## 向量数据库

向量数据库在 LLM 领域的应用主要可以分为以下六类:

1. **管理私有数据和知识库**
2. **为大模型提供实时数据更新**
3. **实现大模型的个性化和增强**
4. **提供智能体的记忆**
5. **保存大模型的处理结果**
6. **构建更复杂的 AI 系统**

**最大内积搜索 (MIPS):**

以下几种常见的ANN算法都可以用于MIPS：

**「LSH」**（Locality-Sensitive Hashing）**」**它引入了一种哈希函数，使得相似的输入能以更高的概率映射到相同的桶中，其中桶的数量远小于输入的数量。

**「ANNOY**（Approximate Nearest Neighbors）**」**它的核心数据结构是随机投影树，实际是一组二叉树，其中每个非叶子节点表示一个将输入空间分成两半的超平面，每个叶子节点存储一个数据。二叉树是独立且随机构建的，因此在某种程度上，它模仿了哈希函数。ANNOY会在所有树中迭代地搜索最接近查询的那一半，然后不断聚合结果。这个想法与 KD 树非常相关，但更具可扩展性。

**「HNSW**（Hierarchical Navigable Small World）**」**它受到小世界网络思想的启发，其中大多数节点可以在很少的步骤内被任何其他节点到触达；例如社交网络的“六度分隔”理论。HNSW构建这些小世界图的层次结构，其中底层结构包含实际数据。中间的层创建快捷方式以加快搜索速度。执行搜索时，HNSW从顶层的随机节点开始，导航至目标。当它无法靠近时，它会向下移动到下一层，直到到达最底层。上层中的每个移动都可能覆盖数据空间中的很长一段距离，而下层中的每个移动都可以细化搜索质量。

**「**FAISS（facebook AI Similarity Search）**」**它运行的假设是：高维空间中节点之间的距离服从高斯分布，因此这些数据点之间存在着聚类点。faiss通过将向量空间划分为簇，然后在簇内使用用向量量化。faiss首先使用粗粒度量化方法来查找候选簇，然后进一步使用更精细的量化方法来查找每个簇。

**「**ScaNN（Scalable Nearest Neighbors）**」**的主要创新在于各向异性向量量化。它将数据点量化为一个向量，使得它们的内积与原始距离尽可能相似，而不是选择最接近的量化质心点。

### Pincone

Pincone 可以算是当前最火的商业向量数据库产品了，它是OpenAI官方首推的向量数据库。目标是为大语言模型带来长期记忆功能。

它的逻辑如下：

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

当用户输入文本内容后，通过嵌入模型计算出文本对应的向量数据。将向量数据带入向量数据库利用距离函数计算出高维空间中距离相似的内容，也就是我们常说的语义搜索。最后将结果返回给应用层处理后反馈给用户。

Pincone有以下几个特点

* 高性能搜索

Pincone只提供了网络部署，通过高性能服务器搭配自研索引查找算法，可以快速的计算内容之间的相似性，即便有数十亿数量级的内容利用Pincone也能低延迟响应，给用户提供最好的体验。

