智弈锋穹–AI 驱动的攻防自治演训靶场

项目文档

作者：何树贤、易梦哲、周君赢、陈易东、张军、牟航、周望、张顾峰、黄睿哲、孙明皓

组织：武汉大学国家网络安全学院

时间：2025 年 8 月 14 日

指导老师：张立强、何艺、严飞

目录

第 1 章 项目背景和概述

内容提要

- 赛题背景
- 基本知识背景

1.1 赛题背景

党的十八大以来，以习近平同志为核心的党中央高度重视网络安全工作，特别在目前日趋复杂的背景下，深刻认识和有力防范网络安全风险，切实维护网络空间安全，已成为事关全局的重大课题。

随着《网络安全法》和《国家网络空间安全战略》的深入推进，实战化网络攻防能力的建设成为网络安全体系中的关键任务之一，尤其在“十五五”规划对关键信息基础设施安全防护提出更高要求的背景下，传统的网络安全演练方式已难以满足动态演训和智能对抗的发展趋势，目前在实际操作中面临三方面的主要瓶颈：

一是传统靶场场景构建模式过于静态，往往采用人工预设的方式搭建演练环境，更新周期长、调整代价高，导致其难以及时引入现实世界中快速变化的业务架构、操作流程以及技术组件，也无法涵盖近年来不断涌现的新型攻击方式与复杂威胁模型，例如 APT、供应链渗透、0day 漏洞利用等，这种脱节使得演练内容始终滞后于现实威胁的发展态势，无法为参与者提供贴近实战的演练体验，也难以检验系统和人员在面对未知攻击时的应对能力，从而削弱了靶场演练在实战能力培养中的实际价值。

二是攻击路径往往依赖人工预设，主要根据经验规则构建固定的攻击链条，缺乏动态生成机制与策略调整能力，导致攻击行为表现出明显的模式化特征，不仅在路径选择上缺乏多样性，也难以模拟出真实渗透过程中攻击者面对防护机制时的适应性调整与策略转移，无法体现现代网络攻防中攻击面广泛、路径不确定、阶段交错的复杂局面，这种设计上的单一性和僵化性直接削弱了演练的覆盖广度与挑战强度，最终造成演练内容与现实渗透行为之间出现严重脱节。

三是评估机制普遍依赖人工回顾，主要通过事后分析操作日志、观察系统响应或专家打分等方式进行效果判断，不仅耗时耗力，而且评判结果高度依赖评估人员的主观判断，缺乏统一的标准与可复用的指标体系，难以保障结果的一致性与公正性，也制约了后续能力提升、策略优化与人才选拔的科学性与有效性。

这些问题在一定程度上制约了演练体系的有效性和可持续发展，因此亟需以网络攻防动

态推演为核心突破口，引入 AI Agent、数据分析与自动化决策等新技术，实现演练环境的动态演化、攻击过程的智能模拟、防御策略的自适应优化以及评估体系的全面量化，从根本上提升攻防演练的实效性与科学性，推动网络安全实战能力建设迈上新台阶。

1.2 基本知识背景

1.2.1 KVM 虚拟机

KVM 是一种内建于 Linux 内核的虚拟化技术，它将 Linux 操作系统转变为一个完整的 Type-1 虚拟化管理程序。KVM 依赖硬件虚拟化扩展（如 Intel VT 或 AMD-V）来为每个虚拟机提供独立的虚拟 CPU 和内存资源，并通过 QEMU 作为后端模拟设备和系统行为，从而运行不同操作系统的虚拟实例。每台虚拟机以普通进程的形式运行在宿主机上，由内核模块 kvm.ko 管理其特权指令的捕获与转发。KVM 虚拟机具备高性能和低开销的特点，支持资源动态分配、快照保存、虚拟网络构建等功能，适合用于构建多租户的复杂网络环境。通过 Libvirt 工具集与 Virt-Manager 图形界面，用户可以以更直观和统一的方式对虚拟机生命周期进行管理，显著简化了大规模虚拟基础设施的运维难度。

1.2.2 Docker 容器

Docker 是一种轻量级虚拟化技术，它通过操作系统级别的内核功能如命名空间和控制组来实现对进程的隔离与资源限制。容器运行在共享内核的宿主机上，但对每个容器而言，它拥有独立的文件系统、网络堆栈与进程空间。Docker 镜像作为容器的模板，以分层文件系统构建，支持快速部署与复用，显著提升了环境一致性和服务交付效率。相较于虚拟机，容器启动速度更快、占用资源更少，更适合部署短周期、可弹性伸缩的服务。在靶场环境中，Docker 通常用于承载 Web 服务、数据库、日志分析平台等高频变化的组件，通过 Docker Compose 工具可实现多容器服务的集中编排，提升整体自动化部署水平。

1.2.3 virsh 虚拟化管理工具

在本项目中，virsh 作为底层虚拟化管理的核心引擎，为靶场环境提供一致、可重复的虚拟机与网络资源创建能力。借助 Libvirt 的 XML 描述语言，用户可以精确地定义每台虚拟机的 CPU、内存、磁盘以及网络接口参数，并通过简单的命令完成虚拟机的启动、暂停、迁移和销毁操作。virsh 所依赖的网络桥接与 NAT 功能能够为不同实验场景隔离网络流量，同时支持在运行时动态调整虚拟网络拓扑，使得在一次部署中完成多种攻防场景成为可能。通过将

这些配置脚本化并纳入版本管理，全流程的环境还原、回滚与审计都可在几分钟内完成，极大地提升了演练效率与运维可控性。

1.2.4 iptables 转发规则

iptables 在内核空间负责对网络数据包进行精细化处理，是实现外部流量与靶场内部主机通信的关键组件。项目中首先在 NAT 表的 PREROUTING 链中完成端口映射，将来自外部的 SSH、Web、VPN 等请求透明地导向各自的容器或虚拟机；随后在 filter 表的 FORWARD 链中实施基于源网段、目的网段和协议端口的策略过滤，既可按照安全域对流量进行白名单放行，也能针对异常会话进行日志记录与审计。在规则定义上，我们将常用策略模板化，并结合脚本自动生成不同演练场景下的规则集，支持增量更新与快速回滚，从而在保证性能稳定的同时，实现对复杂网络拓扑的灵活管控。

