Introduction to LLM

周赵嘉程

2025 年 12 月 1 日

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录

第 1 节

Introduction

Introduction

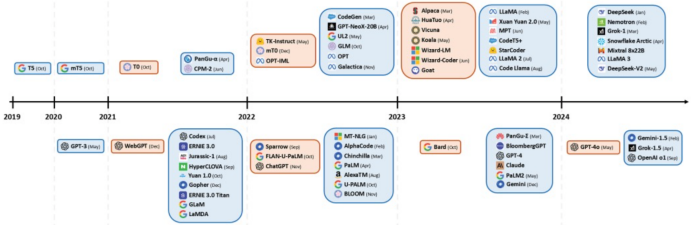
- 自 20 世纪 50 年代图灵测试提出以来，人类一直在探索如何让机器掌握语言智能。
- 语言本质上是一个复杂精妙的人类表达系统，受语法规则的约束。
- 开发能够理解和掌握语言的人工智能（AI）算法是一项巨大的挑战

什么是大语言模型 (What are LLMs?)

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

- 基于深度学习的大规模概率模型，参数量通常在数十亿 (Billions) 至万亿级别。
- 这里的“大”既指**参数量巨大**，也指训练数据的**规模巨大**（万亿级 Tokens）。
- 绝大多数现代 LLM 基于 **Transformer** 架构（特别是 Decoder-only 结构，如 GPT 系列）
- 利用自注意力机制 (Self-Attention) 捕捉长距离的文本依赖关系。
- **自监督学习 (Self-Supervised Learning):** 不需要人工标注，通过“预测下一个词”(Next Token Prediction) 进行训练。
- **知识压缩 (Knowledge Compression):** 模型不仅仅是在记忆文本，而是将世界知识、逻辑规则和语言模式压缩进了神经网络的权重参数中。

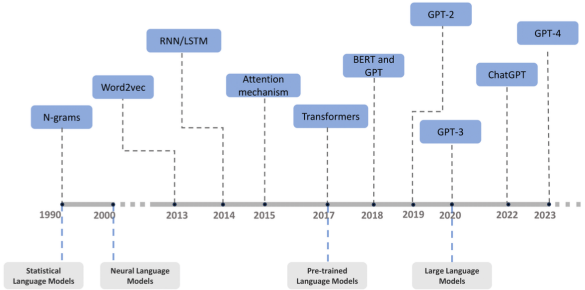
什么是大模型


2019
2020
2021
2022
2023
2024
T5 (Oct)
mT5 (Oct)
T0 (Oct)
PanGu-α (Apr)
CPM-2 (Jun)
TK-Instruct (May)
mT0 (Dec)
GPT-iML
CodeGen (Mar)
GPT-NeoX-20B (Apr)
UL2 (May)
GLM (Oct)
OPT
Galactica (Nov)
Alpaca (Mar)
HuaTuo (Apr)
Vicuna
Koala (May)
Wizard-LM
Wizard-Coder (Jun)
Goat
LLaMA (Feb)
Xuan Yuan 2.0 (May)
MPT (Jun)
CodeT5+
StarCoder
LLaMA 2 (Jul)
Code Llama (Aug)
DeepSeek (Jan)
Nemotron (Feb)
Grok-1 (Mar)
Snowflake Arctic (Apr)
Mixtral 8x22B
LLaMA 3
DeepSeek-V2 (May)
GPT-3 (May)
WebGPT (Dec)
Codex (Jul)
ERNIE 1.0
Jurassic-1 (Aug)
HyperCLOVA (Sep)
Yuan 1.0 (Oct)
Gopher (Dec)
ERNIE 3.0 Titan
GLaM
LaMDA
Sparrow (Sep)
FLAN-U-PaLM (Oct)
ChatGPT (Nov)
MT-NLG (Jan)
AlphaCode (Feb)
Chinchilla (Mar)
PaLM (Apr)
AlexaTM (Aug)
U-PALM (Oct)
BLOOM (Nov)
Bard (Oct)
PanGu-Σ (Mar)
BloombergGPT
GPT-4
Claude
PaLM2 (May)
Gemini (Dec)
GPT-4o (May)
Gemini-1.5 (Feb)
Grok-1.5 (Apr)
OpenAI o1 (Sep)

Historical Timeline of LLMs

- SLM: 统计语言模型
- NLM: 神经语言模型
- PLM: 预训练语言模型
- LLM: 大语言模型

Historical Timeline of LLMs



从 PLM 到 LLM 的范式转变

- PLM (Pre-trained Language Models) 时代 (e.g., BERT, RoBERTa):
  - **范式：** 预训练 + 微调 (Pre-train + Fine-tune)。
  - **局限：** 需要针对特定任务（Task-specific）收集标注数据并更新参数；模型往往是“专才”而非“通才”。
- LLM (Large Language Models) 时代 (e.g., GPT-3, GPT-4, Llama):
  - **范式：** 预训练 + 提示工程/上下文学习 (Pre-train + Prompt/In-Context Learning)。
  - **优势：** 冻结参数即可处理多种任务；通过指令微调 (Instruction Tuning) 实现通用意图理解。

LLM 的核心进步 (Key Advancements)

- **扩展定律 (Scaling Laws):**
  - 随着参数量、数据量和计算量的增加，模型性能呈幂律提升，并未出现瓶颈 [1]。
- **指令微调 (Instruction Tuning):**
  - 通过在多任务指令数据集上微调，模型学会了“遵循指令”而非仅仅“预测下一个词”，显著提高了零样本泛化能力 [2]。
- **人类反馈强化学习 (RLHF):**
  - 将模型输出与人类价值观（有用性、诚实性、安全性）对齐，解决了 PLM 生成内容不可控的问题 [2]。

