大型语言模型（LLM）理论简介

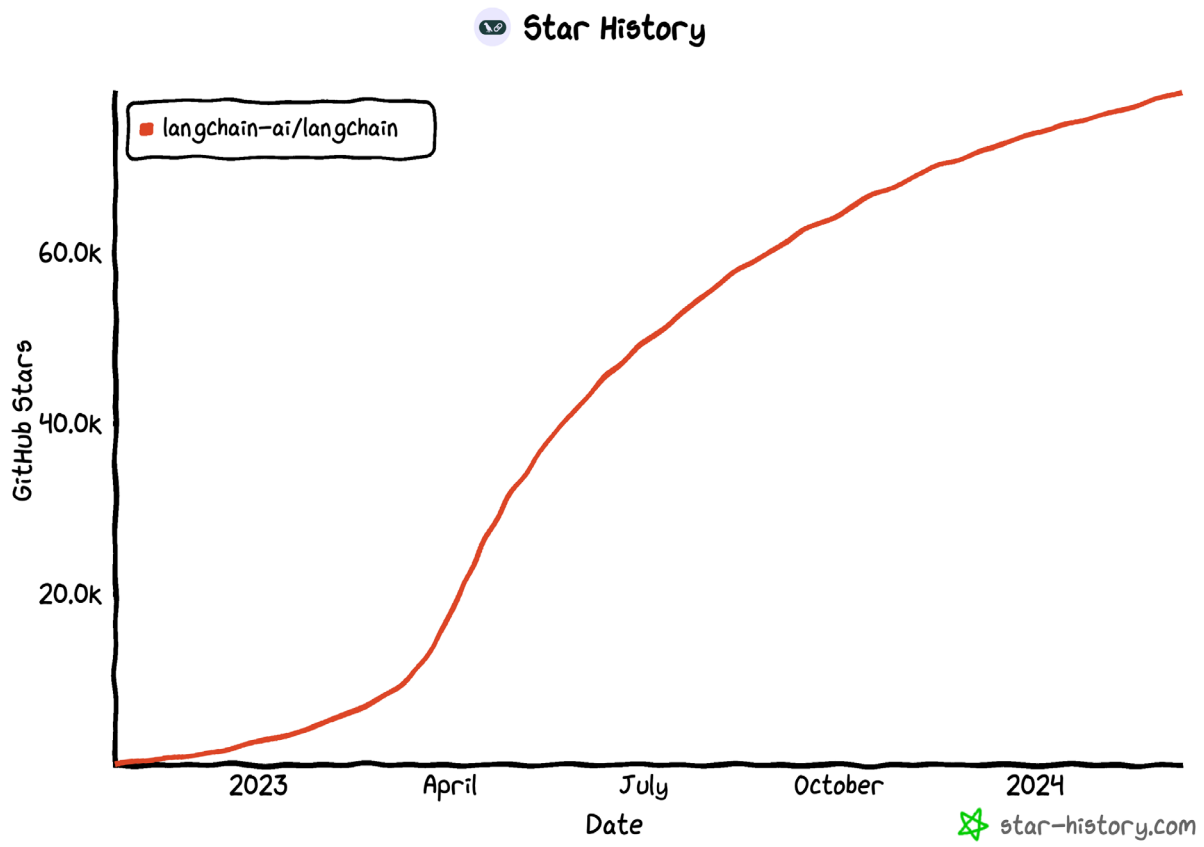
回顾

- LangChain
- 开发LLM应用的流程

什么是 LangChain

- ChatGPT 的巨大成功激发了越来越多的开发者兴趣，他们希望利用 OpenAI 提供的 API 或者私有化模型，来开发基于大型语言模型的应用程序。
- 尽管大型语言模型的调用相对简单，但要创建完整的应用程序，仍然需要大量的定制开发工作，包括 API 集成、互动逻辑、数据存储等等。
- 为了解决这个问题，从 2022 年开始，许多机构和个人相继推出了多个开源项目，旨在**帮助开发者们快速构建基于大型语言模型的端到端应用程序或工作流程**。其中一个备受关注的项目就是 LangChain 框架。
  - Introduction | 🦜🔗 LangChain
  - langchain-ai/langchain: 🦜🔗 Build context-aware reasoning applications (github.com)
- **LangChain 框架是一个开源工具，充分利用了大型语言模型的强大能力，以便开发各种下游应用。**
- **目标是为各种大型语言模型应用提供通用接口，从而简化应用程序的开发流程。**
  - LangChain 框架可以实现数据感知和环境互动，能够让语言模型与其他数据来源连接，并且允许语言模型与其所处的环境进行互动。

langchain-ai/langchain: 🦜🔗 Build context-aware reasoning applications (github.com)



LangChain 的核心组件

- LangChian 作为LLM开发框架，整合 LLM 模型（问答模型、对话模型、embedding 模型等）、向量数据库、交互层 Prompt、外部知识、外部代理工具（AgentTools），构建 LLM 应用。
- LangChain 主要由以下 6 个核心组件组成:
  - **模型输入/输出（Model I/O）**：与语言模型交互的接口
  - **数据连接（Data connection）**：与特定应用程序的数据进行交互的接口
  - **链（Chains）**：将组件组合实现端到端应用，如：搭建检索问答链来完成检索问答
  - **记忆（Memory）**：用于链的多次运行之间持久化应用程序状态
  - **代理（Agents）**：扩展模型的推理能力。用于复杂的应用的调用序列
  - **回调（Callbacks）**：扩展模型的推理能力。用于复杂的应用的调用序列

LangChain 的生态

- **LangChain Community**: 专注于第三方集成，丰富 LangChain 的生态系统，使得开发者可以更容易地构建复杂和强大的应用程序，同时也促进了社区的合作和共享。
- **LangChain Core**: LangChain 框架的核心库、核心组件，提供基础抽象和 LangChain 表达式语言（LCEL），基础架构和工具，用于构建、运行和与 LLM 交互的应用程序，为 LangChain 应用程序的开发提供了坚实的基础。处理文档、格式化 prompt、输出解析等都来自这个库。
- **LangChain CLI**: 命令行工具，使开发者能够通过终端与 LangChain 框架交互，执行项目初始化、测试、部署等任务。提高开发效率，让开发者能够通过简单的命令来管理整个应用程序的生命周期。
- **LangServe**: 部署服务，用于将 LangChain 应用程序部署到云端，提供可扩展、高可用的托管解决方案，并带有监控和日志功能。简化部署流程，让开发者可以专注于应用程序的开发，而不必担心底层的基础设施和运维工作。
- **LangSmith**: 开发者平台，专注于 LangChain 应用程序的开发、调试和测试，提供可视化界面和性能分析工具，旨在帮助开发者提高应用程序的质量，确保它们在部署前达到预期的性能和稳定性标准。

几个基本概念

- **Prompt**
- **Temperature**
- **System Prompt**

Prompt和Completion

- Prompt 最初是 NLP（自然语言处理）为下游任务设计出来的一种任务专属的输入模板
  - 类似于一种任务（例如：分类，聚类等）会对应一种 Prompt
- 在 ChatGPT 推出并应用之后，Prompt 开始被推广为给大模型的所有输入
  - 即，每一次访问大模型的输入为一个 Prompt
  - 大模型返回结果则被称为 Completion

Temperature

- LLM 生成是具有随机性的，在模型的顶层通过选取不同预测概率的预测结果来生成最后的结果
  - 一般可以通过控制 temperature 参数来控制 LLM 生成结果的随机性与创造性
- Temperature 一般取值在 0~1 之间
  - 当取值较低接近 0 时，预测的随机性会较低，产生更保守、可预测的文本，不太可能生成意想不到或不寻常的词。
  - 当取值较高接近 1 时，预测的随机性会较高，所有词被选择的可能性更大，会产生更有创意、多样化的文本，更有可能生成不寻常或意想不到的词。
- 对于不同的问题与应用场景，可能需要设置不同的 temperature，如：
  - 个人助理项目，一般将 temperature 设置为 0，从而保证助手对知识库内容的稳定使用，规避错误内容、模型幻觉；
  - 产品智能客服、科研论文写作等场景中，同样更需要稳定性而不是创造性；
  - 在个性化 AI、创意营销文案生成等场景中，需要创意性，倾向于将 temperature 设置为较高的值。

System Prompt

- System Prompt 是 ChatGPT API 使用的一个新兴概念
  - **并不在大模型本身训练中得到体现，而是大模型服务方为提升用户体验所设置的一种策略**。
- 在使用 ChatGPT API 时，可以设置两种 Prompt：
  - System Prompt，在整个会话过程中持久地影响模型的回复，且相比于普通 Prompt 具有更高的重要性；
  - User Prompt，平时提到的 Prompt，即需要模型做出回复的输入。
- 设置 System Prompt 来对模型进行一些初始化设定，例如：
  - 在 System Prompt 中给模型设定希望它具备的人设（个人知识库助手等）
- System Prompt 一般在一个会话中仅有一个。
  - 通过 System Prompt 设定好模型的人设或是初始设置后，可以通过 User Prompt 给出模型需要遵循的指令。

大模型开发

- **以大语言模型为功能核心、通过大语言模型的强大理解能力和生成能力、结合特殊的数据或业务逻辑来提供独特功能的应用**称为**大模型开发**。
- **将大模型作为一个调用工具**
  - 开发中，不会去大幅度改动模型
  - **通过 Prompt Engineering、数据工程、业务逻辑分解等方法和手段适配应用任务**
  - 不聚焦在优化模型本身上
- **工程问题**
  - 开发大模型相关应用，技术核心点在大语言模型上，一般：
    - 通过调用 API 或开源模型来实现大语言模型的理解与生成
    - 通过 Prompt Enginnering 来实现大语言模型的控制
  - 需要掌握使用大模型的实践技巧

传统的 AI 应用的开发整体思路

- 1. 依次拆解复杂的业务逻辑
- 2. 对于每一个子业务构造训练数据与验证数据
- 3. 对于每一个子业务训练优化模型
- 4. 形成完整的模型链路来解决整个业务逻辑。

LLM 应用的开发整体思路

- 以调用、发挥LLM的核心能力为核心，提供复杂业务逻辑的简单平替方案。
  - LLM的两个核心能力：指令遵循与文本生成
- 1. 用 Prompt Engineering 来替代子模型的训练调优
- 2. 通过 Prompt 链路组合来实现业务逻辑
- 3. 用一个通用大模型 + 若干业务 Prompt 来解决任务
- 将传统的模型训练调优转转为 Prompt 设计调优
  - 更简单、轻松、低成本

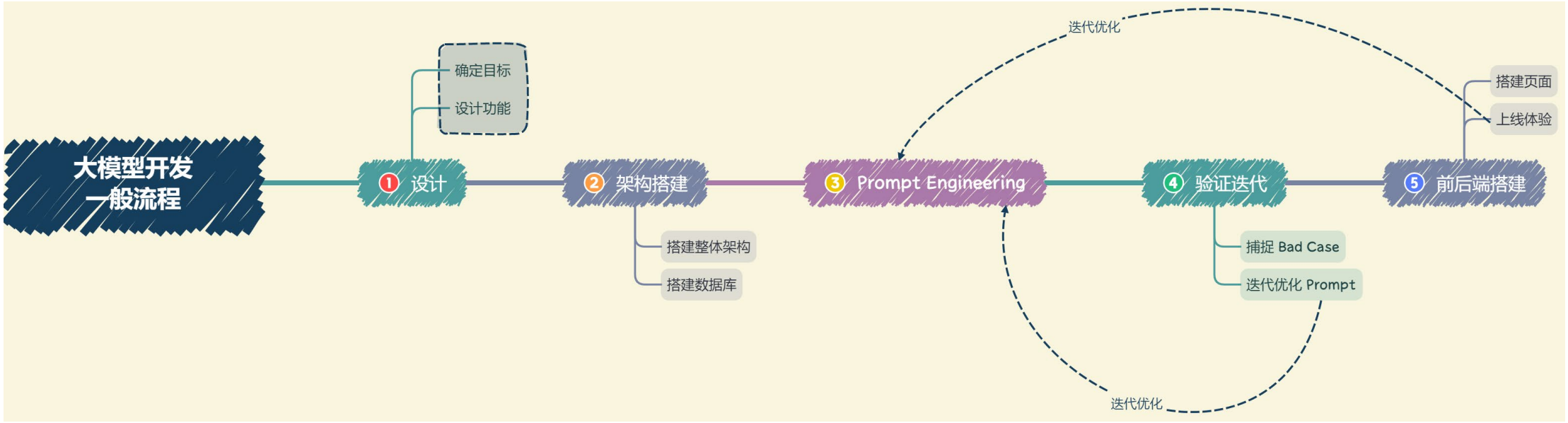
评估思路

- 在评估思路上，LLM开发与传统 AI 开发区别：
  - 传统 AI 开发：需要首先构造训练集、测试集、验证集，通过在训练集上训练模型、在测试集上调优模型、在验证集上最终验证模型效果来实现性能的评估。
  - LLM开发：从实际业务需求出发构造小批量验证集，设计合理 Prompt 来满足验证集效果。从业务逻辑中收集当下 Prompt 的 Bad Case，将 Bad Case 加入到验证集中，针对性优化 Prompt，实现较好的泛化效果。