* 提供容易上手API

提供Python、NodeJS、RestFulAPI，轻松对接数据管理。

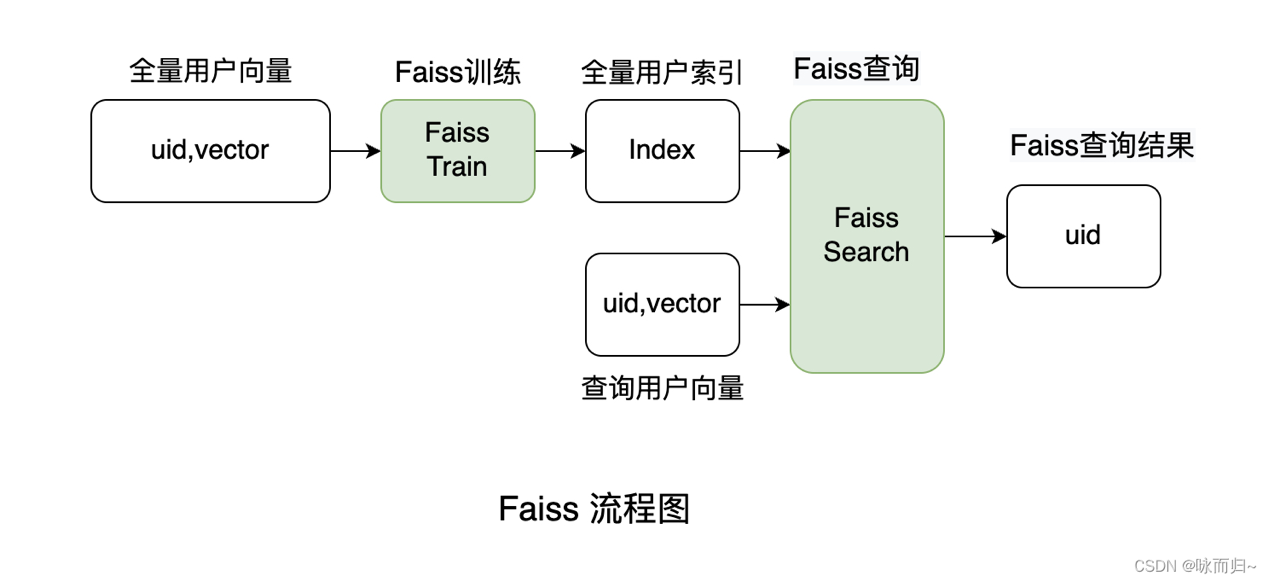
* 全功能管理

通过独特的索引生成算法，对于添加、编辑或删除的数据能进行索引实时更新，立即生效。通过提供的WebUI终端实时就能看到数据变动。另外将矢量数据和元数据进行绑定结合，快速找到对应内容。

* 轻松扩容

完全托管在 AWS 或 Google 的高性能计算平台上，自动扩容，无需担心架构或算法，也无需安排专业运维团队，这一切都自动帮你搞定。一开始提供免费容量，只有超过免费容量后按使用量收费，可根据产品运营状态随时调整成本。

### Faiss



图示

描述已自动生成

图示

描述已自动生成

* **大模型**：

pre\_training 🡪 Fine\_Tuning 🡪 Parameter-Efficient Fine-Tuing，PEFT（Prompt\_Tuning、Adapter-Tuing、LoRA）

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

* **知识问答**：

**LLM + 向量数据库 + 提示词**

Transformer的上下文窗口是它的工作内存。可以将与任务相关的任何信息加载到工作内存中，那么该模型将运行得非常好，因为它可以立即访问所有内存。

可以通过下图了解向量搜索实现GPT Context的大致原理：

图示

描述已自动生成

1. 读取待训练的文档数据并进行向量化，之后存入向量数据库
2. 通过query的向量化结果与数据库向量进行相似度匹配，并返回关联文本结果
3. 结合返回的关联文本和query来构建上下文生成prompt