1.2.5 漏洞挖掘、利用与防护

为了全面评估目标系统的安全性，项目引入漏洞挖掘到防护的闭环流程。首先通过静态代码扫描和动态漏洞扫描工具结合手工代码审计，识别出潜在的配置缺陷与已知安全漏洞；接着利用 Metasploit 框架或自研 Exploit 脚本，对关键服务和应用展开攻击演练，模拟真实攻击链路并生成详细的行为日志；最后基于安全最佳实践和攻防对抗结果，自动化地生成补丁包、访问控制策略或 WAF 规则，实现对已验证漏洞的快速修复与策略下发。整个过程与日志评估模块相结合，可不断校正检测手段的准确性，推动防御策略从经验驱动向数据驱动转型。

1.2.6 模糊测试

为了发现传统漏洞扫描无法覆盖的边界场景和输入验证缺陷，项目集成了基于覆盖率反馈的模糊测试框架。该框架通过自动化生成海量畸形或半结构化输入，用以测试目标应用的协议解析、接口参数和异常状态响应，并实时追踪代码执行路径和崩溃点。结合基于多轮迭代的种子输入优化策略，系统能动态调整变异算法以覆盖更多分支，显著提高漏洞触发率和测试深度。最终，将模糊测试结果与静态或动态扫描报告融合，为安全团队提供多维度的缺陷洞察，从而在未知攻击面前建立更稳健的防护屏障。

1.2.7 AI Agent

AI Agent 是一种具备感知、决策与执行能力的自主体，用于模拟人类在特定任务下的操作行为。在攻防演练中，Agent 通过持续感知环境变化、分析目标系统特征、结合既有知识进

行推理并最终发起攻击或防御行为，体现出智能化对抗的动态性和自适应性。

一个典型的 Agent 包括感知模块、知识表示模块、策略生成模块和行为执行模块，其运行过程涉及对目标资产的信息采集、漏洞推理、攻击路径生成和自动化利用。随着大模型技术的发展，Agent 逐渐具备了更强的自然语言理解能力与代码生成能力，能够从文档、漏洞数据库中抽取并构建攻击利用逻辑。

在本项目中，我们使用了 Gemini CLI 构建攻防 Agent，借助其集成的上下文感知机制、多模态输入解析能力和代码生成能力，完成对渗透链条的智能模拟与自动攻击脚本的生成，探索了基于大模型驱动的 Agent 在靶场环境中的实战应用潜力。另外，我们还对整体框架进行修改编辑，生成 Agent log 使其行为可以被追踪和审计。

第 2 章 项目设计思路与方案

内容提要

- 项目目标设计思路
- 前端设计方案
- 靶场基础设施设计方案
- 攻击模块设计方案
- 防御模块设计方案
- 评价模块设计方案
- 模块协同设计

2.1 项目目标设计思路

攻击防御评价系统的目标设定，源于网络安全演练领域的核心痛点：传统演练多依赖人工搭建环境、手动记录过程，存在场景单一、评估主观、效率低下等问题。基于此，系统目标聚焦于构建“全流程数字化演练平台”，通过拓扑结构展示、AI Agent 驱动的攻击演练、自动化评估三大核心功能，实现从环境搭建到结果分析的闭环管理。

提示 2.1

前后端数据实时交互，杜绝因信息延迟影响演练效果。

从功能边界来看，前端与后端的拆分遵循展示与控制分离的设计原则。前端主要负责构建可视化的交互界面，包括资源监控平台和模拟终端，便于用户直观了解网络运行状态和攻击演进过程，增强了用户的操作体验和对演练环境的理解程度，而后端则不仅承担着靶场底层拓扑结构的可调整参数的搭建与管理，确保整个系统运行的稳定性与评估指标的客观性，同时还集成了 AI Agent 自主攻击模块、防御控制模块以及完整的演训日志记录系统，使得整个平台具备自动化、智能化的攻防能力和可审计的追踪能力，这种架构设计不仅有效满足了用户在操作便捷性方面的需求，也充分保障了平台在功能性、扩展性与安全性方面的可靠性，最终形成了一个高效、智能、可信的网络安全实战演练平台，为防护能力的验证与提升提供了可量化、可复现、可定制的技术依据。

总结 2.1

明确系统旨在解决传统演练痛点：

方案布局上，通过“可调参数的基础设施 + AI 赋能的行为演训”破除传统靶场痛点难题；

设计架构上，通过“前端展示 + 后端控制”架构实现全流程数字化可视化闭环管理。

2.2 前端部分设计方案

前端设计的核心矛盾在于“动态场景多样性”与“前端展示架构稳定性”的平衡。为解决这一矛盾，方案采用“大模型生成 + 预设框架约束”的混合模式，既发挥大模型在动态内容生成上的优势，又通过框架控制不确定性。

在内容生成层面，选择大模型的核心原因是应对复杂网络场景的多样性需求。网络拓扑、攻击路径等展示内容需根据用户输入的业务场景动态调整，大模型可基于预设规则生成符合逻辑的可视化元素，大幅降低手动绘制的成本。但大模型生成存在随机性，因此引入预设框架作为约束——基于 Vue 构建组件库，包含拓扑图组件、攻击流程日志、LLM 评估结果展示等标准化模块，确保生成内容在布局、交互逻辑上的一致性。

交互设计上，输入框则支持自定义需求，如“添加自定义业务系统图标”“修改拓扑连线样式”等，满足个性化场景。同时，界面美观性通过“分层视觉设计”实现：底层为网络拓扑基础图层，中层为攻击 / 防御动态效果层，顶层为操作交互层，配合蓝白主色调，确保运维与安全人员能快速捕捉关键信息。

总结 2.2

前端通过“大模型 + 预设框架”平衡动态性与稳定性，交互设计兼顾易用性与个性化，视觉分层提升信息获取效率。

2.3 靶场基础设施设计方案

2.3.1 混合式靶场构建架构

为精确模拟现实企业网络中的复杂性和层次性，我们采用虚拟机（VM）与容器（Docker）混合构建的方式，搭建出具备多网段分区、可控攻防流程的综合性靶场环境。这种混合式架构在单一 Linux 宿主机上部署，结合 KVM 虚拟化技术与 Docker 容器技术的优势，兼顾资源利用率、部署灵活性与网络隔离性，特别适用于网络安全研究与攻防演练场景。

具体来说，KVM 提供完整的系统虚拟化能力，适合部署如办公终端、数据库服务器、安全监控平台等需要独立操作系统环境的资产；其高度的隔离性有助于精确控制不同主机间的攻击面暴露。Docker 容器则主要用于部署 Web 服务、邮件服务、VPN 节点、日志收集器等应用层服务，其部署迅速、资源占用小，便于模拟海量业务服务的动态上线与下线过程。

