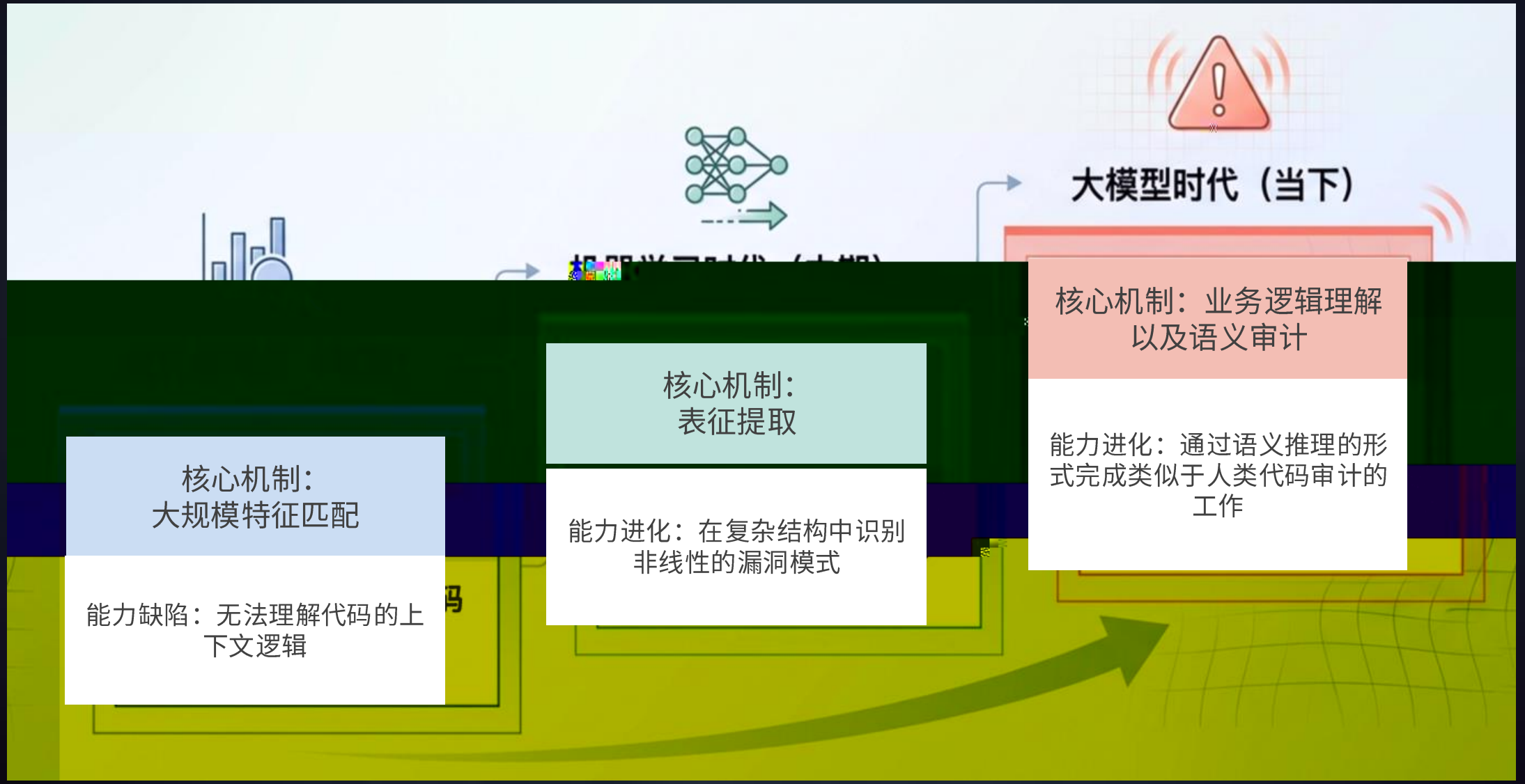
关于AI驱动自动代码审计的实践与思考

漏洞研究院 冯骁韬

AI辅助漏洞检测方法发展演进



目录

CONTENTS

PART 01

Case Study
“逻辑自洽”的幻觉

Case Study - 一次由AI引导的漏洞审计

Claude Code + Opus-4-6

安全漏洞分析报告

存在高危路径遍历漏洞 (Path Traversal / Arbitrary File Write)

