大模型重构恶意软件检测

提示词 | 智能体 |RAG 实践

林敬勤　著

清華大學出版社
北 京

内容简介

本书系统阐述大语言模型在恶意代码检测与分析领域的应用，旨在通过 LLM 提升安全分析的效率与准确性。

全书共 11 章，结构清晰、层层递进：第 1、2 章回顾恶意软件检测技术的发展历程，并介绍 LLM 的基本原理与本地部署方法；第 3、4 章聚焦静态特征建模与 Prompt 工程，为样本处理提供标准化方案；核心实战部分（第 5、6、9 章）涵盖三大典型场景——基于 LLM 的静态特征分析、沙箱报告的自动化生成，以及基于 radare2 的逆向智能体开发，展示如何利用本地或云端模型实现自动化分析；第 7、8 章聚焦能力增强方向，引入 Agent 概念以构建简易的恶意脚本自动化分析系统，并进一步探讨检索增强生成（RAG）与多模型协作机制；第 10、11 章分析 LLM 在安全应用中的潜在风险，并展望由大模型驱动的安全自动化未来。

本书提供配套代码与模板资源，适合安全研究人员、恶意代码分析师、AI 安全方向开发者及高校相关专业学生阅读参考。

版权所有，侵权必究。举报：010-62782989，beiqinquan@tup.tsinghua.edu.cn。

图书在版编目（CIP）数据

大模型重构恶意软件检测 ： 提示词、智能体、RAG实践 / 林敬勤著. -- 北京 ： 清华大学出版社, 2026. 6. -- ISBN 978-7-302-71765-2

Ⅰ. TP393.08

中国国家版本馆CIP数据核字第20261Z42A4号

责任编辑：王中英
封面设计：杨玉兰
责任校对：胡伟民
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
配套资源下载：https://www.tup.com.cn，010-83470236
印 装 者：三河市春园印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张：**18.75　　**字　　数：**315千字
版　　次：2026年6月第1版　　**印　　次：**2026年6月第1次印刷
定　　价：89.00元

产品编号：114333-01

推荐语

从一位同样深耕恶意代码分析多年的研究者视角来看，这本书令人惊艳——一位不到 20 岁的年轻人，能从恶意代码专家分析经验中汲取养分，进而系统性地构建出由大模型驱动的新型检测框架，实属难得。书中 AutoREX 自动化逆向分析系统，将 ReAct Agent、radare2 工具与多角色 Prompt 巧妙融合，真正让大模型“动手”分析恶意样本，既有工程落地的扎实感，又充满新一代安全研究者的敏锐创造力。强烈推荐所有希望拥抱网络安全智能化的读者阅读。

诸葛建伟博士　清华大学网络研究院研究员，
蓝莲花战队共同创始人及领队

本书聚焦大语言模型在恶意软件检测与分析中的落地应用，系统串联提示词工程、智能体架构、RAG 增强与多模型协作等关键技术，并通过静态特征分析、沙箱报告生成、逆向分析 Agent 等实战项目，探索智能化安全分析的新路径。内容兼具理论深度与工程实践，适合安全研究人员、恶意代码分析师及 AI 安全方向读者阅读参考。

段钢　看雪学苑创始人

自从深度学习、大模型兴起，整个安全行业就发生了翻天覆地的变化。而行业的对手——黑灰产业者都已经先行一步，AI 技术在各种网络恶意行

为中被广泛应用。正因如此，在安全方面如不能拥抱 AI，就将完全失去前进的可能。非常高兴在这潮流兴起的时刻，能看到林敬勤的这本书，它必将是我们共同进步前行的关键助力！

谭文　《万径寻踪：Windows 入侵检测与防御编程》
《Windows 内核编程》等书作者

在智能化背景下，本书构建了一条理论到实践的链路：从恶意样本静态特征建模、Prompt 工程设计，到结合 Cuckoo 沙箱实现行为日志的自动化分析，再进一步延伸至基于 radare2、Agent 与 RAG 技术构建自主化逆向分析体系，实现了技术方法到工程落地的闭环。与近年业界在安全智能体、安全自动化以及 AI 驱动安全的趋势高度一致，具有很强的前瞻性和实践价值，同时也具有较高的教学价值，作者展示了主要设计思路，还提供了可直接复现的源码示例，使读者能够深入理解技术实现细节并进行实践验证。

姜政伟　中国科学院信息工程研究所正高级工程师，
威胁情报方向负责人

致　　谢

感谢家人在这段写作时光中的默默支持与鼓励，你们的理解与支持是我最坚实的后盾。感谢我成长路上的良师益友，你们给予了我极大的鼓励与支持。或许此刻你们正在读这段话，我想说，谢谢你们。

感谢清华大学出版社给予的机会，也特别感谢出版团队的专业支持，尤其是我这本书的责任编辑王中英老师——她在出版过程中教会了我很多，本书的顺利出版离不开她的专业指导。

本书的创作跨越了三座城市：洛杉矶、珠海和武汉。这三座热情且浪漫的城市，给予了我极大的创作动力与灵感。

最后，还要诚挚地感谢阅读本书的每一位读者。技术之路漫长，很荣幸能与各位一路同行。

前言 /PREFACE

在这个恶意软件攻防快速发展的时代，新技术不断涌现。2022 年 11 月，OpenAI 公司发布的对话式人工智能 ChatGPT 横空出世。一经问世便引起了巨大的反响。在随后的几年里，我们一同见证了大模型的迅猛发展，其强大的自然语言理解和生成能力正在重塑传统的人机交互方式。

现在，大模型正在以一种极快的速度融入千行百业之中。在计算机领域，大模型的出现不仅提升了开发效率，也深刻地影响了计算机安全领域的攻防格局。大模型的引入使得攻击手段更加多样化与智能化，防御方如果仅仅依赖于传统的规则或动静态特征识别，就难以应对新型的、由大模型驱动的动态变种攻击，包括新型 Agent 恶意软件。尤其当攻击方利用大模型实现自动化的策略调整时，二进制的攻防对抗就已逐步从传统的人机博弈转向了大模型及智能体的协同对抗的新型关系。

因此，构建与大模型相结合的二进制安全分析体系，已经成为当前难以回避的核心议题。大模型在安全领域的价值，不仅体现在其能解析恶意代码中的反汇编语义逻辑，还体现在它能利用上下文窗口自主地推断样本中的潜在威胁。

在这种能力之下，安全防御就不再局限于规则的匹配，而是逐渐演变为了一场关于攻击者攻击意图的理解与样本行为预测的博弈。

1. 本书内容与适读人群

本书由理论知识与项目实战构成，总体呈现一种线性的排布关系。因此读者按顺序阅读就可以很好地了解本书中的全部内容。与现有的安全书籍不同，本书系统性地探索了大模型技术在恶意软件检测中的全流程应用。

本书共有 11 章，各章内容如下：

- **第 1 章**：简述恶意软件检测技术的演进过程，包括从传统特征码检测到机器学习模型的转变。
- **第 2 章**：详细介绍大语言模型的基础原理及应用，并简单尝试部署一个本地模型进行测试。
- **第 3 章**：介绍 PE 文件格式及恶意样本的静态特征建模方式，这也是本书后续章节中特征建模的参考与示范。
- **第 4 章**：详细介绍大模型提示词工程（Prompt Engineering）的安全实战，包括通过不同形式的提示词实现大模型的安全领域任务。
- **第 5 章**：进入项目实战，结合前面四章的内容，通过一个简易的大模型样本静态特征分类实验，展示最基础的大模型安全应用。
- **第 6 章**：项目实战，将大模型引入动态分析领域，通过将大模型与著名的开源沙箱 Cuckoo Sandbox 结合，实现由大模型主导的自动化样本动态分析，并自动提炼生成美观的样本动态分析报告。
- **第 7 章**：引入 Agent 的概念，主要讲解大模型中的 Agent 架构与智能分析系统原理，并实现一个简单的脚本恶意软件自动化分析 Agent，为后续的增强实践做准备。
- **第 8 章**：引入对 Agent 系统的进阶 RAG 与多模型协作概念，从而增强分析系统能力，使其具备更强大的知识检索与协同推理功能。通过构建恶意软件分析领域的专属知识库，Agent 可结合实时上下文进行精准查询与结果融合。
- **第 9 章**：项目实战，结合第 7、8 章的 Agent 架构与 RAG 增强能力，结合开源 radare2 反汇编引擎，构建一个具备自主规划与工具调用能力的反汇编逆向分析 Agent。

- **第 10 章**：浅谈大模型安全应用中存在的多个维度的局限性与潜在风险。
- **第 11 章**：主要展望大模型驱动的安全自动化体系的未来发展，包括防御体系的转变以及多 Agent 分析框架等。

上述各章内容层层递进，适合绝大部分具备基础 Python 编程能力的高校学生、安全从业者、安全爱好者等群体系统地学习与实践。本书通过由浅入深的结构设计，帮助读者逐步了解并掌握大模型在恶意软件分析中的应用方法，并可在自己的实际项目中加以验证与优化。笔者也非常希望能看到各位读者在此书的基础上探索出更多创新应用并加以实践。

2. 如何高效阅读本书并动手实践

笔者在构思本书内容时就曾被一个问题所困扰，由于每位读者所使用的本地环境不同，而大模型在本地部署时非常依赖于本地性能，因此本书难以以一种通用的方法设计全部实验。在权衡后，笔者在第 5 章的项目实战之大模型判定恶意样本中对比了本地 7B 模型部署与云端大模型 API 调用的方法。后续项目实战默认使用更快捷的云端 API 调用，建议各位读者根据自身条件选择。

本书为每个实验配套了详细的代码资源，同时，也为读者准备了详尽的视频课程资源，书中难以详尽说明的安装配置步骤均包含在视频课程中。建议读者结合视频课程与代码资源同步进行学习。

同时，笔者还借助大模型为读者总结了每章的思维导图，帮助读者快速梳理核心知识点与逻辑脉络。因此在阅读各章前，不妨先看看对应的思维导图，对内容大致有个了解，然后再结合正文深入学习，可以有效地提升理解效率。

配套视频可在相应章节扫码观看，其他资源可扫描封底“本书资源”二维码获取。

3. 笔者的一些话

自萌生写作念头至今，本书历经了数次框架调整与内容迭代，其间笔者有过迷茫，也有过瓶颈，但始终没有改变笔者对技术本质的追求与分享的初心。本书的诞生，也恰逢大模型技术爆发式演进的关键时期，这让我们站在潮头之上，洞见新技术如何赋能恶意软件安全攻防。

本书中的实验与系统设计，基于笔者写作阶段主流的大模型能力与工程实践。随着模型能力的演进，部分的实现细节可能会发生变化，但书中所讨论的能力边界、系统风险及设计原则，仍具有长期的参考价值。笔者希望读者从本书中获得的不仅是技术方案，更是一种思维方式，在技术洪流中保持独立思考，在攻防对抗中把握不变的本质。面对当下这样一个快速迭代的技术环境，唯有掌握问题本质，才能以不变应万变。

