ArkFS- AI File System

OSH-2024

杨柄权、刘明乐、李岱峰、常圣

摘要

近年来，大语言模型发展迅猛，已经在许多领域取得了广泛的应用。AIOS 的出现为科研人员展示了大语言模型嵌入操作系统的广阔的发展前景，同时也指明了在此基础上的改进方向。本小组拟在 AIOS 的基础上对文件系统做一些优化，将本地文件系统升级为大语言模型的本地数据库，以求实现更好的文件管理、更快的索引以及更便于使用的数据备份与恢复。

目录

1 研究背景

大语言模型在近几年的发展中取得了巨大的突破。目前已有许多研究人员将大模型应用于操作系统。今年 3 月，AIOS[1] 为我们展示了大语言模型接入操作系统的重要性和优越性。文件系统是操作系统的重要组成部分，也有许多在探索其优化方向以及改进方法。本小组调研了三种大语言模型、AIOS 以及其文件系统目前存在的部分问题。

1.1 大语言模型

1.1.1 Transformers

Transformer 模型是一种仅使用注意力机制在输入和输出之间建立依赖关系的神经网络架构 (如图 1 所示)，它由 Google Brain 在 2017 年提出。该模型的核心思想是注意力机制，可以使模型在生成输出时，根据输入序列中不同 token 的重要性而分配不同的权重[2]。

Transformers 库是一个基于 TensorFlow 和 Pytorch 的 Python 库，用于训练和使用 Transformer 模型。该库由美国人工智能公司 Hugging Face 开发，提供了各种预训练模型

和工具，用于完成各种 NLP 任务。

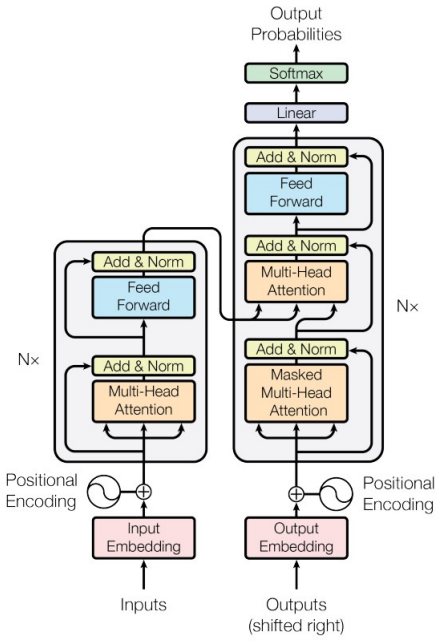

图 1: The Transformer-model architecture[2]

近年来，人们开始探索将 Transformers 的模型应用于文件系统中的一些应用场景：

（1）文件分类。Transformers 模型可以用于自动对文件进行分类。通过对文件内容的语义分析，Transformers 模型可以将文件归入到相应的类别中，这可以提高文件管理的效率。

（2）文件检索。Transformers 模型可以用于提高文件检索的准确性。传统的文件检索依赖关键字匹配，而 Transformers 模型可以做到理解文件内容的语义，从而进一步提高文件检索的准确性。

（3）文件摘要。Transformers 模型可以用于自动生成文件摘要。通过对大篇幅文件内容的语义分析，Transformers 模型可以有效提取关键信息，生成简短摘要，帮助用户更快了解文本内容，提高阅读效率。

（4）文件翻译。Transformers 模型可以用于提高文本翻译的准确性。通过对文本语义的理解，Transformers 模型可以生成更准确的多语言翻译结果，使文件共享与传递更加便捷。

（5）文件安全。Transformers 模型可以用于保护文件安全。通过对文件文本语义的理解，根据文本内容，设立不同安全等级，并提醒用户文件相关敏感信息，保护文件安全。

基于上述种种应用场景，在 transformers 库中可找到与拥有所需相似功能的模型。比如，BERT 模型[3] 是一种双向编码器模型，能够学习上下文文义，并且仅需微调就可完成专门的 NLP 任务。同时，随着 BERT 模型的不断发展，衍生出了 RoBERTa[4]、XLNet[5]、DistilBERT[6]、ALBERT[7] 等模型。