漏洞位置

文件:
行号：第 158-162 行

```
158 |         if write_to is True:
```

o 参数作为文件路径，**没有任何路径验证或清理**

```
(create_yaml_template(write_to="..."))
ig["yaml_file"] 值
```

1. 缺少路径验证

- 第 161 行直接使用 write_t
- write_to 参数可以来自:
  - 方法调用时传入的外部参数
  - 配置文件中的 model_conf

2. 攻击场景

场景一：直接调用攻击

```
el_obj)

te(write_to="/etc/passwd")  # 覆盖系统文件
te(write_to="../../sensitive/file.txt")  # 路径遍历
```

```
template = YamlTemplate(mod
# 攻击者控制 write_to 参数
template.create_yaml_templa
template.create_yaml_templa
```

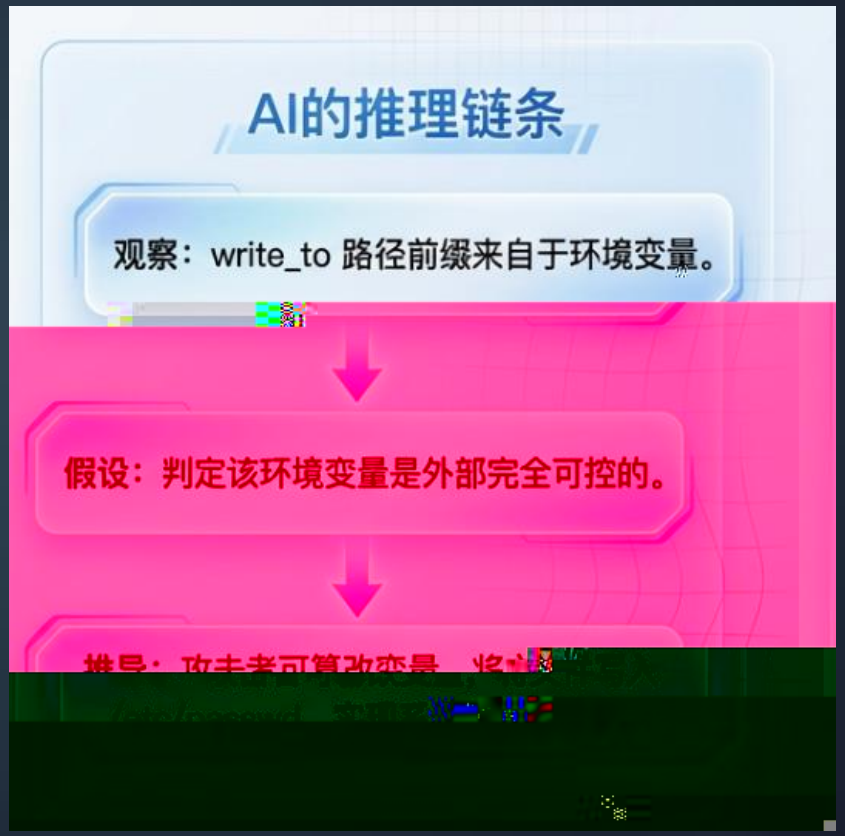
ig（例如通过配置文件注入），可以设置：

```
etc/cron.d/malicious"  # 写入定时任务
```

场景二：配置注入攻击

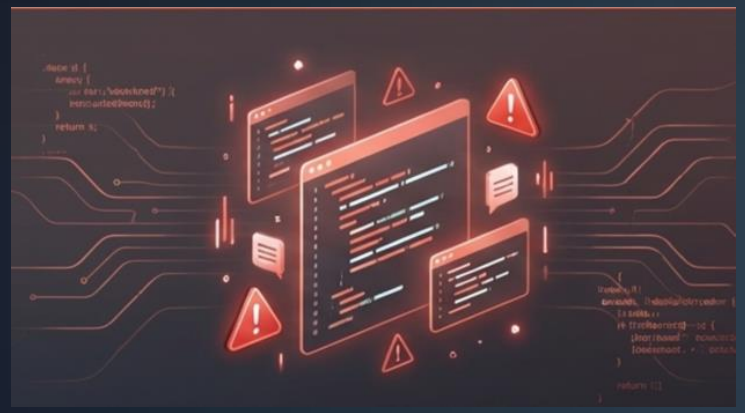
如果攻击者能控制 model_conf

```
model_config = {
    "yaml_file": "../../../
}
```