小结：

- 传统AI开发：
  - 训练集训练、测试集调优、验证集验证
- LLM开发：
  - 初始化验证集/初始Prompt、收集Bad Case、迭代优化Prompt

开发 LLM 应用的整体流程



开发 LLM 应用的整体流程

1. 确定目标
    - 应用场景、目标人群、核心价值、从构建 MVP（最小可行性产品）开始
2. 设计功能
    - 基于功能设计业务逻辑、生成Prompt、确定核心功能，延展设计上下游功能
3. 搭建整体架构
    - 针对所设计的功能，搭建项目的整体架构（数据库 + Prompt + 通用大模型）
    - 基于LangChain实现从用户输入到应用输出的全流程贯通
4. 搭建数据库
    - 预处理数据，切片、向量化构建数据库
5. Prompt Engineering
    - 迭代构建优质的 Prompt Engineering
6. 验证迭代
    - 发现 Bad Case 并针对性改进 Prompt Engineering
7. 前后端搭建
    - 可视化界面
8. 体验优化
    - 收集数据，持续改进

搭建 LLM 项目的流程 模板

- **步骤一：项目规划与需求分析**
  - **1.项目目标：**
  - **2.核心功能：**
  - **3.确定技术架构和工具**
    1. **框架**：LangChain
    2. **Embedding 模型**：GPT、智谱、M3E
    3. **数据库**：Chroma
    4. **大模型**：GPT、讯飞星火、文心一言、GLM 等
    5. **前后端**：如：Flask+VUE 或者 FastAPI+VUE

搭建 LLM 项目的流程 模板

- **步骤二：数据准备与向量知识库构建**
  - 实现原理：如：
    - 加载本地文档 -> 读取文本 -> 文本分割 -> 文本向量化 -> question 向量化 -> 在文本向量中匹配出与问句向量最相似的 top k 个 -> 匹配出的文本作为上下文和问题一起添加到 Prompt 中 -> 提交给 LLM 生成回答。
  - **1.收集和整理用户提供的文档**
  - **2.将文档词向量化**
  - **3.将向量化后的文档导入 Chroma 知识库，建立知识库索引**

搭建 LLM 项目的流程 模板

- **步骤三：大模型集成与 API 连接**
  - **如：**

  1.集成 GPT、星火、文心、GLM 等大模型，配置 API 连接

  2.编写代码，实现与大模型 API 的交互，以便获取问题回答

搭建 LLM 项目的流程 模板

- **步骤四：核心功能实现**
  - **如：**

  1.构建 Prompt Engineering，实现LLM回答功能，根据用户提问和知识库内容生成回答。

  2.实现流式回复，允许用户进行多轮对话。

  3.添加历史对话记录功能，保存用户与助手的交互历史。

搭建 LLM 项目的流程 模板

- **步骤五：核心功能迭代优化**
  - **如：**

  1.进行验证评估，收集 Bad Case

  2.根据 Bad Case 迭代优化核心功能实现

搭建 LLM 项目的流程 模板

- **步骤六：前端与用户交互界面开发**
  - **如：**

  1.搭建前端界面

  2.实现用户上传文档、创建知识库的功能

  3.设计用户界面，包括问题输入、知识库选择、历史记录展示等

搭建 LLM 项目的流程 模板

- **步骤七：部署测试与上线**
  - **如：**

  1.部署项目到服务器或云平台，确保可在互联网上访问

  2.进行生产环境测试，确保系统稳定

  3.上线并向用户发布

搭建 LLM 项目的流程 模板

- **步骤八：维护与持续改进**
  - **如：**

  1.监测系统性能和用户反馈，及时处理问题

  2.定期更新知识库，添加新的文档和信息

  3.收集用户需求，进行系统改进和功能扩展

基于远程服务器的开发

- 选择 VSCode 连接远程服务器，在本地编辑器中直接操作
  - 安装 SSH 插件 打开 VSCODE 的插件市场，搜索 SSH，找到Remote - SSH 插件并安装
  - 获取服务器 IP
  - 配置 SSH 打开刚刚下好的远程资源管理器插件，添加服务器的 SSH， ssh -p port username@ip port 一般配置为 22, username 可以用 root 或者自定义的用户名 IP 替换成服务器的 IP 选择本地的 SSH 配置文件
  - 连接 之后连接时，可以继续点击左侧的远程资源管理器找到我们的服务器，右边有两个选项。
    - 右下角：本窗口打开
    - 左上角有加号：新窗口打开
  - 打开目录 之后点击打开文件夹，输入需要的目录即可打开
- Using GitHub Codespaces in Visual Studio Code - GitHub Docs

基于远程服务器的开发

- **Jupyter Notebook 使用**
- …

GitHub Codespaces

- Codespaces 文档 - GitHub 文档
  - codespace 可以通过网页访问
  - 免费额度 找到 GitHub 的账户设置后，可以在**Plans and usage**中看到剩余的免费额度
  - 调整挂起时间（时间过长会浪费额度）
- **创建第一个 codespace**

1. 打开网址链接：https://github.com/features/codespaces
2. 登录你的 GitHub 账户
3. 点击图示 **Your repositories**
4. 进入存储库列表后，点击图示 New，新建一个存储库
5. 根据需要设置即可，为方便和安全起见 Add a README file 建议勾上，同时选择 Private（因为用到 API key），设置完成后点击 Create repository
6. 创建好存储库后，点击 **code** 选择 **Codespaces**, 点击图示 **Create codespace on main**
7. 接下来操作与 VSCode 相同，可根据需要安装插件调整设置

8. **本地 VSCode 连接 Codespace（非必需）**

几篇论文

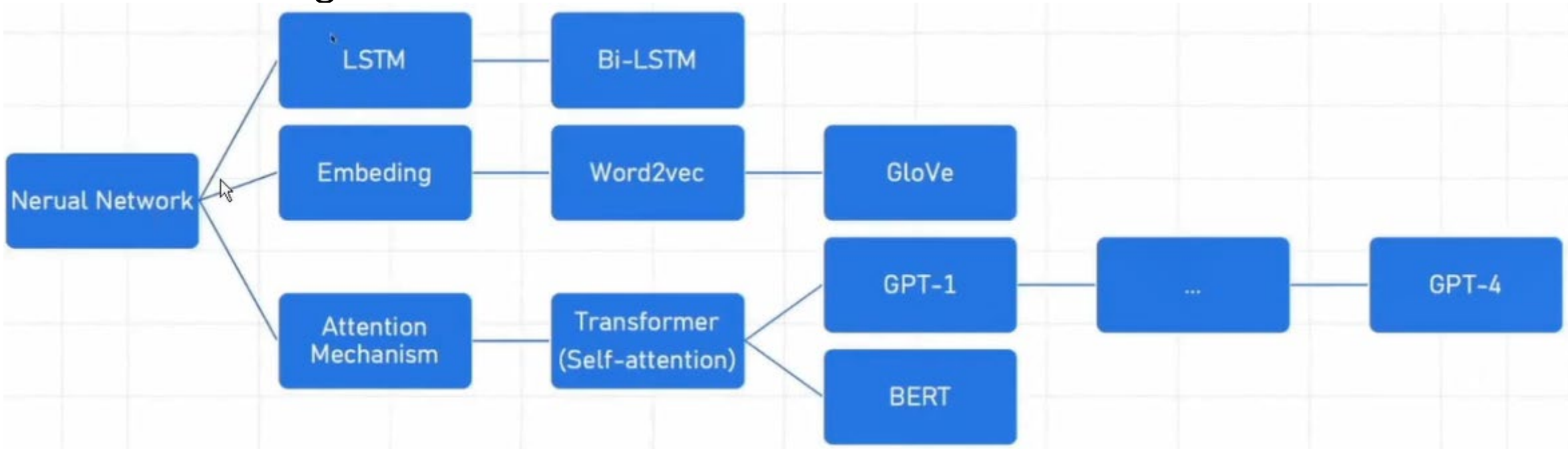
- 0.深度学习的思想融入到语言模型（2003）
  - bengio03a.pdf (jmlr.org)
  - bengio03a.pdf
- 1. 注意力机制的提出（2014）
  - 提出注意力机制，并引入到神经网络中，解决NLP中最典型的机器翻译问题。
  - [1409.0473] Neural Machine Translation by Jointly Learning to Align and Translate (arxiv.org)
  - 1409.0473v7.pdf
- 2. 自注意力机制的提出（Tansformer）（2017）
  - 提出了（Transformer模型）自注意机制的模型架构，给出了替代CNN/RNN的新的深度学习模型架构
    - Attention：叠加的堆栈（encode+decode）
  - [1706.03762] Attention Is All You Need (arxiv.org)
  - 1706.03762v7.pdf
- 3. GPT1（2018）
  - language_understanding_paper.pdf (openai.com)
  - language_understanding_paper.pdf
  - Code: huggingface/transformers: 🤗 Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX. (github.com)
- 4. BERT（双向Transformer）（2019）
  - [1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (arxiv.org)
  - 1810.04805v2.pdf

几篇论文

- 5. Embedding
  - Tomas Mikolov –Word2Vec -3篇
    - [1301.3781] Efficient Estimation of Word Representations in Vector Space (arxiv.org)
    - [1405.4053] Distributed Representations of Sentences and Documents (arxiv.org)
    - [1607.04606] Enriching Word Vectors with Subword Information (arxiv.org)
    - 将词（中文、英文…）转换为词向量，同一个语言空间中同一个词有相同的词向量，即：将语言学的问题转化为数学问题。
    - 不同的语言，就是不同的高维向量空间。就可以使用数学的方法解决问题，如：语言间的翻译、判断词（语义）的相似度…
  - GloVe: Global Vectors for Word Representation (stanford.edu)
    - glove.pdf
- 6. GPT4（2023）
  - [2303.08774] GPT-4 Technical Report (arxiv.org)
    - 2303.08774v6.pdf
  - [2303.12712] Sparks of Artificial General Intelligence: Early experiments with GPT-4 (arxiv.org)
    - 2303.12712v5.pdf
  - [2309.17421] The Dawn of LMMs: Preliminary Explorations with GPT-4V(ision) (arxiv.org)
    - 2309.17421v2.pdf

论文的脉络

- NN → CNN → RNN → LSTM
- 注意力机制 → 自注意力机制 → 多头注意力机制
- Embedding → Word2vec → Global Vectors



