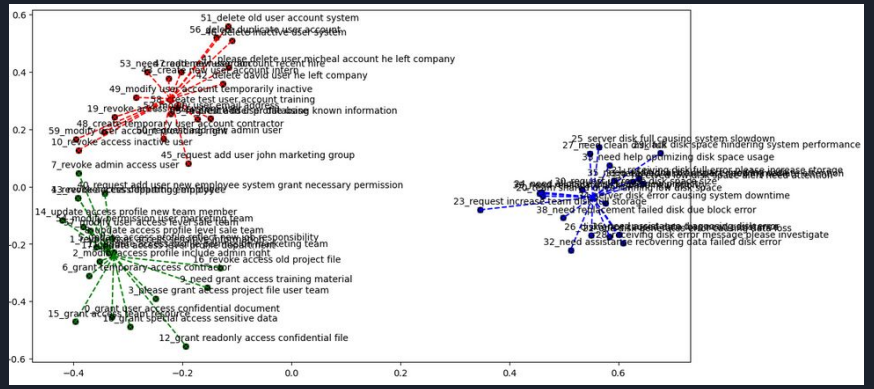
Embedding



	subject	description	key_phrase	label
0	Grant Access to Confidential Files	Grant user access to confidential documents.	grant access	0
1	Revoke Access to Sensitive Data	Revoke user access to sensitive information.	revoke access	0
2	Update Access Profile	Modify my access profile to include admin rights	access profile	0
3	Grant Access Request for New Project	Please grant access to project files for users...	grant access	0
4	Access Removal for Departing Employee	Revoke access for departing employee.	revoke access	0



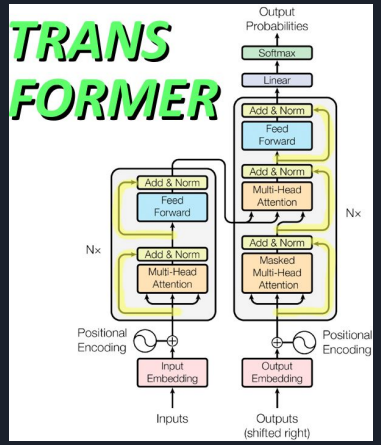




Embedding & Support Ticket Classification

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

Skip Connection



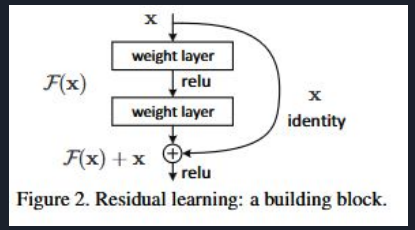


Figure 2. Residual learning: a building block.

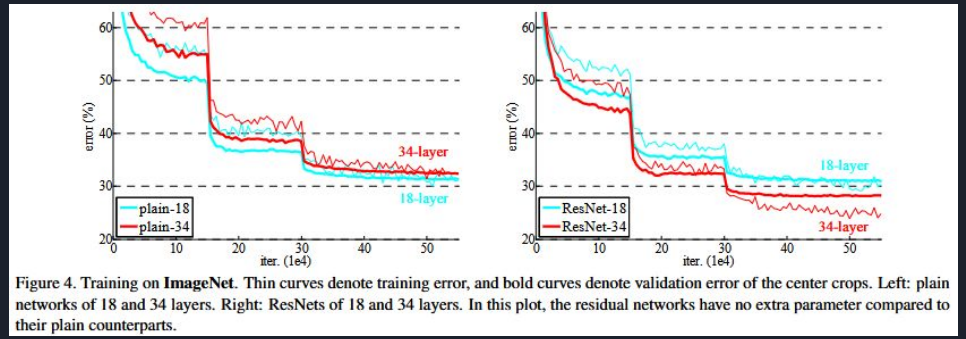
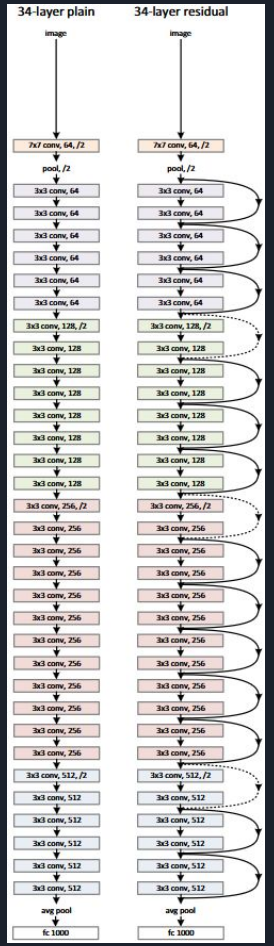
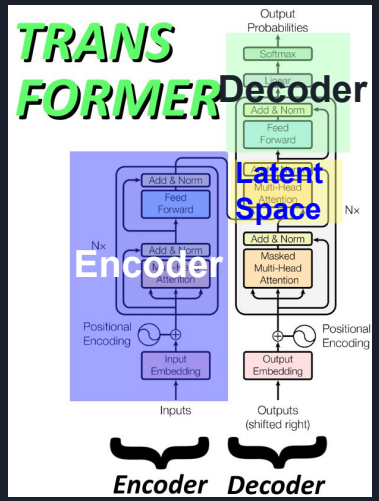


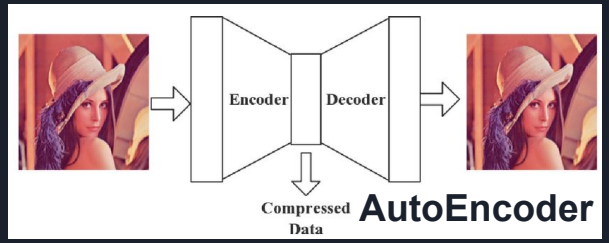
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