LLM 的特殊能力 (Emergent Abilities)

当模型规模突破一定临界值（如 10^{22} FLOPs）时，会出现小模型不具备的**涌现能力 (Emergent Abilities)** [3]：

- **上下文学习 (In-Context Learning, ICL):**
  - 模型无需梯度更新，仅通过 Prompt 中的几个示例即可学会新任务 [4]。
- **思维链 (Chain-of-Thought, CoT):**
  - 通过生成中间推理步骤，显著提升了处理复杂逻辑、数学和常识推理任务的能力 [5]。
- **指令遵循与泛化:**
  - 能够处理训练集中从未见过的任务描述，表现出真正的通用智能特征。

大语言模型的当前应用

- **编程智能体:** GitHub Copilot, Cursor, CodeT5, 以及基于 GPT 的编程助手。
- **对话式 AI:** ChatGPT, Claude, Gemini, 用于客户服务及个人助理。
- **内容生成 (Content Generation):** 自动化文章撰写、创意写作、营销内容生成。
- **科研辅助 (Research Assistance):** Alpha evolve, GPT Deepresearch, 文献综述加速、研究假设生成、辅助数据分析。

Attention Mechanism

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

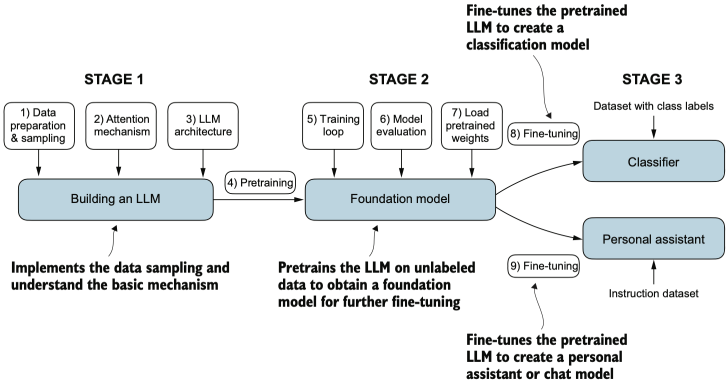
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Building LLM



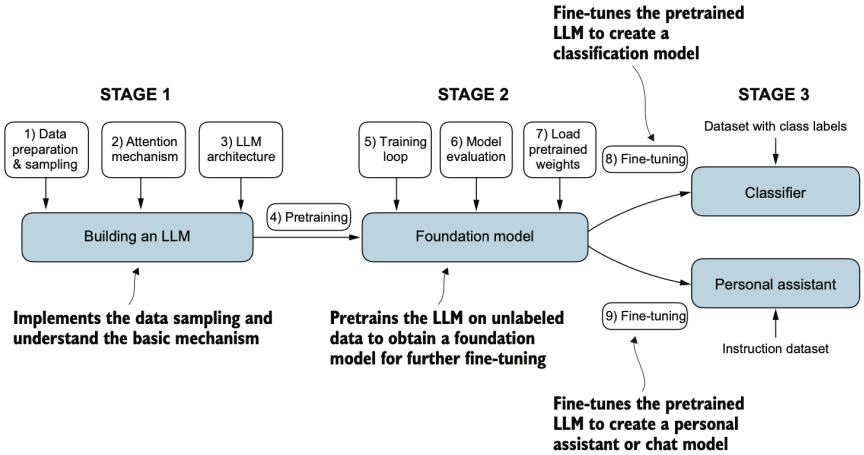
第 2 节

Overview

Credit

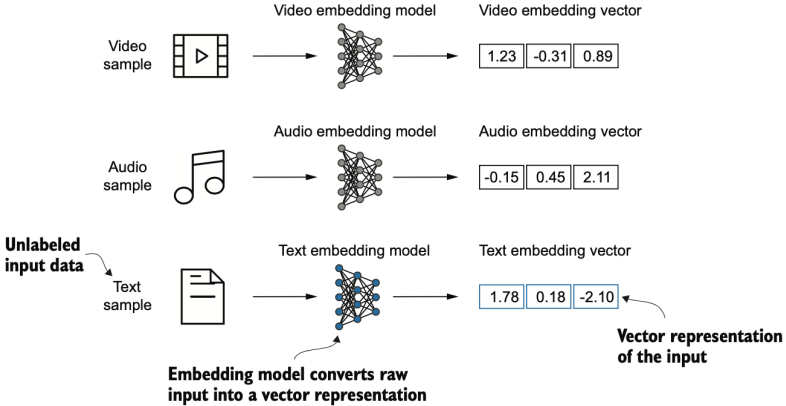
以下部分主要来自 [6]