最后，笔者并非完人，写作过程也是自我成长的旅程，书中或许能看到思路的逐步深化。古语有云“尽信书，则不如无书”，希望读者能以批判性思维阅读本书，敢于质疑和验证。尽管笔者已经竭尽全力反复打磨书稿，但难免有疏漏之处。若各位读者发现了书中的任何错误，恳请谅解并指正。若各位读者有宝贵的建议或想法，也欢迎随时交流。安全技术的进步正是在不断碰撞与分享中发展起来的。在技术奔涌向前的浪潮中，愿本书能成为各位读者探索中的一盏微光！

林敬勤

2026 年 4 月于珠海

目录 / CONTENTS

第 1 章　恶意软件检测技术演进 …… 1

1.1　从签名到启发式：传统技术的回顾 …… 1

1.1.1　签名式检测 …… 2

1.1.2　启发式分析检测 …… 4

1.2　迈向自动化与规模化：沙箱与云查杀的技术主导时期 …… 7

1.2.1　自动化沙箱技术 …… 7

1.2.2　云查杀与大数据协同 …… 10

1.3　AI 的引入：从传统机器学习到大语言模型 …… 11

1.3.1　传统机器学习的应用 …… 11

1.3.2　深度学习的崛起 …… 14

1.3.3　大语言模型的出现 …… 16

1.4　LLM 安全应用核心优势 …… 16

1.4.1　文本理解与生成能力 …… 17

1.4.2　模式识别与异常检测能力 …… 17

1.4.3　推理与关联分析能力 …… 18

1.5　本章总结 …… 19

第 2 章　大语言模型基础 …… 20

2.1　大语言模型概述 …… 20

2.1.1　LLM 的定义 …… 21

2.1.2 LLM 的发展历程与里程碑 …… 22
2.1.3 LLM 的核心能力与应用领域 …… 27
2.2 LLM 的核心架构与技术原理概览 …… 27
2.2.1 Transformer 架构：LLM 的基石 …… 28
2.2.2 预训练与微调 …… 31
2.3 主流开源本地模型介绍 …… 35
2.3.1 DeepSeek 系列 …… 35
2.3.2 Qwen（千问）系列 …… 35
2.3.3 Mistral 系列 …… 36
2.3.4 LLaMA 系列 …… 36
2.3.5 ChatGLM 系列 …… 36
2.4 开源 vs 商业：部署、许可、性能差异 …… 37
2.4.1 部署方式与灵活性差异 …… 37
2.4.2 性能差异与资源消耗 …… 39
2.5 开源 LLM 的本地部署 …… 39
2.5.1 硬件准备 …… 39
2.5.2 Anaconda 的安装与配置 …… 40
2.5.3 模型下载与加载 …… 41
2.5.4 模型推理测试流程 …… 42
2.5.5 pipeline 接口 …… 46
2.6 本章总结 …… 48

第 3 章 恶意样本的静态特征建模 …… 50

3.1 速通 Windows PE 文件结构 …… 50
3.1.1 PE 文件格式概述 …… 51
3.1.2 PE 文件主要组成部分 …… 52
3.1.3 关键数据目录表 …… 54
3.1.4 PE 文件的加载过程 …… 56

3.2 字符串提取、熵值、节结构与导入表 …… 58
3.2.1 字符串提取与分析 …… 59
3.2.2 熵值计算与意义 …… 63
3.2.3 节结构特征分析 …… 65
3.2.4 导入表分析 …… 66
3.3 特征提取的自动化脚本 …… 66
3.3.1 PE 文件解析 …… 67
3.3.2 静态特征提取优化策略 …… 71
3.4 样本静态特征画像 …… 72
3.5 本章总结 …… 73

第 4 章 Prompt Engineering 安全实战篇 …… 74

4.1 Prompt Engineering 基础 …… 74
4.1.1 Prompt Engineering 的定义 …… 75
4.1.2 模型采样参数 …… 78
4.1.3 常见 Prompt 类型与设计技巧 …… 79
4.1.4 恶意软件分析中 Prompt 的独特挑战 …… 87
4.2 恶意软件分析 Prompt 设计原则 …… 88
4.2.1 角色与任务设定 …… 88
4.2.2 输入输出安全与可解释性 …… 90
4.3 恶意样本分析 Prompt 模板集 …… 91
4.3.1 静态分析 Prompt 模板 …… 91
4.3.2 动态分析 Prompt 模板 …… 92
4.4 动态构造 Prompt 的自动化机制 …… 94
4.4.1 字符串与变量注入 …… 94
4.4.2 模板引擎 …… 95
4.4.3 配置化管理 …… 96
4.5 本章总结 …… 97

第 5 章　项目实战之大模型判定恶意样本 …… 98
5.1　样本静态特征提取 …… 98
5.1.1　特征的选择 …… 99
5.1.2　数据的提取与清洗 …… 100
5.2　Prompt 构造与样本输入格式标准化 …… 105
5.3　搭建分析工具链 …… 113
5.3.1　模型选择 …… 113
5.3.2　模型部署 …… 115
5.4　实现样本分析功能 …… 117
5.4.1　模块集成与流程串联 …… 117
5.4.2　样本分析功能实现与示例 …… 118
5.4.3　本地模型局限性与云端推理增强 …… 122
5.5　本章总结 …… 125
第 6 章　项目实战之大模型分析沙箱行为日志 …… 127
6.1　Cuckoo 沙箱简介 …… 129
6.1.1　什么是 Cuckoo 沙箱 …… 130
6.1.2　Cuckoo 沙箱的工作原理 …… 131
6.2　Cuckoo 沙箱安装配置 …… 132
6.2.1　环境准备 …… 132
6.2.2　Cuckoo 沙箱核心组件安装 …… 133
6.2.3　虚拟机配置 …… 136
6.2.4　Cuckoo 沙箱初始化与网络配置 …… 138
6.3　沙箱行为日志获取 …… 141
6.3.1　Cuckoo Sandbox 分析任务提交 …… 142
6.3.2　分析结果提取 …… 144
6.3.3　核心行为日志格式与内容 …… 145

6.4 Prompt 设计与模型适配 …… 147
6.4.1 样本行为画像构建 …… 147
6.4.2 行为分析 Prompt 设计 …… 156
6.4.3 模型选择与适配 …… 161
6.5 可视化报告生成 …… 162
6.5.1 可视化图表生成 …… 162
6.5.2 报告模板渲染 …… 163
6.6 本章总结 …… 165

第 7 章 Agent 架构与智能分析系统原理 …… 166

7.1 Agent 核心概念及架构 …… 167
7.1.1 传统线性分析流程的局限性 …… 167
7.1.2 Agent 概念与四大能力 …… 168
7.1.3 Agent 系统核心架构 …… 171
7.1.4 Agent 驱动的安全分析新范式 …… 173
7.2 Agent 实现机制之 ReAct 决策循环 …… 174
7.2.1 ReAct 决策循环核心原理 …… 174
7.2.2 文本解析型工具调用 …… 178
7.2.3 Function Calling 与工具集成 …… 178
7.3 Agent 基础实践之脚本程序分析 …… 181
7.3.1 工具实现 …… 181
7.3.2 Function Calling 工具注册 …… 184
7.3.3 Prompt 设计 …… 185
7.3.4 ReAct 决策循环核心实现 …… 187
7.3.5 测试与观察 …… 189
7.4 本章总结 …… 192

第 8 章　进阶探索 RAG 与多模型协作 …… 194
8.1　Agent 架构的局限性 …… 194
8.2　RAG 基础与架构 …… 195
8.2.1　RAG 核心思想 …… 195
8.2.2　典型 RAG 流程 …… 198
8.2.3　RAG 的安全分析应用 …… 204
8.2.4　RAG 具体实现方式 …… 204
8.3　多模型体系与模型路由 …… 208
8.3.1　什么是多模型体系 …… 209
8.3.2　模型路由机制 …… 210
8.4　多模型及多角色协作 …… 212
8.4.1　单模型多角色协作 …… 212
8.4.2　多模型多角色协作 …… 213
8.5　本章总结 …… 213
第 9 章　反汇编逆向 Agent 实战 …… 215
9.1　项目简介与设计目标 …… 215
9.2　系统架构设计 …… 216
9.2.1　系统总体结构设计 …… 217
9.2.2　系统数据流与调用链 …… 218
9.3　radare2 工具接口封装 …… 219
9.3.1　radare2 工具安装 …… 220
9.3.2　radare2 分析指令 …… 220
9.3.3　工具封装实现 …… 223
9.3.4　工具文档与注册 …… 227
9.3.5　项目工具列表 …… 229
9.4　Prompt 与多角色协作机制的设计 …… 230
9.4.1　ReAct 模板设计 …… 231

9.4.2 反汇编函数代码清理角色设计 …… 234
9.4.3 工具文档生成与注入 …… 236
9.5 Agent 的核心控制逻辑 …… 238
9.5.1 ReAct 主循环与上下文 …… 239
9.5.2 Action 解析与工具调度 …… 241
9.5.3 反汇编清理角色流程 …… 242
9.5.4 小结 …… 243
9.6 实验运行与测试 …… 244
9.6.1 实验环境准备 …… 244
9.6.2 启动 AutoREX 进行样本分析 …… 246
9.6.3 ReAct 推理循环示例 …… 246
9.6.4 总结报告查看 …… 249
9.7 本章总结 …… 252

第 10 章 浅谈大模型安全应用局限与风险分析 …… 253

10.1 模型能力局限 …… 254
10.1.1 语义理解局限 …… 254
10.1.2 推理稳定性局限 …… 255
10.1.3 知识覆盖局限 …… 256
10.2 系统层风险 …… 257
10.2.1 上下文复杂性 …… 258
10.2.2 推理过程风险 …… 258
10.2.3 推理成本与性能限制 …… 259
10.3 攻防中的对抗风险 …… 260
10.3.1 模型利用攻击 …… 261
10.3.2 模型欺骗与逃逸 …… 263
10.4 浅探可控分析体系 …… 267
10.4.1 外部安全限制 …… 267

10.4.2 内部结构化设计 …… 269
10.4.3 独立验证与审计 …… 269
10.5 本章总结 …… 270

第 11 章 大模型驱动的安全自动化体系展望 …… 272
11.1 从特征匹配到意图理解 …… 272
11.1.1 规则驱动范式 …… 272
11.1.2 数据驱动范式 …… 273
11.1.3 认知驱动范式 …… 273
11.1.4 工程实践与融合架构 …… 274
11.2 从单 Agent 到多 Agent 智能体协作 …… 275
11.2.1 单 Agent 系统局限 …… 275
11.2.2 多 Agent 系统架构 …… 276
11.3 大模型下的新攻防关系 …… 279
11.4 人机协同的安全体系 …… 280
11.5 本章总结 …… 281