GPT模型的历史

- 自然语言处理的局限
- 机器学习的引入
- 深度学习的引入
- Transformer
- GPT模型
- LLM时代

早期NLP的局限

- 自然语言处理（Natural Language Processing, NLP）是人工智能领域（AI）的一个重要分支
  - NLP旨在使计算机能够理解、处理和生成人类语言
    - NLP = NLU + NLG
- 早期基于规则、统计模型实现
  - 采用**统计学习方法**来预测词汇，通过分析前面的词汇来预测下一个词汇
- 局限：
  - 对复杂语境理解能力不足、生成自然流畅文本的能力不够
  - 只能完成简单、生硬、且固定模板下的对话

机器学习的引入

- 机器学习（Machine Learning, ML）是AI领域的一个重要分支，随着机器学习技术的不断进步和普及，NLP领域也迎来了新的机遇。
- 机器学习技术主要包括无监督学习、有监督学习、强化学习。
  - 通过大规模数据的学习和模式识别，使得计算机能够更好地理解和处理自然语言。
  - 传统的NLP任务，如文本分类、命名实体识别等，采用机器学习方法，取得了显著的进展。
- 局限：
  - 传统的机器学习方法在处理复杂的自然语言任务时仍然存在一些局限：
    - 如需要手动提取特征、模型泛化能力有限…

深度学习的引入

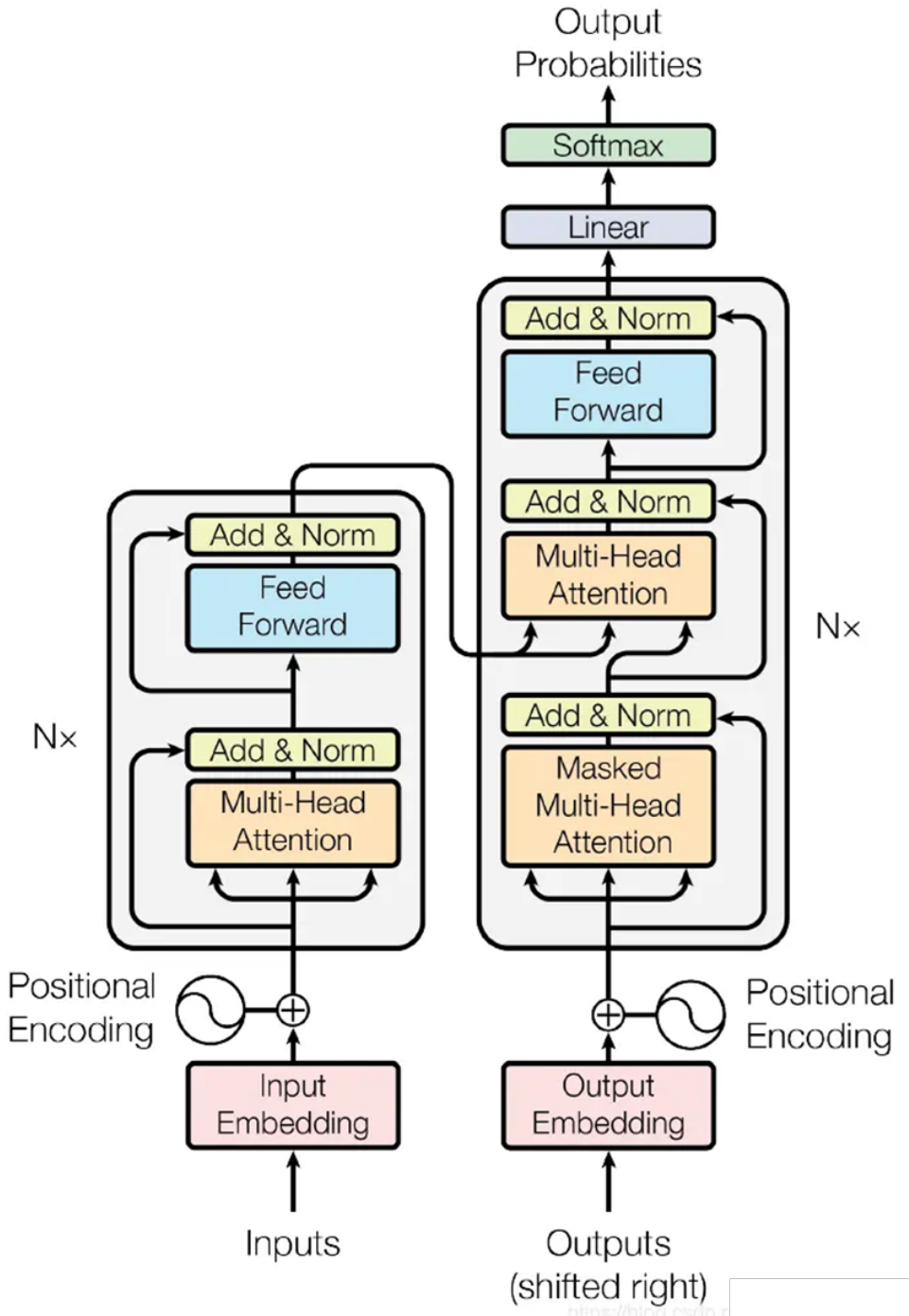
- 深度学习（Deep Learning, DL）是机器学习中有监督学习的一种实现，模仿人类大脑的结构和工作方式，通过构建深层神经网络来处理和理解复杂的数据。
- 常见的神经网络模型有传统的循环神经网络（RNN）和卷积神经网络（CNN）等。
- 深度学习的出现，为NLP提供了工具，使用深度学习模型利用大规模数据的学习能力以及复杂的神经网络结构，可以从原始的文本数据中提取高层次的语义表示，从而实现更准确和更灵活的NLP任务。
- 局限：
  - 记忆长度： 传统的神经网络在处理长序列时往往会出现记忆衰减的问题
  - 并行性 ： 传统的神经网络在处理序列数据时通常是逐步顺序处理的，因此难以利用并行计算的优势。
  - 长距离依赖性： 传统的神经网络在处理长距离文本时，会存在梯度消失和梯度爆炸问题，导致在处理长序列时性能下降

Transformer

- 2017年的论文《Attention is All You Need》中提出Transformer模型
  - 引入自注意力机制（Self-Attention Mechanism）和位置编码（Positional Encoding）等，在处理序列数据时取得了巨大成功。
  - Transformer模型具有更高的并行性和更长的记忆长度，能够更好地处理序列数据中的长距离依赖关系，解决了传统神经网络的局限。
  - 基于Transformer可以生成自然、流畅、复杂的文本，提高了NLP的能力。
- Transformer模型的出现标志着NLP领域迈入了一个新的阶段
  - 可以进行语言模型预训练、序列到序列学习等任务
  - 是后续更多基于深度学习的NLP模型的基础，如BERT、GPT等

Transformer 模型架构

- ① Input Embedding
- ② Positional Encoding
- ③ encoder编码器
- ④ decoder解码器
- ⑤ 理解整个过程



Transformer架构-以GPT为例

- 在LLM中，Transformer架构也是至关重要的组成部分之一。
- Transformer模型的核心是由多个编码器和解码器层组成的深度神经网络结构。
  - 编码器负责将输入文本进行编码，提取语义信息
  - 解码器则负责生成输出文本或者解码任务特定的表示
- Transformer架构具有高度的并行性和可扩展性，能够有效地处理长序列数据。

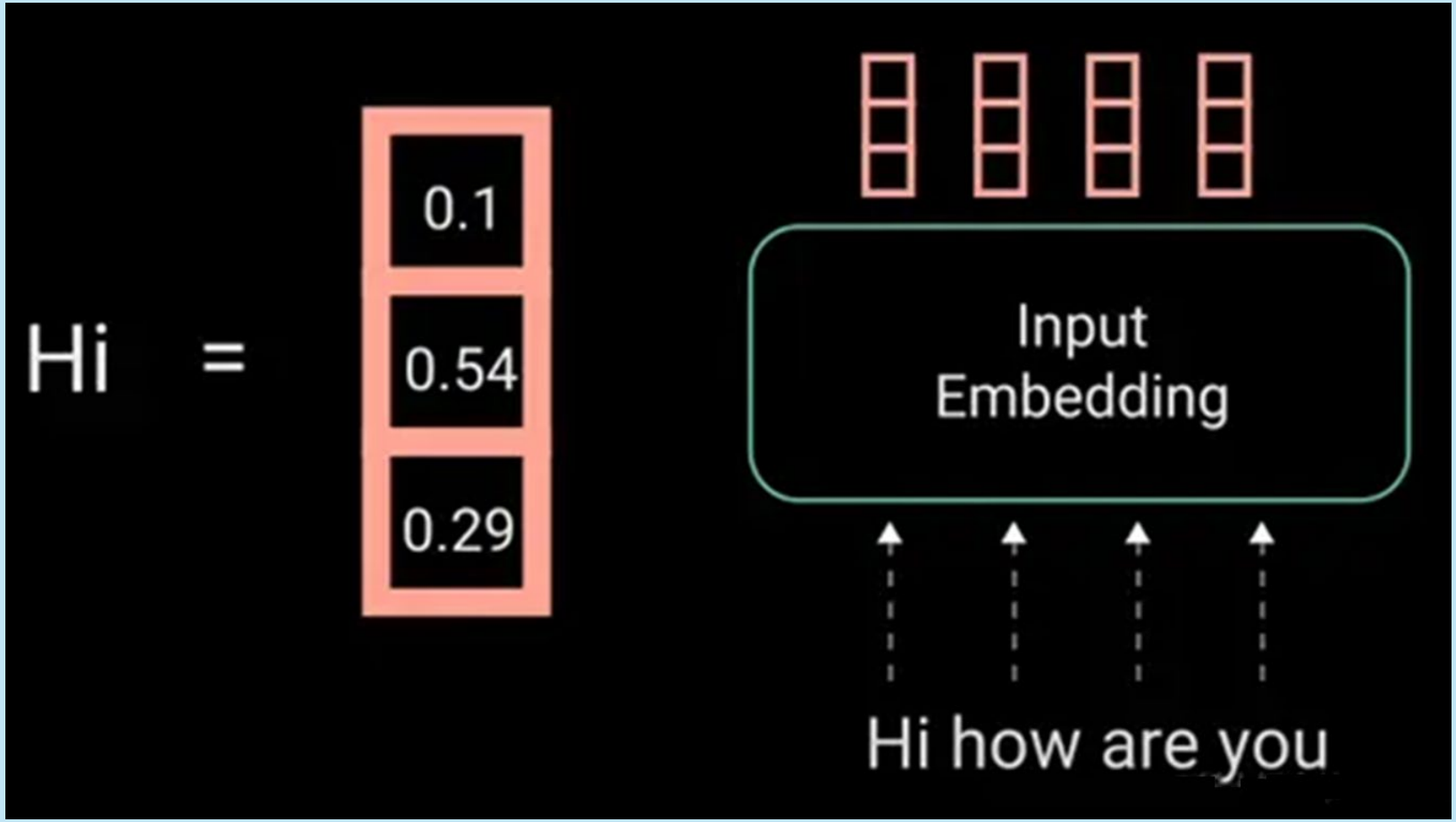
① Input Embedding

- 神经网络需要数字作为输入（神经网络中大部分是数学函数，比如线性函数等），需要将输入的文本数据转换为能应用于线性函数的数据。
  - one-hot encode，即：用唯一的向量替换每个可能的变量值（文本数据）的技术。
- 首先需要对输入的文本，进行向量化处理，也称为Embedding:

Embedding

- Input : Hi how are you
- Tokens : [Hi, how, are, you]
  - 先对原始文本进行分词处理，依次找到每个token
- Token Ids : [10, 15, 26, 30]
  - 再定义一个token表，然后依次找到每个token对应的tokenID
- Input Embedding :
  - [[0.1, 0.54, 0.29], [0.3, 0.12, 0.62], [0.13, 0.14, 0.9], [0.1, 0.2, 0.19]]