Building Overview



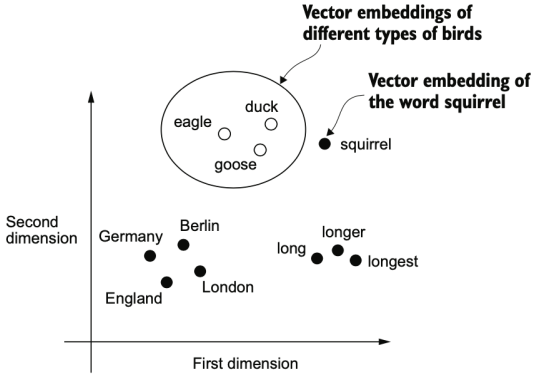
Overview

大模型内部处理的是向量，需要将多模态数据转化为大模型可以处理的向量



Overview

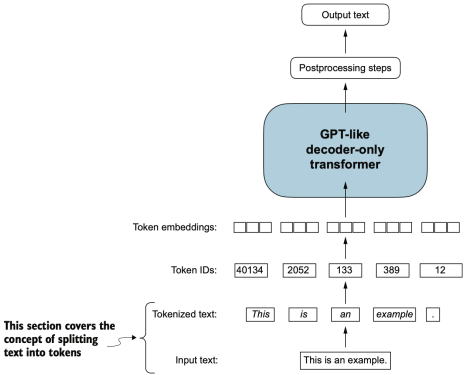
有类似语意的向量在空间中有一定聚集性



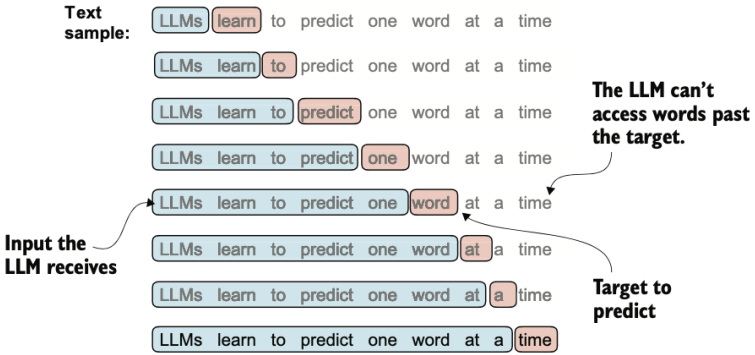
第 3 节

文本处理

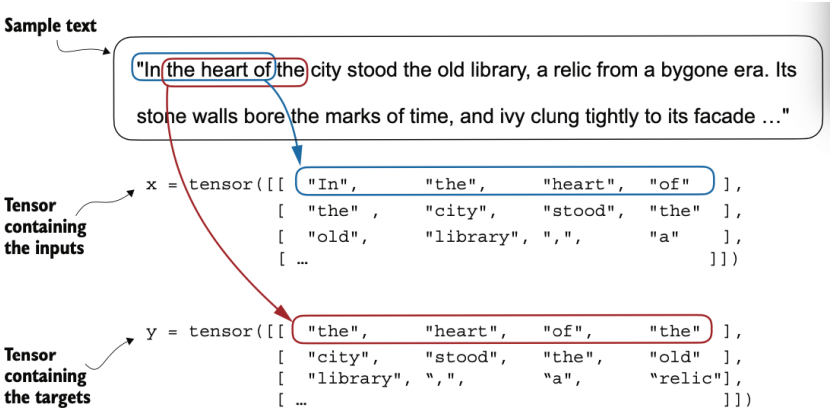
GPT Workflow



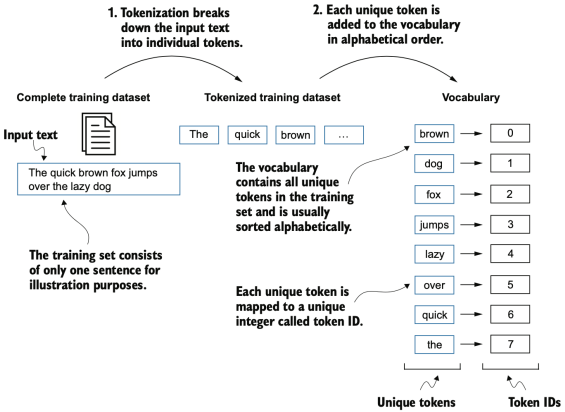
GPT Workflow



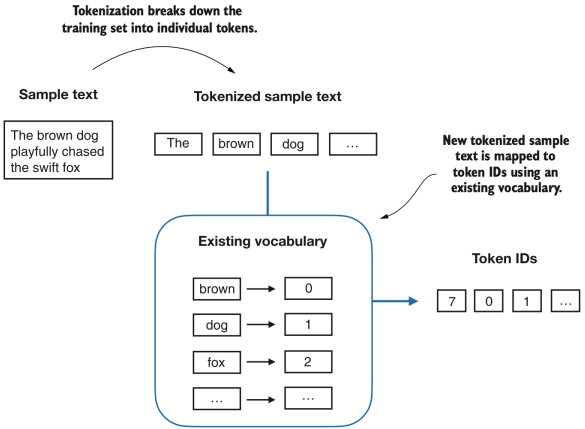
GPT Workflow



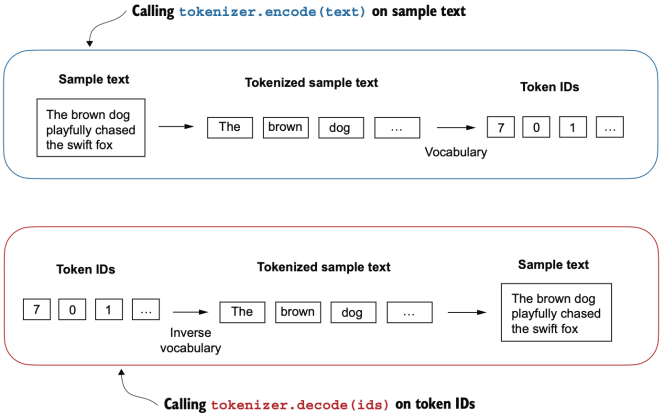
Tokenizer


1. Tokenization breaks down the input text into individual tokens.
2. Each unique token is added to the vocabulary in alphabetical order.
Complete training dataset
Tokenized training dataset
Vocabulary
Input text
The quick brown fox jumps over the lazy dog
The
quick
brown
...
The training set consists of only one sentence for illustration purposes.
The vocabulary contains all unique tokens in the training set and is usually sorted alphabetically.
Each unique token is mapped to a unique integer called token ID.
brown
dog
fox
jumps
lazy
over
quick
the
0
1
2
3
4
5
6
7
Unique tokens
Token IDs