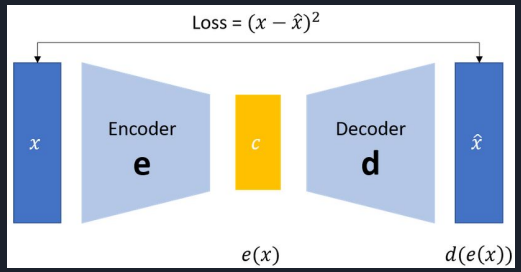


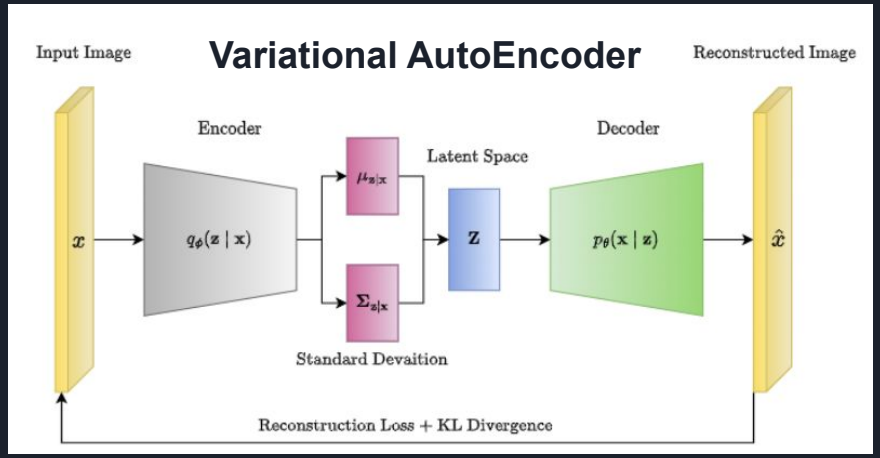
He et al. (2015) https://arxiv.org/pdf/1512.03385

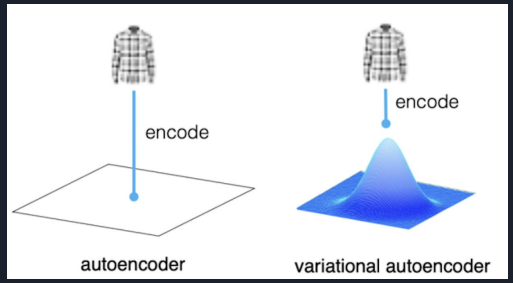
AutoEncoders







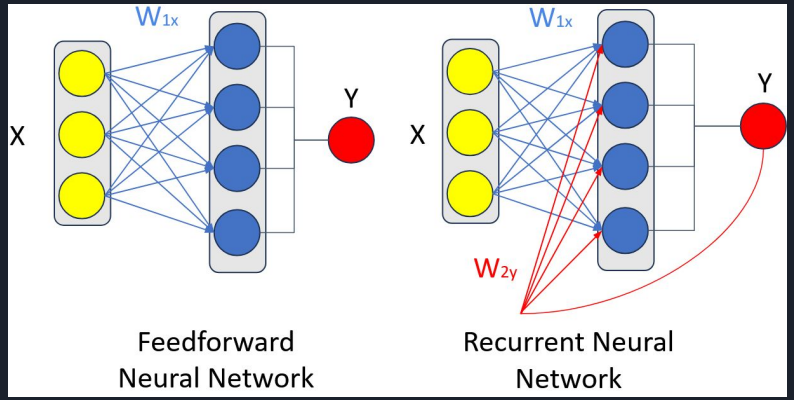


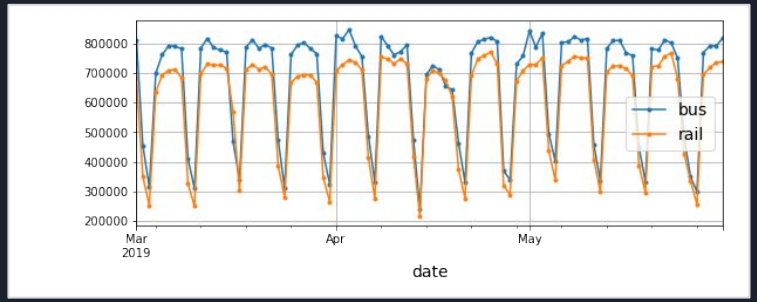


https://sushant-kumar.com/blog/vae-variational-autoencoder

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

Recurrent Neural Networks



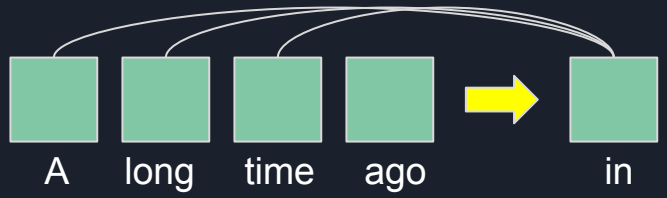


$Y = \text{sigmoid}(Wx*X + b)$

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

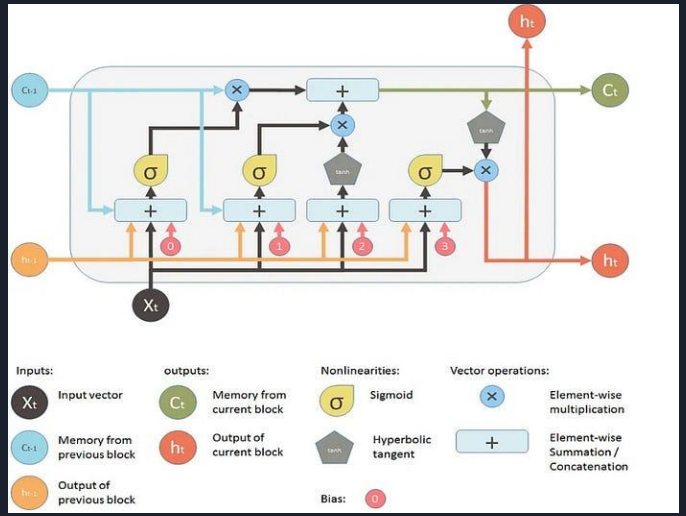
$Y(0) = \text{sigmoid}(Wx*X(0) + b)$

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



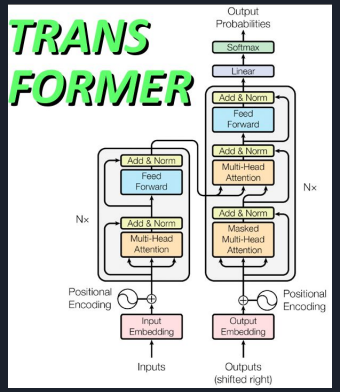
https://github.com/ageron/handson-ml3/blob/main/15_processing_sequences_using_rnns_and_cnns.ipynb