整个靶场通过宿主机 iptables 规则和 NAT 配置进行网络调度和访问控制，不同虚拟资产被划分至逻辑隔离的子网中（如外部网络、DMZ 区、内部网、管理网等），通过有序的访问链

条构建典型攻击路径，为自动化攻击链生成与防御部署策略验证提供实验平台。该架构既能保障底层网络和系统级别的真实还原，又能支持快速迭代的攻防实验，兼顾研究深度与开发效率。

2.3.2 libvirt 与 virsh 结合的管理机制

为了有效管理靶场中众多 KVM 虚拟机资源，我们引入 libvirt 虚拟化抽象层及其命令行接口工具 `virsh` 进行集中控制与自动化操作。libvirt 提供统一的 API 和后端支持，允许用户在多种虚拟化平台（如 KVM、QEMU）上以一致的方式创建、配置、运行与监控虚拟机资源。

virsh 是 libvirt 提供的核心命令行管理工具，具备如下功能：

- 创建、导入与删除虚拟机定义（通过 XML 配置或克隆方式）
- 实时启动、关闭、挂起与恢复虚拟机状态
- 配置虚拟网络、网桥、磁盘挂载等底层资源
- 监控虚拟机运行状态与资源消耗

通过 libvirt 虚拟网络功能，我们可以为不同虚拟机分配逻辑子网，配置独立网桥或 NAT 模式网络，并与宿主机 iptables 规则配合，实现复杂的内部通信结构与访问边界控制。例如，可为 net-ext、net-dmz、net-int 等网络区域分别定义独立虚拟网络，并将其绑定至不同的 KVM VM。

此外，libvirt 还支持使用 XML 文件模板进行大规模虚拟机的自动部署与配置，使得整个靶场环境具备良好的可复现性与脚本化能力，便于后续的版本迭代与实验重构。

综上，通过 libvirt + virsh 的管理机制，我们不仅实现了对虚拟机资源的集中调度，还为混合靶场的生命周期管理提供了强有力的底层支撑。

2.3.3 KVM 虚拟网络配置

在本靶场平台中，虚拟网络被设计为与企业典型分区一致的四个逻辑区域：外部网络、隔离区（DMZ）、内部业务网和管理网。每个区域在宿主机上对应一条 Linux Bridge，并由 libvirt 进行统一生命周期管理。宿主机的物理网卡直接接入外部网络桥，其余网络桥保持纯虚拟化，以便通过 iptables 或 nftables 精细控制跨区访问。

外部网络桥主要用于模拟互联网流量，为靶场提供真实的出入口；DMZ 区承载对外服务，如 Web 前端或跳板机，是攻击者进入内部之前的缓冲带；内部业务网聚合数据库、文件服务等核心资产，默认与外界隔离；管理网专供日志收集、监控与自动化运维流量，通过独立的地址段与更严格的策略实现最小暴露面。

在虚拟机层面，每台 KVM 实例根据角色被分配一张或多张网卡，分别连接到对应网络桥。这样既能还原多网卡设备（如堡垒机或防火墙）的真实部署，也能在同一宿主机内部构建完整的东西向与南北向流量路径。若需进一步细分租户或场景，可在桥上叠加 VLAN 或通过 VXLAN 隧道把同一网段扩展到多台宿主机。

配合 virtio-net 与 vhost-net 加速，以及统一 MTU 和流量整形策略，靶场网络既保证了高性能转发，又能通过包过滤和速率限制将潜在的恶意流量控制在预期范围内。这样的网络拓扑为后续攻击链编排、流量回放和安全策略验证提供了稳定、灵活且接近真实生产环境的基础设施。

说明 2.1 (关于自动化设置的几点说明)

为了实现场景的自动生成和参数的实时调整，我们在靶场控制面中引入一套轻量级 Agent 作为核心调度器。所有虚拟机与容器均以预构建镜像形式存放于本地仓库，启动时仅需加载模板即可完成系统级配置；而业务层面的变更（如服务端口、日志路径）则通过一份声明式 XML 文件描述。Agent 负责解析该文件，为每台实例分配不冲突的子网地址、更新 virsh 或 Docker 网络映射和网段分配，并基于场景标签自动生成对应的 *iptables* 规则集。

在运行过程中，Agent 持续订阅宿主机资源指标与攻击事件流。当检测到资源瓶颈或攻击阶段变更时，它会实时调整 CPU 内存配额、动态收缩或扩展网络带宽，并对现有防火墙策略进行增删改，确保实验流量始终处于可控范围。所有调整操作均通过热更新接口完成，无需停止虚拟机或容器即可生效，从而实现靶场环境的秒级重塑与精细化调优。

2.4 攻击模块设计方案

2.4.1 关键场景攻击链构建

攻击模块设计以模拟真实网络环境中的典型高级持续性威胁为目标，打勾选取基础场景，通过构建完整的攻击链路，展示从初始渗透到数据窃取的全过程。在渗透阶段，AI Agent 将尝试执行社会工程攻击或已知漏洞利用以获取初始访问权限；在横向移动阶段，攻击体主动识别内网结构，渗透并控制更多节点，扩大攻击面；在数据窃取阶段，模拟攻击者对已入侵主机进行文件扫描与关键数据提取，目标文件人工预设于用户目录下，以增强演练的真实性与针对性。整个过程由攻击 AI Agent 主导执行，路径选择具备自主决策能力，可根据目标系统环境自动调整策略，反映出复杂攻击链在真实场景中的演化过程。

2.4.2 人工辅助漏洞利用与 EXP 构建流程

除自动化攻击路径外，系统也支持人工参与的漏洞利用流程，用于检验攻击分析能力与 EXP 生成能力。在内核层面，演练人员可选择已知存在或根据实际生产环境构建的高危漏洞的虚拟机镜像。

对于 nday 漏洞演练：通过调试与运行验证 EXP 的有效性；在 Web 服务层面，结合 xray 和 nuclei 等漏洞扫描工具，对目标站点进行扫描与信息收集，识别存在的安全隐患，并在社区或数据库中查找已公开的 PoC，修改参数以适配当前演练环境，从而实现漏洞的成功利用。整个过程强调人工分析能力与漏洞利用链构建能力的锻炼，辅助 AI Agent 形成更完善的攻击库资源。