Transformers 模型在文件系统上应用前景广阔，但也存在了一些缺陷：

（1）模型规模。Transformers 模型规模通常较大，在资源有限的设备上部署可能较为困难。

（2）模型训练。Transformers 模型的训练需要大量的数据以及算力，训练成本高昂。

（3）模型解释。Transformers 模型的黑盒性质使得其内部难以解释和理解，使得其在一些安全敏感的场景难以运用，很难解释 Transformers 模型是如何检测到敏感信息和恶意文件的。

（4）模型兼容性。不同框架训练的模型可能无法直接部署到其他框架中。

综上，Transformers 模型作为近年来新兴的自然语言处理模型，虽然仍有一定缺陷，但是若能够合理部署，对文件系统是大有裨益的。

1.1.2 AutoGPT

AutoGPT 是一个由 Significant Gravitas 在 2023 年 3 月 16 日提出的实验性项目，旨在探索 GPT-4 模型在没有人为干预的情况下，

能否在人类商业世界中自主生存和执行任务。该项目的核心在于不断向 GPT-4 发送请求，并执行其做出的商业决策，以此来观察和评估 GPT-4 的表现。

经过调研，本人总结出 AutoGPT 的几项关键特性如下：[8]

（1）自动编程。AutoGPT 能够协助完成编程任务，如生成代码片段、修复 bug，甚至帮助创建整个项目。这对于提高开发效率和降低人力成本具有重要意义。

（2）内容创作。在内容创作方面，AutoGPT 可以撰写文章、生成创意文案、编写故事或诗歌。这种自动化的内容生成能力可以帮助快速产出大量文本资料，对于媒体行业和市场营销等领域尤其有价值。

（3）客户服务。AutoGPT 可以作为智能客服的一部分，通过自然语言处理和生成能力，提供客户咨询和支持服务。这可以提高客户满意度并减轻人工客服的工作负担。

（4）知识问答。作为一个认知智能模型，AutoGPT 能够回答各种知识性问题，为用户提供信息查询和学习的帮助。

（5）个性化推荐。例如，一个基于 AutoGPT 的程序可以搜索互联网并生成适合特定活动或假期的独特食谱，这样的个性化推荐能够满足用户的特定需求。

（6）教育和研究。AutoGPT 可以作为教育工具，帮助学生学习编程、数学等科目，或者作为研究人员的工具，帮助他们进行科学实验和数据分析。

（7）自动决策制定。结合外部资源和自主决策能力，AutoGPT 可以在没有人工干预的情况下执行任务，并通过循环评估策略实时评估目标达成程度，以决定任务是否完成。[9]

此外，本人还总结出 AutoGPT 的几项可能的应用领域如下：

（1）人机交互。它可以作为用户与其他系统之间的中介，提升用户体验。

（2）教育和培训。在教育领域，它可以作为一个认知智能模型，提供定制化的学习材料和辅导。

（3）辅助研究。在科学研究中，它可以协助文献搜索、数据分析等任务。

（4）创意生成。AutoGPT 还可以在艺术和设计领域发挥作用，激发创意思考。

总的来说，AutoGPT 的核心逻辑是一个 Prompt Loop。它会基于一定策略自动组装 Command Prompt，这些 Prompt 会包含用户输入的信息，如姓名、任务要求和任务目标等，之后将组装的 Command Prompt 发送给 GPT-4，生成执行下一步任务所需要的指令 Command，再执行 Command 以完成任务，同时生成能够实现下一步任务的 Prompt，不断地循环，以此来实现对复杂任务的自主完成。

1.1.3 Lang Chain

Lang Chain 是一个开源框架，旨在促进由大型语言模型驱动的应用程序的开发。它是一套全面的工具和组件，使构建上下文感知应用程序变得更加容易，使这些应用程序可以使用自然语言处理进行推理和交互。

下面是我找到 Lang Chain 的一些关键特性:

（1）上下文感知。LangChain 允许应用程序将语言模型连接到各种上下文源，例如提示指令、示例或内容。

（2）推理。该框架支持依赖语言模型来推理如何根据提供的上下文或采取的操作进行回答的应用程序。

（3）组件。它提供了用于处理语言模型的可组合工具和集成，使它们模块化且易于使用。

(4) 现成链。Lang Chain 包括用于完成高级任务的内置组件组合，这简化了入门过程。

(5) Lang Chain 表达式语言 (LCEL)。这是一种组合链的声明性方式，支持在不更改代码的情况下将原型投入生产。

(6) 开发工具。Lang Chain 支持 Python 和 JavaScript，提供模板，以及像测试平台，用于调试、测试、评估和监控链。

(7) 易于部署。使用 LangServe，开发人员可以将任何链转换为 API，从而简化应用程序的部署。

总而言之，langchain 可以作为一个程序接口，给普通应用程序接上" 挂件"，实现更自动化的功能。

我查找了一些 langchain 的应用实例，例如分析 PDF 内容并提取关键信息[10]。这篇论文仔细地讲述了如何应用 langchain，指出了 langchain 的功能优点：