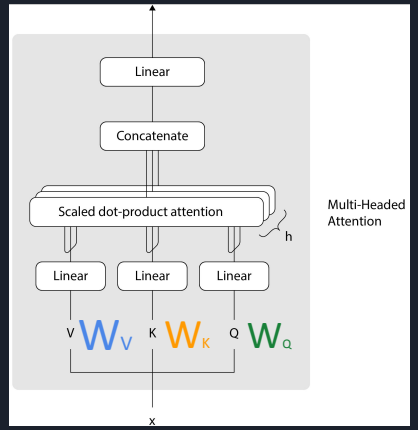
Recurrent Neural Networks: LSTM



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

Attention vs. RNNs

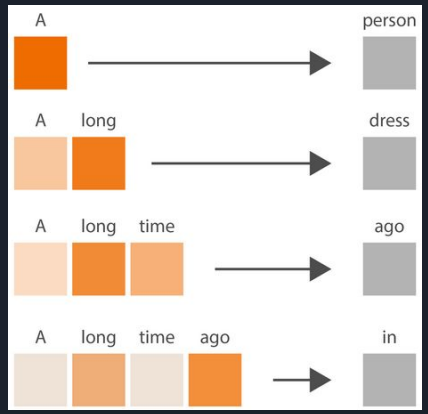




$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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.

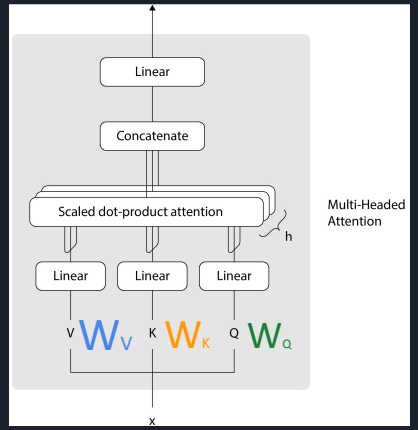
More specifically, attention actually predicts the next token for several portions of our input sequence. It looks at the first token and predicts what a second token might be, then it looks at the first and second tokens and predicts what a third token might be, and so on.



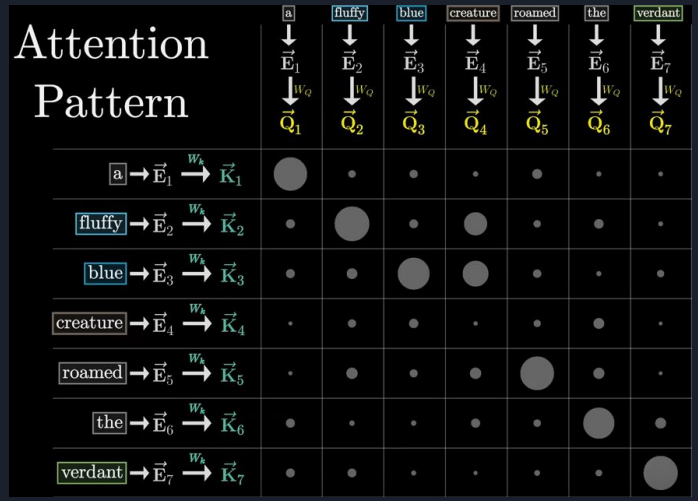
https://bea.stollnitz.com/blog/gpt-transformer/

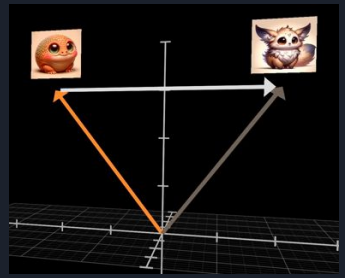
https://www.youtube.com/watch?v=eMlx5fFNoYc

Transformer LLMs: Attention



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





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

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.