对于 0day 漏洞以及 APT 攻击演练：

2.4.3 攻击 Agent 的自动化执行机制

我们使用开源 Agent 客户端框架 Gemini Cli 作为我们的 AI Agent 基本架构，该框架支持多种主流操作系统（如 Windows、Linux、MacOS 等），也可适配 OpenEuler 等国产开源系统。可配置工作目录自主设置 RAG，并能操作终端执行命令，并自反馈的给出相应回答和调整

为了实现攻击行为的自主执行，攻击 Agent 具备版本识别、漏洞关联与自动配置功能。在目标识别阶段，Agent 基于资产信息与系统指纹确定目标组件与版本号，在项目构建的 Agent RAG 知识检索系统中查找与之对应的已知漏洞信息，并获取匹配的 PoC 样本；在利用准备阶段，Agent 自动对 PoC 进行重构与配置，主要完成攻击目标的地址替换、参数填充与脚本适配；最终执行攻击操作，并使用 Agent log 监控其效果反馈，包括系统响应、异常状态与返回值，判断攻击是否成功，并在必要时根据失败原因调整路径或切换攻击手段，实现攻击链的持续推进与策略自适应。

2.4.4 攻击行为自动化评估与结果统计

为支持攻击模块的自动化评估机制，我们在 Agent 的架构中设计了基于日志驱动的攻击效果量化方案，攻击 Agent 在每次执行攻击任务时将自动记录完整的行为的 shell 执行日志，在这些日志中，信息包括攻击起始时间、攻击方法、所用 EXP 等。

然后，系统使用量化模型以及 LLM 对这些数据进行自动归类与统计，生成攻击尝试与成功次数的对比结果，并以可视化方式呈现给用户，辅助评估攻击路径的有效性、Agent 策略的合理性以及环境部署的防护能力，从而实现攻击行为从执行到评估的全流程自动、闭环管理。

2.5 防御模块设计方案

防御模块以“精准识别 + 高效响应”为目标，通过“日志审计 + AI 检测 + 独立分析”的三层架构，实现对攻击行为的全链路感知与评估。

日志审计层负责数据采集，覆盖容器日志（如 K8s 事件日志）、网络流量日志（如防火墙规则命中记录）、应用日志（如 Web 服务器访问日志）等全维度信息。采用 ELK（Elasticsearch+Logstash+Kibana）栈进行日志聚合，通过 Filebeat 代理实时采集日志，经 Logstash 清洗（如过滤冗余字段、标准化时间格式）后存入 Elasticsearch，确保数据的完整性与一致性。

AI 检测层是防御核心，采用“特征工程 + 深度学习”混合模型。特征工程提取攻击行为的静态特征（如异常端口访问、恶意 User-Agent）；深度学习模型（如 LSTM 神经网络）则通过训练历史攻击日志，学习动态攻击模式（如 SQL 注入的字符变异规律、勒索病毒的文件加密行为序列）。模型部署采用 TensorFlow Serving 容器化方案，支持实时接收日志数据并输出攻击概率评分（0-100 分），评分超过阈值（如 70 分）则触发告警。

为提升检测准确性，设计独立日志分析模块。该模块采用微服务架构，与攻击、防御其他模块完全解耦，通过消息队列（如 Kafka）异步接收日志数据，避免资源竞争与干扰。同时，模块内置“误报修正机制”——当 AI 检测到攻击行为后，自动关联历史日志中的正常操作记录进行交叉验证，例如，某 IP 触发“异常登录”告警时，若该 IP 在过去 30 天有多次正常登录记录且属于企业内网段，则降低告警等级，减少误报率。

说明 2.2

（防御模块可能缺乏的说明）

2.6 评价模块设计方案

评价模块旨在通过对攻击机 Shell 行为日志的深度分析，为整体防御体系提供精准、可落地的改进建议。首先，模块会持续采集 Agent 在目标机器上记录的所有交互式命令和系统调用等关键日志，通过统一的预处理管道清洗、聚合出高效特征向量，并将其按照语义和行为模式分批次推送至大规模语言模型（LLM）中，以便对潜在威胁行为进行上下文理解与行为溯源。整个过程采用流式数据传输与异步调度相结合的方式，既保障了日志传输的实时性，又在保证系统吞吐的同时防止了过载风险。

在日志解析阶段，评价模块依托 LLM 强大的自然语言理解与推理能力，针对命令执行先后顺序、参数使用逻辑以及与已知攻击场景的相似度等维度，生成多维度行为画像，并对可能存在误报、漏报或策略盲区的环节进行重点标注。随后，模块结合内置的评分模型与安全

最佳实践库，对每一次攻击尝试给出可量化的安全得分，通过连续监测得分变化来评估防御策略的有效性。同时，为了使反馈更具可操作性，评价模块以“风险描述 + 改进建议 + 优先级指引”的形式输出详细报告，帮助安全运维团队从策略调整、权限收敛和审计日志配置等多个层面迅速响应。

为了实现持续优化，评价模块内置自我学习机制。每次运维人员对报告中建议的采纳情况和后续安全事件的响应结果，都会被回馈至评估引擎，用于修正后续行为分析的权重分配与建议生成逻辑。在实践中，随着样本累积和模型微调，评价模块将不断提升对新型威胁模式的识别能力，进一步缩短从发现异常到落地防御的周期，真正做到“检测—评估—响应—复盘”的闭环迭代。

2.7 模块协同设计

各模块并非独立运行，而是通过数据流转形成闭环：前端将用户配置（如演练场景参数）传递给后端；后端根据参数部署容器环境、生成网络拓扑，并将环境信息同步至攻击模块；攻击模块完成漏洞扫描与攻击演示后，将攻击过程数据（如时间、路径、结果）推送至防御模块；防御模块的日志分析与 AI 检测结果，经后端自动化评估引擎处理后，转化为可视化图表在前端展示。这种协同机制确保了“环境 - 攻击 - 防御 - 评估”全流程的连贯性，最终实现系统的核心价值——为网络安全防护提供可信赖的演练与评估平台。