(1) 查询系统：使用 LangChain，用户可以根据自己的输入查询 PDF 文档。

(2) 索引和检索技术：LangChain 使用有效的索引和检索技术来提高搜索体验。

(3) 可移动过滤器：用户可以根据需要调整过滤器，以获取更准确的搜索结果。

(4) 简单的搜索界面：LangChain 提供了一个简单易用的搜索界面。用户上传 PDF 文件，然后在 Web 应用程序中提出查询。

这篇论文实现了 LangChain 和 Streamlit 协同工作，使用户能够从 PDF 文档中检索准确的信息，对我们的工作很有指导意义。

1.2 AIOS - AI embeded Operating System

AIOS 是一种 LLM 代理操作系统，将大语言模型嵌入操作系统(OS)作为 OS 的大脑，实现了 "有灵魂" 的操作系统。AIOS 旨在解决 LLM 代理集成和部署过程中的多种挑战，如代理请求的次优调度和资源分配、在代理与 LLM 交互中维持上下文的困难，以及与具有不同能力和专长的异构代理集成的复杂性。这些问题通常导致资源利用不佳和瓶颈现象。AIOS 通过优化资源分配、支持代理间的上下文切换、实现代理的并发执行、为代理提供工具服务、以及维护代理的访问控制来解决这些挑战。

AIOS 提出了一个 LLM 专用的内核设计，该设计区分了与 LLM 相关的任务和非 LLM 任务之间可能出现的冲突。通过这种区分，LLM 内核旨在增强对 LLM 相关活动的管理和协调。AIOS 内核中包括了一系列专门解决 LLM 操作相关不同功能的模块：

代理调度器（Agent Scheduler）：优先处理和调度代理请求，以优化 LLM 的使用。

AIOS 的代理调度器采用了先进的调度算法，如先入先出(FIFO)和循环轮询(Round Robin)，这些算法通过优化任务执行顺序和平衡资源使用，提高了处理大量并发请求的能力。通过并发执行，调度器能够显著减少代理的等待时间，同时保持高效的任务处理。

上下文管理器（Context Manager）：支持在 LLM 中的中间生成状态的快照和恢复以及 LLM 的上下文窗口管理。

对于需要处理大规模数据和长上下文的复杂任务，上下文管理器提供了快照和恢复功能，使得代理可以在资源受限时暂停并在资源可用时恢复，而不会丢失上下文信息。此外，上下文窗口管理功能通过文本摘要和上下文扩展技术，帮助代理有效处理超出 LLM 处理能力的长上下文情况。

内存管理器（Memory Manager）：为每个代理的交互日志提供短期内存。

存储管理器（Storage Manager）：将代理交互日志持久化到长期存储以供将来检索。

这两个模块共同工作，确保了大规模数据的有效存储和快速访问。内存管理器为每个代理的交互日志提供短期内存，支持快速数据处理和访问。存储管理器负责长期数据保存，使用了本地文件、数据库或云解决方案，保障了数据的持久性和可靠性。

工具管理器（Tool Manager）：管理代理调用外部 API 工具（例如搜索、科学计算）。

工具管理器集成了多种 API 工具，支持代理执行复杂任务时调用外部资源。这为处理大规模数据集、执行复杂的计算或查询提供了强大的扩展能力。

访问管理器（Access Manager）：在代理之间执行隐私和访问控制策略。

此外，内核通过 LLM 系统调用接口暴露服务，使代理可以透明地利用这些服务。AIOS 还设计了 AIOS SDK 来进一步封装 LLM 系统调用，为代理开发者提供更便利的代理库函数。通过 AIOS 架构，例如旅行计划代理可以将其任务分解为结合 LLM 推理（例如计划生成和工具调用决策）和操作系统级别行为（例如访问存储和执行软件服务）的步骤。这种能力的协同组合使得多个 LLM 代理能够处理越来越复杂的多模态任务，这些任务需要推理、执行和与物理世界的交互。