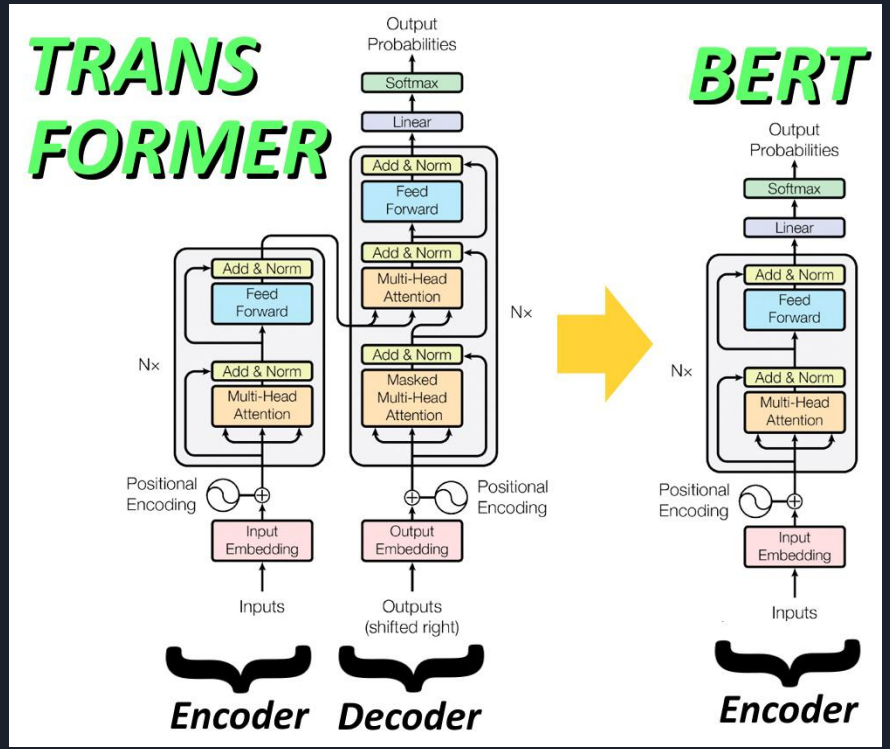
5. Memory Bottlenecks in Long Sequences

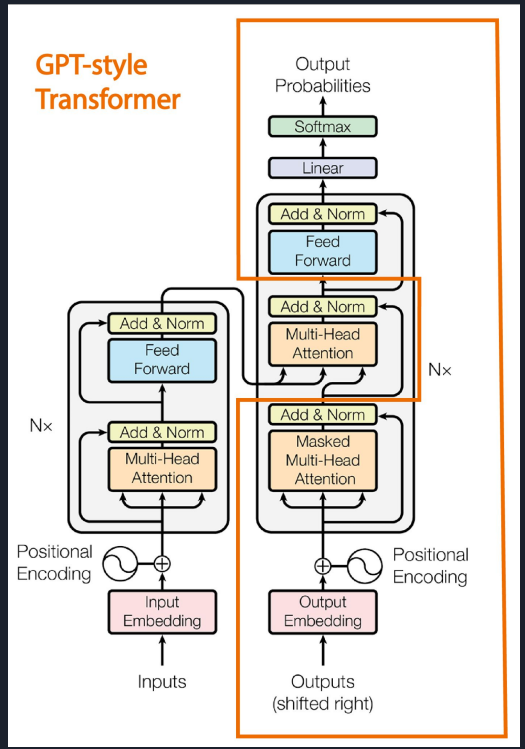
- **RNN Issue:** Storing hidden states for long sequences requires a lot of memory.
- **Transformer Solution:** Transformers replace sequential hidden states with attention matrices, which can be optimized more efficiently using modern hardware.