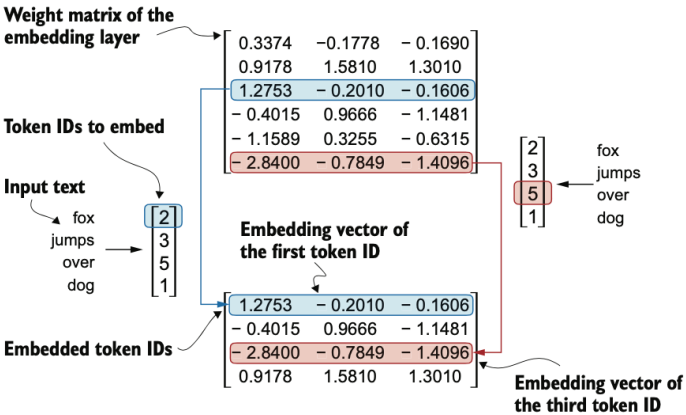
Tokenizer


Tokenization breaks down the training set into individual tokens.
Sample text
The brown dog playfully chased the swift fox
Tokenized sample text
The
brown
dog
…
New tokenized sample text is mapped to token IDs using an existing vocabulary.
Existing vocabulary
brown
0
dog
1
fox
2
…
…
Token IDs
7
0
1
…

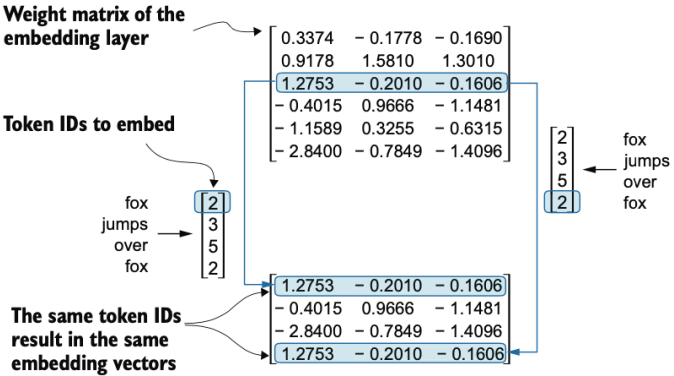
Tokenizer Encoding & Decoding



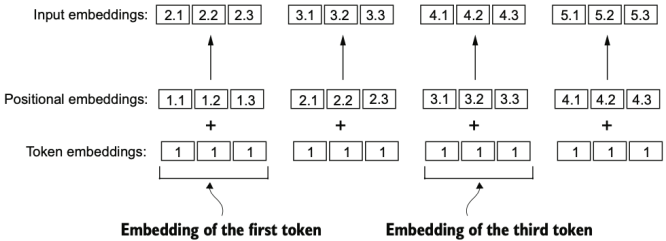
Embedding


Weight matrix of the embedding layer
0.3374 −0.1778 − 0.1690
0.9178 1.5810 1.3010
1.2753 − 0.2010 − 0.1606
− 0.4015 0.9666 − 1.1481
− 1.1589 0.3255 − 0.6315
− 2.8400 − 0.7849 − 1.4096
Token IDs to embed
Input text
fox
jumps
over
dog
2
3
5
1
Embedding vector of the first token ID
1.2753 − 0.2010 − 0.1606
− 0.4015 0.9666 − 1.1481
− 2.8400 − 0.7849 − 1.4096
0.9178 1.5810 1.3010
Embedded token IDs
2
3
5
1
fox
jumps
over
dog
Embedding vector of the third token ID

Embedding


Weight matrix of the embedding layer
0.3374 − 0.1778 − 0.1690
0.9178 1.5810 1.3010
1.2753 − 0.2010 − 0.1606
− 0.4015 0.9666 − 1.1481
− 1.1589 0.3255 − 0.6315
− 2.8400 − 0.7849 − 1.4096
Token IDs to embed
fox
jumps
over
fox
2
3
5
2
1.2753 − 0.2010 − 0.1606
− 0.4015 0.9666 − 1.1481
− 2.8400 − 0.7849 − 1.4096
1.2753 − 0.2010 − 0.1606
The same token IDs result in the same embedding vectors
2
3
5
2
fox
jumps
over
fox

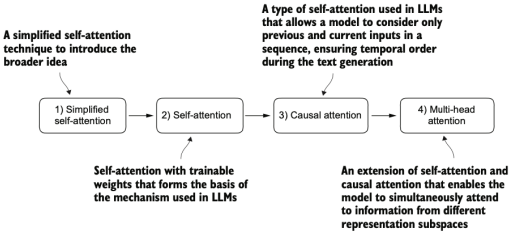
Embedding


Input embeddings:
2.1 2.2 2.3
3.1 3.2 3.3
4.1 4.2 4.3
5.1 5.2 5.3
Positional embeddings:
1.1 1.2 1.3
2.1 2.2 2.3
3.1 3.2 3.3
4.1 4.2 4.3
Token embeddings:
1 1 1
1 1 1
1 1 1
1 1 1
Embedding of the first token
Embedding of the third token

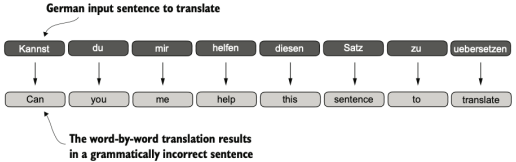
第 4 节

注意力机制

Attention Evolution



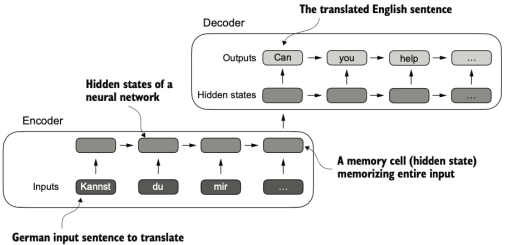
Problem


German input sentence to translate
Kannst du mir helfen diesen Satz zu uebersetzen
Can you me help this sentence to translate
The word-by-word translation results in a grammatically incorrect sentence
Kannst du mir helfen diesen Satz zu uebersetzen
Can you help me to translate this sentence
The correct translation
Certain words in the generated translation require access to words that appear earlier or later in the original sentence.