Embedding



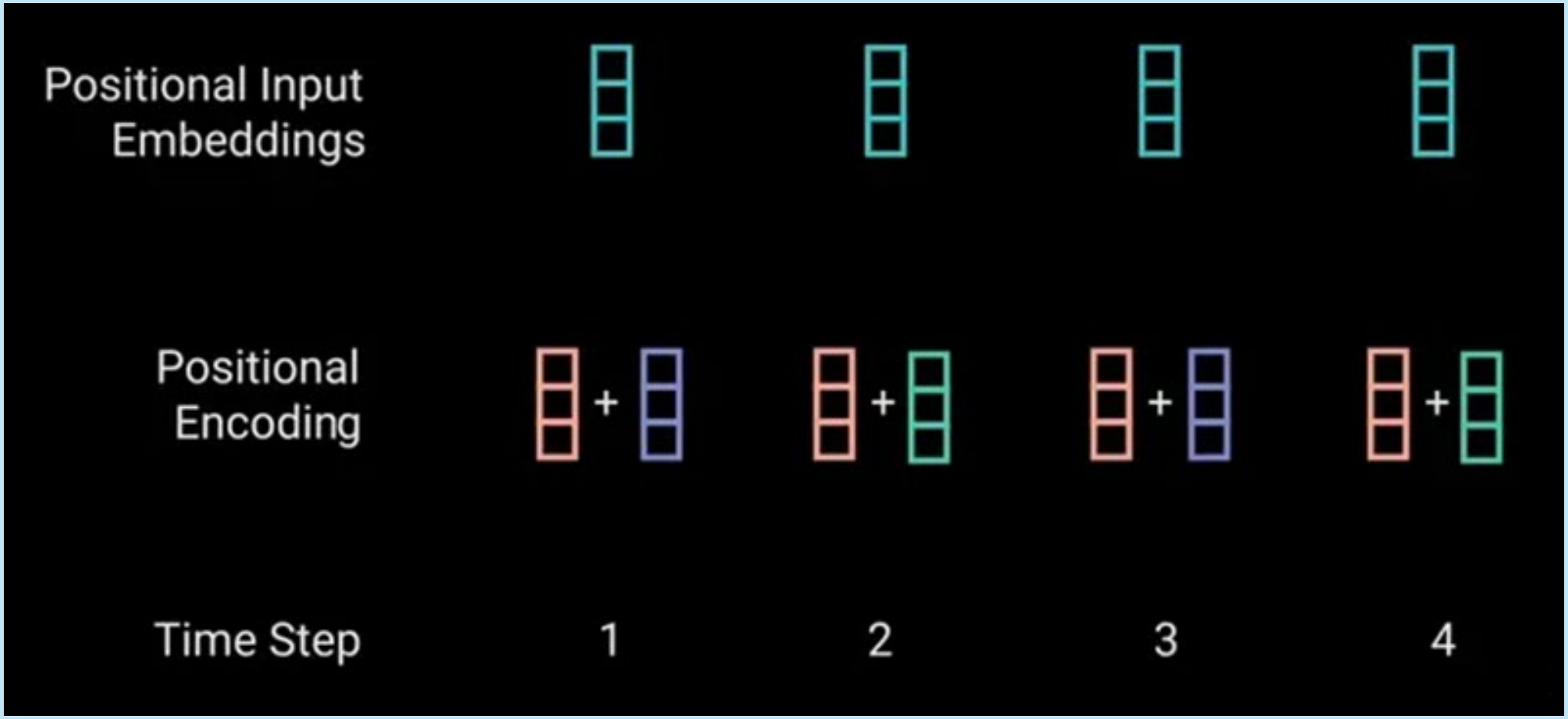
② Positional Encoding

- Positional Encoding用于解决长距离依赖
  - 将序列中的位置信息嵌入到词向量中的技术
    - 在词向量中添加位置编码来表示每个词的位置信息
  - 模型在接收输入时，不仅可以学习词的语义信息，还可以了解词在序列中的顺序

Positional Encoding

- Input : Hi how are you
- Positional index : 0, 1, 2, 3
- Positional Embedding : [[0.2, 0.3, 0.0], [0.2, 0.3, 0.1], [0.2, 0.3, 0.2], [0.2, 0.3, 0.3],]
- Positional Embedding + Input Embedding : [[...], [...], [...]]

Positional Encoding



③ encoder编码器

- 编码器是负责将输入序列转换为一系列隐藏表示的部分
- 负责对输入序列进行编码和提取特征，为后续的任务提供语义信息
- Transformer中的编码器由多个相同的层堆叠而成（多层堆叠）
  - 每一层都由两个子层组成：自注意力机制层和前馈神经网络层

每一层的工作流程

- 自注意力机制层（Self-Attention Layer）
  - 负责计算输入序列中每个位置的注意力权重，以便模型能够在整个序列中捕捉上下文信息和长距离依赖关系。
  - 每个位置的隐藏表示会根据其他位置的信息进行加权求和，从而得到更丰富的表示。
- 前馈神经网络层（Feedforward Neural Network Layer）
  - 对每个位置的隐藏表示进行非线性变换和映射，以提取更高级的特征
  - 通常包括一个全连接层和一个激活函数，用于增强编码器的表征能力

多层编码器的堆叠和输出

- 通过多层编码器的堆叠，Transformer模型能够逐层地提取输入序列中的特征，并将其转换为适合后续任务处理的表示形式。
- 编码器的输出可以被用于解码器进行下游任务的执行，如语言建模、机器翻译等。
- 隐藏表示（Hidden Representation）是指输入数据经过一系列的线性变换和非线性变换后得到的一个向量，它用来表示输入数据的抽象特征。
  - 比如在解码器中，隐藏表示是关于当前位置的输入序列信息的一个抽象表示，它包含了输入序列的语义信息，但已经被转换成了一个更加紧凑的形式，便于后续的处理和生成。
- 在长距离文本计算中，使用到了成对卷积层Pairwise Convolutional Layer技术

④ decoder解码器

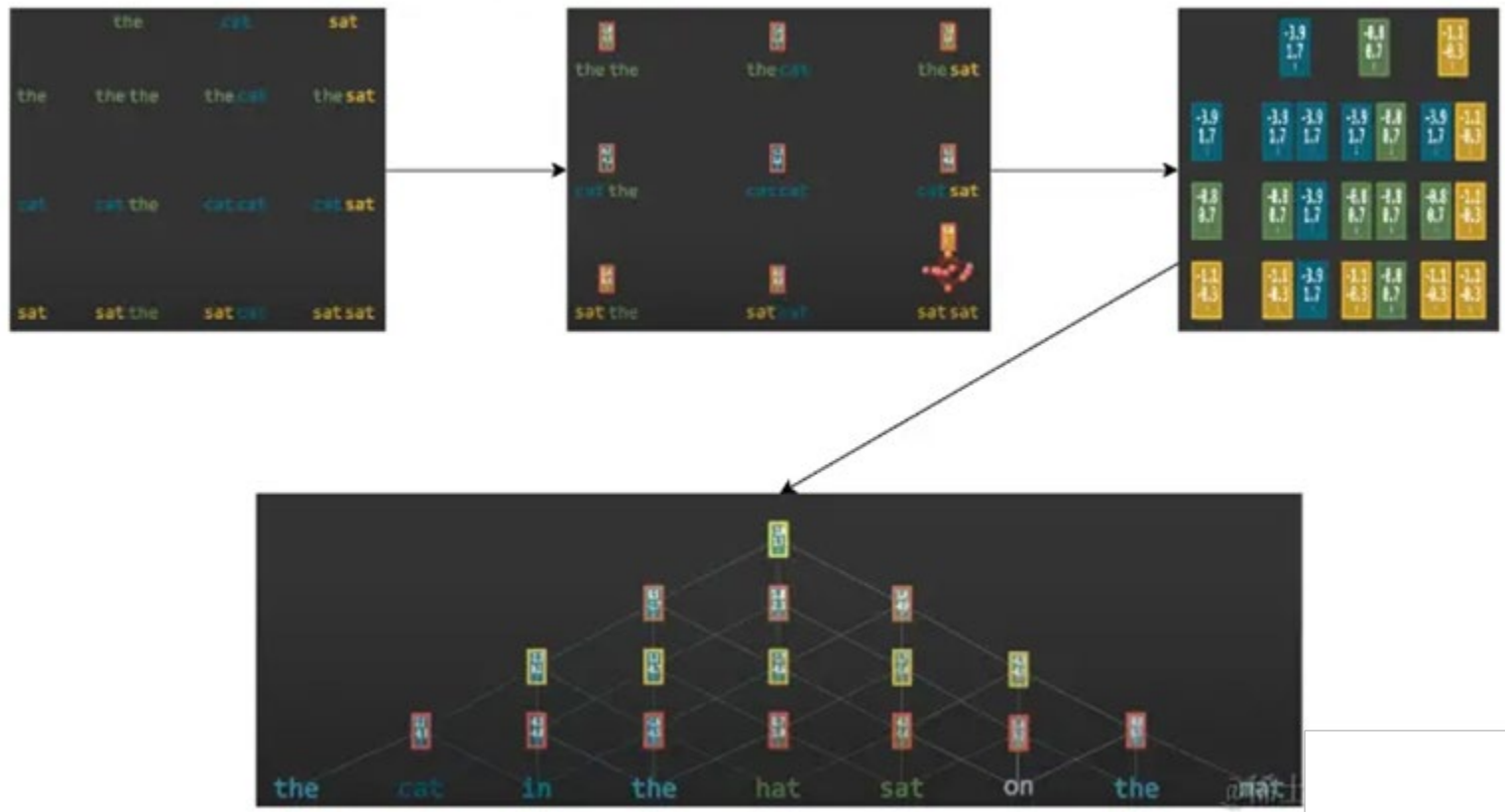
- 解码器是负责将编码器产生的隐藏表示转换为目标序列的部分。
- 主要通过自注意力机制和编码-解码注意力机制来理解输入序列的语义信息，并将其转换为目标序列的表示。

decoder解码器工作流程

- 自注意力机制层（Self-Attention Layer）
  - 解码器的每个位置都会对自己位置的词进行注意力计算，捕捉目标序列中的内部依赖关系。
- 编码-解码注意力机制层（Encoder-Decoder Attention Layer）
  - 这一层允许解码器在生成每个位置的输出时，与编码器的隐藏表示进行交互。
  - 通过计算解码器当前位置与编码器各个位置之间的注意力权重，解码器可以聚焦于输入序列的不同部分，提取相关信息。
- 前馈神经网络层（Feedforward Neural Network Layer）
  - 最后一层是一个前馈神经网络层，将解码器当前位置的隐藏表示转换为目标序列的输出。
  - 通过多层的全连接层和激活函数，前馈神经网络层提取更高级的特征，产生最终的输出。
- 通过上述三个步骤，解码器能够逐步生成并调整目标序列的每个位置的输出，并在生成过程中根据输入序列和先前生成的部分来进行动态调整。

⑤ 如何理解整个过程

- 1. 输入向量化（向量）
  - 先把输入文本和每一个词的位置信息进行向量化并计算得出每一个词的向量值
- 2. 编码器处理（压缩）
  - 在编码器中对输入的向量值进行一层又一层的无数次数学计算，保证每一层都有上一层的所有上下文信息，最终输出一系列高度抽象的隐藏表示（向量数据）。
  - 这些隐藏表示包含了输入序列的语义信息，但已经被转换成了一个更为紧凑的形式，以便后续的处理。


the cat sat
the the the cat the sat
cat cat the cat cat cat sat
sat sat the sat cat sat sat
the the the cat the sat
cat the cat cat cat sat
sat the sat cat sat sat
the cat in the hat sat on the mat