结语 …… 282

第 9 章

反汇编逆向 Agent 实战



第 8 章介绍了可以动态增强 Agent 知识的主要技术——RAG 及多模型协作架构，还尝试实现了一个简单的大模型 + RAG 问答程序，展示了 RAG 技术如何提升大模型的回复质量。

在本章中，我们将把这些能力应用到真正的恶意样本分析场景中。本章将基于一款轻量级的开源反汇编引擎 radare2，构建可调用的底层分析工具链，并使用大模型作为系统的核心“大脑”，让模型能够自主规划分析步骤、调用工具、解释结果，最终完成对未知样本的行为推断与报告生成。

最终，我们将实现一个具备自主规划、工具协作、行为链构建能力的逆向工程 Agent。

9.1 项目简介与设计目标



与前几章的实战内容相比，本章的项目规模更大、功能更完善，其核心思想是通过工具链 + 策略层的方式，让模型能够像一名真正的逆向工程分析师那样工作。

传统的逆向流程往往依赖于人工经验的判断，例如，作为一位逆向工程分析师，工作流程可能是这样的：

定位程序入口点 → 检查字符串、导入表等基础信息 → 分析并推测样本行为 → 分析关键函数的反汇编 → 汇总报告

这一过程高度依赖人工经验，且步骤复杂，包含大量的重复性工作。而大模型的出现显然给予了这种场景自动化实现的可能。因此，本章的项目实战的核心目标就是将本书前面介绍过的技术（ReAct 决策方式、工具调用、RAG 增强、多角色协作等）整合起来，实现一个真正可以自主完成逆向分析流程的智能体。

围绕这一个核心目标，本章的主要任务分为以下三个部分：

- **将大模型融入传统的恶意软件逆向分析工作流中**：大模型在分析工作流中将作为能分析、能决策、能调用工具、能根据结果调整思路的专业分析师。
- **构建多技术融合的智能分析体系**：通过多种技术的融合匹配，提升系统的分析能力。其中包括 ReAct 决策循环、工具调用与参数规划、RAG 逆向知识增强、多角色协作等。
- **分析全流程自主化、自动化**：从加载样本，到逐步展开静态分析，再到行为链的重构与最终的分析报告生成，所有步骤均由智能体自主完成，从而最大化提升逆向分析过程的智能化与自动化水平。

9.2 系统架构设计



在明确了项目的主要任务与设计目标后，不仅需要从整体视角规划系统的架构布局，还要明确它们在智能分析流程中的职责分工与交互方式。因此，本节将分别从系统总体的宏观结构以及数据流与调用链两方面展开，详细介绍本章项目的整体架构设计思路。

9.2.1　系统总体结构设计

本项目采用三层结构进行设计，包含工具层、推理层及知识层，如图 9-1 所示。

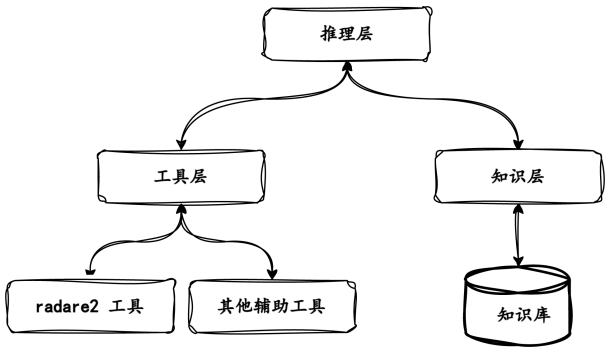


图 9–1　系统层级设计划分

其中，各层之间可以通过明确的接口和数据格式进行交互，共同构成一个可调度、可解释、可扩展的自动化逆向分析体系：

- **工具层**（Tool Layer）：工具层为系统的基础层，负责与样本进行直接的交互。因此这一层需要做好反汇编逆向工具接口的封装与调用约定，其中也包含其他的辅助工具，例如报告保存、RAG 查询等工具。
- **推理层**（Inference Layer）：推理层是系统的核心决策单元，这一层需要做的是处理系统中有关大模型推理调用的问题。其中包括任务的规划（也就是 ReAct 决策循环）、工具的调用、返回数据的解释等工作。推理层不会直接处理样本，而是通过思维链 + 工具调用的方式来驱动整个分析的过程。
- **知识层**（Knowledge Layer）：知识层用于弥补模型在逆向领域专业性不足的问题，该层将主要由 RAG 技术进行构建，并且使用模型主动调用方式，而非被动注入，因此当推理层遇到需要专业解释的问题时，模型可自主查询知识进行增强回复，从而减少因知识缺失而造成的分析误判问题。

本章项目的实现将会通过上述三个核心层的紧密协同工作完成。



注意：在第 7 章介绍过基于 OpenAI function calling 的 Agent 实现方式，但本章项目将采用文本解析型的架构，原因主要在于后者的兼容性更强。部分开源或国产模型并不支持 function calling，而通过文本解析则几乎适用于所有的模型，只需在 Prompt 中指导模型按格式输出即可。这一点相信大家在测试第 7 章的实验时也有所体会。其次文本解析架构的 Agent 流程更灵活，本章将使用多步骤多角色分析流程更适合让模型自由地规划分析步骤并自主协同各角色完成样本的深度判研。

9.2.2 系统数据流与调用链

在设计好系统的整体结构后，需要对其中的数据流和调用链进行设计。整个逆向分析智能体的运行可以被视为一个连续的决策循环，模型不断通过调用工具获取信息，并在循环中自行推理并分析样本。

本章项目整个系统的数据流与调用链如图 9-2 所示。

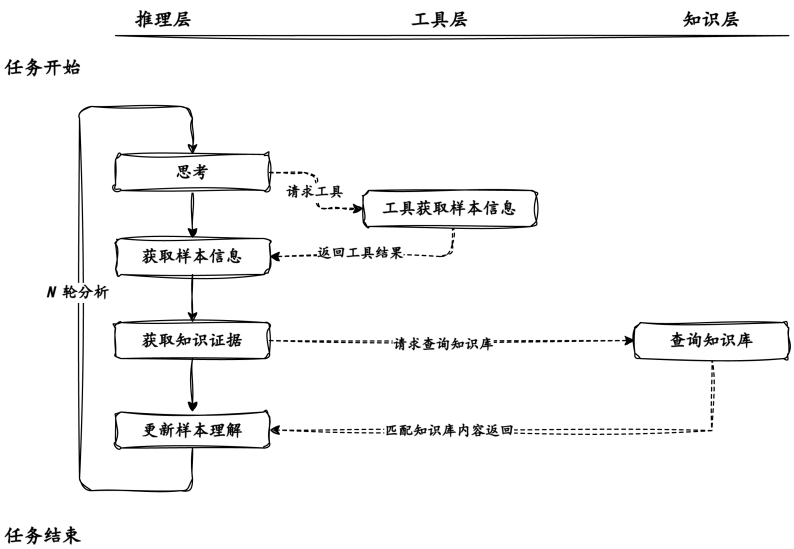


图 9-2 项目数据流与调用链示意图

其核心流程包括以下 5 个阶段：

（1）**分析任务初始化**：系统在接收到样本路径后，反汇编引擎会打开样本，而推理层也会收到其初始输入，但此时推理层尚未收到包含样本的任何信息。

（2）**推理层动作请求**：推理层将会基于当前的任务状态，决定第一步需要获取的技术信息，例如获取样本的基本结构等信息；此时推理层会输出一条动作指令，例如 get_imports 等。

（3）**动作执行与结果返回**：工具层在接收到推理层的动作指令后，会通过封装好的 radare2 接口执行对应的操作，例如提取样本导入函数列表等，并将这些数据原样或清理后返回给推理层。

（4）**结果解释与推理链更新**：推理层在接收到工具返回的结果后，会对其进行解释与推断，当推理层在解释过程中遇到知识空缺时，则会向知识层请求补充信息。最后推理层将会基于上述信息更新推理链路，并指导下一步的动作请求。

（5）**样本结果与报告生成**：步骤（2）至步骤（4）将会循环多轮，直到推理层认为证据充分或样本意图明确时，推理层会整理样本的关键行为链、函数、核心逻辑等证据，最终跳出分析循环并生成分析报告。

9.3　radare2 工具接口封装



在构建智能分析系统之前，首先需要为智能体准备一款可靠的逆向分析引擎。本章中所有反汇编与样本的静态解析能力，均基于轻量级的开源工具 radare2 实现。

相比 IDA、Ghidra 等大型的桌面工具，radare2 更轻量、启动快且具备较强的脚本化优势。并且通过其 Python 接口 r2pipe，可以方便地将函数列表、反汇编、导入表、字符串等分析操作封装为可供直接调用的工具，从而使模型具备直接操作样本分析的能力。

9.3.1 radare2 工具安装

radare2 是一款开源分析工具，你可以在 GitHub 中找到该项目（https://github.com/radareorg/radare2），并下载该项目预编译的 Windows 安装包，同样也可以在本书所提供的配套资源包中找到对应文件（resource/C9）。

下载好后，可以在目录 radare2-6.0.6-w64\bin 下运行命令 r2 –v，验证是否可以使用，当输出 radare2 的版本信息时即可。

9.3.2 radare2 分析指令

radare2 的核心是基于命令的分析流程。由于本书并不会全面讲解 radare2 的使用方法，因此本节将列出本项目中最常用的基础指令，读者只需掌握这些指令即可支持后续实验。

其 radare2 常用分析指令如表 9-1 所示。

表 9–1 radare2 常用分析指令

指令	功能说明
r2 <file>	打开样本文件
aaaa	执行深度自动分析
afl / aflj	列出函数列表
pdf @ <func_or_addr> / pdfj @ <func_or_addr>	查看指定函数的反汇编
iz / izj	获取字符串
ii / iij	获取导入表
iS / iSj	查看节区信息
s <func_or_addr>	跳转到指定函数或地址
pd 20	查看当前位置起 20 条指令



提示：如果在指令后加上字母 j，radare2 会以 JSON 格式返回数据，否则以纯文本形式返回。

接下来我们尝试先使用 radare2 控制台以指令的形式对样本进行基础分析。

1. 打开样本并执行自动分析

首先使用 r2 指令打开样本，并通过 aaaa 指令进行深度自动分析，如代码 9-2 所示。

代码 9-2　打开样本并执行自动分析

```
r2 sample.exe
[0x00400000]> aaaa
```

其中，r2 sample.exe 用于加载目标二进制文件，而 aaaa 指令会执行深度分析，包括符号识别、函数边界识别及控制流分析等，这是后续所有操作的基础。

2. 查看样本的函数列表

在 radare2 加载完样本并执行自动分析后，可以尝试使用 afl 或 aflj 指令获取样本的函数列表，其中指令 afl 会返回纯文本的函数列表，如代码 9-3 所示。