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



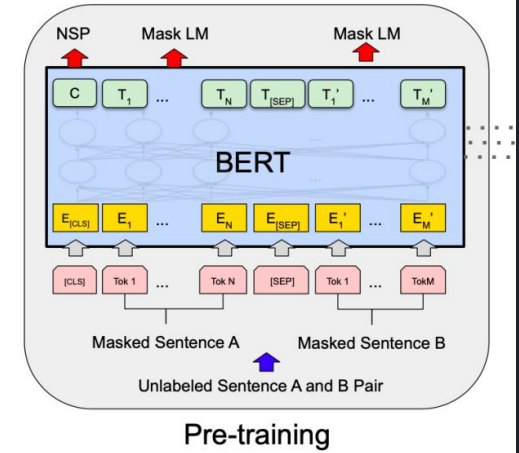


Transformers and LLMs

BERT

GPT-style Transformer

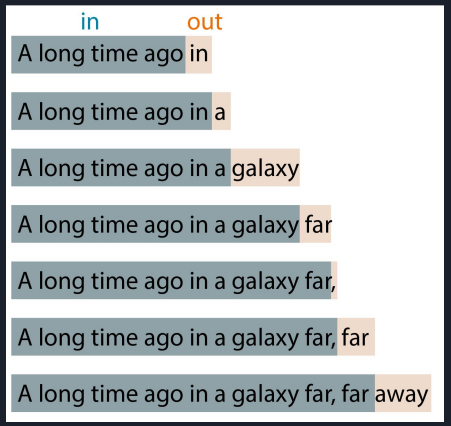
AutoEncoding Generation



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

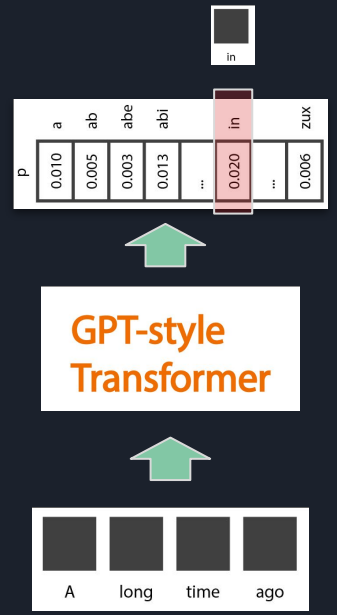
AutoRegressive Generation



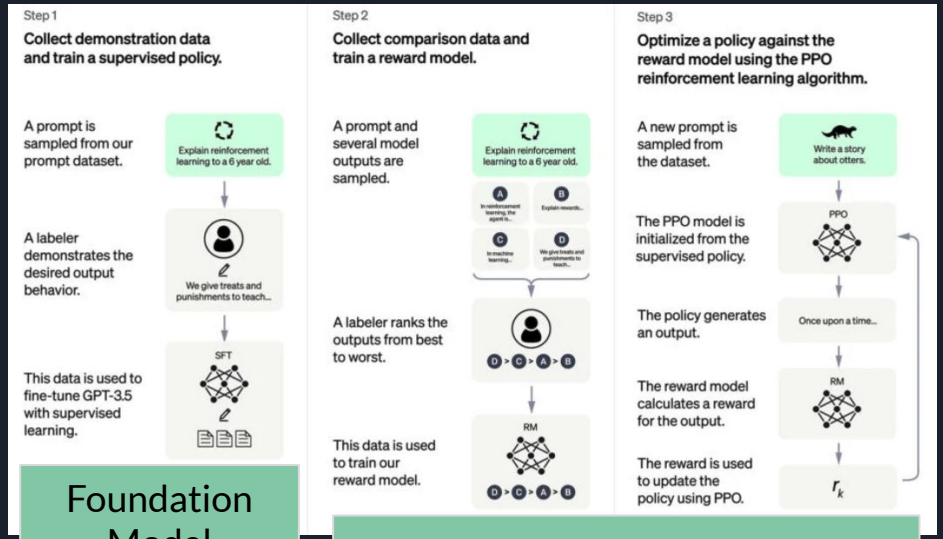
Transformers and LLMs

AutoEncoding Generation

AutoRegressive Generation



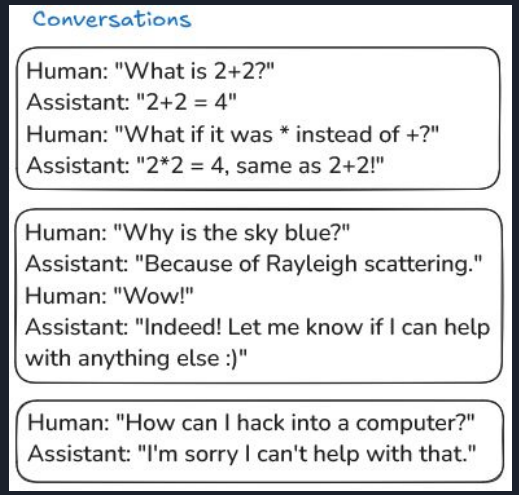
Training LLMs

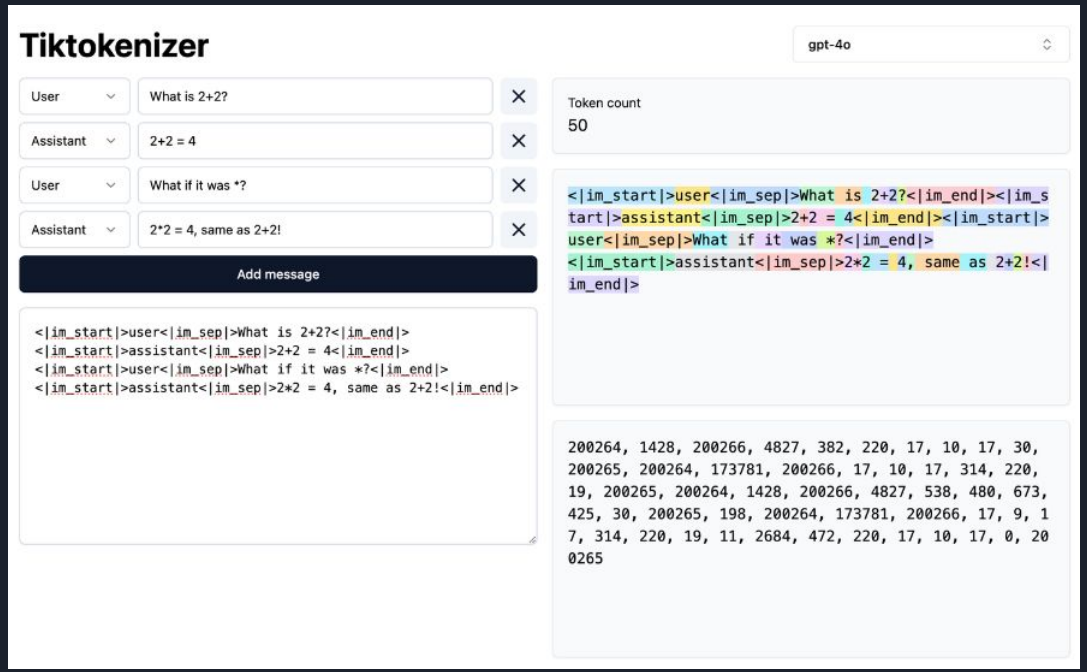


Foundation Model

Instruct Model

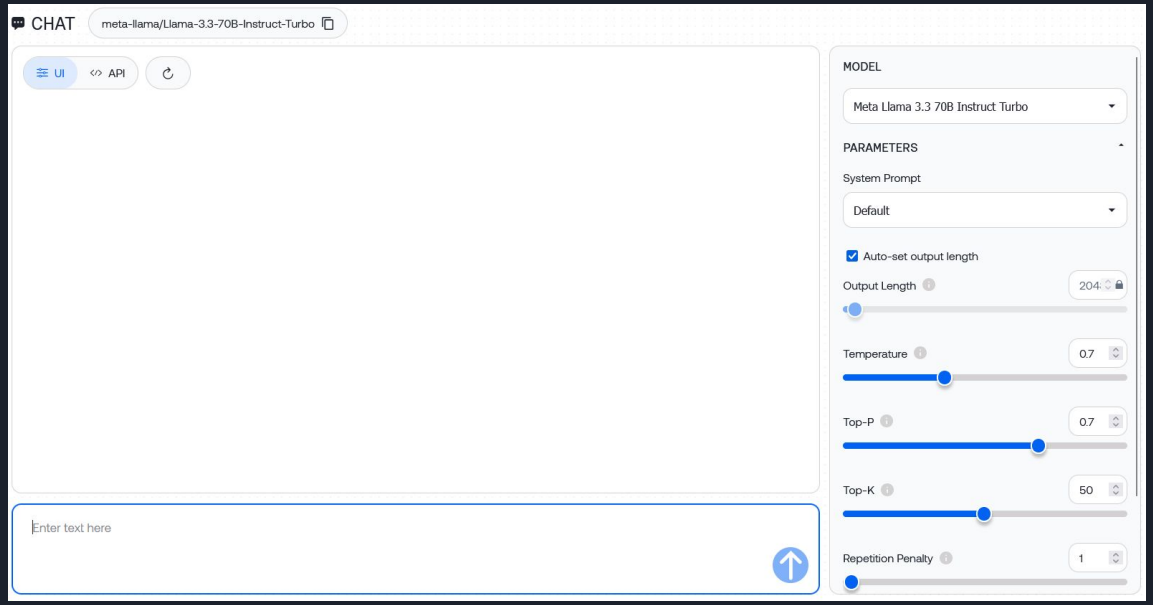
Training LLMs: Instruct Model





Part 2. Basic Applications of LLMs

QA, Summary, Translation



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

Semantic Search

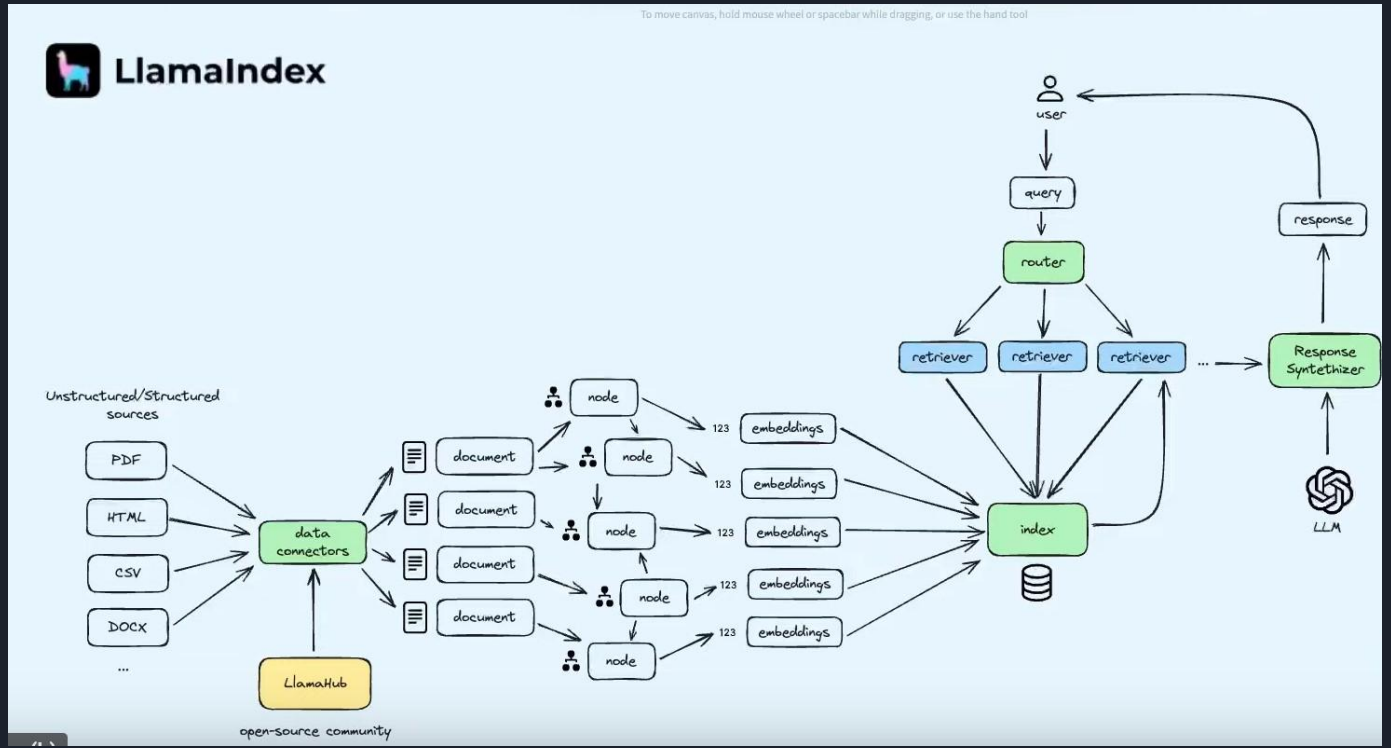
```
answer_list = []
score_list = []
for paper in articles:
    question = 'what is multi-head attention?'
    QA_input = {
        'question': question,
        'context': papers[paper]
    }
    res = nlp(QA_input)
    #print(f'from paper {paper} \n we learn that answer to {question} is \n {res} \n')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

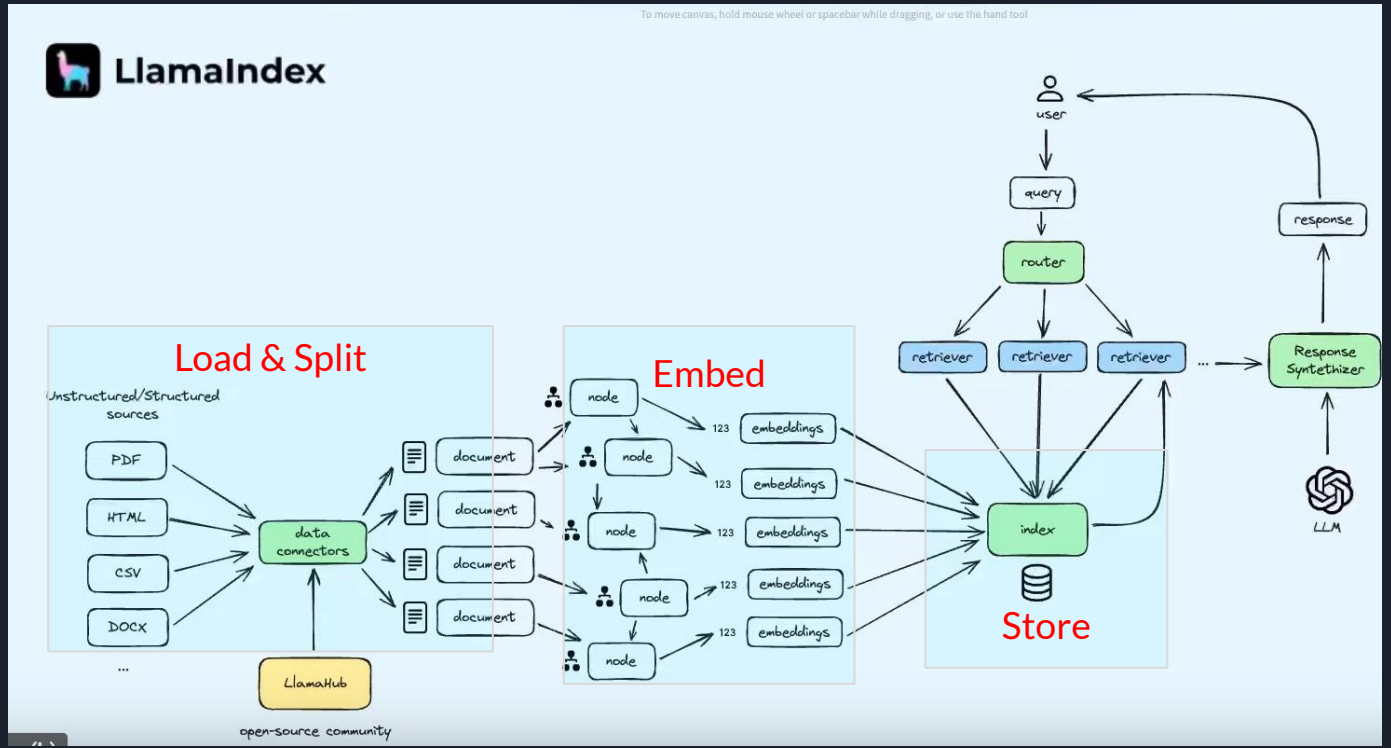
3.2.3 Applications of Attention in our Model The Transformer uses multi-head attention in three different ways: • In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder Score:0.50 With a single attention head, averaging inhibits this Score:0.46 In this work we employ h = 8 parallel attention layers, or heads Score:0.46 Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality Score:0.43 Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions Score:0.43 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 Score:0.37 • The encoder contains self-attention layers Score:0.37 This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9] Score:0.36 • Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position Score:0.33 3.3 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 Score:0.33 We implement this inside of scaled dot-product attention by masking out (setting to −∞) all values in the input of the softmax which correspond to illegal connections Score:0.26 MultiHead(Q, K, V) = Concat(head1, ..., headh)WO where headi = Attention(QWQ i , KWK i , V WV i) Where the projections are parameter matricesWQ i ∈ Rdmodel×dk , WK i ∈ Rdmodel×dk , WV i ∈ Rdmodel×dv and WO ∈ Rhdv×dmodel Score:0.24 This allows every position in the decoder to attend over all positions in the input sequence Score:0.20 Each position in the encoder can attend to all positions in the previous layer of the encoder Score:0.15 This consists of two linear transformations with a ReLU activation in between Score:0.15 3.4 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 dmodel Score:0.14 See Figure 2 Score:0.14 We also use the usual learned linear transfor- mation and softmax function to convert the decoder output to predicted next-token probabilities Score:0.13 Score:0.13 Another way of describing this is as two convolutions with kernel size 1 Score:0.12 The dimensionality of input and output is dmodel = 512, and the inner-layer has dimensionality dff = 2048 Score:0.10 output values Score:0.10 We need to prevent leftward information flow in the decoder