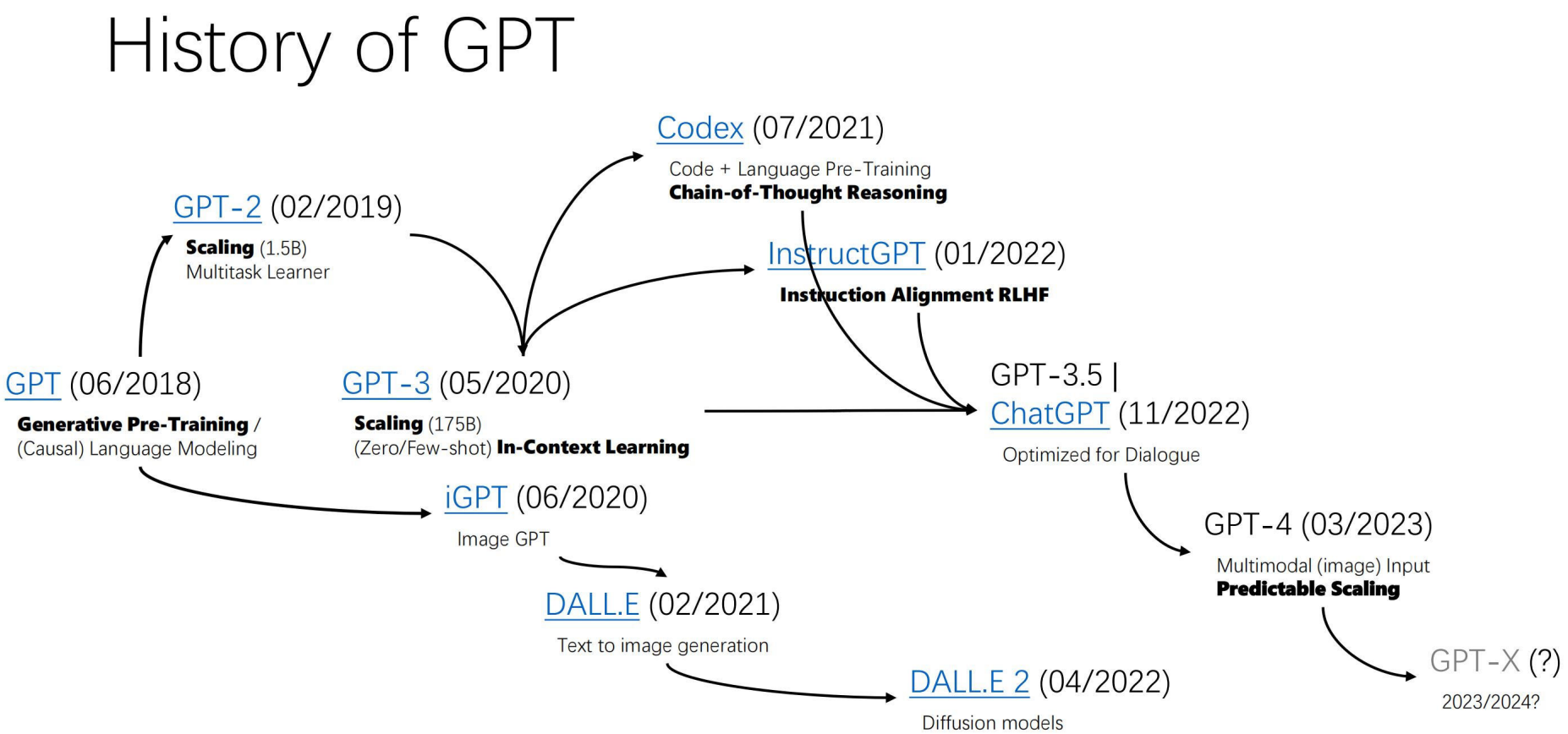
⑤ 如何理解整个过程

- 3. 解码器理解（解压）
  - 编码器产生的隐藏表示并不直接包含目标序列的信息。
  - 需要解码器对这些隐藏表示进行解压，将其中包含的输入序列的语义信息转换为目标序列。
  - 解码器通过多次迭代生成和调整的过程，逐步将编码器的输出转化为完整的目标序列。
- 4. 解码器输出（输出）
  - 解码器最终负责逐步生成和调整目标序列的每个位置的输出。
  - 解码器根据先前已生成的部分目标序列以及编码器的输出，不断地预测并生成下一个位置的目标序列元素。
  - 通过多次迭代，解码器逐步生成完整的目标序列，从而完成了序列到序列的转换任务。

GPT模型

- OpenAI于2018年提出GPT（Generative Pre-trained Transformer）
- 生成型预训练变换模型 3 （Generative Pre-trained Transformer 3，简称 GPT-3）是一个自回归语言模型，目的是为了使用深度学习生成人类可以理解的自然语言。
  - GPT-3 - 维基百科，自由的百科全书 (wikipedia.org)
- ChatGPT，全称聊天生成预训练转换器[2]（英语：Chat Generative Pre-trained Transformer[3]），是OpenAI开发的人工智能聊天机器人程序，于2022年12月推出。
  - ChatGPT - 维基百科，自由的百科全书 (wikipedia.org)
- 生成型预训练变换模型 4（Generative Pre-trained Transformer 4，简称GPT-4）是由OpenAI公司开发并于2023年3月14日发布。
  - GPT-4 - 维基百科，自由的百科全书 (wikipedia.org)
- GPT-4o（Generative Pre-trained Transformer 4 Omni）是由OpenAI训练的多语言、多模态（多种类型数据，例如文本、图像、音频等）GPT大型语言模型，于2024年5月13日发布。
  - GPT-4o - 维基百科，自由的百科全书 (wikipedia.org)

GPT历史



GPT: Generative Pre-trained Transformer

GPT特点

- GPT是一种
  - ①基于Transformer架构
  - ②预训练的
  - ③生成式的 NLP模型
- GPT-3的神经网络包含1750亿个参数，需要700GB来存储
- GPT-4的参数规模是在GPT-3的10倍以上

LLM时代

- 在技术创新方面，GPT模型并未带来全新的技术突破
  - 大力出奇迹…
- 通过大规模预训练来学习语言模型，直接推动了行业内大型语言模型（Large Language Model, LLM）的兴起，更引发了生成式人工智能（Artificial Intelligence Generated Content, AIGC）领域的革命。
  - LLM = RAG + AIGC

LLM

- LLM是什么
- 常见LLM
- LLM特点
- LLM工作机制
- LLM工作场景

什么是LLM

- LLM（large language model，LLM）大语言模型是一种语言模型，由具有许多参数（通常数十/百亿个权重或更多）的人工神经网络组成，使用无监督学习对大量未标记语料（文本数据）进行预训练生成通用模型，获得对语言深层次的理解，然后再在不同特定任务下进行微调。
  - 是一种旨在理解和生成人类语言的人工智能模型。
- LLM （large language model，LLM）：
  - Large（大型）： 该模型具有大量的参数、大量的语料、复杂庞大的结构
  - Language（语言）： 该模型用于NLP任务，可以处理和生成自然语言，如英语、中文等
  - Model（模型）： 该模型是基于深度学习构建的神经网络模型，一般都是基于Transformer架构
- GPT模型是LLM的重要代表和标杆，行业内的LLM大都和GPT模型类似，是标准模型
  - 国外的有 GPT-3.5、GPT-4、PaLM、Claude 和 LLaMA 等
  - 国内的有文心一言、讯飞星火、通义千问、ChatGLM、百川等
- **涌现**

涌现能力（emergent abilities）

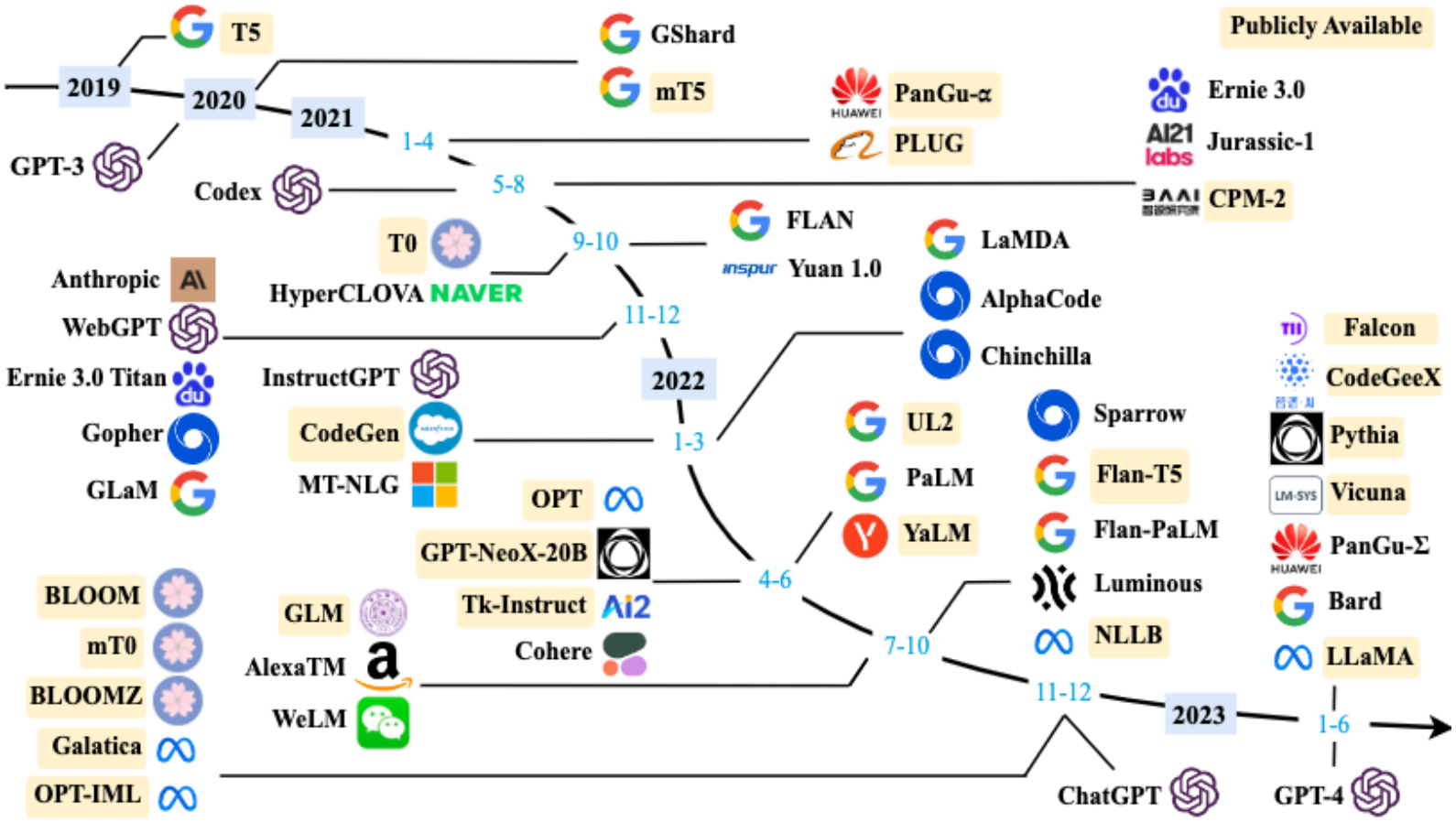
- 涌现 - 维基百科，自由的百科全书 (wikipedia.org)
- **涌现能力**
  - 区分大语言模型（LLM）与以前的预训练语言模型（PLM）最显著的特征
  - 在小型模型中不明显，在大型模型中特别突出
    - 类似物理学中的相变现象
  - 模型性能随着规模增大而迅速提升，超过了随机水平
    - 量变引起质变
- 涌现能力可以与复杂任务有关，但更关注的是其通用能力。
  - 让 LLM 在处理各种任务时表现出色，成为解决复杂问题和应用于多领域的工具。

LLM 典型的涌现能力

- 上下文学习
  - 上下文学习能力是由 GPT-3 首次引入的。
  - 允许语言模型在提供自然语言指令或多个任务示例的情况下，通过理解上下文并生成相应输出的方式来执行任务，而无需额外的训练或参数更新。
- 指令遵循
  - 指令微调：通过使用自然语言描述的多任务数据进行微调
  - LLM 在使用指令形式化描述的未见过的任务上表现良好
    - LLM 能够根据任务指令执行任务，而无需事先见过具体示例，具有强大的泛化能力。
- 逐步推理
  - 小型语言模型难以解决涉及多个推理步骤的复杂任务，例如数学问题
  - LLM 采用**思维链（CoT, Chain of Thought）**推理策略
    - 利用包含中间推理步骤的提示机制解决任务，从而得出最终答案
    - **据推测，这种能力可能是通过对代码的训练获得的**