代码 9-3　afl 指令获取样本纯文本格式函数列表

```
[0x1400112b7]> afl
0x1400112b7   23    379 entry0
0x1400120b0    8    269 main
0x1400113ed    6     65 fcn.1400113ed
0x140011028    2     50 fcn.140011028
0x1400110a5    5     83 fcn.1400110a5
0x1400111bd    2    165 fcn.1400111bd
......
```

而指令 aflj 则会返回 JSON 格式的函数列表，如代码 9-4 所示。

代码 9-4　aflj 指令获取样本 JSON 格式函数列表

```
[0x1400112b7]> aflj
[
  {"name": "entry0","addr": 5368779447,"size": 7555},
  {"name": "main","addr": 5368783024,"size": 269},
```

```
  {"name": "fcn.1400113ed","addr": 5368779757,"size": 5903}
  ......
]
```



提示：上述 JSON 结构为示例化简格式。实际的 radare2 输出会包含更多字段，涵盖基本块、控制流、调用关系、栈结构、局部变量、指令数量等内容。但在本项目中主要使用其中的 name、addr、size 三个字段。

3. 查看函数 main 的反汇编内容

获取样本函数列表后即可使用 pdf 指令获取对应函数的反汇编代码，如代码 9-5 所示。

代码 9–5　pdf 指令获取指定函数反汇编代码

```
[0x1400112b7]> pdf @ main
            ; CODE XREF from fcn.1400112b2 @ 0x1400112b2(x)
/ 269: int main (int argc, char **argv, char **envp);
| afv: vars(5:sp[0x38..0x170])
|           0x1400120b0    4055         push rbp
|           0x1400120b2    57           push rdi
|           0x1400120b3    4881ec3801.. sub rsp, 0x138
|           0x1400120ba    488d6c2420   lea rbp, [var_20h]
|           0x1400120bf    488d7c2420   lea rdi, [var_20h]
|           0x1400120c4    b916000000   mov ecx, socket ; 0x16 ; 22
......
```

其中 @ 后紧跟着的为函数名或函数地址，radare2 会自动跳转到该函数的入口地址并输出对应的反汇编代码。同样也可以使用 pdfj 指令输出更易解析且详细的 JSON 格式内容。

4. 提取样本中的字符串

通过 iz 或 izz 指令可以提取样本中的字符串信息。代码 9-6 展示了使用 iz 指令提取字符串的输出结果。

代码 9-6　使用 iz 指令提取样本中的字符串

```
[0x1400112b7]> iz
nth paddr      vaddr       len size section type    string
-------------------------------------------------------------
0   0x00008fb0 0x140019bb0 8   9    .rdata  ascii   _ArgList
1   0x00009030 0x140019c30 8   9    .rdata  ascii   _ArgList
2   0x000090b0 0x140019cb0 4   5    .rdata  ascii   pids
3   0x000090b8 0x140019cb8 6   7    .rdata  ascii   needed
4   0x000090c0 0x140019cc0 8   9    .rdata  ascii   suspects
5   0x000090d0 0x140019cd0 5   6    .rdata  ascii   bytes
6   0x000090d8 0x140019cd8 4   5    .rdata  ascii   name
7   0x000092d0 0x140019ed0 7   8    .rdata  ascii   wsaData
8   0x000092d8 0x140019ed8 5   6    .rdata  ascii   hints
9   0x000092e0 0x140019ee0 8   9    .rdata  ascii   addrInf
```

以上四个示例仅仅展示了 radare2 反汇编引擎在手工分析场景中的几项基础能力。读者可以自行探索其他指令。但是在本章的项目中，读者并不需要熟记所有命令，也不需要频繁地手动分析。接下来将通过 Python r2pipe 库对上述指令进行统一封装，使其成为可供直接调用的工具接口。

9.3.3　工具封装实现

在了解了 radare2 的基础指令后，本节将展示如何将这些指令封装为可供模型调用的工具接口。由于模型无法直接执行控制台命令，因此需要使用 Python r2pipe 库对这些操作进行统一的封装，使所有的底层分析都能够以 Python 函数的形式供智能体进行调用。

在这个过程中，Python 与 radare2 反汇编引擎之间的通信由 r2pipe 库完成。通过 r2pipe.open() 方法可启动一个 radare2 分析会话，而通过 r2.cmd() 方法即可向 radare2 控制台发送一条命令并获取其分析结果，如代码 9-7 所示。

代码 9-7　r2.cmd() 方法执行分析指令

```
import r2pipe
self.r2 = r2pipe.open(self.sample_path, flags=["-2"]) # 打开样本
self.r2.cmd("aaaa")                # 执行自动深度分析，否则后续命令无法执行
```

```
self.r2.cmd("aflj")              # 获取函数列表（JSON）
self.r2.cmd("pdfj @ main")       # 获取 main 函数的反汇编（JSON）
self.r2.cmd("izj")               # 获取字符串（JSON）
```

其中，cmd() 的返回值始终为字符串，因此对于分析指令获取的 JSON 结果，需要再使用 json.loads() 将输出转换为可供解析的 Python 数据结构。这也是工具层能够向智能体提供结构化数据的关键能力。

在工具的封装过程中，还需要重点关注以下几点：

- **屏蔽底层命令**：智能体在实际执行工具的过程中，不需要知道 radare2 的具体指令，只需要调用统一的函数，例如模型调用 get_strings() 工具即可获取样本的字符串列表。
- **统一输出格式**：所有工具返回的内容必须是可被模型理解的文本格式（例如 JSON）。
- **可扩展性**：工具层应允许轻松增加新的工具（例如节区分析、调用图提取等）。

接下来将以本章项目内的函数调用图（CallGraph）提取工具作为示例，展示如何通过 r2pipe 库封装 radare2 的多条分析指令，并将分析结果转换为统一的结构化形式。

其中，该工具会对样本中每个函数进行分析，提取其内部的调用目标，从而构建出一个简化的调用关系图，其核心流程如下：

（1）遍历样本函数列表。

（2）提取每个函数内部的 call 指令。

（3）分析每个 call 的目标地址。

（4）最终构建出统一格式的调用关系条目，也就是 CallGraph。

其中，CallGraph 工具的设计可以分为两个部分，如代码 9-8 所示。

代码 9-8　CallGraph 工具

```
def _get_calls(self, addr):
    ...
def get_core_callgraph(self):
    ...
```

其中 _get_calls() 为内部函数，用于分析指定函数的反汇编内容，并提取所有 call 指令跳转的目标地址，而 get_core_callgraph() 则负责遍历所有函数，并汇总最终结果。

首先来看一下内部函数 _get_calls() 的实现过程，如代码 9-9 所示。

代码 9-9　_get_calls() 实现

```
    def _get_calls(self, addr):
        try:
            self.r2.cmd(f"s {hex(addr)}")
            ops = json.loads(self.r2.cmd("pdfj") or "{}")

            calls = set()
            for op in ops.get("ops", []):
                    if op.get("type") == "call" and isinstance(op.
get("jump"), int):
                    calls.add(hex(op["jump"]))
                    if len(calls) >= 20:
                        break

                        calls = sorted(calls, key=lambda x: int(x, 16))
                        return calls if calls else None
        except:
            return None
```

在函数中，仅用到了两条 radare2 的分析指令，分别是 s 和 pdfj 指令。

首先函数接受一个地址，接下来通过 radare2 的 s <addr> 指令将 radare2 的分析位置（seek）跳转到该地址。接着，使用 pdfj 指令获取该函数的完整反汇编信息，并将其以 JSON 格式解析并存储至变量 ops 中。

在获得函数反汇编的指令列表后，可以通过遍历 ops["ops"] 中的每条反汇编指令，检查其 type 字段是否为“call”，同时确认 jump 字段是否为有效的跳转地址，并将满足这两项条件的跳转目标加入 calls 集合中。这里使用集合结构可以自动去重，同时限制 calls 的最大长度，从而避免因函数过大导致输出过长的问题。

最后，对收集到的调用地址按数值大小进行排序，并以十六进制字符串的形式返回。若未找到任何调用指令，则返回 None 即可。

在完成了对单个函数内部调用关系的提取之后，下一步就是将其应用到整个样本函数的分析中。也就是说我们需要遍历样本中所有被识别出的函数，并逐一调用 _get_calls() 函数获取其调用目标，从而构建出完整的 CallGraph，而这一部分逻辑由外部函数 get_core_callgraph() 实现，该函数最终也将成为提供给推理层的工具之一，其整体流程如代码 9-10 所示。

代码 9-10　get_core_callgraph() 实现

```
    def get_core_callgraph(self):
        if not self.r2:
            return ["错误：请先 open_sample()"]
        try:
            funcs = json.loads(self.r2.cmd("aflj") or "[]")
            items = []

            # 读取并记录函数调用关系
            for f in funcs:
                addr = f.get("addr")
                name = f.get("name")
                calls = self._get_calls(addr)
                if not calls:
                    continue
                  items.append({"name": name,"addr": hex(addr),"calls":
calls})

             items = sorted(items, key=lambda x: int(x["addr"], 16))
              lines = [json.dumps(it, separators=(",", ":")) for it in
items]
             return lines

        except Exception as e:
            return [f"get_core_callgraph 失败：{e}"]
```

首先，函数通过 aflj 指令获取 radare2 自动分析后的函数列表 funcs。与前面的工具封装一致，使用 JSON 格式输出便于后续的解析与结构化处理。

接着解析 funcs 并提取每条函数的函数名与地址，紧接着调用前面提到的 _get_calls() 函数提取对应函数调用的所有目标地址列表。

最后使用 sorted 方法按照函数地址递增进行排序，即可得到样本内每个函数的 CallGraph 调用关系，如代码 9-11 所示。

代码 9-11　get_core_callgraph() 工具运行示意

```
    [
      { "name": "entry0", "addr": "0x1400112b7",
          "calls": ["0x140011055","0x140011087","0x14001114f","
0x140011168", ...] },

      { "name": "main", "addr": "0x1400120b0",
          "calls": ["0x140011159","0x1400111bd","0x1400112f3","
0x140011375", ...] },
      ...
    ]
```

上述结果展示了样本中每个函数的调用目标，本工具的实现为推理层识别样本关键逻辑路径、分析核心行为链和定位样本核心功能提供了结构化的支持。

9.3.4　工具文档与注册

在准备好工具函数后，需要让推理层能够正确识别、理解并调用工具。因此，需要给工具编写文档描述并注册工具，以便大模型在推理过程中根据工具说明自动选择合适的分析动作。

在本项目中，每个工具都需要配套一段简要的文档描述，用于告诉大模型以下四个有关工具的核心知识：

- **工具名**：大模型将会直接以该名称调用工具。
- **工具的核心功能**：简要描述工具的用途，以便模型按需选择合适的工具。
- **工具输入提示**：如果工具包含输入参数，则还需要解释工具的输入，模型才能正确地传入参数并调用工具。
- **工具返回格式**：这部分需要说明工具会以何种结构返回结果，例如纯文本、JSON 或者特定的键值格式，以便模型在 ReAct 推理阶段正确地解析工具参数。