总结 2.3

各模块通过数据闭环协同，实现从场景配置到评估展示的全流程贯通，保障系统功能的完整性与连贯性。

第 3 章 方案实现

内容提要

- 靶场环境构建及其自动化、参数调整
- 防御模块设计方案
- 攻击模块设计方案
- 评估模块设计方案

3.1 靶场环境构建方案及其自动化、参数调整

靶场网络的三层架构由主机层、网段层和转发层按功能职责依次递进构成，既保证了环境搭建的可复用性，也为后续的自动化和参数化调整提供了清晰的模块化边界。

3.1.1 主机层

主机层以虚拟化实例与容器化服务共同构成靶场的算力基础。通过在宿主机上并行部署若干虚拟机实例，同时在指定的 VM 中运行 Docker 容器群，实现对攻防目标的多样化呈现。为了打通内外网访问，主机层还集成了一台基于 OpenVPN 的跳板机，该节点既作为运维管理的入口，也承担了虚拟网关的职责。所有靶场主机通过桥接或路由方式与跳板机互联，形成了可控的逻辑拓扑，同时在该层面预留了 Agent 工作目录，用于后续收集日志、下发配置及执行自动化脚本。

3.1.2 网段层

网段层借助 XML 模板化定义虚拟网络的逻辑结构，按照管理网、外网、DMZ 区和内网等安全域划分多个子网。通过 Libvirt 提供的网络描述语法，手动编写或由 LLM Agent 生成网络段配置文件，然后使用“virsh net-define”完成初始注册，再经由“virsh net-start”激活各网段。该流程可将网段划分、网关地址规划及 DHCP 参数一并注入，形成一组可自动化复用的网络定义。Agent 在工作目录中保留若干 Few-Shot 模板，负责动态填充网段名称、网关 IP 与 CIDR 范围，从而实现基于场景的网络拓扑快速构建。

3.1.3 转发层

转发层依托 iptables 在宿主机内核空间实施精细化的报文转发与策略控制。首先在 NAT 表的 PREROUTING 链中完成端口和地址的 DNAT 映射，使外部请求可透明访问内网 VM 或

容器；接着在 filter 表的 FORWARD 链中，通过策略链条对不同网段间的流量进行放行或丢弃。整个规则集同样通过 Agent 调用预置模板生成脚本文件，支持对协议、端口、源/目的网段的多维度筛选，并以异步批量下发方式加载到系统中。当需要增量调整或扩展拓扑时，只需更新少量模板参数，Agent 即可在数秒内完成新规则的下发与生效。

这一三层架构设计将计算资源、网络拓扑与策略控制有机分离，配合 LLM Agent 的模板化与 Few-Shot 能力，不仅极大提升了环境搭建的效率，也为后续的持续集成、动态演练和安全评估奠定了坚实基础。

3.1.4 动态场景生成和参数调整机制

本靶场将网络拓扑、IP 规划与防火墙策略抽象为声明式场景描述，所有配置均由 LLM 驱动的 Agent 自动生成。流程如下：首先，运维人员编写包含“网段定义、桥接名称、地址池、策略语句”四大部分的 XML 模板；Agent 接收模板后，调用内置的语义解析模块将“允许 DMZ→内网数据库”“外网至 Web 只开 80/443”等自然语言规则转译为网段配置块和 iptables 语句；随后通过 libvirt API 和 Docker API 分别下发网络定义（XML/CLI）、创建对应 Linux Bridge 以及 MacVLAN 接口；最后注入生成的防火墙规则至宿主机的 nftables/iptables 中。整个过程无需人工编写脚本，也不需手动维护 XML 或规则文件。

在拓扑变更或安全策略更新时，只需修改 XML 中的场景描述，Agent 即可对比差异并在线更新网络配置：

- 如果新增子网或租户，自动生成对应桥接和 VLAN 接口，并在 XML 中同步标注；
- 如果调整访问策略，Agent 通过热重载方式追加、删除或替换现有规则，无需中断任何虚拟机或容器；
- 对于网段 IP 池扩容或收缩，Agent 调用 IPAM 子模块实时调整地址范围，并下发新的 DHCP/DNS 配置。

由此，网络拓扑与防火墙策略的“场景—定义—执行”闭环得以在数秒内完成，可在演练过程中任意切换或重构，保证了环境的高度一致性与可控性。

3.1.5 架构及部署可扩展性

尽管当前部署集中在单节点，但设计上具有多层次的扩展能力：

1. 模板复用与多实例复制所有模板支持参数化占位符，可通过批量渲染生成 N 份场景定义，Agent 可并发在多宿主或多命名空间中同步部署。
2. 插件化网络驱动 Agent 核心分为「解析」「IPAM」「桥接」「策略」四大插件，未来可替

换为 Kubernetes CNI、SDN 控制器或公有云网络模块，只需编写对应适配层即可接入原有流程。

3. 跨宿主 VXLAN/EVPN 扩展当需要跨机房或多宿主机协同时，Agent 可在现有 Linux Bridge 基础上生成 VXLAN 隧道或 EVPN 配置，并自动在 XML 中注入对等节点列表，实现 L2 网络透明延伸。
4. 混合云对接 Agent 支持将同一套场景模板渲染为 Terraform HCL 或云厂商网络定义，通过 API 在公有云中创建 VPC、子网与安全组，再使用 IPSec 或 WireGuard 将云端网络与本地长时互联。

上述扩展策略既保留了模板化和声明式的高层管理方式，又可无缝接入新技术栈与云平台，确保靶场环境可随需求增长而弹性伸缩。

3.2 攻击模块设计方案

3.2.1 架构概述

本攻击模块基于开源的 Gemini CLI Agent 框架，在原有流程之上引入多项扩展，以满足智能化攻击模拟需求。核心组件包括：攻击决策引擎、漏洞与利用库接口、攻击执行器以及行为日志模块。Agent 在初始化时加载各功能插件，形成从“策略决策 → 漏洞检索 → 利用执行 → 结果反馈”的闭环。

3.2.2 对常规漏洞的复现及演练

在常规漏洞演练中，Agent 首先对目标服务进行指纹识别，并将服务版本信息与本地集成的 ExploitDB、Metasploit 模块库进行匹配检索。匹配结果按风险评分和可用性排序后，Agent 自动下载或调用对应的 EXP 脚本，生成针对目标的攻击任务。攻击执行器以并发或顺序方式发起利用请求，并通过内置的会话管理模块检测是否成功获取 Shell 或触发漏洞行为。一个典型的靶场漏洞复现的流程如下：

首先，Agent 根据 XML 中定义的“漏洞扫描范围”加载指纹模块（如 Nmap 脚本或 HTTP banners）；然后 Agent 启用我们内置在 Agent 工作目录中的漏洞检索子系统进行关键字查找，查询本地数据库并返回可用 EXP 列表；随后攻击执行器调用 EXP，自动填充必要参数（目标 IP、端口、URL 路径、用户名/密码等），并在成功或失败后记录结果；复现结果与原始 EXP 元信息（CVE 编号、漏洞描述、利用条件）一并写入 Shell 行为日志，供后续分析与复测使用。

