**项目总结报告**

日期：2025/7/17

|  |  |  |  |
| --- | --- | --- | --- |
| 组号 | 17 | 项目名称 | Robotrain：具身智能训练平台 |
| 编程语言 | python | 开发平台和框架 | React+FastAPI+minIO+ray |

|  |
| --- |
| **项目工作小结** |
| 1.采用哪种架构风格？哪些设计模式？  从整体上看，系统采用了经典的**分层架构**风格，将系统清晰地划分为用户界面层、应用层、通信层、数据层和训练层 ，每个层次都有明确的职责，有助于降低模块间的耦合度。  同时，该项目也体现了微服务架构的思想，通过将不同的功能进程（如前端、后端、数据库、训练端等）部署在各自独立的Docker容器中 ，实现了服务的独立部署和扩展。  项目主要运用了两种设计模式来优化系统结构和提升健壮性。  首先是外观模式（Facade），其中Backend服务扮演了核心的外观角色，它封装了与后端多个独立服务（如Trainer、MinIO和postgres\_db）之间的复杂交互，为前端提供了一套统一且简洁的API接口，从而极大地简化了前端的开发逻辑，并增强了后端组件的独立性和可维护性。  最后是状态模式（State），该模式被用来优雅地管理训练任务（TrainingTask）复杂的生命周期，项目将任务的每一种状态（如pending、running、completed、failed）的行为逻辑分别封装到独立的状态类中，TrainingTask对象仅持有对当前状态对象的引用并委托其处理操作，状态的转换也由状态对象自身负责，这种设计避免了在主类中使用大量条件判断，使得代码更加清晰、稳定，并且易于扩展新的状态 。  2.技术方案有哪些亮点？  该项目的技术方案紧密围绕解决机器人大模型训练与部署中的实际痛点展开，展现了多个维度的创新与亮点。  首先，项目明确地基于**实际问题**，旨在解决研究者在微调和部署大模型时面临的流程繁琐、资源门槛高和工具链缺失的困境 ，通过提供一个端到端的云平台，极大地降低了机器人大模型应用的门槛。  为了实现高效和可维护的部署，项目全面采用**Docker部署**方案 ，将前端、后端、数据库和训练端等不同进程容器化，并使用Docker Compose进行编排，实现了服务的模块化和一键化部署。  在资源管理上，项目引入了精巧的**调度器**机制，通过一个自动化的训练任务调度器来管理和序列化对底层GPU资源的访问，确保了在有限的硬件条件下多用户任务能够有序执行。  为了应对大规模模型训练的挑战，项目深度整合了**分布式训练**框架，后端集成了分布式数据并行（DDP）库，能够自动处理多GPU并行和显存优化，使得在单机多卡环境下高效微调大型VLM成为可能 。  在数据管理上，项目采用了**分布式存储系统**MinIO作为对象存储服务，专门用于存放数据集和模型权重等大型文件，这不仅解耦了数据与应用，也为未来系统的横向扩展奠定了基础。  架构上，项目实现了**计算端与后端的分离**，用户通过FastAPI后端提交任务，后端将任务信息存入数据库后，通过消息队列与Ray远程调用独立的训练层执行计算，这种解耦设计提升了系统的灵活性和可靠性 。  此外，项目初步探索了**3D仿真建模**，提供了轨迹数据可视化回放功能，允许用户在训练前对数据质量进行直观的检查和校验 。  在核心的**训练与存储优化**方面，项目通过支持torch.compile静态图生成、实现了对训练过程的精细化控制和性能加速 ；同时，通过对数据集规模的约束和对模型资产的统一管理，使用高效的对象存储也兼顾了存储效率 。  3.是否做了单元测试？是否做了系统的功能测试、性能测试、兼容性测试和易用性测试等？  本项目各个子模块都进行了单元测试。前端部分主要利用Vitest测试框架，针对每个页面和大部分组件进行单元测试；后端部分利用Pytest测试框架，针对service和api进行了单元测试，后端代码覆盖率达到94%；训练端依据整个训练流程的实际逻辑，在其中分模块进行单元测试，覆盖整个训练流程以及评估流程。  本项目也完成了系统的功能测试、性能测试、兼容性测试和易用性测试。  功能测试主要针对  测试注册正常数据是否成功，测试注册非法数据是否正确处理，测试登录正常，测试登录失败时是否正常处理，测试上传数据集正常，测试回放已上传的数据集正常，测试训练流程正常，测试评估已训练的流程正常。  易用性测试针对：  测试用户不需要培训就能使用系统，测试网站各个界面的信息图标提示是否正常显示。  兼容性测试：测试不同浏览器下的兼容性，测试不同操作系统下的兼容性。  性能测试：  测试10000人同时登录，测试10000人同时获取数据集列表，测试10000人同时获取评估列表。  4.是否采用大模型来辅助开发？对开发生产率约有百分之几的提升？  使用Cursor，AI studio工具进行辅助。  对于测试用例的生成，使用cursor辅助生成可以达到相当好的效果，提升90%。  对于系统构建和大型项目的理解，cursor明显能力不足，经常自己产生新的bug，耗费我们大量时间验证和修改它生产的代码。最佳实践是把任务明确相关的代码部分发给AI studio，使用Gemini 2.5 Pro模型生成并确认。可以使用多个ai相互确认。最后首先经过人工检查，再在代码仓库上进行修改，保证代码始终在自己修改之中，不过度依赖cursor类直接修改代码的ai工具。 |
| **项目组成员对项目的贡献度（%）** |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 姓名 | 需求 | 设计 | 编码 | 测试 | 项目管理\* | 小计 | | 师文翔 | 25 | 25 | 25 | 25 | 25 | 25 | | 徐丁杰 | 25 | 25 | 25 | 25 | 25 | 25 | | 潘奕宇 | 25 | 25 | 25 | 25 | 25 | 25 | | 石辰阳 | 25 | 25 | 25 | 25 | 25 | 25 |   注： 项目管理包括项目计划、报告、沟通与协调等。 |

|  |  |
| --- | --- |
| **软件规模** | |
| 前端的代码行数（不包括注解行、空行和复用代码）： | 21425 |
| 后端的代码行数（不包括注解行、空行和复用代码）： | 29613 |

|  |
| --- |
| **经验、教训和建议** |
| 在项目开发中，我们认识到不能盲目迷信AI，尽管它能高效辅助开发，但其产出仍需人类专家的严格审查与验证以确保质量。我们应将AI视为强大的初稿生成器或建议者，但最终的决策和把关必须由人来完成。  我们获得的核心经验是，编写全面细致的测试用例是保障软件质量的基石，能够从源头保证模块的稳定可靠。这不仅能在早期发现并修复问题，也为后续的重构和功能迭代提供了坚实的安全网。  规划备选方案是应对不确定性的关键策略，如此次我们从Ray备选到DDP方案的平稳切换，保证了项目的顺利推进。这个过程也让我们更加重视模块化的解耦设计，通过定义稳定的接口来增强系统的灵活性和可维护性。  我们深刻体会到，各功能模块需要通过稳定抽象的接口实现解耦，并且前后端对接的标准应在项目早期就完成沟通和对齐。这样不仅能保证并行开发的效率，还能在后期极大地方便集成与维护。  最后，团队成员间积极的沟通、明确的早期规划和清晰的分工是项目成功的核心。通过在项目早期就预先规划好方案并进行明确分工，我们确保了团队目标的一致性和协作的高效性。 |

项目组各成员签字：

师文翔

徐丁杰

潘奕宇

石辰阳