例如前面提到的 get_core_callgraph() 工具的文档就如代码 9-12 所示。

代码 9-12　get_core_callgraph() 工具文档

```
    self.tools_list = [
    ("get_core_callgraph","get_core_callgraph(void): 返回紧凑版核心调用图，
每个函数一行 JSON，字段为 {name, addr, calls[]}。")
    ]
```



实际项目源码的 tools_list 中还会包含更多的工具文档，这里只展示了 get_core_callgraph() 一种工具。其中文档中清晰地描述了工具名（get_core_callgraph）、工具核心功能（返回紧凑版核心调用图）、工具输入（void）及工具返回的格式内容（{name, addr, calls[]}）。

通过上述文档的描述，模型可以更清晰自由地选择合适的工具，同时以正确的方式解析工具返回的结果。

在第 7 章末尾的实验中，曾使用了简单的 if 判断进行工具的调用，而随着工具的增多，显然使用 if else 并不是明智之举，因此在本项目中可以通过维护一个工具映射表进行管理，该表会将工具名称直接映射至对应的实现方法。

在工具文档准备好后，只需要将工具统一注册到工具映射表中，即可让推理层在后续的 ReAct 循环中通过工具名称直接调用底层函数。

还是以 get_core_callgraph() 工具为例，只需要将函数名称与对应的实现录入工具映射表中即可，如代码 9-13 所示。

代码 9-13　get_core_callgraph() 工具映射（注册）

```
    mapping = {
    "get_core_callgraph": self.tools.get_core_callgraph
    }
```

在工具注册完成后，推理层即可通过“get_core_callgraph”调用底层实现 self.tools.get_core_callgraph() 方法，从而实现底层能力的统一调度。

9.3.5　项目工具列表

前面已经以函数工具 get_core_callgraph() 为例，展示了从 radare2 接口的封装、工具文档的编写到工具的映射的完整流程。而实际上，本项目按照上述方法封装了六个工具，其他的工具封装详见章节资源包（C9）。表 9-2 中给出了本章项目所实际使用的工具列表。该表总结了每个工具的名称、输入参数、核心功能及返回格式，以便读者快速理解本项目工具层的整体能力结构：

表 9-2　项目工具列表

工具名称	输入参数	核心功能说明	返回格式
open_sample	void	打开样本并执行 radare2 深度分析（aaaa），为后续所有工具准备分析上下文	文本提示
list_functions	void	返回样本中所有函数的简要信息（name / addr / size）	JSON 列表
get_disasm	name	获取指定函数的反汇编文本（最多输出前 80 行）	文本字符串
get_strings	void	获取样本中长度 ≥ 6 的字符串列表，用于特征提取和 IOC 提取	JSON 列表
get_imports	void	返回样本导入表（DLL:API），用于识别关键功能依赖	JSON 列表
get_core_callgraph	void	构建样本的函数调用图，提取每个函数 call 的跳转目标	JSON Lines
save_report	filename, content	保存模型生成的最终分析报告	JSON
retrieve_knowledge	query	查询外部知识库，用于逆向知识的增强	文本知识片段

上述这些工具为系统的推理层提供了丰富的底层分析能力，使得大模型能够逐步逆向分析样本，并拼凑出其潜在的行为意图链条。

至此，工具层已经具备了支撑自动化逆向分析的基础能力。接下来需要将注意力放在系统的核心推理层中。

9.4 Prompt 与多角色协作机制的设计



在构建智能体的推理层时，Prompt 设计是核心的组件之一。由于样本的逆向分析本身就是一个高度结构化、步骤化的任务，因此仅仅依靠单一角色的 Prompt，通常无法让模型高质量地完成分析过程中的多阶段推理。

本章项目中将运用到多角色协作机制，通过给模型赋予多个职责不同的角色，使其能够在推理过程中更清晰地分工，更稳定地完成复杂的逆向分析流程。

在本章项目中，设计了两个角色，它们主要承担了如下职责：

- **恶意样本逆向分析师**：该角色为项目的核心角色，通过 ReAct 决策循环指导整体分析方向，因此主要用于统筹调度整体的分析流程。
- **反汇编代码清理角色**：该角色主要用于对获取到的反汇编代码进行清理工作，并按原始逻辑返回更加清晰的 C 语言风格伪代码。

上述两个角色将通过协同工作对样本进行分析，其协同流程如图 9-3 所示。

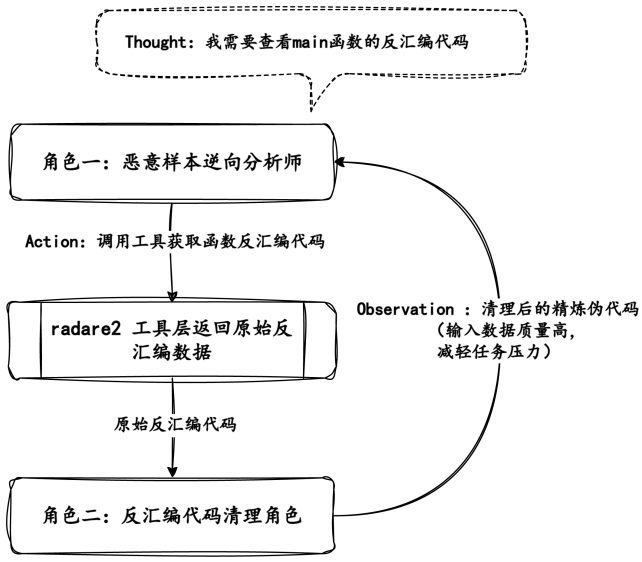


图 9-3　项目双角色协同流程图

通过将烦琐的任务进行角色化拆分，可以避免单个模型既要规划、又要解释、还要写报告的问题，从而使系统结构更加清晰，模型推理质量更高。

9.4.1　ReAct 模板设计

在设计模板之前，我们先回顾一下 ReAct 决策循环的核心流程：通过 Prompt 的设计，模型可以以思考（Thought）—动作（Action）—观察（Observation）的循环形式推进分析过程。而在逆向分析任务中，这种结构具有天然的优势，模型可以循序渐进地通过工具和已有知识探索样本，而不是一次性地输出笼统无关的结论。

接下来将从项目核心角色（恶意样本逆向分析师）开始，逐步介绍如何在项目中设计可靠的 Prompt，以引导模型正确地调用工具进行分析。

- **动态信息部分**：包含模型角色定义、模型核心目标及可供使用的工具列表，这部分内容随时可以改变。
- **分析原则**：这部分内容包含模型在分析过程中需要遵守的分析原则，同时需要指导模型的大致分析流程。
- **工具调用规则**：严格约束模型在工具调用过程中的格式、语法与执行逻辑。
- **禁止事项**：设定模型行为边界，防止其在逆向分析中产生幻想或偏离分析范围。

1. 角色定义与任务说明

一个优秀的 Prompt 开头通常都跟随清晰明确的角色定义、任务说明及可用的工具列表（例如前面的函数列表、反汇编、导入表、字符串、调用图等工具），这里仍然使用 jinja2 的填充功能，为后续的动态填充留足空间，如 Prompt 分块 1 所示。

Prompt 分块 1　角色定义与任务说明

```
{{ role }}

你的目标：
```

```
{{ task }}

你可以调用以下工具：
{{ tools }}
```

这三个部分的动态填充，可以明确模型的能力边界，让模型知道可以通过哪些工具获取证据，从而减轻模型幻想不存在的工具的问题。工具列表可以根据项目的发展动态地调整，这也是一开始就要使用 jinja2 进行 Prompt 管理的原因，越往后，任务越复杂，动态处理 Prompt 的优势就越凸显，可以使系统适应不同的样本、不同的分析阶段与不同的分析目标。

2. 分析指导与原则

这一部分需要补充任务的细节，包含核心分析任务的流程指导及必要规则的说明，这也是模板最核心的部分之一，它定义了模型在逆向分析期间必须遵守的推理逻辑，如 Prompt 分块 2 所示。

Prompt 分块 2　分析原则

```
【分析原则】
1. ** 基于证据分析（不能幻想）**
   - 不能凭空生成 API、字符串、CFG、调用链或反汇编。
   - 所有结论必须由工具返回的内容支撑。

2. ** 只分析关键函数（不是遍历所有）**
   你必须根据以下线索选择 2 ～ 6 个关键函数进行深度分析：
   - 尝试从程序入口处进行分析。
   - 出现在 get_core_callgraph 中的核心调度节点。
   - 与网络、配置、反沙箱、加密相关。
   - main、entry0、特殊命名。

3. ** 必要工具的正确使用（严格）**
   - get_disasm(funcname) 中的 funcname 必须来自 list_functions 的返回。
   - 不得推测或构造函数名和地址。
   - get_strings：用于确认配置、域名、Key、路径。
   - get_imports：用于推断行为模块。
   - get_core_callgraph：用于定位行为主链。

4. ** 通过证据构建行为链（Behavior Chain）**
   你需要根据工具结果以及 retrieve_knowledge 查询的相关知识（确保分析过程中的关键节点主动调用 retrieve_knowledge 工具并在报告中标记）推断以下可能行为：
   - 反沙箱与环境检查。
```

```
- C2 / 配置加载。
- 键盘记录 / 输入监控。
- 网络通信。
- 文件 / 注册表操作。
- 进程注入或持久化。

5. ** 报告生成规范 **
   当你认为样本已经足够清晰，细节已经足够明确时：
   - 使用 save_report 工具以中文保存包含完整细节的 markdown 格式样本报告。

6. ** 结束分析 **
    当分析结束时，输出 Final Answer: 标签，并在其后面跟上你的分析过程摘要，例如：
    " 在本次样本分析中，我首先观察了……"
```

通过 Prompt 分块 2 中的六条分析指导原则，向模型明确了逆向分析的整体工作方式。在 Prompt 中要求模型从入口点和关键节点入手，严格依赖工具的输出进行分析推理，接着按照每一步中收集到的证据构建样本的潜在行为链，最终生成结构化报告并使用工具保存。

这些细化的规则不仅可以限制模型的出错空间，还能有效地提升模型分析推理路径的稳定性与可控性，使其所表现出来的行为更接近真实逆向工程师的分析过程。

3. 工具调用规范

这一部分主要用于规范模型在 ReAct 循环中的工具调用格式。通过 One-Shot 方式统一格式，不仅便于系统解析模仿，也能防止模型输出错误的动作指令导致推理中断。尤其在涉及多个工具、多步骤推理时，保持 Action 以及 Action Input 的一致性至关重要，如 Prompt 分块 3 所示。

Prompt 分块 3　工具调用规范

```
【ReAct 工具调用规范】

每一次工具调用都必须严格使用以下格式：

Thought: 说明你为什么需要调用该工具。
Action: 工具名
Action Input: JSON 参数
```

```
示例：
Thought: 为了查看入口点逻辑，我需要检查 main 的反汇编。
Action: get_disasm
Action Input: {"funcname": "main"}
```