Case Study - “误报”

AI的“假设”



视角：纯代码级视野（语义推理）
判断逻辑：只要是外部输入，一律视为高危
结果：为了无“漏报”，产生“误报”

真实的部署环境

视角：架构与环境视野
真实环境：系统运行在受控的隔离环境
事实
在网络层面上无法进行干预

Case Study - “盲区”的本质 - AI模型被困于代码孤岛上

PART 02

AI代码审计落地的难题

落地的难题（二） - 通过提示词修复问题

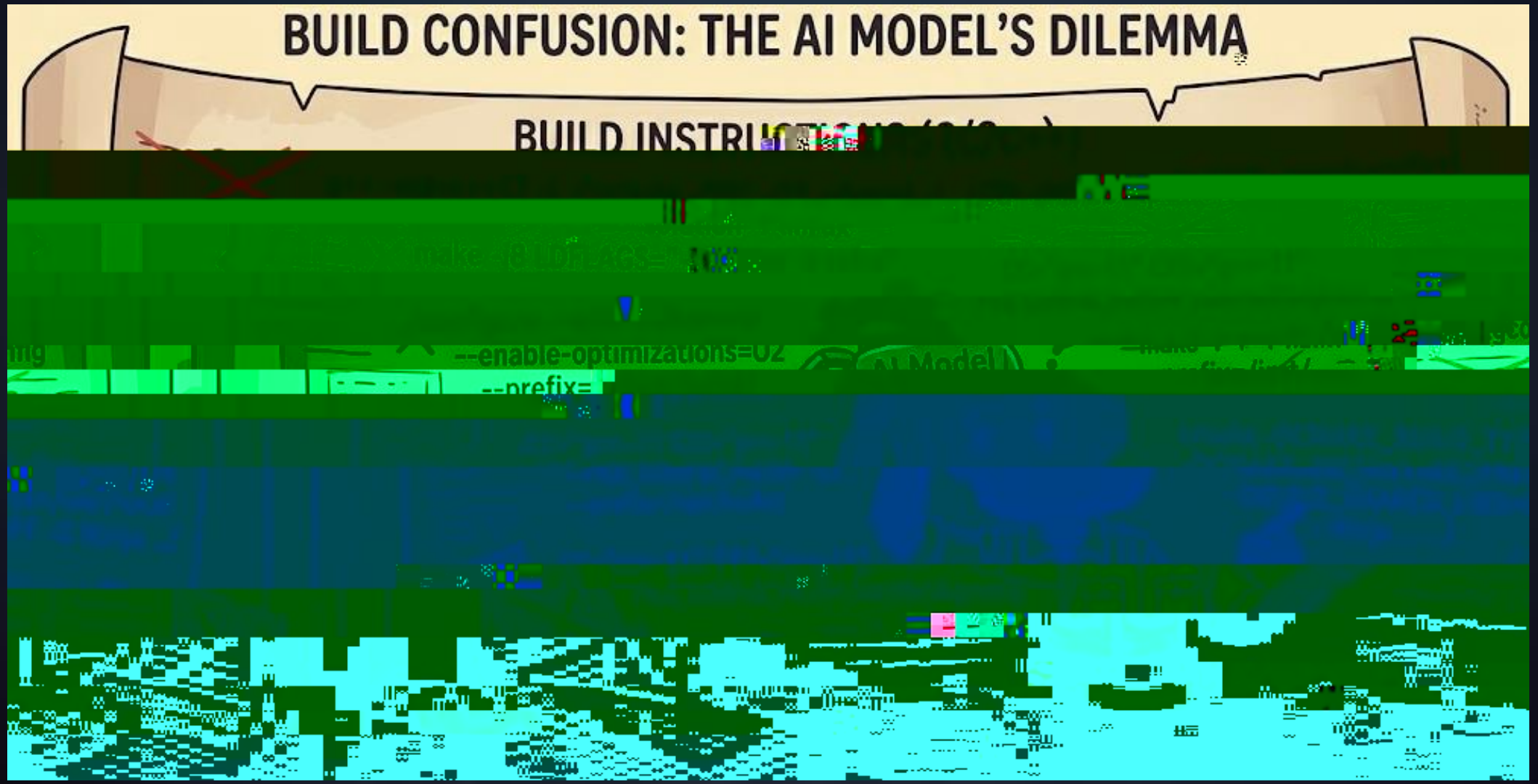