通过这种自动化流程，常规漏洞的复现和演练可在分钟级完成，不仅验证了漏洞利用链

的完整性，也为评测团队提供了详尽的操作记录。

3.2.3 模糊测试挖掘和异常检测

针对零日或未知漏洞，我们在 Agent 中集成了多种 Fuzzing 插件，包括协议模糊、文件格式模糊和 API 参数模糊。Agent 在演练开始时，基于资产清单自动选择合适的 Fuzzer，并生成变异输入流发往目标服务。实时监控模块通过 eBPF 或动态插桩技术捕获目标进程的崩溃信息、异常日志和内存访问违规。典型流程如下：

Agent 读取“模糊测试配置”，确定测试模式（长度变异、格式变异或序列化变异）；Fuzzing 引擎并行输出多路变异样本，按优先级向目标发起请求，并通过覆盖率反馈调整变异策略；监控组件捕捉进程崩溃栈信息和异常日志，将可复现的崩溃用例存储到本地目录；最后 Agent 将发现的异常输入与上下文环境（目标版本、触发条件）自动归档，并将可能的利用点标注到场景描述中，供后续漏洞利用模块调用。

该机制能够在演练过程中主动挖掘潜在缺陷，并实时生成可复现测试用例，显著提升零日漏洞发现效率与准确性。

3.3 防御模块设计方案

3.4 评估模块设计方案

3.4.1 Agent 行为捕获与日志生成

在本方案中，我们对开源 Gemini CLI 进行了深度定制，使其在每次调用 shell 执行分支时能够自动生成结构化日志。具体而言，针对 TypeScript 代码中通过 childprocess.exec 或等效方法发起的命令执行操作，注入了统一的日志入口 log entry；该入口以预定义的 JSON 模板为蓝本，记录命令字符串、执行时间戳、工作目录、用户标识以及执行结果（包括成功、失败、回滚等状态）。在命令执行前后以及异常撤销阶段，都会分别写入对应状态的日志条目，确保整个执行过程中的每一个关键节点都被原子化捕获。日志层面采用异步写入策略，兼顾了性能开销最小化与日志完整性的双重需求。

3.4.2 日志结构解析与图示说明

生成的原始日志条目严格遵循统一字段规范，每条记录包含 id, timestamp, command, status 等维度属性。为便于文档说明，我们在此插入系统生成的示例图（见图），直观展现了单条日

志条目的核心字段与层级关系。该图示不仅阐释了日志中各属性的含义，也帮助读者快速理解在多状态分支下，系统如何保持对每一次命令执行全链路的端到端可追溯。

3.4.3 JSON 化处理与前端交付

日志生成完成后，通过管道化处理模块将离散的日志条目汇总为符合前端接口规范的 JSON 数据包。该处理流程包括按时间窗口聚合、字段类型校验和可选的敏感信息脱敏，最终输出的 JSON 对象结构清晰，便于前端组件在可视化大屏中快速渲染行为流。前端在接收到该数据后，可基于内置的时间轴控件或折线图组件，动态展示 Agent 在靶场中执行的每一步操作，并结合评估模块提供的安全得分，实现从日志解析到可视化反馈的全流程闭环。

第 4 章 效果演示

第 5 章 创新与特色

内容提要

- 三层架构与网络参数动态调整
- 基于 Gemini CLI 的行为日志与决策引擎
- 攻防决策与评估一体化
- 前端可视化与虚拟终端实时交互

5.1 三层架构与网络参数动态调整

本项目率先提出将主机层、网段层与转发层三者进行有机划分与协同管理的设计思路。主机层以虚拟机和容器的混合部署为基础，为每一次攻防演练提供独立隔离的计算单元；网段层通过模板化的 XML 定义，精准刻画各安全域的 CIDR 分片及网关属性；转发层则借助 iptables 完成从外部到内网的 DNAT、以及跨域流量的细粒度策略控制。更为突出的是，在网段层与转发层中，我们引入了基于 LLM 的 Agent 自动化配置能力。Agent 可根据演练场景动态生成或更新虚拟网络定义和防火墙规则，实现随时增删虚拟网段、调整网关 IP、修改转发端口等操作。这种方案打破了传统脚本执行僵化、手工维护成本高的局限，使得网络参数的微调与拓扑重构能够在运行期在线完成，显著缩短了环境重建与测试准备的周期，同时保持了高可用与高隔离的双重目标。

5.2 基于 Gemini CLI 的行为日志与决策引擎

我们对开源 Gemini CLI 进行了深入改造，将其打造成既能下发指令又能实时采集 Agent 行为的智能中枢。在每次通过 TypeScript 原生方法触发 shell 执行时代码中嵌入统一日志出口，按预定义 JSON 模板记录命令内容、执行时长、终端输出、错误信息、退出状态等全链路数据。所有日志条目采用异步写入与批量推送相结合的方式，既保证了对大规模并发命令的支持，也避免了性能瓶颈。日志数据上报后，Agent 可在多轮交互中对历史行为进行回溯，并在 LLM 的推理能力加持下自主优化下一步攻击路径。该机制不仅能全面揭示靶场内潜在风险点，还为后续的防御策略评估和改进提供了可度量、可对比的行为基准。

5.3 攻防决策与评估一体化

攻防演练与效果评估向来是割裂的两端，本项目则通过 LLM Agent 实现二者的无缝融合。Agent 在执行攻防操作时实时调用内置决策模型，根据当前网络态势和历史日志动态生成下一步行动；同时，每一次操作的上下文和结果都会反馈至评估模块，由另一个 LLM 子模型基于定量化指标进行打分并形成安全报告。这种“决策—行为—评估”闭环可在数秒内完成一轮迭代，使得演练过程中的策略调整、漏洞修补与风险复测同步推进。通过将攻防动作与评估分数序列化，运维团队可以直观感知策略改进的效果，驱动系统安全防护从静态配置向数据驱动不断进化。