4. 输出提示

在 Prompt 的最后，还需要补上一小段输出提示，为模型建立并强调明确的结束条件，防止模型在思考时陷入死循环或者忘记生成最终的报告。

在自动化分析场景中，明确的任务终止标志非常重要，其不仅可以使分析流程完成闭环，还能够提示模型在完成分析后所要执行的收尾动作，例如保存报告及输出分析摘要等，如 Prompt 分块 4 所示。

Prompt 分块 4　输出提示

```
现在，请按照上述流程执行完整的 ReAct 分析。
分析结束时，务必使用 save_report 保存你的最终报告，并在 Final Answer 内输出分析摘要。
```

Prompt 分块 4 所展示的，既是分析任务的执行信号，同样也标识了 Prompt 的正式结束位置。所以，从这一行开始，模型将按照既定的 ReAct 流程进入推理阶段，逐步调用工具并进行样本的分析。

9.4.2　反汇编函数代码清理角色设计

由于本章的项目中使用了多角色的协同工作，因此在设计完系统核心的 ReAct Prompt 后，还需要处理系统中反汇编函数代码清理角色的 Prompt。

之所以项目中要使用多角色的方案，是因为在自动化逆向分析流程中，模型在通过工具获得反汇编或伪代码内容后，往往需要对这些内容进行进一步的结构化整理，以更好地理解函数的核心逻辑。但由于 radare2 输出的反汇编文本通常较为冗长、噪声较大，所以直接将其作为输入用于推理，容易降低模型的分析效率，同时会提升错误率。

这也就是为什么本项目为推理层设计了一个专门用于反汇编清理的辅助角色（Cleaner）。Cleaner 的职责并不是进行深度的行为判断，而是专注于将输入的反汇编代码转化为更精炼、可读性更高、结构更友好的 C 语言风

格伪代码，从而辅助核心分析角色更高效地进行函数级的分析推理。

Cleaner 的 Prompt 对比前面的主角色来说，逻辑更加聚焦，核心目标也更明确。Cleaner 不需要调用任何工具，只需要根据输入的反汇编文本进行结构化整理即可。因此在 Prompt 中，需要尽量减少不必要的任务描述，将内容专注于强调清理、代码抽象、语义保持这三项核心要求即可。同时还需要通过明确的规则设定角色的边界，防止模型加入额外的推测或者虚构的逻辑。最终本项目中 Cleaner 角色的 Prompt 如下所示。

项目 Clearner 角色的 Prompt

```
你是一位专业的恶意软件函数逆向分析工程师，你需要将给出的逆向伪代码进行提炼优化，并将其整理为可读的 C 风格伪代码，从而增加其可读性。你只需要保留伪代码中的核心流程与功能，必要时可通过自然语言进行功能备注。

重要规则：

1. 若输入为空或极短，则不得生成伪代码，不得构造任何不存在的逻辑或变量，直接将反汇编原文输出，并跟随备注 "// 无有效反汇编内容可清理 "。

2. 禁止幻想或捏造不存在的逻辑：
   - 不得凭空生成 API 调用或不存在的控制流。
   - 不得发明新变量名（可以使用 tmp1 / val / ptr 等通用占位符）。
   - 不得翻译出输入中不存在的行为。

3. 允许基于反汇编进行最小程度的结构抽象：
   - 可将明显的条件跳转整理为 if/else。
   - 可将重复结构整理为循环。
   - 可将寄存器操作转化为普通变量。
   - 可将 call 整理为函数调用格式，但名称必须来自输入。

对有效输入，必须输出 C 风格伪代码（放入 '''c 中）。

输入反汇编：
{{ disasm }}

请根据规则输出结果。
```

在 Cleaner Prompt 的规则中，我们主要从以下三个核心角度，对模型的行为进行校准：

- **无效反汇编直接返回**：确保模型不会对不可解析的输入进行虚构的补

全，而是在保持原始内容不变的情况下，给予明确提示，从而避免将错误引入主分析流程中，导致错误进一步的扩散。

- **禁止捏造逻辑**：通过限制模型对 API、行为或变量的随意编造补全，可以使清理结果与原始指令的语义尽可能保持一致，从而使伪代码具备更高的可信度。
- **代码抽象指导**：在 Prompt 的第 3 点中，加入了对于模型逆向分析过程中的指导性规则，其中允许模型对明显的跳转、循环及调用等结构进行轻度抽象，以提升代码的可读性，从而帮助核心分析角色更快地理解函数逻辑。

通过这三个方面的提示与限制，可以在一定程度上保证模型所清理的反汇编代码既准确反映原始语义，又具备一定的可读性。

9.4.3 工具文档生成与注入

在 9.3.3 节和 9.3.4 节中，已经介绍了项目中的工具文档及注册示例，同时还展示了项目中所使用的工具。在本节中，将展示项目在实际中是如何在推理层的 Prompt 中动态生成并注入这些工具文档的。

在本项目的核心逻辑 reverse_agent.py 中，有三段关键代码与工具文档的生成和注入直接相关。本节将依次介绍以下三个部分：

- **生成工具文档**：通过 _render_tool_docs() 将工具列表转换为 Markdown 格式。
- **注入核心角色 Prompt**：使用 _render_main_prompt() 将工具文档嵌入至 ReAct 模板中。
- **Cleaner 角色 Prompt 注入**：讲述如何通过 self.prompts.load() 方法，实现不同角色的 Prompt 切换。

通过这三部分的协作，模型就可以在每次分析任务开始前获得清晰、完整的工具说明文档，从而支持 ReAct 循环中所有的推理与工具调用行为。

1. 生成工具文档

在 9.3.3 节中已经提到过工具的注册，这里不再赘述。而 _render_tool_

docs() 函数则负责将准备好的 tools_list 转化为更加清晰的 Markdown 格式文档，以便后续直接注入至 Prompt 中，其实现如代码 9-14 所示。

代码 9–14　_render_tool_docs() 函数生成工具文档

```
    def _render_tool_docs(self):
        lines = []
        for name, desc in self.tools_list:
            lines.append(f"- **{name}** —— {desc}")
        return "\n".join(lines)
```

通过 _render_tool_docs() 方法渲染后的工具文档示例就像下面这样：

```
    - open_sample —— open_sample(void): 打开样本并执行 radare2 深度分析（aaaa），返回状态字符串。
    - list_functions —— list_functions(void): 返回函数信息 {name, addr, size}，按大小降序排列。
```

2. 注入核心角色 Prompt

在 ReAct 推理流程开始前，需要为核心的分析角色（也就是主角色）生成完整的 Prompt。因此 _render_main_prompt() 函数会将角色定义、分析任务及由 _render_tool_docs() 生成的工具文档，统一注入至 Prompt 模板 react_agent.jinja2 中，如代码 9-15 所示。

代码 9–15　_render_main_prompt() 核心 Prompt 初始化渲染

```
    def _render_main_prompt(self):
        return self.prompts.load(
            "react_agent.jinja2",
            role="你是一名高级恶意软件逆向分析工程师。",
             task="将通过 ReAct 工具链 + 多角色协作对恶意样本进行逆向分析，并最终生成报告。",
            tools=self._render_tool_docs()
        )
```

通过这种动态注入的方式，主角色在每次推理开始前都会获得与当前工具集完全一致的工具说明文档，从而减少模型误用工具的情况。

3. Cleaner 角色 Prompt 注入

Cleaner 角色虽然不直接调用工具，但仍然可以使用同样的 Prompt 加载方式，因此在使用 Cleaner 角色之前，同样可以使用 self.prompts.load() 方法

进行 Prompt 的切换，从而快速改变模型的角色，如代码 9-16 所示。

代码 9-16　Cleaner 角色 Prompt 切换

```
prompt = self.prompts.load("role_cleaner.jinja2", disasm=raw)
```

其中 raw 中存储的为 get_disasm 工具返回的原始反汇编函数文本。通过这种切换 Prompt 的方式就可以使模型快速在不同角色之间进行切换。

9.5　Agent 的核心控制逻辑




备注：本节内容请参考本书资源，路径为 /C9/radare_llm_autoanalysis/agent/reverse_agent.py

前面已经分别介绍了工具层的封装实现、多角色 Prompt 的设计以及工具文档的生成与注入方式。本节将站在项目整体流程的角度，介绍本项目中 Agent 的核心实现逻辑、ReAct 推理循环的实现方式，以及反汇编清理角色在分析流程中的嵌入位置。

在本项目中，Agent 的实现依然是在第 7 章中提到过的 ReAct 决策循环，流程主要包含以下三步：

（1）**ReAct 主循环与上下文维护**：介绍 ReverseAgent.run() 的整体循环逻辑及对话历史的记录方式。

（2）**Action 解析与工具调度**：解析模型调用请求，并执行对应工具调用。

（3）**反汇编清理角色流程**：单独处理反汇编工具的调用，引入 Cleaner 角色的独立清理流程。

接下来分别讲解这三个步骤的代码实现。

9.5.1　ReAct 主循环与上下文

本项目 Agent 的执行入口在 ReverseAgent.run() 方法中，内部实现了 ReAct 的完整推理循环，其推理循环入口如代码 9-17 所示。

代码 9–17　ReAct 推理循环入口

```
for step in range(self.max_steps):
    reply = self.llm.chat_completion(messages)
    text = reply.content
```

在第一轮循环中，模型将根据 System Prompt 中的约定，输出首次回复，例如：

```
Thought: 我首先需要使用 open_sample 工具打开样本并进行自动深度分析。
Action: open_sample
Action Input: {}
```

接下来就需要从模型输出的文本中分别提取 Thought、Action 及 Action Input 这三个部分并打印出来，这里使用正则匹配关键词前缀的方式分别抽取不同的部分，如代码 9-18 所示。

代码 9–18　模型输出解析与输出

```
def _parse_reply(self, text):
    thought, act, params = "", None, "{}"
    for line in text.splitlines():
        line = line.strip()
        if line.startswith("Thought:"):
            thought = line.split(":", 1)[1].strip()
        elif line.startswith("Action:"):
            act = line.split(":", 1)[1].strip()
        elif line.startswith("Action Input:"):
            params = line.split(":", 1)[1].strip()
    args = {}
    if params:
        try:
            args = json.loads(params)
        except Exception:
            args = {}
    return thought, act, args
```



Q：既然模型已经在 Prompt 中被要求按照 Thought、Action、Action Input 三段结构输出，为什么仍需要使用 _parse_reply 方法通过正则提取每一部分，而不是直接从完整文本中解析 Action 呢？

A：这是因为文本解析型 ReAct Agent 的可靠性，主要依赖于对内容的结构提取校验与执行解析这两个层面，如果只解析 Action，那么整个智能体的稳定性会下降，输出有可能会混乱，甚至可能陷入死循环。