通过修改提示词来让Agent在判断漏洞时需要注意部署环境

“在漏洞研判中,请考虑项目的部署情况”

Agent能够通过扫描项目的文档以及环境配置来思考外部攻击是否可以实现

落地的难题（三） - PoC生成需求

落地的难题（三） - PoC生成难题



PART 03

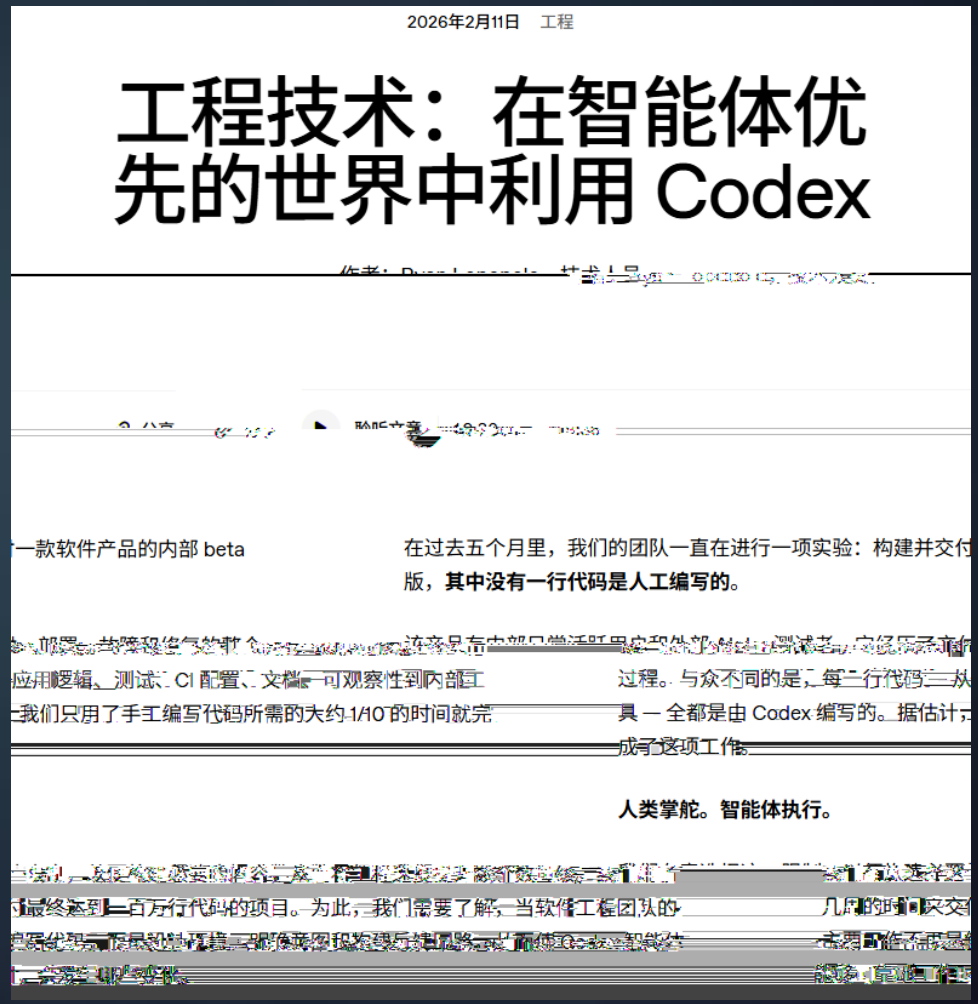
方法论转型与 Harness Engineering

Harness Engineering - Openai

Openai团队将强大的模型比作拥有无限潜能
但是天性狂野难以控制的“烈马”