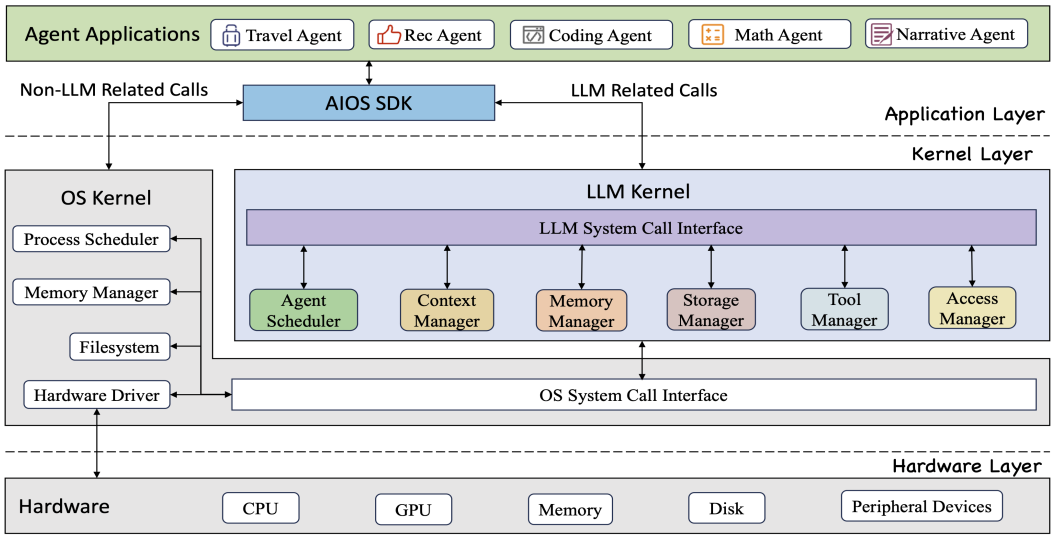

图 2: The AIOS architecture

如图 2 所示，AIOS 架构分为三层：应用层、内核层和硬件层。这种分层架构确保系统各个部分的职责清晰划分。每个上层抽象了下层的复杂性，通过接口或特定模块促进交互，从而提高了模块化程度，简化不同层之间的系统交互。

AIOS 为人工智能（AI）和操作系统（OS）的未来发展提供了一个创新的视角和框架，它将大型语言模型（LLM）作为操作系统的一部分，从而实现了一个“有灵魂”的操作系统。这不仅推动了向更高级别的通用人工智能（AGI）迈进的步伐，还为 AI 代理的开发和部署提供了新的机会和挑战。以下是 AIOS 对于人工智能和操作系统未来发展的几个意义：

1. 提升 AI 代理的效率和性能：通过优化资源分配、上下文管理和调度机制，AIOS 能够提高 AI 代理的运行效率和性能。这对于处理

复杂任务和大量并发请求尤为重要，能够确保系统资源被高效利用，减少等待时间，提升用户体验。

2. 促进 AI 与操作系统的融合: AIOS 将 LLM 作为操作系统的核心组成部分，这代表了 AI 和操作系统融合的一个重要步骤。通过将 AI 深度集成到操作系统中，AIOS 为 AI 应用的开发和运行提供了一个更为直接和高效的平台，这可能会促进新型 AI 应用的涌现。

3. 推动操作系统的进化: AIOS 提出了一个关于如何将 AI 能力嵌入到操作系统中的新范式。这不仅仅是在操作系统上运行 AI 应用，而是让 AI 成为操作系统的一部分，参与到操作系统的各个方面，如资源管理、调度和安全性控制。这可能会引发操作系统设计和实现方式的根本变革。

4. 开拓新的研究和开发领域: AIOS 的出现开辟了一个跨学科的研究和开发领域，涉及人工智能、操作系统、软件工程等多个领域。它为研究社区提出了一系列新的研究问题和挑战，如高级调度算法、上下文管理技术、资源优化策略等。

5. 推进向 AGI 的转变: 通过为 LLM 代理提供一个高效、灵活和安全的运行平台，AIOS 为实现更高级别的智能，即通用人工智能（AGI）提供了可能。AIOS 的模块化和灵活性能够支持更加复杂的 AI 代理开发，这些代理能够在更广泛的环境和情境中独立工作，展现出更高层次的理解和决策能力。