Previous Solution

上海交通大學 Shanghai Jiao Tong University

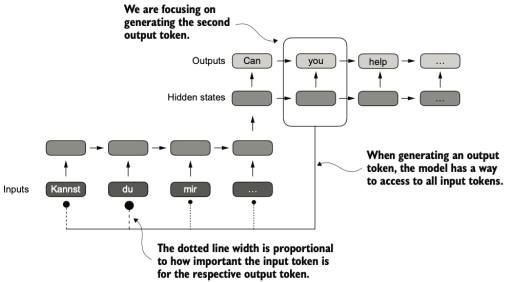
This figure is an encoder–decoder RNN.



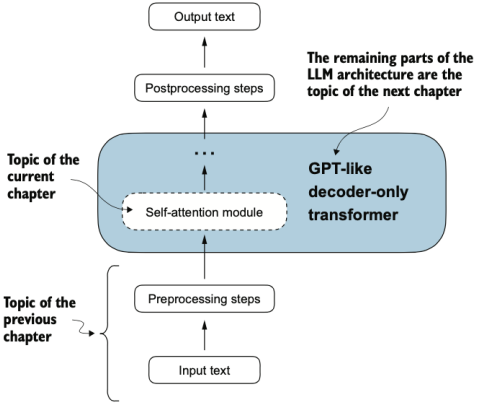
Previous Solution

- Encoder-Decoder RNN decoder 不能获取 encoder 中间状态的 hidden state
- 这代表所有的信息需要压缩到一个 hidden state 中
- 后果是可能带来上下文损失
- 这是设计 self attention 机制的 motivation

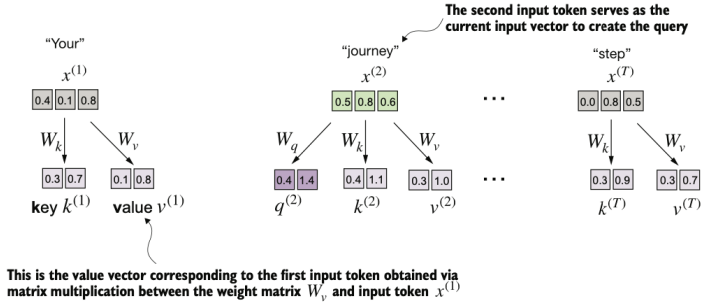
Self Attention



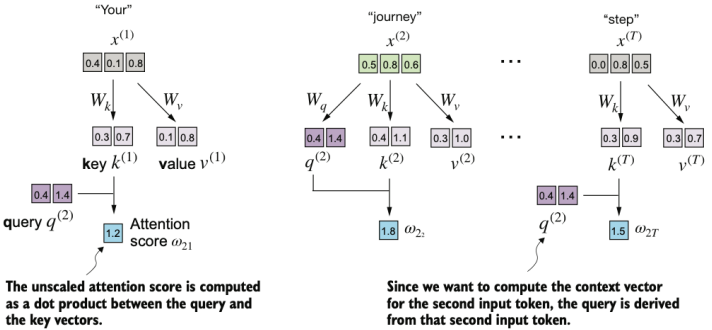
Self Attention Module



Compute Self Attention



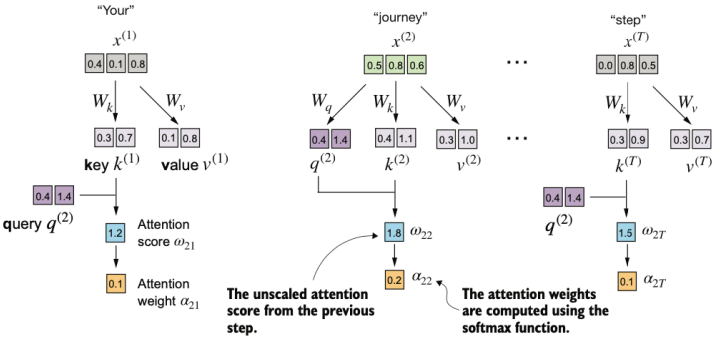
Compute Self Attention



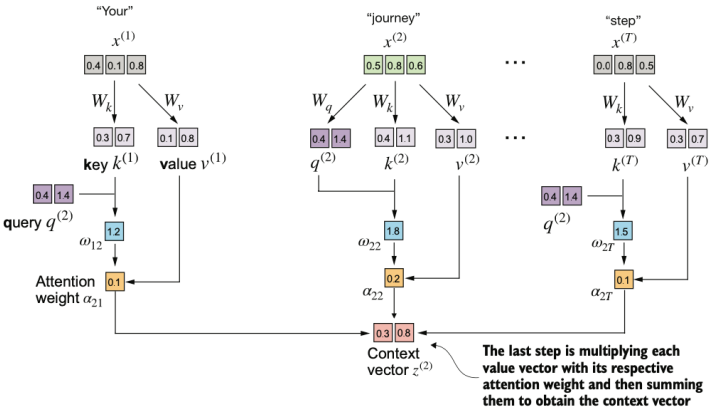
The unscaled attention score is computed as a dot product between the query and the key vectors.

Since we want to compute the context vector for the second input token, the query is derived from that second input token.