而Harness则是为这匹烈马量身定制的“马具”

Harness Engineering的本质就是旨在将模型的能力转化为受治理的可靠的行动



2026年2月11日 工程

工程技术：在智能体优先的世界中利用 Codex

一款软件产品的内部 beta

在过去五个月里，我们的团队一直在进行一项实验：构建并交付
版，**其中没有一行代码是人工编写的。**

应用逻辑、测试、CI 配置、文档、可观察性到内部工
我们只用了手工编写代码所需的大约 1/10 的时间就完

过程。与众不同的是，每一行代码——从
具 — 全都是由 Codex 编写的。据估计，
成了这项工作。

人类掌舵。智能体执行。

最终达到一百万行代码的项目。为此，我们需要了解，当软件工程团队的
几周的时间交付

方法论的改变（一）

方法论的改变

从“模型驱动”转向“工程约束“：

核心公式：智能体 = 模型 + 驾驭系统

底层模型是通用的，而决定审计质量的“护城河”是[illegible]约束系统。我们[illegible]应该通过设计精密的CAR（[illegible]Control-Agency-Runtime）来引导AI

破除”代码孤岛“ - 仓库即现实（Repo-as-truth）：

“只是模型在运行时无法访问的东西，对它而言就是不存在的”

必须将所有的“隐性语义”显性化。除了代码，应将项目文档、设计文件、部署架构、环境变量约束以及历史补丁等全部写入知识库并进行索引

方法论的改变（二）

方法论的改变

[illegible]
在方法论的各个环节中AI都会出现幻觉从而引起错误。
利用Runtime（运行时层）的监控机制建立自愈循环（self-healing loop）。例如，当AI生成PoC因编译报错或者断言不符合事实而失败时，系统应自动捕获错误日志并回传给系统，要求模型基于“错误经验”重新推理修正认知。

状态外部化与成本控制：

在对一个项目的审计过程中，一些背景以及逻辑意图的推理可能会在不同的case中复用，重复消耗推理token
需要建立项目知识库实现状态外部化（State Externalization）。例如将AI对项目环境流程重复的推理结论存储在知识库，避免每次都消耗token进行相同的逻辑推理，从而降低模型的消耗。

基于harness engineering思想的CAR架构模型

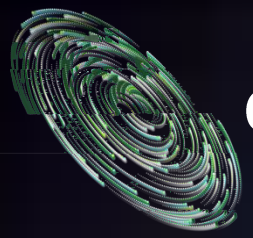
CAR模型的核心目的是为了实现从“模型驱动”到“系统可靠”的范式跃迁。

CAR模型的构成：

控制层（Control - C）：包含在任务开始前塑造行为的持久化制品。者包括项目指令（如AGENTS.md）、架构规则、代码检查器（Linter）、权限政策和验收标准。本质上是将“人类判断”转化成“机器刻度约束”的地方。