常见LLM

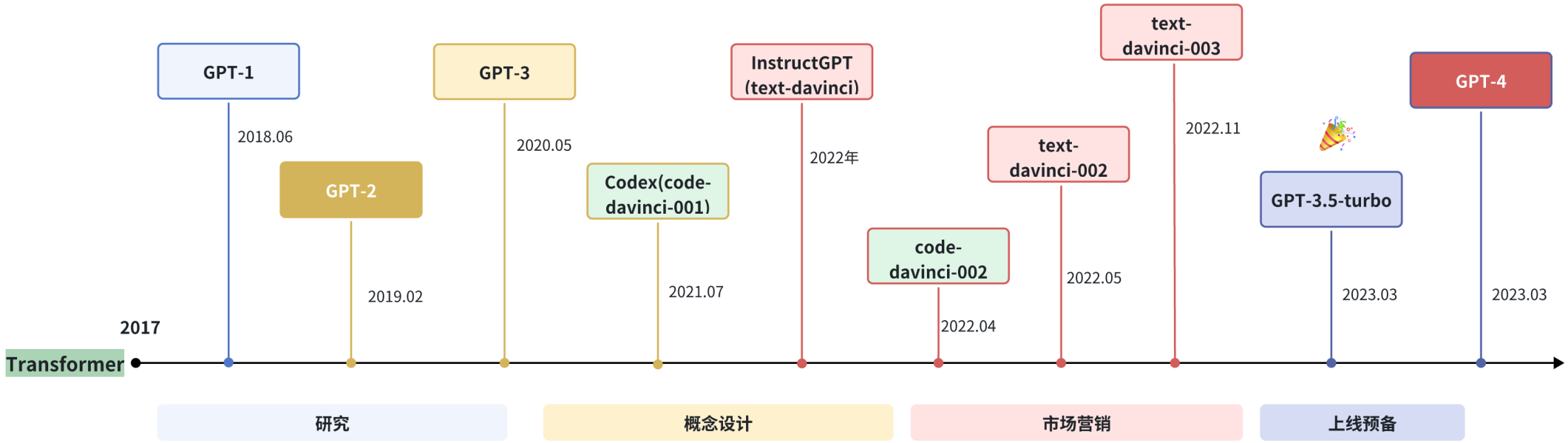
[2303.18223] A Survey of Large Language Models (arxiv.org)



闭源： **GPT 系列**

- OpenAI 公司在 2018 年提出的 GPT（Generative Pre-Training）模型是典型的生成式预训练语言模型 之一。
- GPT 模型的基本原则是通过语言建模将世界知识压缩到仅解码器(decoder-only) 的 Transformer 模型中，这样它就可以恢复(或记忆)世界知识的语义，并充当通用任务求解器。
- 两个关键点：
  - 训练能够准确预测下一个单词的 decoder-only 的 Transformer 语言模型
  - 扩展语言模型的大小

OpenAI 在 LLM 上的研究阶段



ChatGPT

- 2022 年 11 月，OpenAI 发布了基于 GPT 模型（GPT-3.5 和 GPT-4） 的会话应用 ChatGPT。
  - ChatGPT 是基于强大的 GPT 模型开发的，具有特别优化的会话能力。
    - 展现出与人类交流的出色能力
- ChatGPT 从本质上来说是一个 LLM 应用，是基于基座模型开发出来的，与基座模型有本质的区别。
  - 其支持 GPT-3.5 和 GPT-4 两个版本。
- 随便说几点：
  - ChatGPT 支持最长达 32,000 个字符，知识截止日期是 2021 年 9 月（？）。
  - 可以执行各种任务，包括代码编写、数学问题求解、写作建议等。
  - 拥有丰富的知识储备，对数学问题进行推理的技能，在多回合对话中准确追踪上下文，并且与人类安全使用的价值观非常一致。
  - ChatGPT 支持插件机制，这进一步扩展了 ChatGPT 与现有工具或应用程序的能力。
  - 到目前为止，是人工智能历史上最强大的聊天机器人。

GPT-4

- 2023 年 3 月发布的 GPT-4，将文本输入扩展到多模态信号
  - GPT3.5 拥有 1750 亿 个参数，GPT4 的参数量官方并没有公布，但有相关人员猜测，GPT-4 在 120 层中总共包含了 1.8 万亿参数，也就是说，GPT-4 的规模是 GPT-3 的 10 倍以上。
  - GPT-4 比 GPT-3.5 解决复杂任务的能力更强，在许多评估任务上表现出较大的性能提升。
  - 由于六个月的迭代校准(**RLHF** 训练中有额外的安全奖励信号)，GPT-4 对恶意或挑衅性查询的响应更安全，并应用了一些干预策略来缓解 LLM 可能出现的问题，如幻觉、隐私和过度依赖。
- 2023 年 11 月 7 日， OpenAI 召开了首个开发者大会，会上推出了最新的大语言模型 GPT-4 Turbo
  - 上下文长度扩展到 128k（相当于 300 页文本），训练知识更新到 2023 年 4 月
- 2024 年 5 月 14 日，新一代旗舰生成模型 GPT-4o 正式发布
  - GPT-4o 具备了对文本、语音、图像三种模态的深度理解能力
  - 反应迅速且富有情感色彩，极具人性化
  - GPT-4o 是完全免费的（每天的免费使用次数是有限的）
    - GPT3.5 是免费的，而GPT-4 是收费的。需要开通 plus 会员 20 美元/月

主流GPT模型API对比

语言模型名称	上下文长度	特点	input 费用($/million tokens)	output 费用($/1M tokens)	知识截止日期
GPT-3.5-turbo-0125	16k	经济，专门对话	0.5	1.5	2021 年 9 月
GPT-3.5-turbo-instruct	4k	指令模型	1.5	2	2021 年 9 月
GPT-4	8k	性能更强	30	60	2021 年 9 月
GPT-4-32k	32k	性能强，长上下文	60	120	2021 年 9 月
GPT-4-turbo	128k	性能更强	10	30	2023 年 12 月
GPT-4o	128k	性能最强，速度更快	5	15	2023 年 10 月

Embedding 模型名称	维度	特点	费用($/ 1M tokens)
text-embedding-3-small	512/1536	较小	0.02
text-embedding-3-large	256/1024/3072	较大	0.13
ada v2	1536	传统	0.1

闭源： **Claude 系列**

- Claude 系列模型是由 OpenAI 离职人员创建的 Anthropic 公司开发的闭源语言大模型。
  - https://claude.ai/chats
- Claude 3 系列包括三个不同的模型，分别是 Claude 3 Haiku、Claude 3 Sonnet 和 Claude 3 Opus。

模型名称	上下文长度	特点	input 费用 ($/1M tokens)	output 费用 ($/1M tokens)
Claude 3 Haiku	200k	速度最快	0.25	1.25
Claude 3 Sonnet	200k	平衡	3	15
Claude 3 Opus	200k	性能最强	15	75

闭源： **PaLM/Gemini 系列**

- PaLM 系列语言大模型由 Google 开发。
  - 初始版本于 2022 年 4 月发布，并在 2023 年 3 月公开了 API。
  - 2023 年 5 月，Google 发布了 PaLM 2。
  - 2024 年 2 月 1 日，Google 将 Bard(之前发布的对话应用) 的底层大模型驱动由 PaLM2 更改为 Gemini，同时也将原先的 Bard 更名为 Gemini。
    - https://ai.google/discover/palm2/
    - https://gemini.google.com/
    - 目前的 Gemini 是第一个版本，即 Gemini 1.0，根据参数量不同分为 Ultra, Pro 和 Nano 三个版本。

闭源：文心一言

- 文心一言是基于百度文心大模型的知识增强语言大模型
  - 2023 年 3 月在国内开启邀测，中文能力相对来说非常不错
  - 文心一言的基础模型文心大模型于 2019 年发布 1.0 版，现已更新到 4.0 版本，包括：
    - NLP 大模型、CV 大模型、跨模态大模型、生物计算大模型、行业大模型
  - 千帆大模型…
- 文心一言网页版分为免费版和专业版
  - https://yiyan.baidu.com/

闭源：星火大模型

- 讯飞星火认知大模型是科大讯飞发布的语言大模型，支持多种自然语言处理任务。
  - https://xinghuo.xfyun.cn/
  - 2023 年 5 月首次发布
  - 2023 年 10 月，讯飞发布讯飞星火认知大模型 V3.0
  - 2024 年 1 月，讯飞发布了讯飞星火认知大模型 V3.5
    - 支持 system 指令，插件调用等多项功能

开源：**LLaMA 系列**

https://llama.meta.com/
https://github.com/facebookresearch/llama

- LLaMA 系列模型是 Meta 开源的一组参数规模 从 7B 到 70B 的基础语言模型。
- LLaMA 于2023 年 2 月发布，2023 年 7 月发布了 LLaMA2 模型，并于 2024 年 4 月 18 日发布了 LLaMA3 模型。
- 在数万亿个字符的语料集上训练，仅使用公开可用的数据集来训练最先进的模型，不需要依赖专有或不可访问的数据集。
  - 数据集包括 Common Crawl、Wikipedia、OpenWebText2、RealNews、Books 等。
- 特点和优势：
  - 使用大规模的数据过滤和清洗技术，以提高数据质量和多样性，减少噪声和偏见。
  - 使用高效的数据并行和流水线并行技术，以加速模型的训练和扩展。
  - LLaMA 13B 在 CommonsenseQA 等 9 个基准测试中超过了 GPT-3 (175B)，而 LLaMA 65B 与最优秀的模型 Chinchilla-70B 和 PaLM-540B 相媲美。
  - LLaMA 通过使用更少的字符来达到最佳性能，从而在各种推理预算下具有优势。

开源：通义千问

https://tongyi.aliyun.com/
https://github.com/QwenLM/Qwen2

- 通义千问由阿里巴巴基于“通义”大模型研发
- 2023 年 4 月正式发布
- 2023 年 9 月，阿里云开源了 Qwen（通义千问）系列工作
- 2024 年 2 月 5 日，开源了 Qwen1.5（Qwen2 的测试版）
- 2024 年 6 月 6 日正式开源了 Qwen2。
  - Qwen2 是一个 decoder-Only 的模型，采用 SwiGLU 激活、RoPE、GQA的架构
  - 中文能力相对来说非常不错的开源模型
- 目前，已经开源了 5 种模型大小：0.5B、1.5B、7B、72B 的 Dense 模型和 57B (A14B)的 MoE 模型；所有模型均支持长度为 32768 token 的上下文。将 Qwen2-7B-Instruct 和 Qwen2-72B-Instruct 的上下文长度扩展至 128K token。

开源：**GLM 系列**

https://chatglm.cn/
https://github.com/THUDM/GLM-4

- GLM 系列模型是清华大学和智谱 AI 等合作研发的语言大模型
  - 2023 年 3 月 发布 ChatGLM、6 月发布 ChatGLM 2、10 月推出 ChatGLM3
  - 2024 年 1 月 16 日 发布了 GLM4、6 月 6 日正式开源
    - GLM-4-9B-Chat 支持多轮对话、网页浏览、代码执行、自定义工具调用（Function Call）和长文本推理（支持最大 128K 上下文）等功能。
- 开源了对话模型 GLM-4-9B-Chat、基础模型 GLM-4-9B、长文本对话模型 GLM-4-9B-Chat-1M（支持 1M 上下文长度）、多模态模型 GLM-4V-9B 等全面对标 OpenAI。
- 2024 年 1 月 16 日 发布了 ChatGLM4，但目前还没有开源。