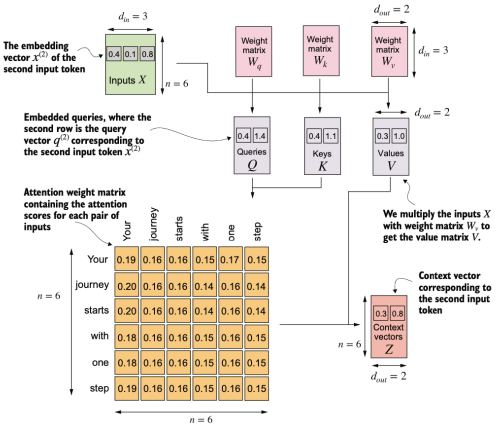
Compute Self Attention



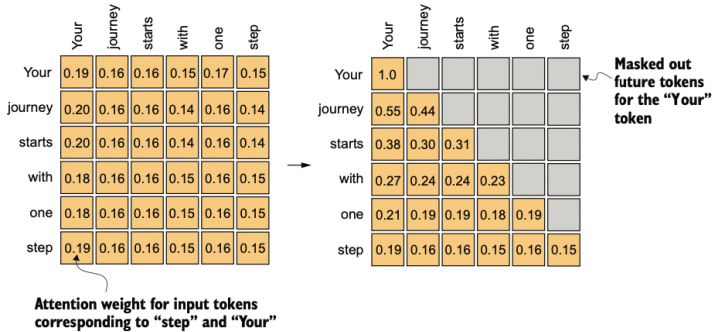
Compute Self Attention



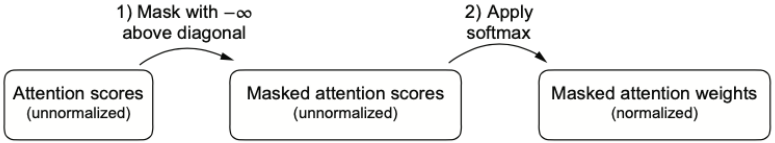
Compute Self Attention



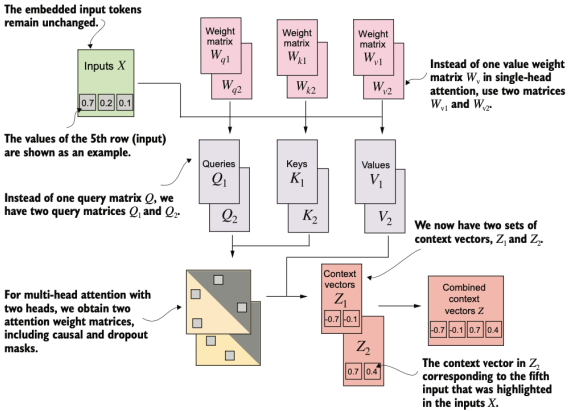
Causal Attention



Masked Attention



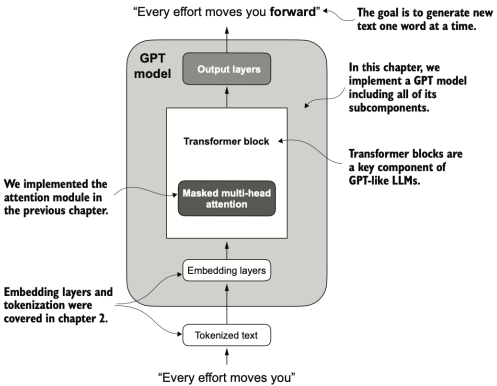
Multihead Attention



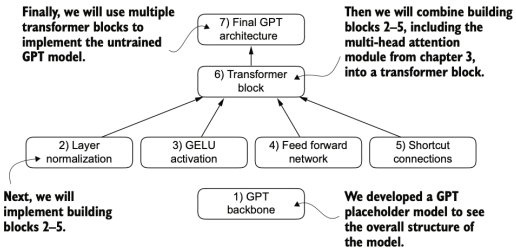
第 5 节

GPT 实现

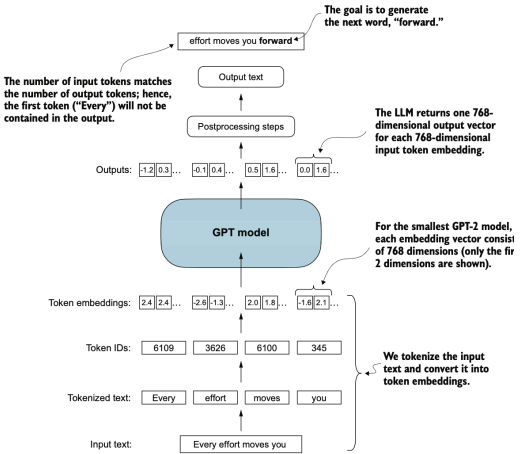
Quick Review



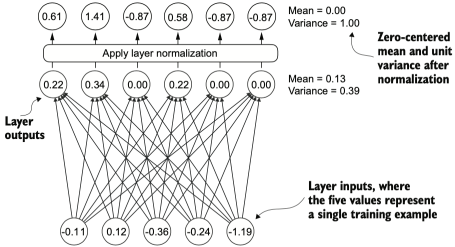
核心组件


Finally, we will use multiple transformer blocks to implement the untrained GPT model.
7) Final GPT architecture
Then we will combine building blocks 2–5, including the multi-head attention module from chapter 3, into a transformer block.
6) Transformer block
2) Layer normalization
3) GELU activation
4) Feed forward network
5) Shortcut connections
Next, we will implement building blocks 2–5.
1) GPT backbone
We developed a GPT placeholder model to see the overall structure of the model.