代理层（Agency - A）[illegible]集（API）、执行环境（如代码沙盒、浏览器）、以及分工结构（如多智能体协作模式）。

运行时层（Runtime - R）：管理任务随实践推移的执行稳定性。这包括上下文压缩与外部状态化（如建立知识库）、重试与回滚策略、审计轨迹追踪以及预算控制。



CAR

信息·趋势·感悟

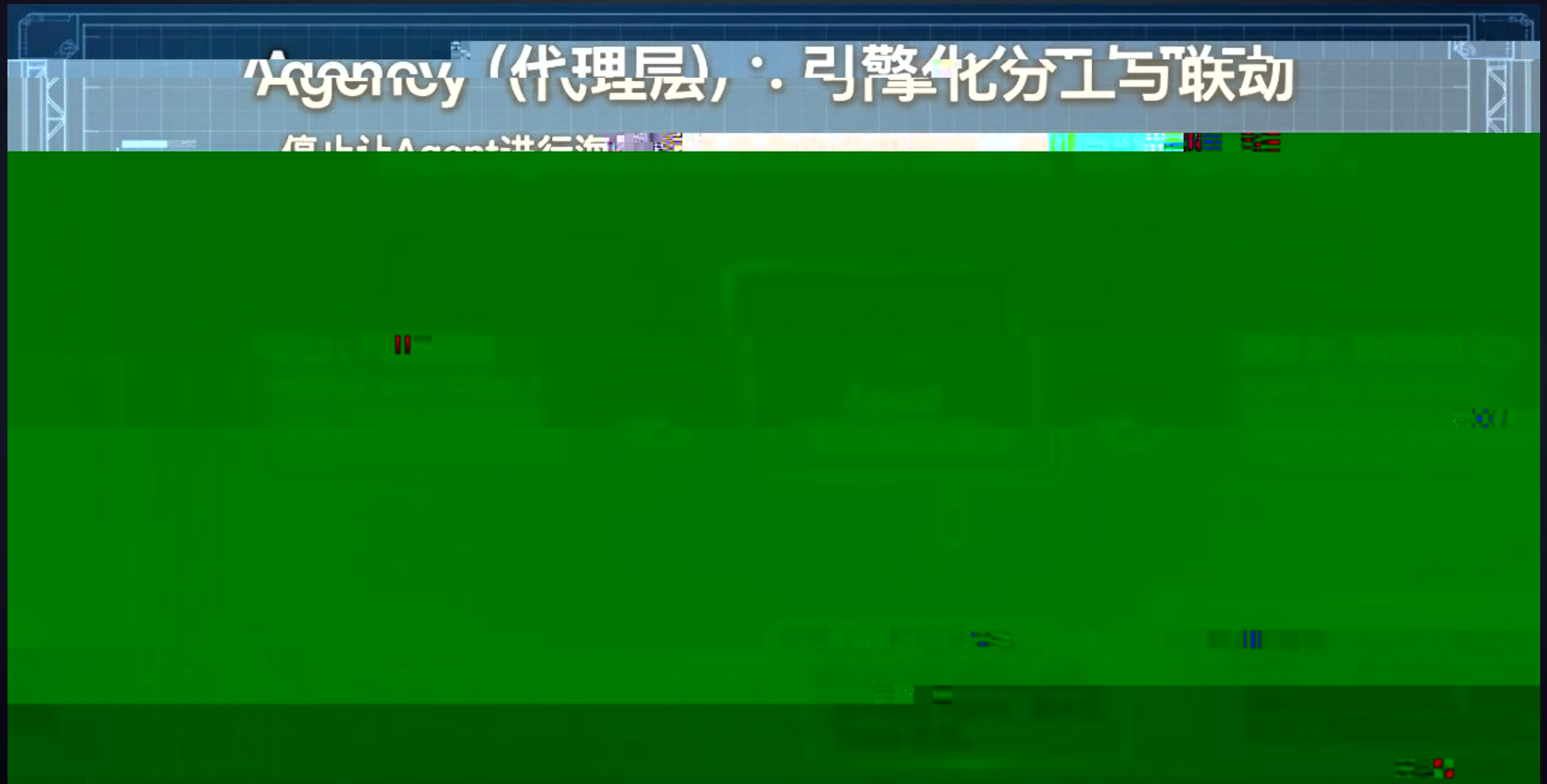
THE POWER OF COMMUNITY STARTS WITH YOU

美国2026RSAC热点研讨

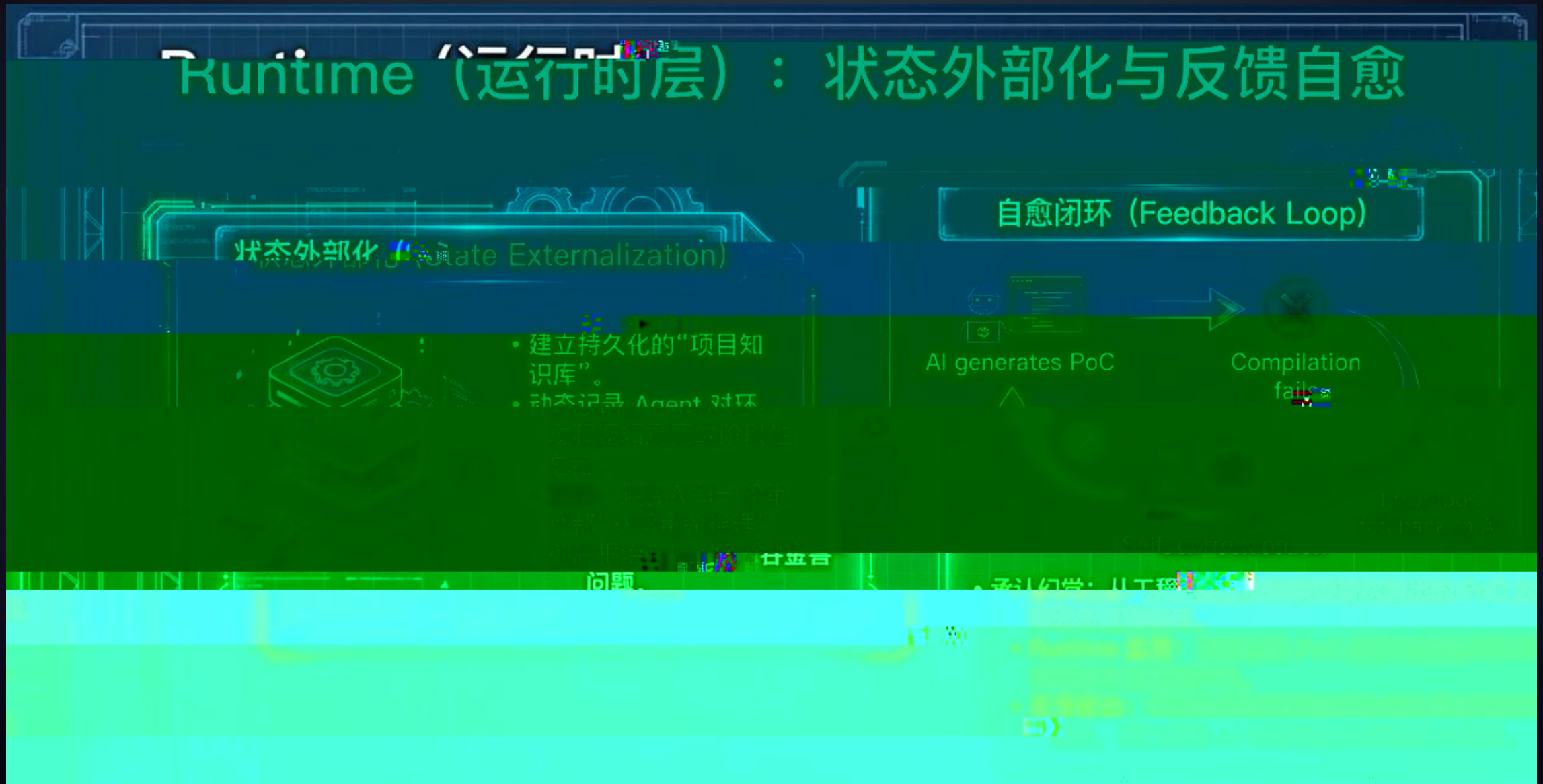
暨第十八届信息安全高级论坛

INFORMATION SECURITY FORUM 2026

CAR架构模型 - Agency 代理层



CAR架构模型 - Runtime 运行时层



PART 04

总结

总结

核心观点：Agent = Model + Harness

视角转变：不再把Agent当作独立的“专家”，而是将其作为整个“安全审计系统”中的一个特定引擎

安全审计系统的CAR架构：

Control：定义边界、引入文档、Patch

Agency：感知能力、引擎化分工、多工具（SAST、Linter、沙盒）联动

Runtime日志：确保持续化运行，状态外部化（知识库）解决成本问题，自愈闭环（回传）解决幻觉问题。

未来的核心竞争力不再是如何写好Prompt，而是构建严谨、可审计、自愈的Harness Engineering体系

THANKS!