图示

描述已自动生成

langchain引入了一个叫Agents代理的模块。代理可以将很多外部操作融入到业务链条当中。**所谓代理就是代为处理**。

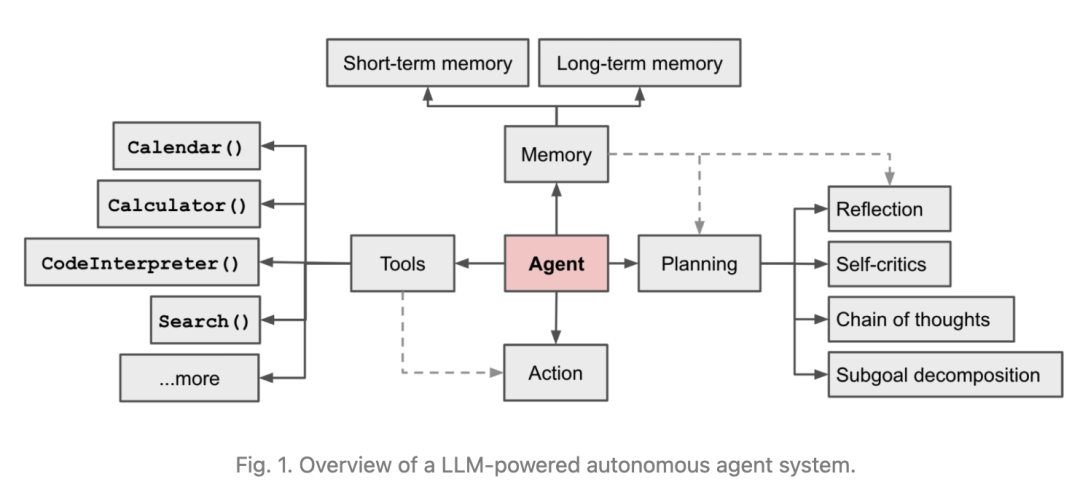
图示

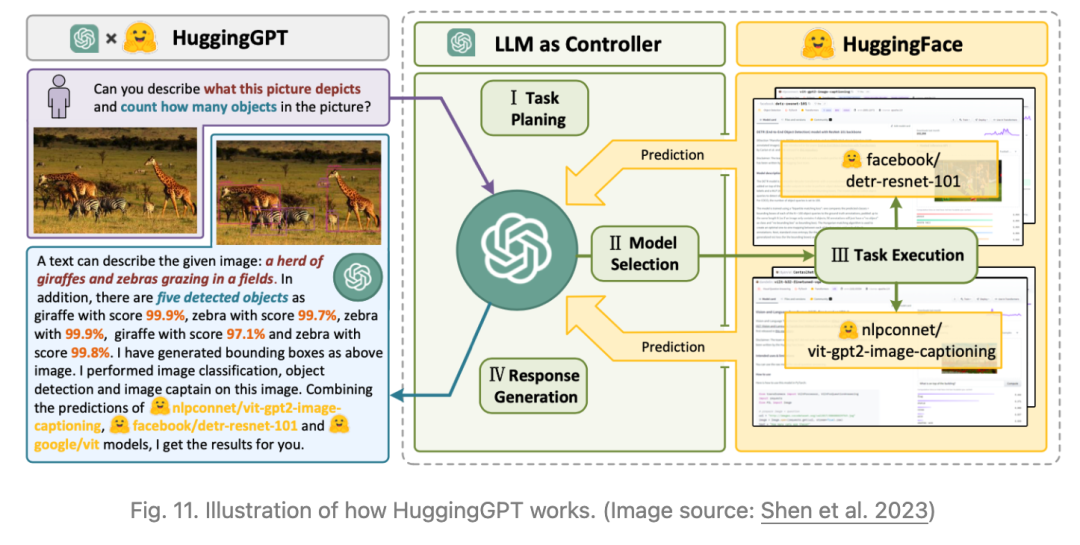
描述已自动生成

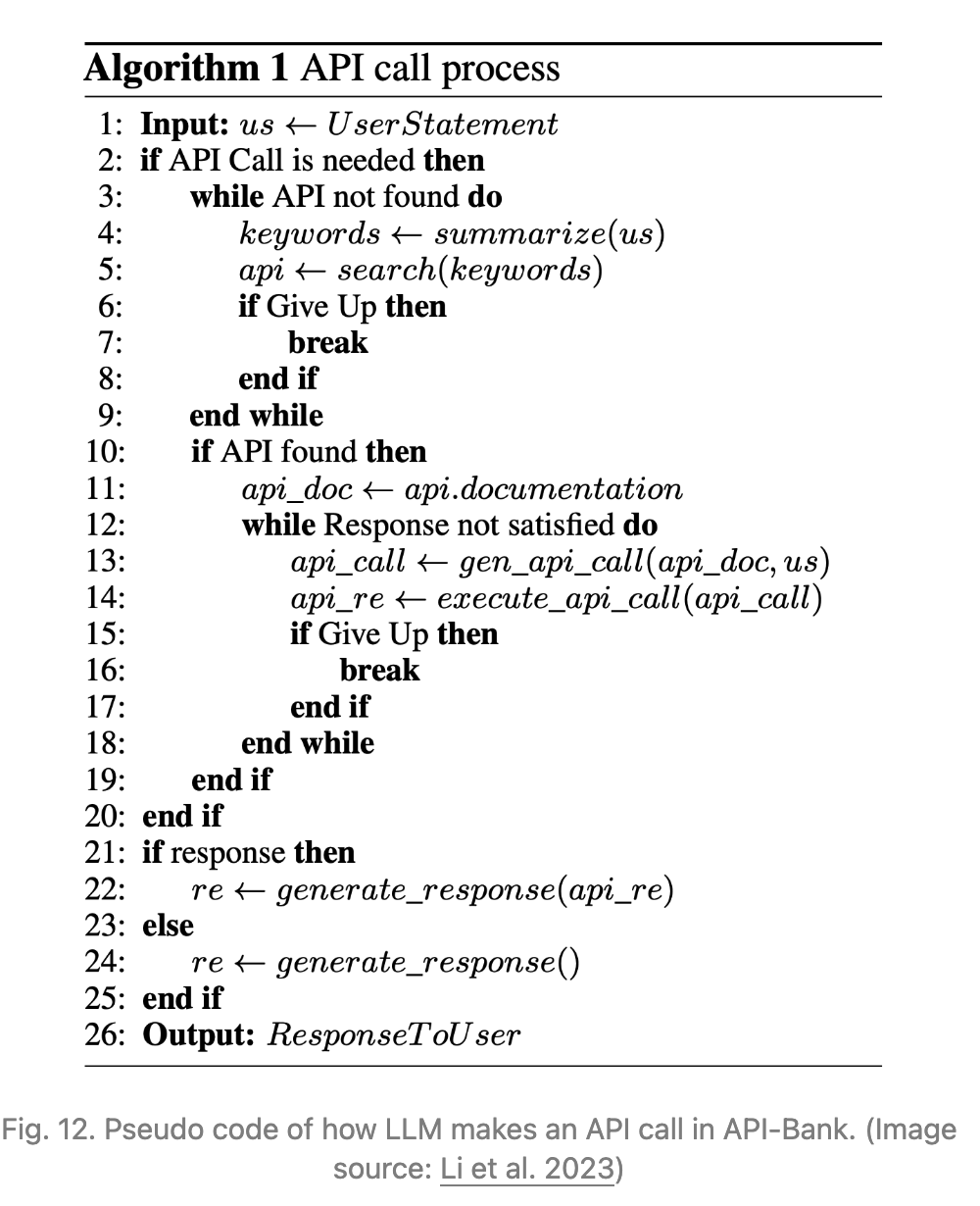
Agent = LLM（大型语言模型）+ 记忆 + 规划技能 + 工具使用

在以LLM驱动的自主代理系统中，LLM作为代理的大脑，辅以几个关键组件：

* **规划**
  + 子目标和分解：代理将大型任务分解为较小，可管理的子目标，从而有效地处理复杂的任务。
  + 反思和改进：代理可以对过去的行动进行自我批评和自我反思，从错误中学习并改进未来的步骤，从而提高最终结果的质量
* **记忆**
  + 我将所有的上下文学习（参考Prompt Engineering）都看成是利用模型的短期记忆来学习。
  + 长期记忆：长期记忆为代理提供了长期存储和召回（无限）信息的能力，它们通常通过利用外部的向量存储和快速检索来存储和召回（无限）信息。
* **使用工具**
  + 代理通过学会调用外部API来获取模型权重（通常在预训练后很难修改）中缺少的额外信息，包括当前信息，代码执行能力，访问专有信息源等。