总之，AIOS 的提出和实现为 AI 和操作系统的未来发展开辟了新的道路，预示着 AI 技术将在系统设计、资源管理和用户交互等方面发挥越来越重要的作用。

1.3 文件系统老化——BetrFS

文件系统的使用速度随着使用时长的增加而减慢，这种现象被称为文件系统的老化。

在文件系统的整个生命周期中，有大量的文件生成、删除、附加、移动操作，当逻辑上连续的文件块（来自同一目录中的大文件或小文件）分散在磁盘上时，就会发生存储碎片化，并带来使用效率的降低。

在 The Linux System Administrator's Guide[11] 一书中提出，现代 Linux 文件系统将文件中的所有块紧密地放在一起，即使它们不能存储在连续的扇区中，也能将碎片化程度降至最低。一些文件系统，如 ext3，有效地分配了离文件中其他块最近的空闲块。因此，不必担心 Linux 系统中的碎片化问题。然而 Conway 等[12] 指出，由于带宽的增长速度大约为存储容量增长速度的平方根，随着顺序 IO 和随机 IO 的差距逐渐增大，碎片化将成为一个越来越重要的问题。为此，Conway 等研发了名为 BetrFS 的新文件系统以从文件布局的角度减缓碎片化对文件系统老化的影响。

1.4 文件备份与恢复——Next4

文件备份与恢复在数据作为一种战略资产的价值不断增长的当今变得愈加重要，然而目前在 linux 上可用的数据备份方法不够有效，因为在运行时，大多数方法都会阻塞 I/O 以保证数据的完整性，这会导致大量 I/O 数据的丢失，且备份需要巨大的存储容量，这导致文件的备份频率不能超过一天一次，导致每次数据的恢复都会带来巨大损失。为了解决这一问题，Dani 等[13] 提出了使用快照实现文件备份和恢复的思路，建立了 Next4 文件系统。实验表明，快照技术可以有效降低备份所需的时间和空间开销，且具有广泛的应用价值。

2 立项依据及理论知识

2.1 局部性原理

局部性原理是指 CPU 访问存储器时，无论是存取指令还是存取数据，所访问的存储单元都趋于聚集在一个较小的连续区域中。局部性原理表现在三个方面：

1. 时间局部性（Temporal Locality）：如果一个信息项正在被访问，那么在近期它很可能还会被再次访问。程序循环、堆栈等是产生时间局部性的原因。

2. 空间局部性（Spatial Locality）：在最近的将来将用到的信息很可能与正在使用的信息在空间地址上是临近的。

3. 顺序局部性（Order Locality）：在典型程序中，除转移类指令外，大部分指令是顺序进行的。顺序执行和非顺序执行的比例大致是 5:1。此外，对大型数组访问也是顺序的。指令的顺序执行、数组的连续存放等是产生顺序局部性的原因。

大语言模型的引入可以更好地利用时空局部性，可以根据文件的使用频率等参数合理安排文件存储和文件调用，在此基础上提高存储效率。

2.2 Ext4 文件系统

Ext4 是第四代扩展文件系统，是 Linux 系统下的日志文件系统，是 Ext3 文件系统的后继版本。

相较于 Ext3，Ext4 有许多方面的优化，例如：

1. Ext3 文件系统最多只能支持 32TB 的文件系统和 2TB 的文件，根据使用的具体架构和系统设置，实际容量上限可能比这个数字还要低，即只能容纳 2TB 的文件系统和 16GB 的文件。而 Ext4 的文件系统容量达到 1EB，而文件容量则达到 16TB。

2. Ext3 目前只支持 32000 个子目录，而 Ext4 取消了这一限制，理论上支持无限数量的子目录。

3. Ext3 文件系统使用 32 位空间记录块数量和 i-节点数量，而 Ext4 文件系统将它们扩充到 64 位。

4. 如果一个应用程序需要在实际使用磁盘空间之前对它进行分配，大部分文件系统都是通过向未使用的磁盘空间写入 0 来实现分配。为了保证下载文件有足够的空间存放，常常会预先创建一个与所下载文件大小相同的空文件，以免未来的数小时或数天之内磁盘空间不足导致下载失败。而 Ext4 在文件系统层面实现了持久预分配并提供相应的 API，比应用软件自己实现更有效率。5. 日志是文件系统最常用的结构，日志也很容易损坏，而从损坏的日志中恢复数据会导致更多的数据损坏。Ext4 给日志数据添加了校验功能，日志校验功能可以很方便地判断日志数据是否损坏。而且 Ext4 将 Ext3 的两阶段日志机制合并成一个阶段，在增加安全性的同时提高了性能