开源：Baichuan 系列

https://www.baichuan-ai.com/chat
https://github.com/baichuan-inc

- Baichuan 是由百川智能开发的开源可商用的语言大模型，基于 Transformer 解码器架构（decoder-only）。
  - 2023 年 6 月 15 日发布 Baichuan-7B 和 Baichuan-13B
    - 百川同时开源了预训练和对齐模型
      - 预训练模型是面向开发者的“基座”
      - 对齐模型则面向广大需要对话功能的普通用户
  - Baichuan2 于 2023年 9 月 6 日推出
    - 发布了 7B、13B 的 Base 和 Chat 版本，并提供了 Chat 版本的 4bits 量化
  - 2024 年 1 月 29 日 发布 Baichuan 3，但是目前还没有开源。

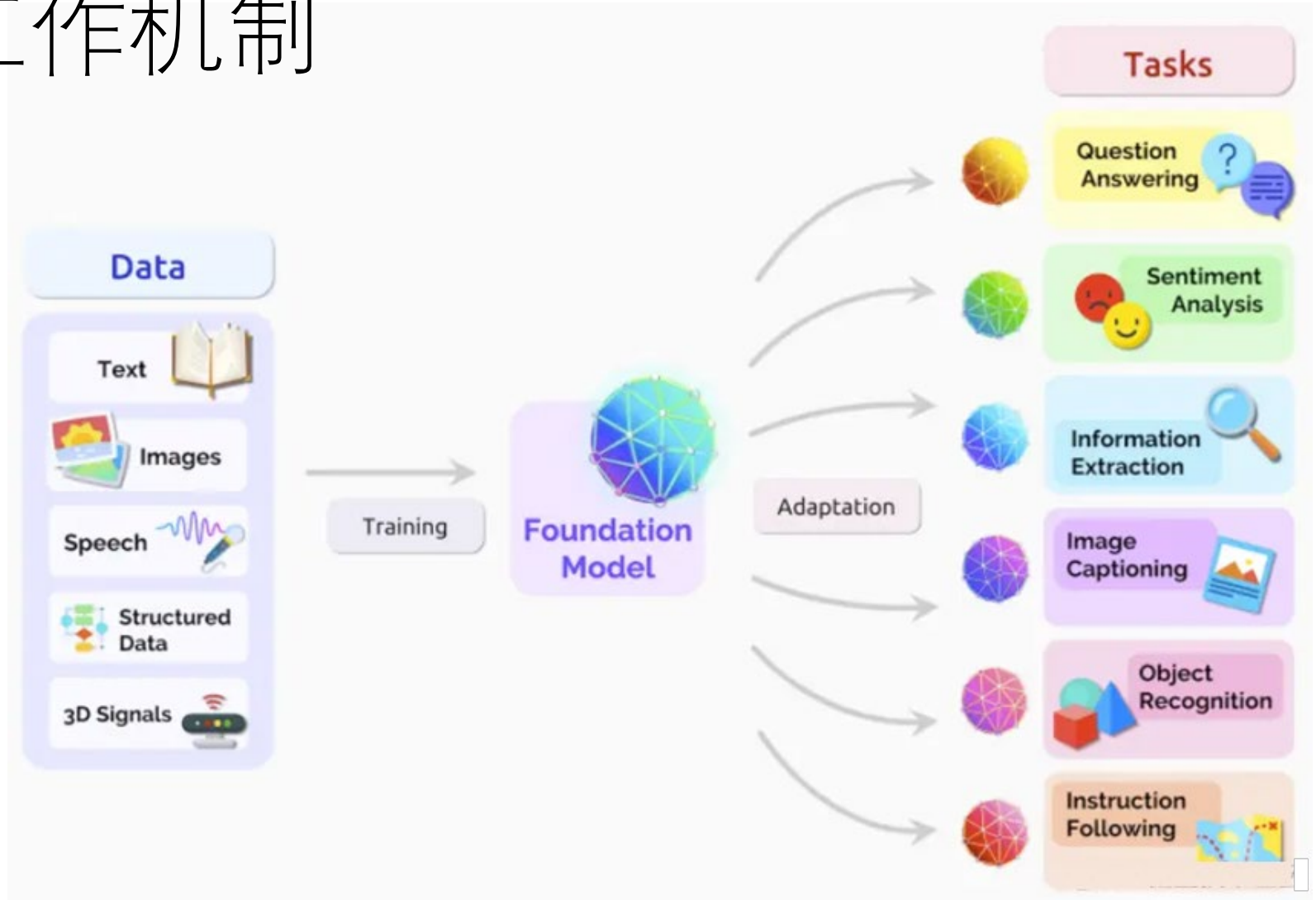
LLM特点

- 巨大的规模
- 预训练和微调
- 上下文感知
- 多语言支持
- 多模态支持
- 伦理和风险问题
- 高计算资源需求

LLM工作机制

- LLM通过预训练和微调两个阶段来完成自然语言处理任务
- 预训练阶段
  - 通过大规模语料库学习语言模型的普适知识
- 微调阶段
  - 通过在特定任务上的有标注数据上进行训练，使模型适应具体应用场景
  - 或者 RAG

LLM工作机制


Data
Text
Images
Speech
Structured Data
3D Signals
Training
Foundation Model
Adaptation
Tasks
Question Answering
Sentiment Analysis
Information Extraction
Image Captioning
Object Recognition
Instruction Following

(1) 预训练阶段

- 在预训练阶段，LLM使用大规模的文本数据来学习语言模型。
  - 这些文本数据通常是来自于互联网等大规模文本语料库，包含了各种类型的文本，如新闻、百科、社交媒体等。
  - LLM通过预训练阶段学习文本中的语言模式、语法结构和语义信息，从而掌握语言的普遍规律和知识。
- 在预训练过程中，LLM模型会经过多轮迭代，通过自监督学习的方式来训练模型。
  - 通常采用的方法是掩盖（masking）部分输入文本中的词语或句子，然后让模型预测被掩盖的部分。
  - 通过这种方式，模型能够学习到词语之间的关联性、语义信息以及上下文的语境。
- 因为LLM提前预训练了海量的语料数据，向GPT提问各行各业问题时，它大部分都可以比较正确输出。
  - 为什么是大部分？

(2) 微调阶段

- 在预训练完成后，LLM模型通常需要在特定任务上进行微调，以适应具体的应用场景。
- 微调阶段的目的是通过在特定任务上的有标注数据上进行训练，进一步提升模型的性能和泛化能力。
- 在微调阶段，LLM模型会被输入一些与任务相关的有标注数据，比如文本分类、命名实体识别、机器翻译等任务的训练数据。然后，通过在这些数据上进行训练，模型可以调整自身参数，使得模型在特定任务上表现更好。

LLM工作场景

- RAG + AIGC
- RAG场景
  - LLM可以与信息检索模型相结合，根据用户的查询检索相关的信息，并生成与检索结果相关的文本。
  - 常见的应用包括智能问答/对话系统、文档总结/摘要生成等。
- AIGC场景
  - LLM作为一个通用的生成模型，可以灵活应用于各种生成式任务中，为内容创作和生成提供强大支持。
  - 常见的应用包括自动化的内容生成和创作等。

RAG场景

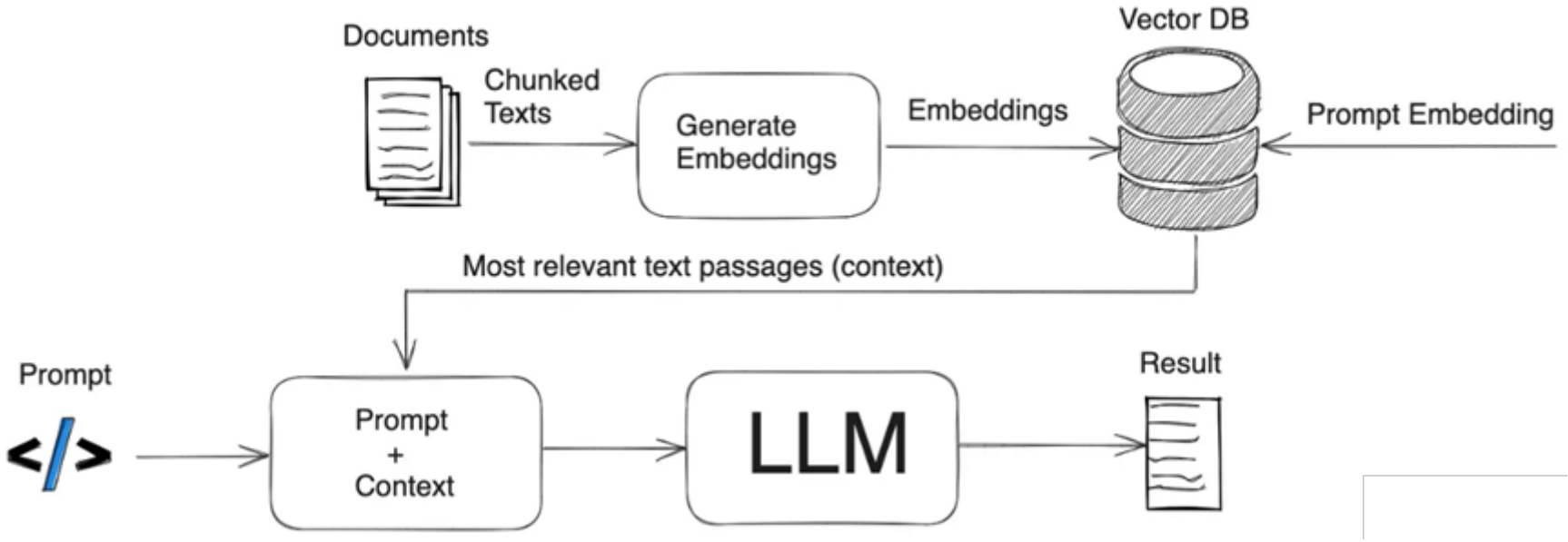
- LLM在哪些情况下会输出不正确呢？
  - 比如GPT训练的数据可能只截止到22年左右，如果向GPT问发生在24年的事情，肯定不知道。
- 原因：LLM存在：
  - 时效性不及时：LLM依赖训练的语料，语料的时效性就决定了LLM回答的正确性
  - 数据源不充足：LLM依赖训练的语料，如果缺少某一个领域内的语料，那么LLM也无法回答正确
- 如何解决？
  - 换句话说如何让LLM可以更准确的回答问题呢？