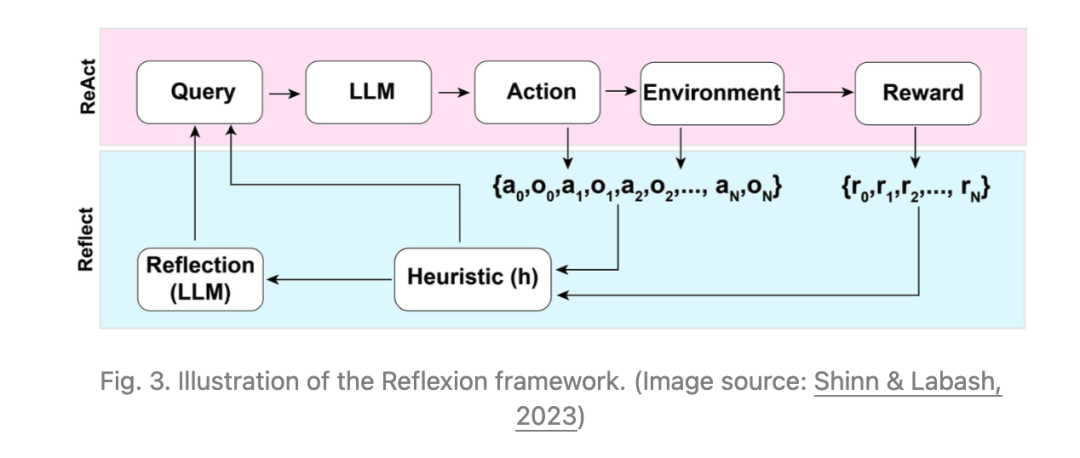
在API-Bank工作流程中，LLM需要做出一些决策，而在每个步骤中，我们都可以评估该决策的准确性。这些决策包括：

1. 判断是否需要进行API调用。
2. 确定要调用的正确API：如果不够好，LLM需要迭代修改API输入（例如，为搜索引擎API决定搜索关键字）。
3. 根据API结果进行响应：如果结果不满意，模型可以选择进行改进并再次调用。

该基准评估了代理程序在三个层面上的工具使用能力：

* Level-1评估调用API的能力。在给定API的描述的情况下，模型需要确定是否调用给定的API，正确调用它，并对API返回做出适当的响应。
* Level-2检查检索API的能力。模型需要搜索可能解决用户需求的API，并通过阅读文档学习如何使用它们。
* Level-3评估计划API超越检索和调用的能力。在用户请求不明确的情况下（例如，安排团队会议，为旅行预订航班/酒店/餐厅），模型可能需要进行多个API调用来解决问题。

**「反思」**是一个框架，它为代理提供动态记忆和自我反思的能力，以提高它的推理技能。



AutoGPT 是一个我认为最近炒得沸沸扬扬的项目，但我仍然觉得它有点鼓舞人心。它是一个允许 LLM 保留任务列表并继续递归分解任务的项目。AutoGPT有点像让我们的模型系统思考

两种方式的差别：

* OpenAIEmbeddings：
  + 使用简单，并且效果比较好；
  + 会消耗openai的token，特别是大段文本时，消耗的token还不少，如果知识库是比较固定的，可以考虑将每次生成的embedding做持久化，这样就不需要再调用openai了，可以大大节约token的消耗；
  + 可能会有数据泄露的风险，如果是一些高度私密的数据，不建议直接调用。
* HuggingFaceEmbeddings：
  + 可以在[HuggingFace](https://link.juejin.cn?target=https%3A%2F%2Fhuggingface.co%2F" \o "https://huggingface.co/" \t "_blank)上面选择各种[sentence-similarity](https://link.juejin.cn?target=https%3A%2F%2Fhuggingface.co%2Fmodels%3Fpipeline_tag%3Dsentence-similarity%26sort%3Ddownloads" \o "https://huggingface.co/models?pipeline_tag=sentence-similarity&sort=downloads" \t "_blank)模型来进行实验，数据都是在本机上进行计算
  + 需要一定的硬件支持，最好是有GPU支持，不然生成数据可能会非常慢
  + 生成的向量效果可能不是很好，并且HuggingFace上的中文向量模型不是很多。