6. 尽管延迟分配、多块分配和盘区功能可以有效减少文件的碎片，但碎片还是不可避免会产生。Ext4 支持在线碎片整理，并将提供 e4defrag 工具进行个别文件或整个文件系统的碎片整理。

Ext4 文件系统功能强大，我们也将基于 Ext4 引入大模型，实现各方面的优化。

3 实验目的

本小组拟在 AIOS 的基础上，在应用层和内核层优化文件索引技术、存储架构和数据备份及恢复技术，预期实现目标（可能会修改）为:

1. 使 LLM 分层安排文件存储，根据文件的调用频率、在一定时空内的使用概率确定索引的优先级，以优化存储效率和索引速度，类似 cache 的实现及其对 CPU 的优化作用;

2. 实现 LLM agents 之间的存储共享，以改善 LLM agents 的决策能力;

3. 将本地数据库与云端相结合，引入 Next4 的快照备份技术，提高文件系统的数据备份和恢复效率;

4. （如果有余力的话,）提高文件系统的稳定性、安全性。

4 运行环境

要实现一个基于 Linux 的文件管理系统并接入大型模型，我们需要考虑以下运行环境要求:

4.1 硬件要求

1. 内存: 根据大模型的规模和实际需求，选择内存容量适中的设备。对于大型模型，可以考虑使用具有高内存容量的服务器或工作站。

2. 硬盘速度: 为了提高数据读写速度，建议使用固态硬盘 (SSD) 来存储和读取模型训练和推理所需的数据。

3. 网络要求: 确保所使用的硬件设备具有稳定的网络连接和高网络带宽，以满足大模型对数据传输和通信的需求。

4.2 操作系统要求

Linux: 与 Windows 或 macOS 相比，Linux 在大模型推理和微调方面提供了更完善的功能支持，并且是工业场景中最常使用的操作系统。

4.3 大模型环境配置

(langchain 例)[14]:

Python 版本 > 3.10.8, < 3.11

Cuda 版本: == 12.2

chromadb: 存储和查询程式语言片段的数据库。

gradio: 将模型、数据集、文本、图像等内容部署成简单的界面。

4.4 综合

综合小组的目标以及编程语言对于大模型的适配度而言，本小组将尝试在虚拟机或者 vlab 上进行工作，编程语言选择 Python 或 c++，仿真平台选择 QEMU，以上运行环境相关条目将综合之后可行性调研以及老师的建议进行灵活修改。

5 重要性分析

本项目可以对 AIOS 的文件系统做以下优化:
1. 优化文件组织: LLM 根据文件在一定时空内的使用频率确定文件的存储位置，将热点文件存储在高速存储介质上，将冷门文件存储在低速存储介质上，以提高文件的访问速度。并且，LLM 可以根据文件的内容和用户的行为进行文件的自动分类和检索，以提高文件的组织和检索效率。
2. 共享存储: 实现 LLM agents 之间的存储共享，以改进模型的决策能力和调度性能，因为一个代理可以从其他代理的内存或存储中受益。
3. 本地与云端相结合: 将本地数据库与云端相结合，以加强文件系统的数据备份和恢复能力。
4. 文件安全: 引入 LLM 的自然语言处理技术，对文件进行内容分析和风险评估，以提高文件的安全性。

参考文献

[1] Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. 2024.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[5] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020.

[6] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[7] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[8] Tianle Han Yiheng Liu et al. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1, 2023.

[9] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions. 2023.

[10] Adith Sreeram A S. An effective query system using llms and langchain. 12(6), 2023.

[11] Lars Wirzenius, Joanna Oja, Stephen Stafford, and Alex Weeks. The linux system administrator's guide. 2004.

[12] Alex Conway, Ainesh Bakshi, Arghya Bhattacharya, Rory Bennett, Yizheng Jiao, Eric Knorr, Michael A. Bender Yang Zhan, William Jannen, Rob Johnson, Bradley C. Kuszmaul, Donald E. Porter, Jun Yuan, and Martin Farach-Colton. File system aging. 2024.

[13] Aditya Dani, Shardul Mangade, Piyush Nimbalkar, and Harshad Shirwadkar. Next4: Snapshots in ext4 file system. 2024.

[14] langchain-chatchat. 2023.