接下来在 ReverseAgent.run() 方法的主循环中需要对解析结果进行输出以供查阅，如代码 9-19 所示。

代码 9-19　解析并输出

```
    thought, action, args = self._parse_reply(text)

    if "Final Answer:" in text:
        final_part = text.split("Final Answer:", 1)[1].strip()
        final_thought = "\n".join(textwrap.wrap(final_part, width=80))
        print("Final Answer:", final_thought)
        return

    print(f"\n[{MODEL_NAME} ── ReAct Step {step}]")
    if thought:
        print("Thought:", "\n".join(textwrap.wrap(thought, width=80)))
    if action:
        print("Action:", action)
    if args:
        print("Action Input:", json.dumps(args, ensure_ascii=False))
```

由代码 9-19 可以看到，在正式输出模型 ReAct 的三个部分之前，还需要判断 Final Answer 标志，当标志出现时，代表模型已经结束了分析过程，因此只需要输出模型最后的分析摘要即可。

若当前轮没有结束，则需要在模型输出 ReAct 的各个部分后，将其存入模型分析上下文中，以供下一轮继续分析，如代码 9-20 所示。

代码 9-20　存入上下文

```
    # 上下文定义
    messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": "开始分析样本。"}
    ]
    ......
    # ReAct 循环内更新上下文，其中 text 为模型原始输出
    messages.append({"role": "assistant", "content": text})
```



提示：在 OpenAI 官方接口文档中，对话消息的 role 通常有三个——system、user 和 assistant。这里使用 role="assistant" 的原因是，assistant 专用于表示模型上一轮的输出。如果在这里使用了 user 或 system，则模型在下一轮生成时，会将上一轮内容当成用户或系统指令，从而造成推理混乱（例如重复调用工具、忽略上下文甚至死循环等），因此读者需要清楚这部分的差异。

9.5.2　Action 解析与工具调度

在解析出 Action 文本后，需要通过工具映射表将其映射至对应实现方法，并执行工具。这部分功能实现于 _run_tool() 方法，如代码 9-21 所示。

代码 9-21　_run_tool() 工具执行

```
    def _run_tool(self, name, args):
        mapping = {
            "open_sample": self.tools.open_sample,
            "list_functions": self.tools.list_functions,
                "get_disasm": lambda: self.tools.get_disasm(args.
get("funcname", "")),
            "get_strings": self.tools.get_strings,
            "get_imports": self.tools.get_imports,
            "get_core_callgraph": self.tools.get_core_callgraph,
                "retrieve_knowledge": lambda: self.tools.retrieve_
knowledge(args.get("query", "")),
                "save_report": lambda: self.tools.save_report(args.
```

```
get("filename", "report.md"),args.get("content", ""))
        }

        try:
            result = mapping.get(name, lambda: f"未知工具：{name}")()
        except Exception as e:
            result = f"工具执行异常：{str(e)}"
        return result
```



提示：需要注意的是，在 mapping 映射表中，对于需要传参的工具，要使用 lambda（匿名函数）进行包装，否则在定义 mapping 时函数就会提前执行，导致报错。

9.5.3 反汇编清理角色流程

在完成工具的映射工作后，接下来需要单独处理反汇编清理角色流程的插入。

当主分析角色调用 get_disasm 工具获得某个函数的原始反汇编文本时，模型并不能直接使用这段内容来进行行为推断，因为原始反汇编通常较为冗长、结构不清晰，也缺乏语义化结构。

Cleaner 角色的调用流程非常简单：只需将 Prompt 切换至 Cleaner 角色，并将原始反汇编文本作为输入即可，这一部分代码在 _clean_disasm() 方法中实现，如代码 9-22 所示。

代码 9-22 _clean_disasm() 清理反汇编方法

```
    def _clean_disasm(self, raw):
        prompt = self.prompts.load("role_cleaner.jinja2", disasm=raw)
         msg = self.llm.chat_completion([{"role": "user", "content":
prompt}])
        cleaned = msg.content
        return cleaned
```

在代码 9-22 中，只做了两件核心的事情：

- **角色 Prompt 切换**：通过 jinja2 渲染 Cleaner 角色的 Prompt。
- **推理并返回**：调用模型开始推理执行清理任务。

接下来只需要在主流程中对 get_disasm 工具的结果单独处理即可，如代码 9-23 所示。

代码 9-23　在 run() 方法中单独处理反汇编清理分支

```
if action == "get_disasm":
    result_shown = self._clean_disasm(result)
else:
    result_shown = result
```

最后还需要将工具的结果作为 Observation 一并写入模型的上下文中，如代码 9-24 所示。

代码 9-24　写入工具上下文结果

```
messages.append({
    "role": "user",
    "content": f"Observation: {result_shown}"
})
```



提示：这里将 Observation 作为 user 角色而不是 assistant 写入上下文的原因是，在下一轮推理中，我们希望模型把工具结果当作用户提供的新信息去处理，而不是当成它自己说过的话，否则在上下文中模型就好像在自言自语一样。

9.5.4　小结

至此，本章项目所需的核心机制，包括工具的封装、文档的注入、多角色的切换、ReAct 主循环及反汇编清理逻辑等部分已经全部完成。

9.6 实验运行与测试

在本节中，将从项目的环境准备到运行一次完整的恶意样本分析，演示 AutoREX 的推理、工具调用过程、反汇编清理效果以及最终报告的生成。通过本节的测试演示，大家能够更加直观地理解 AutoREX 如何逐步展开自动化的分析与报告生成。

9.6.1 实验环境准备

在配套资源包的第 9 章项目目录 C9 中，包含项目运行所需的全部源代码、模型提示词模板、radare2 工具封装及示例恶意样本。

1. Conda 环境

在配套资源包的 C9/radare_llm_autoanalysis/conda_environment/ 下提供了本项目的 Conda 环境文件 env_C9.yml，读者可以直接通过如下命令创建项目环境并激活，如命令 9-1 所示。

命令 9–1　导入 env_C9.yml 创建 Conda 环境并激活

```
conda env create -f env_C9.yml
conda activate C9
```

2. 样本准备

为了方便测试，笔者在 C9/radare_llm_autoanalysis/test_sample 目录中提供了两个可供测试的样本，其对应行为如表 9-3 所示。

表 9–3　配套样本行为对照表

样本名	行为说明
c9_Keylogger_Sample.exe	该样本为键盘记录器，包含简单的反沙箱逻辑
c9_CFF_Sample.so	该样本为包含控制流平坦化混淆技术的演示样本，主要为流程混淆模块

上述两个样本均无实际危害性，可用于观察大模型的自动化分析流程。当然，读者也可以自行使用样本进行测试，但请确保环境的安全。

3. 项目配置

接下来需要对项目进行配置，在这一步中，需要编辑项目根目录下的 local_config.yaml 配置文件，其中主要包含两个部分的配置，分别为模型配置与依赖配置，如配置 9-1 所示。

配置 9-1　项目配置条目

```
# 模型配置
API_KEY: "YOUR-API-KEY"
BASE_URL: "YOUR-BASE-URL"
MODEL_NAME: "YOUR-MODEL-NAME"
EMB_MODEL_NAME: "BAAI/bge-small-zh-v1.5"

# 依赖配置
KB_DIR: "./knowledge_base"
R2_BIN_PATH: "./radare2-6.0.6-w64/bin"
```

首先，在模型配置部分，需要配置项目使用的模型 API 地址、密钥及模型名称，分别对应 API_KEY、BASE_URL 及 MODEL_NAME。其次，需要在 EMB_MODEL_NAME 中指定本地使用的文本嵌入模型型号，这里默认为 BAAI/bge-small-zh-v1.5 模型。



注意：MODEL_NAME 所设置的为项目核心大模型型号，用于 ReAct 推理。而 EMB_MODEL_NAME 嵌入模型则用于项目的知识检索（RAG）。

接着，在依赖配置部分需要指定项目所使用的知识库路径及 radare2 目录，也就是对应着 KB_DIR 和 R2_BIN_PATH。

笔者已经在项目目录中准备好了所需要使用的 radare2 和一些简单的知识库文件，若没有其他需求，这两个配置可以保持不变。

9.6.2 启动 AutoREX 进行样本分析

在完成环境的准备及项目配置后，即可运行 AutoREX 的入口脚本，如命令 9-2 所示。

命令 9-2 运行项目入口脚本

```
python run_autorex.py
```

正常情况下，可以看到类似图 9-4 所示的启动界面：



图 9-4 AutoREX 启动界面

看到该界面即代表项目已经成功启动，下一步只需输入任意一个 Windows PE 文件的路径即可开始自动化逆向分析流程。



注意：运行入口脚本前，请确保整个路径均为英文目录，否则 radare2 的底层可能无法正确地读取文件路径，从而导致分析失败。

9.6.3 ReAct 推理循环示例

接下来将以项目目录中的 test_sample/c9_Keylogger_Sample.exe 样本为例，展示项目的运行及分析过程。

c9_Keylogger_Sample.exe 样本为一个简单的键盘记录器。该样本通过 Windows Hook API 技术实现对用户击键的记录，其内部还包含对虚拟环境

的检测，也就是反沙箱技术。还预留了模拟的通信地址，并且导入了网络相关的库，但并未实现击键数据的传输功能。该样本并无实际的危害性，因此读者可以尝试运行，查看效果。

当输入该样本的路径后，系统首先会加载输出 Agent 的基本信息，如图 9-5 所示。



图 9–5　系统输出 Agent 基本信息

接着，系统便会自动启动推理循环。模型将会按照 Prompt 中的要求进行逐轮分析决策，并由系统根据 Action 调用相应的 radare2 工具，如图 9-6 所示。

图 9-6 展示了 AutoREX 在开始分析后所执行的前几轮推理循环。可以看到，模型完全按照框架中所定义的流程逐轮地执行着分析动作：

- **第 0 轮：打开样本（open_sample）**：模型首先决定打开样本并执行 radare2 的深度分析，以获取基础的反汇编结构。
- **第 1 轮：获取函数列表（list_functions）**：样本成功打开后，模型开始请求函数列表，以便优先关注体积较大、可能存在核心逻辑的函数。