LLM 面临的主要问题

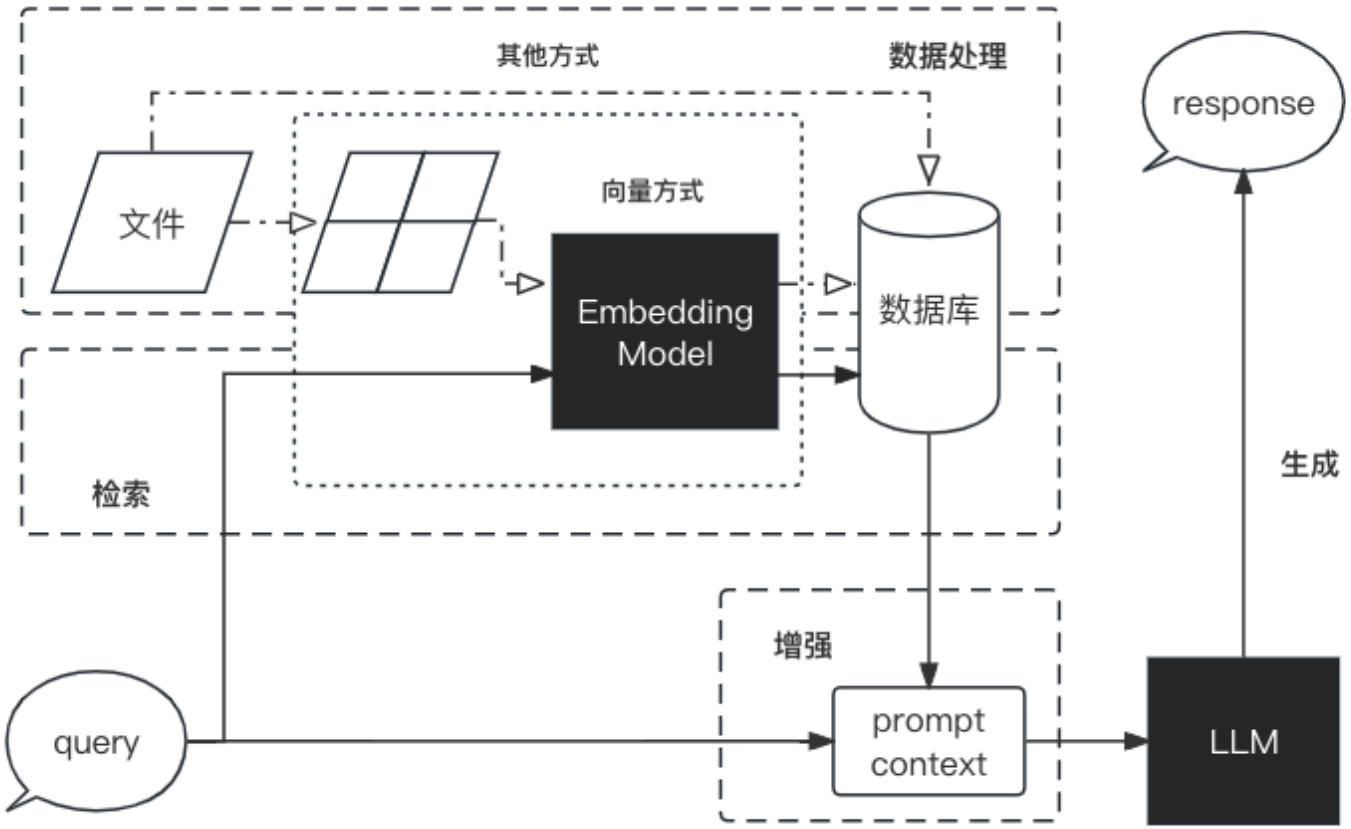
- 信息偏差/幻觉： LLM 会产生与客观事实不符的信息，导致用户接收到的信息不准确。
  - RAG 通过检索数据源，辅助模型生成过程，确保输出内容的精确性和可信度，减少信息偏差。
- 知识更新滞后性： LLM 基于静态的数据集训练，导致模型的知识更新滞后，无法及时反映最新的信息动态。
  - RAG 通过实时检索最新数据，保持内容的时效性，确保信息的持续更新和准确性。
- 内容不可追溯： LLM 生成的内容缺乏明确的信息来源，影响内容的可信度。
  - RAG 将生成内容与检索到的原始资料建立链接，增强了内容的可追溯性，从而提升了用户对生成内容的信任度。
- 领域专业知识能力欠缺： LLM 在处理特定领域的专业知识时，效果不理想，会影响到其在相关领域的回答质量。
  - RAG 通过检索特定领域的相关文档，为模型提供上下文信息，从而提升了在专业领域内的问题回答质量和深度。
- 推理能力限制： 面对复杂问题时，LLM 缺乏必要的推理能力，影响了对问题的理解和回答。
  - RAG 结合检索到的信息和模型的生成能力，通过提供额外的背景知识和数据支持，增强了模型的推理和理解能力。
- 应用场景适应性受限： LLM 需在多样化的应用场景中保持高效和准确，单一模型可能难以全面适应所有场景。
  - RAG 使得 LLM 能够通过检索对应应用场景数据的方式，灵活适应问答系统、推荐系统等多种应用场景。
- 长文本处理能力较弱： LLM 在理解和生成长篇内容时受限于有限的上下文窗口，且必须按顺序处理内容，输入越长，速度越慢。
  - RAG 通过检索和整合长文本信息，强化了模型对长上下文的理解和生成，有效突破了输入长度的限制，同时降低了调用成本，并提升了整体的处理效率。

什么是RAG

- **检索增强生成（RAG, Retrieval-Augmented Generation）**
  - 一种结合检索和生成的方法，**整合知识库中检索到的相关信息为基础，指导LLM生成答案**
  - 思路：由用户传递并提供语料，解决LLM语料时效性和数据源的问题



RAG 的工作流程


其他方式
数据处理
response
文件
向量方式
Embedding Model
数据库
检索
生成
增强
query
prompt context
LLM

RAG工作阶段

- 数据处理阶段
- 检索阶段（Retrieval）
- 增强阶段
- 生成阶段（Generation）
- 调整阶段（Adjustment）

数据处理阶段

- 对原始数据进行清洗和处理
- 将处理后的数据转化为检索模型可以使用的格式
- 将处理后的数据存储在对应的数据库中

检索阶段（Retrieval）

- 将用户的问题输入到检索系统中，从数据库中检索相关信息
- 使用检索式方法从大型知识库或文档集合中检索与当前任务相关的文本片段或文档
  - 语料通常是几十MB甚至更大的文件，不可能全部传递过去，
  - 使用检索，即：只传递相关性最高的数据，以减少资源的消耗。
- 使用向量存储的方式
  - 使用向量存储的方式提高相关性搜索的准确率，从知识库中检索到准确性更高的数据。
- 将检索到的内容作为LLM的输入
  - 检索到的文本通常包含与待生成内容相关的信息（比如问题回答等），作为LLM的输入，提供给LLM作为上下文环境
  - LLM在生成阶段生成更加相关和准确的内容
- 小结：
  - 检索阶段通常是由一个专门的检索系统来完成的，而不是由大型语言模型（LLM）直接执行。
  - 为保证检索到相关性最高的数据，一般情况下语料数据会使用向量存储，通过向量计算的方式，提高检索的相关性。

增强阶段

- 对检索到的信息进行处理和增强
- 以便生成模型可以更好地理解和使用

生成阶段（Generation）

- 将增强后的信息输入到生成模型中，生成模型根据这些信息生成答案
  - 在检索阶段检索到对应的文本后，传递给LLM
  - LLM获得文本后，利用这些文本作为上下文，进行生成式任务
- 生成阶段的输入是文本数据，即：传递给LLM模型的通常是文本数据，而非向量数据。

调整阶段（Adjustment）

- 需要有调整
- 在调整阶段，可以根据用户的反馈或后处理方法，对生成的文本进行进一步的修改或优化，以满足特定的需求。

RAG VS Finetune

微调: 通过在特定数据集上进一步训练大语言模型，来提升模型在特定任务上的表现

- 提升LLM效果的主流方法：RAG 和 微调（Finetune）

特征比较	RAG	微调
知识更新	直接更新检索知识库，无需重新训练。信息更新成本低，适合动态变化的数据。	通常需要重新训练来保持知识和数据的更新。更新成本高，适合静态数据。
外部知识	擅长利用外部资源，特别适合处理文档或其他结构化/非结构化数据库。	将外部知识学习到 LLM 内部。
数据处理	对数据的处理和操作要求极低。	依赖于构建高质量的数据集，有限的数据集可能无法显著提高性能。
模型定制	侧重于信息检索和融合外部知识，但可能无法充分定制模型行为或写作风格。	可以根据特定风格或术语调整 LLM 行为、写作风格或特定领域知识。
可解释性	可以追溯到具体的数据来源，有较好的可解释性和可追踪性。	黑盒子，可解释性相对较低。
计算资源	需要额外的资源来支持检索机制和数据库的维护。	依赖高质量的训练数据集和微调目标，对计算资源的要求较高。
推理延迟	增加了检索步骤的耗时	单纯 LLM 生成的耗时
降低幻觉	通过检索到的真实信息生成回答，降低了产生幻觉的概率。	模型学习特定领域的数据有助于减少幻觉，但面对未见过的输入时仍可能出现幻觉。
伦理隐私	检索和使用外部数据可能引发伦理和隐私方面的问题。	训练数据中的敏感信息需要妥善处理，以防泄露。

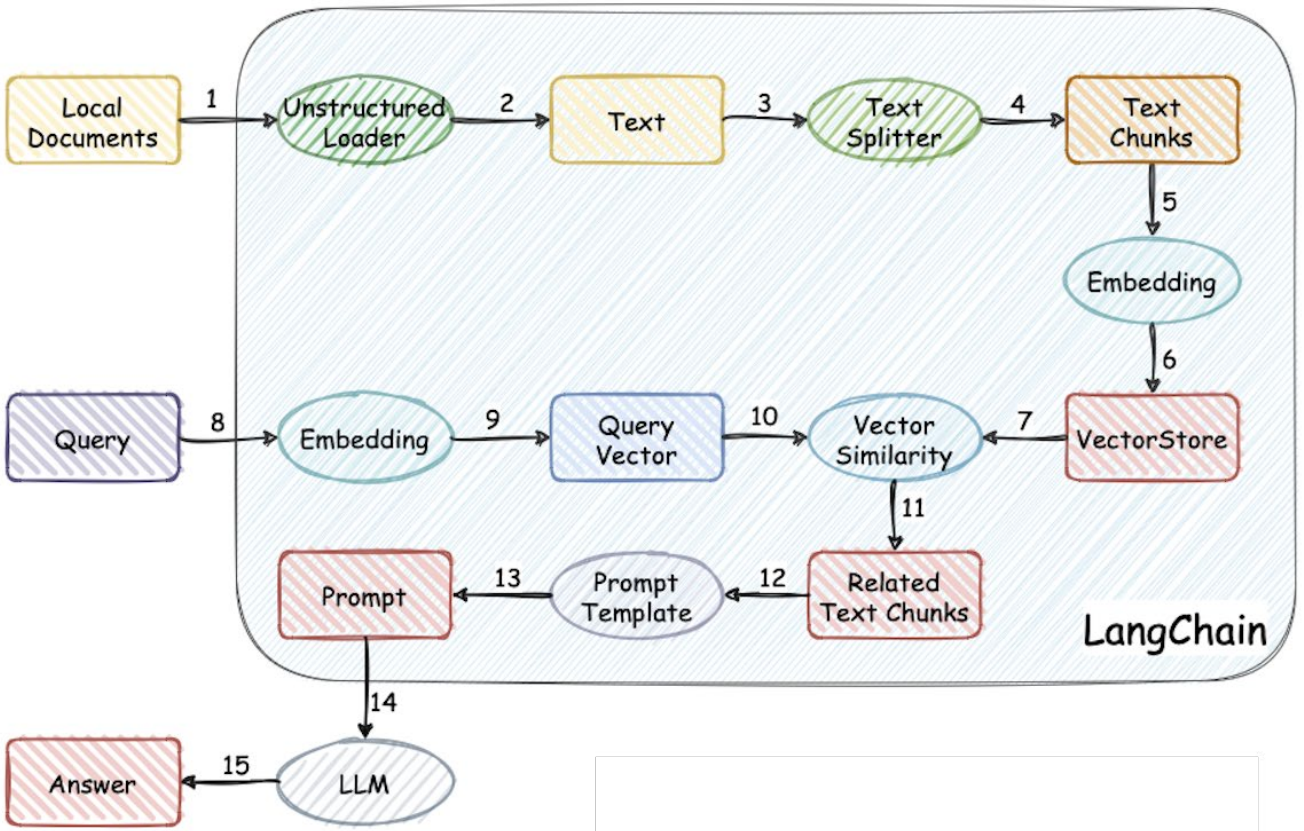
LangChain 构建 RAG 应用

椭圆形代表LangChain 的模块
如：数据收集模块或预处理模块。

矩形代表数据状态
如：原始数据或预处理后的数据。

箭头表示数据流的方向，从一个模块流向另一个模块。

每一步骤，LangChain 都可以提供对应的解决方案处理各种任务。



AIGC场景

- AIGC（Artificial Intelligence Generated Content，人工智能生成内容）是一个涵盖多种生成式任务的场景，包括文本生成、图片生成、代码生成、视频生成、语音生成等等。
- 在这些任务中，LLM（Large Language Model，大语言模型）可以发挥重要作用，解决各种问题，为各种应用场景提供支持。

总结