# 大模型参数高效微调(Parameter-Efficient Fine-Tuning, PEFT)

随着模型变得越来越大，在消费级硬件上对模型进行全部参数的微调变得不可行。此外，为每个下游任务独立存储和部署微调模型变得非常昂贵，因为微调模型与原始预训练模型的大小相同。参数高效微调(PEFT) 方法旨在解决这两个问题！PEFT 方法仅微调少量或额外的模型参数，固定大部分预训练参数，大大降低了计算和存储成本，同时最先进的 PEFT 技术也能实现了与全量微调相当的性能。

HuggingFace开源的一个高效微调大模型的库: PEFT。目前大致可分为三大类，不同的方法对PLM的不同部分进行下游任务的适配:

* Prefix-Tuing/Prompt-Tuing: 对模型的输入或者隐藏层添加k个额外可训练的前缀tokens，只训练这些前缀参数；
* Adapter-Tuing：将较小的神经网络层或者模块插入预训练模型的每一层，这些新插入的神经网络模块被称为adapter(适配器)，下游任务微调时也只是训练这些适配器参数；
* LoRA：通过学习小参数的低秩矩阵来近似模型的权重矩阵W的参数更新，训练时只优化低秩矩阵参数；

1. Prefix-Tuning: Optimizing Continuous Prompts for Generation

[图形用户界面, 文本, 应用程序

描述已自动生成](https://aclanthology.org/2021.acl-long.353.pdf)

文本, 信件

描述已自动生成

图片包含 游戏机, 家具, 桌子, 凳子

描述已自动生成

卡通人物

中度可信度描述已自动生成

1. P-Tuning：GPT Understands, Too

P-Tuning 的方法思路与 Prefix-Tuning 很相近，**P-Tuning** 利用少量连续的 embedding 参数作为 prompt 使 GPT 更好的应用于 **NLU** 任务（For instance, in the pre-training, x refers to the unmasked tokens while y refers to the [MASK] ones; and in the sentence classification, x refers to the sentence tokens while y often refers to the [CLS].），而 **Prefix-Tuning** 是针对 **NLG** 任务设计，同时，**P-Tuning 只在 embedding 层增加参数，而 Prefix-Tuning 在每一层都添加可训练参数**。

[图示

描述已自动生成](https://arxiv.org/pdf/2103.10385.pdf)

文本

描述已自动生成

图片包含 形状

描述已自动生成

Prompt Encoder是由一个 Bi-LSTM 和一个两层的前馈神经网络组成，对 prompt embedding 序列进行编码后再传入到语言模型中。

**P-Tuning v2**: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks的思路其实和 Prefix-Tuning 相似，在**模型的每一层都应用连续的 prompts**并对 prompts 参数进行更新优化。同时，该方法是**针对 NLU 任务优化和适配**的。

[图形用户界面, 应用程序

中度可信度描述已自动生成](https://arxiv.org/pdf/2110.07602.pdf)

1. The Power of Scale for Parameter-Efficient Prompt Tuning

Prompt Tuning 方式可以看做是 Prefix Tuning 的简化，**固定整个预训练模型参数，只允许将每个下游任务的额外**�**个可更新的 tokens 前置到输入文本中，也没有使用额外的编码层或任务特定的输出层**。如下图所示，在模型大小增加到一定规模时，仅仅使用 Prompt Tuning 就足以达到 Fine Tuning 的性能。

图表, 折线图

描述已自动生成

[文本

描述已自动生成](https://arxiv.org/pdf/2104.08691.pdf)