to preserve the auto-regressive property Score:0.08 For each of these we use dk = dv = dmodel/h = 64 Score:0.04 These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2 Score:0.03 In the embedding layers, we multiply those weights by √dmodel Score:0.02 In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30] Score:0.02 FFN(x) = max(0, xW1 + b1)W2 + b2 (2) While the linear transformations are the same across different positions, they use different parameters from layer to layer Score:-0.01

	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

Retrieval Augmented Generation

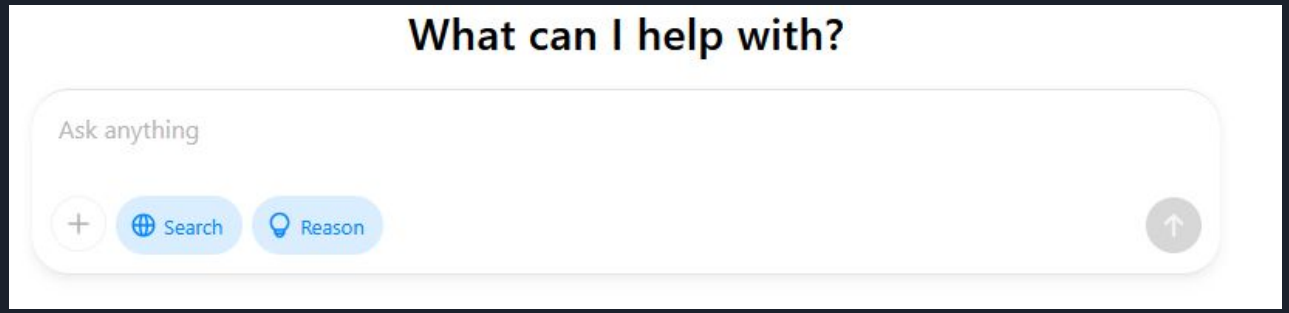


Retrieval Augmented Generation



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 × 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

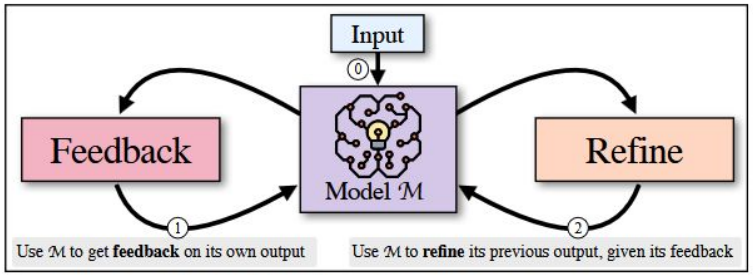


Figure 1: Given an input (⓪), SELF-REFINE starts by generating an output and passing it back to the same model $\mathcal{M}$ to get feedback (①). The feedback is passed back to $\mathcal{M}$, which refines the previously generated output (②). Steps (①) and (②) iterate until a stopping condition is met. SELF-REFINE is instantiated with a language model such as GPT-3.5 and does not involve human assistance.

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

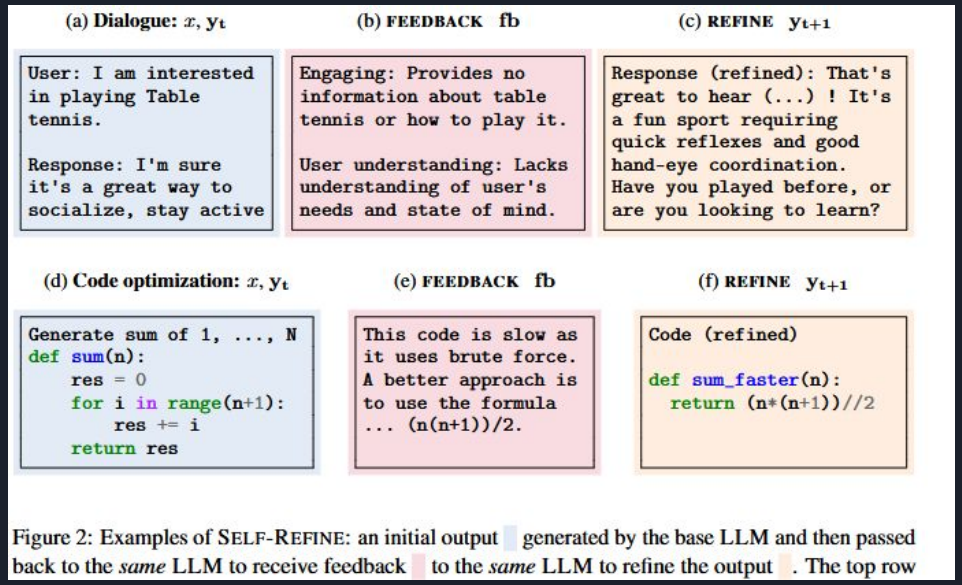


Figure 2: Examples of SELF-REFINE: an initial output generated by the base LLM and then passed back to the *same* LLM to receive feedback to the *same* LLM to refine the output . The top row

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.

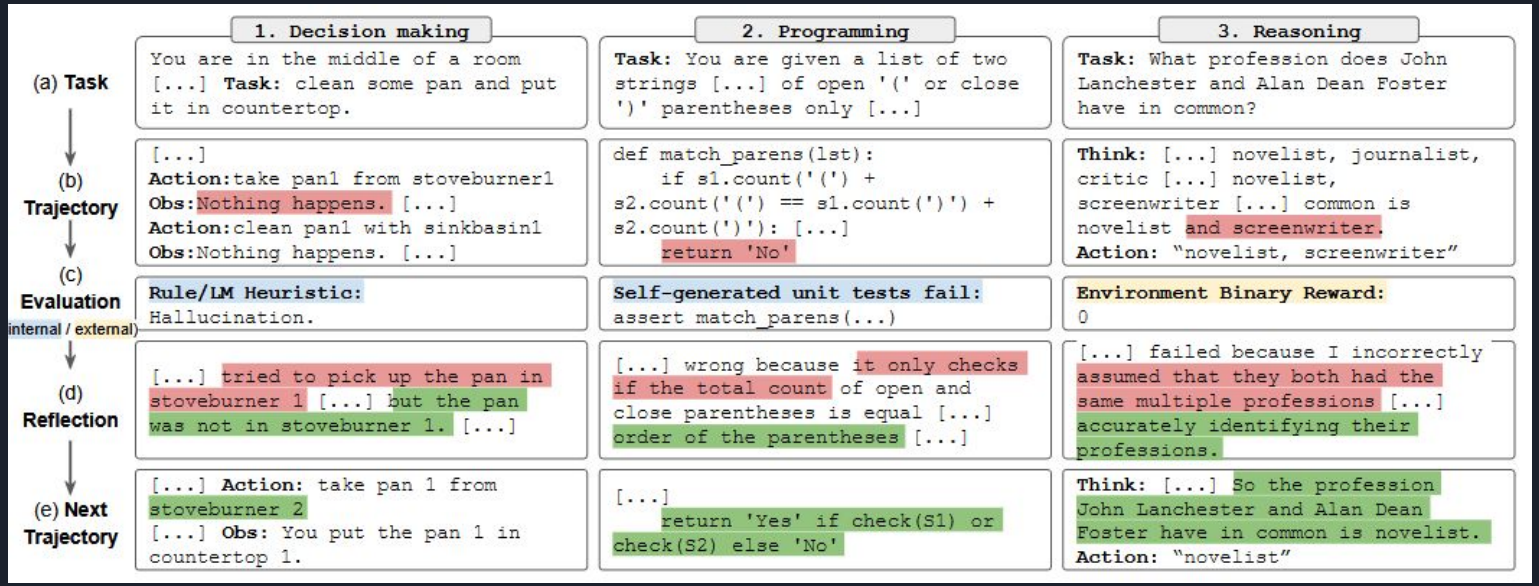
Prompt Example for Reflection:

```
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

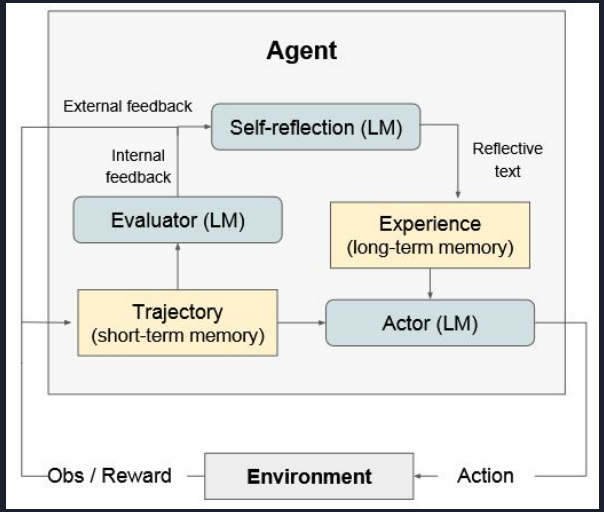
Agents



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=PLIkpOeXz5iWxXFb7Guh-CB2oiIJsd1hma

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

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...
- Tiktokenizer: https://tiktokenizer.vercel.app/
- Transformer Neural Net 3D visualizer: https://bbycroft.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.