5.4 前端可视化与虚拟终端实时交互

项目在前端设计上同样秉持创新理念，将网络拓扑和攻防态势以可视化图表、拓扑图与时间轴形式直观呈现。用户在浏览器中即可清晰查看各虚拟网段、主机节点及其相互连通关系。同时，内嵌的虚拟终端组件与后端实时对接，运维人员可在同一页面内发起命令、动态调整网络参数或防火墙规则，并即时在可视化界面中观测到其影响。该方案打通了监控与操作的最后一公里，摆脱了传统切换多窗口或终端的低效模式，大幅提升了调试效率和用户体验，并为后续自动化运维和智能告警奠定了基础。

第 6 章 项目总结与展望

内容提要

- 项目总结
- 存在的不足与展望

6.1 项目总结

（这个总结可以放到最后写）

6.2 存在的不足与展望

6.2.1 关于 IPV6 的支持问题

在本次靶场设计与实施过程中，我们主要围绕传统 IPv4 协议体系进行网络规划与流量控制，所有的 NAT、iptables 规则、服务监听与访问控制策略均基于 IPv4 进行构建与验证。然而，IPv6 作为下一代互联网协议，其在地址空间、点对点通信能力、自动地址配置、安全性机制等方面具有显著优势，尤其在企业级网络部署、移动互联网、物联网等场景中正逐渐被广泛采纳。

未在靶场中引入 IPv6 支持，暴露出当前系统在协议兼容性和前瞻性方面的局限。具体而言，现有的 iptables 规则集并未对 IPv6 流量进行任何处理，宿主机及虚拟网络默认可能启用了 IPv6 栈但未加以管理，理论上存在绕过现有防火墙策略的可能性。此外，部分服务程序默认监听 IPv6 接口，一旦未对其明确限制，也可能导致不必要的暴露面。

从实验价值角度看，缺失 IPv6 网络的实验能力也限制了对某些真实世界攻击场景的还原。例如，针对 IPv6 SLAAC 自动配置的钓鱼攻击、RA（Router Advertisement）欺骗等常见手段无法在当前环境中演练；同时，也无法测试支持 IPv6 的恶意软件行为或入侵检测系统对 IPv6 流量的识别能力。

未来的靶场迭代应将 IPv6 纳入核心网络设计范畴，建立 IPv6 地址规划方案，配置与 IPv4 规则并行的防火墙策略，明确虚拟资源的 IPv6 监听范围，同时探索基于 IPv6 的攻击链建构与防御机制。只有充分考虑双栈网络环境，靶场系统的仿真能力和安全性测试覆盖度才能更加贴近真实企业环境的演进趋势。

6.2.2 关于 1-day 和 0-day 漏洞场景下 EXP 自动生成的问题

在安全评估和漏洞利用流程中，自动化生成针对 1-day 或 0-day 漏洞的 Exploit（EXP）一直是学界和工业界共同面临的难题。对于 1-day 漏洞而言，虽然漏洞细节和修补补丁公开可查，但补丁往往并未暴露完整的利用链信息，且不同环境下的内存布局、编译选项及运行时依赖千差万别，使得单纯依靠语言模型或模板化 Agent 生成可靠的 EXP 难度极高。一旦漏洞触发条件涉及复杂的程序状态或多阶段交互，这种自动化方法更容易产生失败甚至误导性的伪 EXP。

对于 0-day 漏洞的场景，难度则更为显著。此类漏洞尚未被公开披露，缺乏任何先验的利用案例或社区讨论，Agent 在训练数据中往往完全无法覆盖相应的漏洞模式。即便具备自动化静态分析或污点追踪等传统方法，结合大规模语言模型生成 EXP 的机制也无法弥补对未知漏洞的直观感知和对运行环境微妙差异的判断，从而难以形成可执行的利用脚本。

尽管自动化 EXP 生成尚未成熟，我们仍能够在评估模块中提供故障样本以供后续分析。具体而言，系统可在漏洞触发过程中捕获崩溃文件、堆栈信息及内存转储等核心数据，统一以标准化的样本格式输出。安全研究人员可以基于这些样本在调试器中重现漏洞上下文，为手动构建 EXP 或引入符号化调试提供必要的参考。通过持续积累不同环境和补丁版本下的故障样本库，可在一定程度上降低后续漏洞利用开发的重复成本。

在未来展望中，借助多模态神经网络与符号执行相结合的探索路线，或将为 EXP 自动生成提供新的可能性。但目前而言，在 1-day 和 0-day 漏洞场景下，我们更倾向于将 Agent 的作用聚焦于自动化样本收集与格式化，以辅助手动分析流程，并促成“样本驱动 + 专家复审”的混合模式，从而在保持自动化效率的同时兼顾漏洞利用的精确性与可靠性。

第 7 章 部署说明和注意事项

内容提要

- 开源项目地址
- 宿主机部署前置工作
- 靶场部署
- 攻防模块部署
- 前端部署
- 注意事项

7.1 开源项目地址

我们演示时的虚拟机的容量十分巨大，这里不直接提供地址，有复现和评审需要的可以联系任一参赛队员。

我们基于开源框架而修改的，用于靶场攻防和评测的 Agent 客户端代码：

https://github.com/Grablocker/Pentest-Gemini.git

我们前端的开源地址：https://github.com/din0sauria/smart-range

我们文档的开源地址：https://github.com/Grablocker/Documentation

7.2 宿主机部署前置工作

为了构建一个高性能且稳定的 KVM 与 Docker 混合靶场环境，宿主机需要在硬件资源、操作系统支持和关键虚拟化组件等方面做好充分准备。以下从硬件配置、系统选择、虚拟化环境搭建和容器平台部署四个方面展开说明。

说明 7.1

建议使用基于 amd64 架构并支持硬件虚拟化的处理器，例如支持 Intel VT-x 或 AMD-V 的多核 CPU，至少 4 个物理核心，以保障能够并发运行多个虚拟机和容器服务。内存方面最低为 16 GB，推荐配置 32 GB 或以上内存。磁盘空间方面至少保留 50 GB 的可用空间，强烈建议选用 SSD，以提供更高的 I/O 吞吐能力，便于高速运行。

操作系统方面，选择支持上述架构的主流操作即可。本项目支持 OpenEuler 等国产开源操作系统。

7.2.1 操作系统选择与配置

宿主机推荐安装 Ubuntu Server 的长期支持版本，例如 Ubuntu 20.04 或 24.04，该系统对 KVM 与 Docker 支持良好，并拥有活跃的社区生态和丰富的文档资源。在部署前需检查并启用 CPU 的硬件虚拟化功能，可通过如下命令进行确认：

```
$ lscpu | grep -E 'vmx|svm'
```