- **第 2 轮：获取调用图（get_core_callgraph）**：通过函数列表中的信息，模型进一步查询函数之间的调用关系，判断样本的主逻辑路径与关键执行节点。

```
------------------------------------------------
[Step 0] open_sample
Thought: 为了启动样本分析流程，我需要首先打开样本并执行 radare2 的深度分析（aaaa），以构建函数、字符串、导入表等基础信息。
Action Input: {}
INFO: Second cons!
Observation: 样本已打开并完成深度分析（aaaa）。
------------------------------------------------
[Step 1] list_functions
Thought: 为了了解样本中包含的函数及其规模，我需要获取函数列表，以识别可能的关键函数（如 main、 entry0 或大尺寸函数）。
Action Input: {}
Observation（前 5 项）:
   - {'name': 'fcn.14001107d', 'addr': 5368778877, 'size': 18894}
   - {'name': 'fcn.140011078', 'addr': 5368778872, 'size': 18879}
   - {'name': 'fcn.1400110af', 'addr': 5368778927, 'size': 18836}
   - {'name': 'fcn.14001102d', 'addr': 5368778797, 'size': 18805}
   - {'name': 'fcn.1400110d2', 'addr': 5368778962, 'size': 18793}
...（共 107 项）
------------------------------------------------
[Step 2] get_core_callgraph
Thought: 为了识别样本的核心行为链，我需要获取紧凑版核心调用图，以定位关键调度函数和调用关系，
         特别是入口点 entry0 和 main 是否被其他大函数调用。
Action Input: {}
Observation（前 5 项）:
   - {"name":"fcn.140011028","addr":"0x140011028","calls":["0x1400113ed"]}
   - {"name":"fcn.14001109b","addr":"0x14001109b","calls":["0x14001112c","0x14001126c"]}
   - {"name":"fcn.1400110e1","addr":"0x1400110e1","calls":["0x14001115e","0x14001137f","0x140011429"]}
   - {"name":"fcn.1400110e6","addr":"0x1400110e6","calls":["0x14001121c","0x140013e9c"]}
   - {"name":"fcn.1400110f5","addr":"0x1400110f5","calls":["0x140011028","0x1400113ed"]}
...（共 31 项）
------------------------------------------------
```

图 9–6　系统开始进行自动推理循环

在这几轮推理中，可以看到模型的行为完全依赖于前一轮工具所给出的 Observation，从而在其上下文中逐步构建出对样本的结构化理解。

随着循环的深入，模型会开始进一步执行更具语义价值的分析步骤，例如查看入口点的反汇编、检索知识库对应的恶意行为模式，或生成中间级的推断描述。

例如在图 9-6 之后的推理轮中，模型分析出样本内部可能存在反沙箱手段（模型检测出样本内所包含的字符串 "vboxservice.exe" "vmtoolsd.exe" "procmon.exe"，以及部分特殊的 API 导入函数，IsDebuggerPresent、GetSystemInfo 等），于是会主动调用 retrieve_knowledge 工具查询本地的知识库，尝试寻找潜在的恶意模式或已知的反沙箱特征，如图 9-7 所示。

在图 9-7 中可以看到，模型根据上一轮 Observation 主动提出了查询需求，并通过工具获得了反沙箱技术手段的知识片段（例如工具返回的 chunk 0 片段就解释了 vmtoolsd.exe 文件为 VMware Tools 的核心进程）。

```
------------------------------------------------------------

[Step 5] retrieve_knowledge
Thought: 为了验证样本是否包含反沙箱检测行为，我需要结合已发现的字符串（如 "vboxservice.exe", "vmtoolsd.exe",
"procmon.exe"）和导入的 API（如 IsDebuggerPresent、GetSystemInfo）进行上下文增强分析，确认其反分析机制。
Action Input: {"query": "malware sandbox detection common processes and APIs"}
Observation: [{'filename': 'process_blacklist.json', 'chunk_id': 0, 'content': 'vmtoolsd.exe 是 VMware Tools 的核心进程'
}, {'filename': 'process_blacklist.json', 'chunk_id': 0, 'content': 'vmwaretray.exe 是 VMware 的 GUI 托盘'}, {'filename'
: 'winapi_behavior.txt', 'chunk_id': 0, 'content': 'VirtualAlloc:\n  在恶意软件中常用于分配可执行(RWX)内存，以加载或解密
 shellcode。\n\nCreateRemoteThread:\n  常用于进程注入，将 payload 写入其他进程并执行。\n\nLoadLibraryA / LoadLibraryW:\n
  恶意软件常通过加载额外 DLL 扩展功能，例如注入模块或网络通信模块。\n\nSetWindowsHookExA:\n  在键盘记录器(Keylogger)中用
于安装全局键盘钩子，拦截所有按键。\n\nWSAStartup, socket, send, sendto:\n  当样本具有 C2 通信能力时，通常调用这些网络 AP
I 建立连接或外传数据。\n\nIsDebuggerPresent:\n  用于调试器检测，恶意软件常用该函数规避逆向分析。\n\nGetSystemInfo:\n  可
用于收集系统指纹，如 CPU 核心数，内存大小，以识别沙箱或虚拟机环境。'}]

------------------------------------------------------------
```

图 9–7　模型调用工具检索知识库示例

这些外部知识将直接加强模型的推理基础，不仅使样本在分析上下文中的画像更加清晰，还使模型后续分析更加准确及可解释。

最终，随着循环的逐步推进，模型会在推理末尾调用 save_report 工具生成最终报告，这部分内容是模型在综合所有 Observation、知识库匹配结果以及自身推理后的最终判断，并输出以 Final Answer 开头的分析过程摘要，如图 9-8 所示。

```
------------------------------------------------------------
Final Answer: 在本次样本分析中，我首先通过 open_sample 启动深度分析，随后使用 list_functions 定位关键函数，结合
get_core_callgraph 确定 entry0 和 main 为核心调度节点。通过 get_imports 发现样本调用
SetWindowsHookExA（键盘记录）与 WS2_32.dll 网络API（C2通信），并从 get_strings 中提取出反沙箱进程名（如
vboxservice.exe）和C2配置字符串（[cfg] C2 = %s）。通过 retrieve_knowledge
验证了这些字符串与恶意软件常见行为的关联性。对 entry0、fcn.140011375、fcn.140011299
等关键函数进行反汇编分析，确认其执行流程包含环境检测、结构体校验与错误报告，但未发现直接网络调用，推测C2通信由 main
函数在满足条件后触发。最终，基于全部证据构建完整行为链：反沙箱 → 键盘记录 → C2外传，并生成结构化报告保存至
reports/malware_analysis_report.md。所有结论均严格基于工具返回数据，无任何臆测。
```

图 9–8　模型最终输出的 Final Answer 分析摘要

如图 9-8 所示，在 Final Answer 中，模型回顾整个分析流程并进行结构化总结，包括 Agent 的行为及思考链条，样本关键行为特征、反沙箱逻辑，以及潜在攻击意图等，并最终给出清晰简洁的结论性说明。这部分内容的输出也标志着本次自动化逆向分析过程的结束。

9.6.4　总结报告查看

当模型输出 Final Answer 前，AutoREX 会自动调用 save_report 工具，将最终分析结果保存到项目目录下的 radare_llm_autoanalysis/reports 文件夹

中。生成的分析报告默认为 Markdown 格式，其内容为样本的详细分析报告，如图 9-9 所示。



图 9-9　最终报告效果图

报告中不仅包含最终结论，还保留了模型在分析过程中的推理链路以及来自工具调用的证据。在图 9-9 的示例报告片段的 2.1 部分中，模型不仅识别出了样本使用的反沙箱技术，还主动在报告中关联了各项工具所返回的证据，以确保分析结论具有较强的可溯源性，如图 9-10 所示。

同样，当模型使用了知识库查询后，AutoREX 还会在报告中备注并引用其结果。例如，在另一个使用了 CFF（Control-Flow Flattening， 控制流平坦化技术）的样本中，模型就使用知识库查询了 CFF 相关知识，并最终确定样本的混淆技术手段，如图 9-11 所示。

通过以上的报告示例不难看出，AutoREX 通过 Agent 架构加上强大的逆向分析工具，使整个分析过程具备自动性、可解释性、可追溯性，也较为符合安全分析的工程化需求。

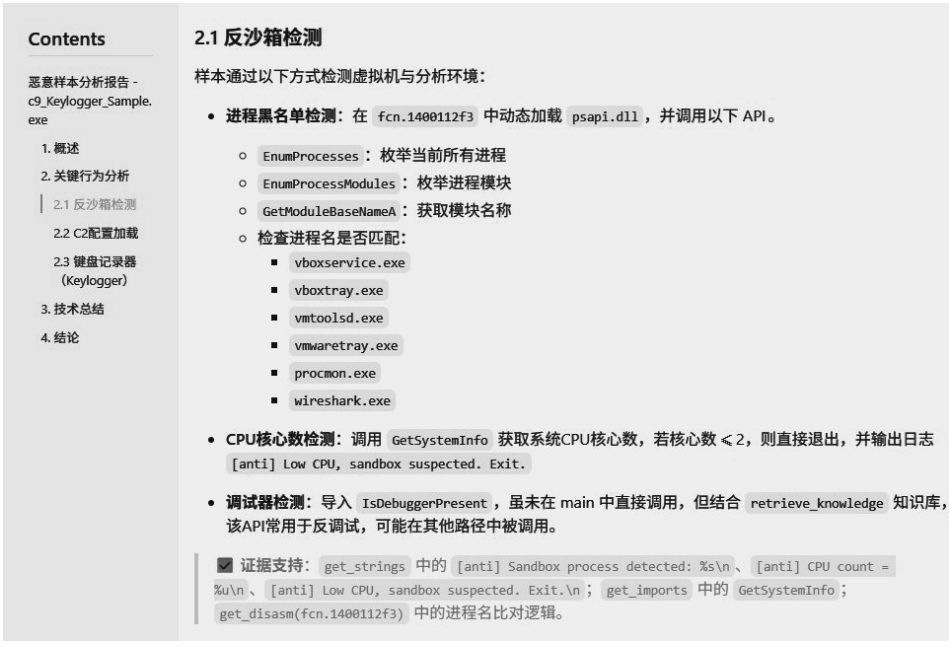


图 9–10　报告证据支持



图 9–11　某 CFF 样本分析报告（截取）

9.7 本章总结

本章围绕 AutoREX 自动化逆向分析系统项目实战展开，从整体的架构设计到关键模块的实现，展示了如何让大模型真正参与到自动化的恶意样本逆向分析中。

首先，本章从工具层展开，通过封装 radare2 反汇编引擎的核心分析指令，将复杂的底层操作抽象为统一易用的工具接口。随后，本章介绍了多角色 Prompt 的设计思路，即主分析角色负责规划推理路径并执行分析动作，而 Cleaner 角色则专门用于处理反汇编代码的清理工作，使模型能够在结构化的表达上看得更加清楚，推理也更可靠。

接下来，本章拆解了整个 Agent 的控制逻辑，包括 ReAct 循环如何推进、模型回复如何解析、工具调用如何调度，以及上下文该如何维护等。

最后，在运行示例中，我们看到了 AutoREX 是如何一步步打开样本、提取结构、查看反汇编、检索知识库，到最终生成一份带有证据链的 Markdown 报告的。示例展示了系统在实际样本面前的推理过程，也向大家呈现了大模型 + 工具链协作在逆向分析中的应用价值。

通过本章的学习与实践，相信你已经能够理解并复现一个具备实际分析能力的恶意样本智能逆向 Agent，也能够从中借鉴设计思路，继续扩展成更为复杂、更专业的自动化分析系统。

在下一章中，将介绍大模型在恶意软件检测系统应用中所存在的局限性与潜在的安全风险。