1. Adapter Tuning

与 Prefix Tuning 和 Prompt Tuning 这类在输入前可训练添加 prompt embedding 参数来以少量参数适配下游任务，**Adapter Tuning 则是在预训练模型内部的网络层之间添加新的网络层或模块来适配下游任务**。假设预训练模型函数表示为  ，对于 Adapter Tuning ，添加适配器之后模型函数更新为：  ， w是预训练模型的参数，是新添加的适配器的参数，在训练过程中，w 被固定，只有被更新。 || ≪ |w| ，这使得不同下游任务只需要添加少量可训练的参数即可，节省计算和存储开销，同时共享大规模预训练模型。

Adapter 主要包括 Series Adapter（串行） 和 Parallel Adapter（并行）：

* Series Adapter的适配器结构和与 Transformer 的集成如下图（a）所示。**适配器模块被添加到每个 Transformer 层两次：多头注意力映射之后和两层前馈神经网络之后**。适配器是一个 bottleneck（瓶颈）结构的模块，由一个两层的前馈神经网络（由向下投影矩阵、非线性函数和向上投影矩阵构成）和一个输出输出之间的残差连接组成。
* Parallel Adapter如下图（b）所示。**将适配器模块与每层 Transformer 的多头注意力和前馈层并行计算集成**。

图示

描述已自动生成

1. LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

[图示

描述已自动生成](https://arxiv.org/pdf/2106.09685.pdf)

文本, 信件

描述已自动生成

报纸上的文字

描述已自动生成

FP32 是单精度浮点数，用8bit 表示指数，23bit 表示小数；

FP16半精度浮点数，用5bit 表示指数，10bit 表示小数；

BF16是对FP32单精度浮点数截断数据，即用8bit 表示指数，7bit 表示小数。

在数据表示范围上，FP32和BF16 表示的整数范围是一样的，小数部分表示不一样，存在舍入误差；FP32和FP16 表示的数据范围不一样，在大数据计算中，FP16存在溢出风险。

日程表

描述已自动生成

图片包含 日程表

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

相同prompt相同的模型，每次生成的结果不同：和top\_k、top\_p以及temperature有关。

## Top-k & Top-p

选择输出标记的方法是使用语言模型生成文本的一个关键概念。有几种方法（也称为解码策略）用于选择输出token，其中两种主要方法是 top-k 采样和 top-p 采样。

让我们看一下示例，模型的输入是这个prompt文本 *The name of that country is the*:

图示

描述已自动生成

Example output of a generation language model.

在这种情况下，输出标记 *United* 是在处理的最后一步选择的——在语言模型处理输入并计算其词汇表中每个标记的似然分数之后。 该分数表示它将成为句子中下一个标记的可能性（基于训练模型的所有文本）。

图示

描述已自动生成

The model calculates a likelihood for each token in its vocabulary. The decoding strategy then picks one as the output.

### 1.挑出top token：贪心解码

您可以在此示例中看到，我们选择了可能性最高的标记“United”。

图示

描述已自动生成

Always picking the highest scoring token is called "Greedy Decoding". It's useful but has some drawbacks.

贪心解码是一种合理的策略，但也有一些缺点，例如输出带有重复文本循环。想一想智能手机的自动建议中的建议。当您不断选择最高建议的单词时，它可能会变成重复的句子。

### 2.从top tokens中挑选：top-k

另一种常用的策略是从前 3 个tokens的候选名单中抽样。这种方法允许其他高分tokens有机会被选中。这种采样引入的随机性有助于在很多情况下生成的质量。

图示

描述已自动生成

Adding some randomness helps make output text more natural. In top-3 decoding, we first shortlist three tokens then sample one of them considering their likelihood scores.

更广泛地说，选择前三个tokens意味着将 top-k 参数设置为 3。更改 top-k 参数设置模型在输出每个token时从中抽样的候选列表的大小。将 top-k 设置为 1 可以进行贪心解码。

图片包含 图表

描述已自动生成

Adjusting to the top-k setting.