若输出结果中包含“vmx”或“svm”，则说明虚拟化功能已被 CPU 支持。如未显示，则需进入 BIOS 或 UEFI 设置手动开启虚拟化选项。

7.2.2 KVM 及其管理工具安装

KVM 是基于 Linux 内核的虚拟化机制，可将宿主机作为裸机虚拟化管理器运行多个虚拟机。Libvirt 作为其管理工具集，提供统一的虚拟机配置与管理接口，支持命令行与图形化管理方式。Virt-Manager 作为图形前端，便于对虚拟机状态进行可视化查看与控制。安装步骤如下：

安装 KVM 及相关软件包：

```
$ sudo apt update
$ sudo apt -y install bridge-utils cpu-checker libvirt-clients
$ sudo apt -y install libvirt-daemon qemu qemu-kvm virt-manager
```

添加当前用户至 libvirt 与 kvm 用户组以获得管理权限：

```
$ sudo usermod -aG libvirt $USER
$ sudo usermod -aG kvm $USER
```

验证 KVM 模块是否加载成功：

```
$ kvm-ok
```

若输出显示/dev/kvm 存在并支持加速，即说明 KVM 功能正常。

启动并设定 libvirtd 服务为开机自启：

```
$ sudo systemctl enable --now libvirtd
```

Libvirt 可用于创建虚拟网络、自动配置网桥与 DHCP 服务，并设置 iptables 规则，为后续部署多网段靶场提供良好的支持基础。

7.2.3 Docker 及 Compose 平台安装

Docker 平台将在靶场中承担 Web 服务、邮件系统、数据库与监控栈等容器化服务的部署任务，Docker Compose 用于协调多容器服务的构建与编排。安装步骤如下：

安装基础依赖：

```
$ sudo apt update
$ sudo apt install ca-certificates curl gnupg
```

添加 Docker 官方 GPG 密钥：

```
$ sudo install -m 0755 -d /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg
$ sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

配置 Docker 软件源：

```
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "$VERSION_CODENAM
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

安装 Docker 及其组件：

```
$ sudo apt update
$ sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-com
```

添加用户至 docker 用户组：

```
$ sudo usermod -aG docker $USER
```

完成安装后，可通过运行测试容器确认安装状态：

```
$ docker run hello-world
```

若成功输出欢迎信息，说明 Docker 平台已正确配置完成。

7.3 靶场部署

7.3.1 跳板机

跳板机是靶场接受外网访问的桥梁，也是唯一一个依赖完全手动配置的靶场组件。根据文档描述，我们使用了 openVPN 作为跳板机的网络设施。

openVPN 的原项目地址：https://github.com/OpenVPN/openvpn

这里我们推荐下载器配置，这个下载器是 Github 上的开发人员整理的一个兼容性强的下载脚本，可以一键启用。这也是我们的配置方式；项目地址 https://github.com/Nyr/openvpn-install/

7.3.2 靶场内网和虚拟主机

在靶场内网和虚拟主机的部署中，VM 是用户根据自行需求和复现条件打包设计的，用户可以根据需要选择已有的漏洞镜像或自行构建。而 docker 可以用来充当轻量化的服务和构建单元，当然也可以使用 VulnHub 等镜像网站自行搭建和获取具有特定漏洞的 Docker 镜像，并与 VM 虚拟机任意搭配网段；我们的靶场可以做到 VM 和 docker 的统一管理和自由部署。

打包好虚拟机的 VM 镜像之后，提交靶场网络拓扑结构的表单，使用 Agent 自动构建靶场网络拓扑结构；

如果不需要特意部署带有漏洞的 Docker 容器，那么 Docker 的拉取和运行可以通过 Agent 自动完成。Agent 会自动生成对应的 Docker 容器，并配置好网络桥接和 iptables 防火墙规则。

7.4 攻防模块部署

如文档中介绍所示，我们的攻击 Agent 部署在一个 Kali Linux 虚拟机中。在 Kali Linux 中，我们将上述提到的 Agent 集成到了攻击机中，并使用 npm link 集成了呼出命令，无需额外进行配置，直接终端使用 gemini 命令即可呼出。

7.5 前端部署

首先，通过 Git 克隆项目到本地工作目录：

```
git clone https://github.com/din0sauria/smart_range.git
cd smart_range
```

推荐使用 VSCode 作为主要编辑器，并安装 Volar 插件来替代 Vetur，以获得对 Vue 3 和 TypeScript 更精准的语法提示与类型检查。

进入项目根目录后，执行以下命令安装所有前端依赖：

```
npm install
```

在开发过程中，可以使用热重载功能快速预览修改效果：

```
npm run dev
```

完成开发并准备发布时，运行以下命令对项目进行构建和压缩，生成面向生产环境的静态资源：

```
npm run build
```

若需启用页面中的虚拟终端功能，请在独立终端中启动后端服务：

```
python testvtcmd.py
```

7.6 注意事项

7.6.1 Windows 机器部署要求

在靶场环境中引入 Windows 虚拟机时，必须确保操作系统内已安装并启用了 VirtIO 驱动，以实现网络、存储与串口等虚拟化设备的高效交互。若部署前未在 ISO 或映像中集成 VirtIO 驱动程序，Windows 安装过程将无法识别底层虚拟磁盘及网络接口，此时应在安装选项中启用 SATA 控制器以保障磁盘可见性，但这会带来性能与灵活性上的折中。为避免后续因驱动缺失而导致的网络中断或无法挂载虚拟盘，建议在创建 Windows 模板时进行一次性驱动注入，并在首次引导后验证网络连接与磁盘读写性能。这样既可最大程度地发挥虚拟化平台的吞吐能力，又能确保后续自动化管理与快照操作的稳定性。

7.6.2 Vue 开发环境配置

在构建基于 Vue 的前端项目时，推荐在 VSCode 中使用 Volar 作为主要语言服务插件，并在此基础上禁用旧版本的 Vetur 扩展以避免误判与性能冲突。首先，确保在扩展市场中安装最新版本的 Volar，并在工作区设置中启用 Take Over Mode，使其能够接管对文件的模板、脚本和样式部分的语法提示和错误检查。接着，进入 VSCode 的扩展管理界面，将已安装的 Vetur

扩展标记为禁用或直接卸载；这一步可防止两套诊断服务同时运行导致的重复报错、自动补全迟滞以及类型推断不一致等问题。在工作区配置文件中推荐强化 Volar 的支持力度并关闭 Vetur 相关功能。