2. 它内部的RL训练模块（LightningRL算法）启动工作，将最终奖励`1.0`分配给轨迹中的每一次LLM调用。
3. 然后，它使用这些`(输入, 输出, 奖励)`数据对`Llama-3.2-3B-Instruct`模型进行一次参数更新（梯度下降）。这次更新会使模型在未来更倾向于生成能够得到高奖励的SQL和答案。
4. 模型更新完毕。Server准备好处理下一个任务，此时它内部的LLM已经是“微调”过的新版本了。

这个循环不断重复，Agent通过在客户端的“实战”产生数据，服务器端的模型则根据这些数据不断“进化”。

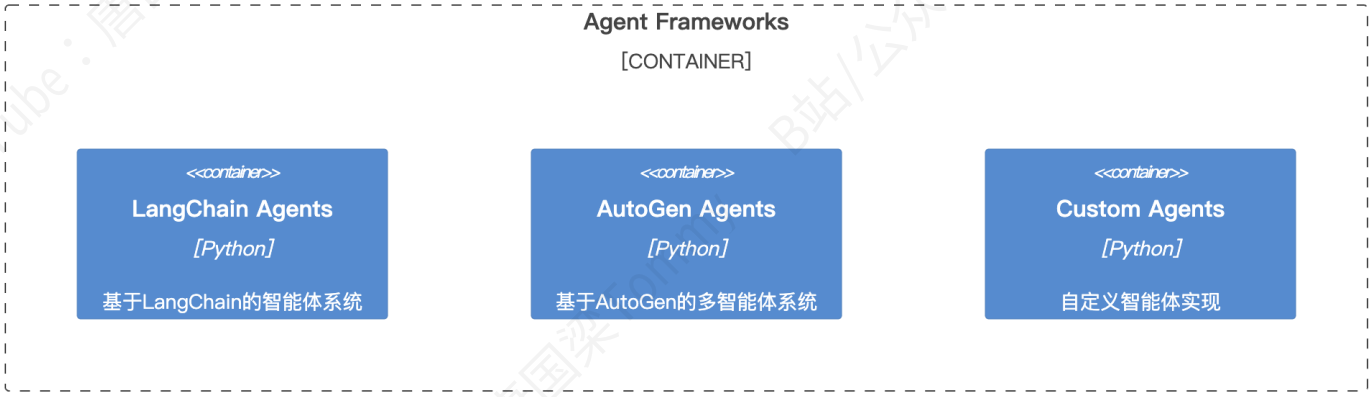
三、系统架构设计

3.1 架构设计

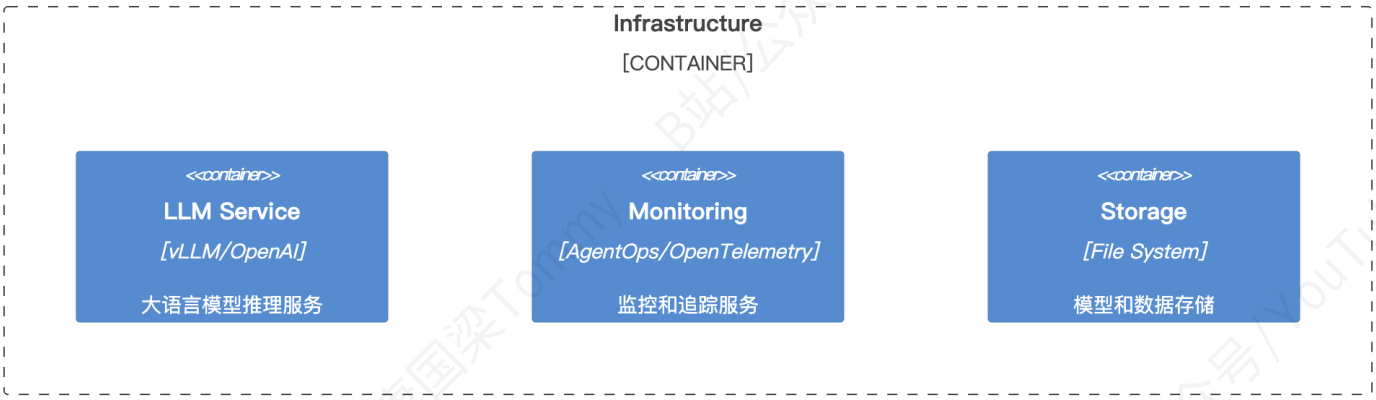
Agent Lightning采用**分布式训练架构**：

- **Agent Lightning Platform**

- **Agent Frameworks**



- **Infrastructure**



3.2 核心模块

3.2.1 AgentLightningServer（训练协调中心）

设计目标：作为分布式训练的中枢神经系统，负责任务分发、资源管理和结果收集

关键职责：

- 任务队列管理
- 版本化资源管理
- 分布式状态同步
- 超时和错误恢复

3.2.2 AgentLightningClient（ Agent 客户端）

设计目标：为 Agent 提供轻量级的训练集成接口，最小化对现有代码的侵入

关键职责：

- 任务轮询和获取
- 资源缓存管理
- 结果上报和重试
- 同步/异步双模式支持

3.2.3 Trainer（分布式训练管理器）

设计目标：管理多进程训练工作流，提供企业级的进程生命周期管理

关键职责：

- 多进程启动和监控

- 优雅关闭和资源清理
- 进程通信和状态同步
- 异常恢复和重启

3.2.4 VERL Engine（强化学习算法引擎）

设计目标：提供高性能的分布式强化学习算法实现

关键职责：

- PPO/GRPO算法实现
- 分布式训练协调（多GPU/多节点支持）
- 模型检查点管理
- 性能指标计算和日志记录

3.3 Agent训练流程图