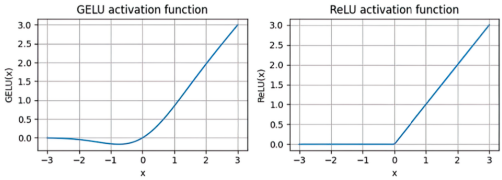
Workflow Review



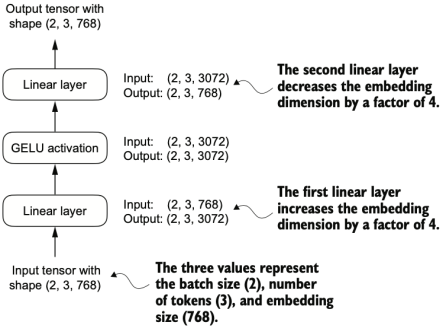
Normalization



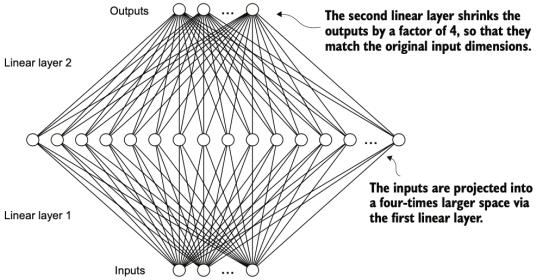
GELU


GELU activation function
GELU(x)
x
ReLU activation function
ReLU(x)
x

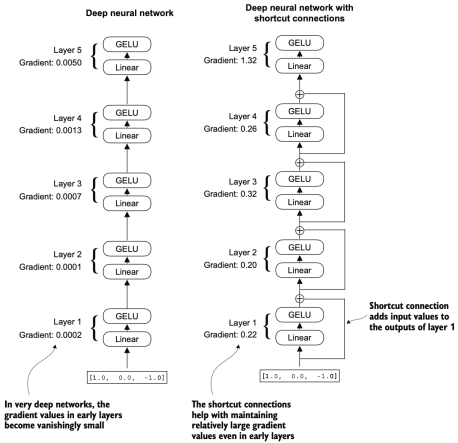
FNN Layer Connection



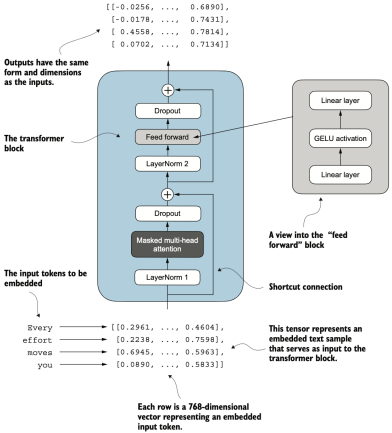
FNN


Outputs
Linear layer 2
Linear layer 1
Inputs
The second linear layer shrinks the outputs by a factor of 4, so that they match the original input dimensions.
The inputs are projected into a four-times larger space via the first linear layer.

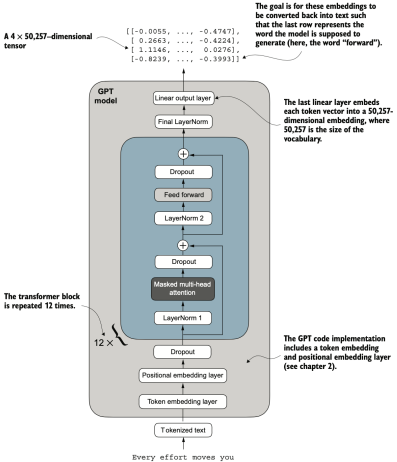
Short Cut


Deep neural network
Deep neural network with shortcut connections
Layer 5
Gradient: 0.0050
Layer 4
Gradient: 0.0013
Layer 3
Gradient: 0.0007
Layer 2
Gradient: 0.0001
Layer 1
Gradient: 0.0002
GELU
Linear
[1.0, 0.0, -1.0]
Layer 5
Gradient: 1.32
Layer 4
Gradient: 0.26
Layer 3
Gradient: 0.32
Layer 2
Gradient: 0.20
Layer 1
Gradient: 0.22
Shortcut connection adds input values to the outputs of layer 1
In very deep networks, the gradient values in early layers become vanishingly small
The shortcut connections help with maintaining relatively large gradient values even in early layers

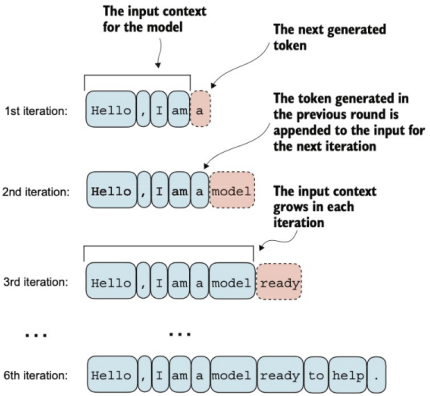
Transformer Block



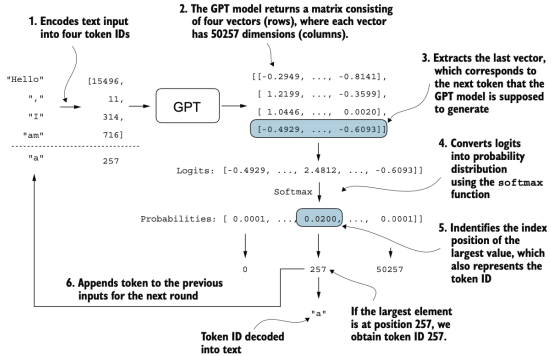
GPT 架构


A 4 × 50,257–dimensional tensor
[[-0.0055, ..., -0.4747],
[0.2663, ..., -0.4224],
[1.1146, ..., 0.0276],
[-0.8239, ..., -0.3993]]
The goal is for these embeddings to be converted back into text such that the last row represents the word the model is supposed to generate (here, the word "forward").
GPT model
Linear output layer
Final LayerNorm
Dropout
Feed forward
LayerNorm 2
Dropout
Masked multi-head attention
LayerNorm 1
12 ×
Dropout
Positional embedding layer
Token embedding layer
Tokenized text
The last linear layer embeds each token vector into a 50,257-dimensional embedding, where 50,257 is the size of the vocabulary.
The transformer block is repeated 12 times.
The GPT code implementation includes a token embedding and positional embedding layer (see chapter 2).
Every effort moves you