### 3.从概率加起来为15%的top tokens中挑选：top-p

选择最佳 top-k 值的困难为流行的解码策略打开了大门，该策略动态设置tokens候选列表的大小。这种称为*Nucleus Sampling 的*方法将可能性之和不超过特定值的top tokens列入候选名单。top-p 值为 0.15 的示例可能如下所示：

图形用户界面, 应用程序, Teams

描述已自动生成

In top-p, the size of the shortlist is dynamically selected based on the sum of likelihood scores reaching some threshold.

Top-p 通常设置为较高的值（如 0.75），目的是限制可能被采样的低概率 token 的长尾。我们可以同时使用 top-k 和 top-p。如果 *k* 和 *p* 都启用，则 *p* 在 *k* 之后起作用。

## Temperature

从生成模型中抽样包含随机性，因此每次点击“生成”时，相同的提示可能会产生不同的输出。温度是用于调整随机程度的数字。

### 采样时如何选择温度

较低的温度意味着较少的随机性；温度为 0 将始终产生相同的输出。执行具有“正确”答案的任务（如问题回答或总结）时，较低的温度（小于 1）更合适。如果模型开始自我重复，则表明温度过低。

高温意味着更多的随机性，这可以帮助模型给出更有创意的输出。如果模型开始偏离主题或给出无意义的输出，则表明温度过高。

图片包含 图示

描述已自动生成

Adjusting the temperature setting

可以针对不同的问题调整温度，但大多数人会发现温度 1 是一个很好的起点。

随着序列变长，模型自然会对其预测更有信心，因此您可以在不偏离主题的情况下为长提示提高温度。相反，在短提示上使用高温会导致输出非常不稳定。

## Likelihood

我们的模型通过阅读从互联网上抓取的文本来学习建模语言。 给定一个句子，例如 *I like to bake cookies*，要求模型重复预测下一个标记 *[?]* 是什么：

I [?]

I like [?]

I like to [?]

I like to bake [?]

I like to bake cookies

该模型了解到单词 *to* 很可能跟在英语中的 *like* 之后，而 *cookies* 很可能跟在 *bake* 之后。

### Intuition

token的可能性可以被认为是一个数字（通常在 -15 和 0 之间），它量化模型对句子中使用该token的**意外程度**。 如果token的可能性很低，则意味着模型不希望使用该token。 相反，如果token具有很高的可能性，则该模型有信心使用它。 例如，使用我们的 Large 模型，句子“*I like to*”中出现“*to*”的可能性大约为 -1.5。 这是相当高的，这意味着模型相当有信心*I like*的标记后面会跟着标记 *to*.. 同样，从句子*I like to bake cookies* 中出现 *cookies* 的可能性大约为 -3.5，略低于 前面的例子（这很直观：brownies 或 cake也是合理的选择），但仍然很高。 但是，如果我们将句子更改为 *I like to bake chairs* ，那么token *chairs*的可能性就会大大降低，大约为 -14。 这意味着该模型对其在句子中的使用感到非常意外。

图片包含 图形用户界面

描述已自动生成

Likelihood of a token

## Number of Generations

当您调用 Generate endpoint 时，您可以选择在一次调用中生成多个结果。 这是通过设置 num\_generations 参数来完成的。

图示

描述已自动生成

Generating multiple outputs in a single API call.

模型的输出将根据您指定的生成设置而有所不同，例如温度、top-k 和 top-p。

每一个生成结果都有其一组似然值，其中包括：

* 每个生成的 token 的可能性
* 所有生成的标记的平均可能性。

图示, 地图

描述已自动生成

在文本生成推理方面，HuggingFace 提供了如下 GPU 建议，A10G（24G），A100（80G）：

* 对于 7B 模型，建议选择 "GPU [medium] - 1x Nvidia A10G"；
* 对于 13B 模型，建议选择 "GPU [xlarge] - 1x Nvidia A100"；
* 对于 70B 模型，建议选择 "GPU [xxxlarge] - 8x Nvidia A100"。