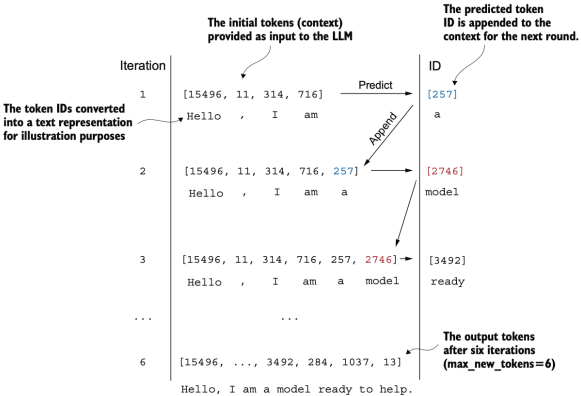
生成模式


The input context for the model
The next generated token
1st iteration:
Hello , I am a
The token generated in the previous round is appended to the input for the next iteration
2nd iteration:
Hello , I am a model
The input context grows in each iteration
3rd iteration:
Hello , I am a model ready
...
...
6th iteration:
Hello , I am a model ready to help .

生成模式


1. Encodes text input into four token IDs
2. The GPT model returns a matrix consisting of four vectors (rows), where each vector has 50257 dimensions (columns).
3. Extracts the last vector, which corresponds to the next token that the GPT model is supposed to generate
4. Converts logits into probability distribution using the softmax function
5. Indentifies the index position of the largest value, which also represents the token ID
6. Appends token to the previous inputs for the next round
"Hello"
","
"I"
"am"
"a"
[15496,
11,
314,
716]
257
GPT
[[-0.2949, ..., -0.8141],
[1.2199, ..., -0.3599],
[1.0446, ..., 0.0020],
[-0.4929, ..., -0.6093]]
Logits: [-0.4929, ..., 2.4812, ..., -0.6093]]
Softmax
Probabilities: [0.0001, ..., 0.0200, ..., 0.0001]]
0
257
50257
"a"
Token ID decoded into text
If the largest element is at position 257, we obtain token ID 257.

Prediction Cycle



第 6 节

Conclusion

检索增强生成 (Retrieval-Augmented Generation)

- **核心痛点 (Pain Points):**
  - **幻觉 (Hallucination):** 模型可能会一本正经地胡说八道。
  - **知识时效性:** 模型的知识截止于训练结束那一刻，无法回答实时问题。
- **工作流程 (Workflow):**
  1. **Retrieve (检索):** 将用户的问题向量化，在外部向量数据库中匹配最相关的文档片段。
  2. **Augment (增强):** 将检索到的事实作为上下文 (Context) 拼接到 Prompt 中。
  3. **Generate (生成):** LLM 基于增强后的 Prompt 生成准确且可溯源的答案。

多智能体系统 (Multi-Agent Systems)

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

- **从单体到群体 (From Solo to Swarm):** 面对复杂的长流程任务（如开发一个游戏），单个 LLM 容易遗忘上下文或逻辑混乱。
- **协作机制 (Collaboration Mechanism):**
  - **角色扮演 (Role Playing):** 不同的 Agent 扮演不同角色（如产品经理、架构师、工程师、测试员）。
  - **流程标准化 (SOP):** 将复杂任务拆解为流水线，Agent 之间通过对话进行规划、执行和互相审查。
- **典型框架:**
  - **AutoGen (Microsoft):** 允许创建可定制、可对话的 Agent 网络。
  - **MetaGPT:** 只要给一句需求，就能输出整个软件工程项目

推理加速关键技术：KV Cache

- **背景 (Background):** LLM 推理是自回归 (Auto-regressive) 的，每生成一个 Token，都要重新计算一次 Attention。
- **空间换时间 (Space-Time Trade-off):**
  - Attention 公式: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$
  - 生成第 t 个 token 时，前 $t-1$ 个 token 的 K (Key) 和 V (Value) 矩阵计算结果是 ** 不变的 **。
  - **KV Cache:** 将历史的 K, V 向量缓存在显存中，每次只计算当前新 token 的 Q, K, V，避免重复计算。
- **挑战 (Challenge):**
  - **显存杀手:** KV Cache 随序列长度线性增长，是长文本推理显存占用过高 (Memory Bound) 的主要原因。
  - 解决方案示例：*PagedAttention (vLLM).*

第 7 节

References

References I

[1] J. Kaplan et al., *Scaling laws for neural language models*, 2020. arXiv: 2001.08361 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2001.08361.

[2] L. Ouyang et al., *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2203.02155.

[3] J. Wei et al., *Emergent abilities of large language models*, 2022. arXiv: 2206.07682 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2206.07682.

References II

[4] T. B. Brown et al., *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.14165.

[5] J. Wei et al., *Chain-of-thought prompting elicits reasoning in large language models*, 2023. arXiv: 2201.11903 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2201.11903.

[6] S. Raschka, *Build A Large Language Model (From Scratch)*. Manning, 2024, ISBN: 978-1633437166. [Online]. Available: https://www.manning.com/books/build-a-large-language-model-from-scratch.

谢 谢