6Analyst v6

基于大语言模型的网络资产智能分析工具，对网络扫描数据进行自动化分析，识别设备厂商、型号、操作系统、固件版本和实际用途。

目录

项目概述

6Analyst 是一个完整的网络资产分析流水线，采用多Agent协作架构：

1. **数据清洗** - 过滤无用字段、压缩Banner、简化HTML，减少Token消耗

2. **产品分析** - 识别设备厂商、型号、操作系统、固件版本
3. **用途分析** - 分析设备用途、行业属性、检测到的服务
4. **结果校验** - 验证分析结果的准确性，修正错误结论

主要特性

特性	说明
多Agent协作	ProductAnalyst + UsageAnalyst + CheckAnalyst 三级分析
数据压缩	智能清洗，Token压缩率可达80%+
多线程并行	支持多工作线程同时处理，大幅提升吞吐量
断点续传	自动保存进度，中断后可继续处理
智能限流	自动检测API限制，指数退避重试
费用预估	支持预估Token消耗和费用开销
提示词管理	多版本提示词切换，可视化配置页面
模型对比	支持85+模型的费用对比和选择

核心功能

1. 数据清洗模块（`data_cleaner.py`）

数据清洗是整个流水线的第一步，负责对原始网络扫描数据进行预处理，目标是在保留关键信息的同时，显著减少Token消耗（压缩率可达80%以上）。

1.1 清洗流程概述

```
原始数据 → 字段过滤 → HTTP响应处理 → HTML简化 → Banner压缩 → 恶意内容过滤 → 字符串清理 → 输出
```

每条记录的处理流程：

1. **顶层字段过滤**：删除无价值的元数据字段
2. **服务遍历**：对每个服务（如http-80、ssh-22）分别处理
3. **Body处理**：检测错误页面、默认页面，或简化HTML
4. **Banner压缩**：针对SSH/TLS/MySQL等协议压缩Banner
5. **恶意内容过滤**：清理可能触发安全软件的内容
6. **字符串清理**：移除特殊字符、合并空白

1.2 字段过滤规则

顶层字段过滤（直接删除）：

字段	删除原因
`IP Index`	索引号，对分析无意义
`Timestamp`	时间戳，不影响设备识别
`OS`	nmap识别的OS，通常不准确，会干扰LLM判断

服务字段过滤（在每个服务内删除）：

字段	删除原因
`Body sha256`	哈希值，对分析无意义

1.3 HTTP响应处理

错误页面检测与过滤：

检测以下HTTP错误响应，直接过滤Body内容（返回null）：

错误码	检测模式	说明
400	`400\s*Bad\s*Request`	请求格式错误
401	`401\s*Unauthorized`	未授权
403	`403\s*Forbidden`	禁止访问
404	`404(\s*Not\s*Found)?`	页面不存在
406	`406\s*Not\s*Acceptable`	不可接受
415	`415\s*Unsupported`	不支持的媒体类型
421	`421\s*Misdirected`	错误定向请求
500	`500\s*Internal\s*Server\s*Error`	服务器内部错误
502	`502\s*Bad\s*Gateway`	网关错误
503	`503\s*Service\s*Unavailable`	服务不可用

默认页面标签化：

将常见的默认页面替换为简短标签，大幅减少Token消耗：

检测模式	替换标签	说明
`Welcome\s*to\s*nginx`	`[DEFAULT:nginx-default]`	Nginx默认页
`Apache2?\s*(Debian|Ubuntu)?\s*Default`	`[DEFAULT:apache-default]`	Apache默认页
`Plesk\s*Default\s*Page`	`[DEFAULT:plesk-default]`	Plesk默认页
`Web\s*Server's\s*Default\s*Page`	`[DEFAULT:plesk-default]`	通用默认页
`IIS\s*Windows\s*Server`	`[DEFAULT:iis-default]`	IIS默认页
`It\s*works!`	`[DEFAULT:apache-works]`	Apache测试页
`Test\s*Page\s*for.*Apache`	`[DEFAULT:apache-test]`	Apache测试页

1.4 HTML内容简化

对有价值的HTML页面，使用 `html_extractor.py` 提取关键信息并压缩为紧凑格式。

提取的信息：

信息类型	提取方式	示例
标题	`<title>` 标签	`title:MikroTik Router`
描述	`<meta name="description">`	`desc:Network management`
关键词	`<meta name="keywords">`	`keywords:router,mikrotik`
生成器	`<meta name="generator">`	`generator:WordPress 5.8`
表单	`<form>` 标签	`form:action=/login method=POST`
输入字段	`<input>` 标签	`inputs:username,password`
链接	`<a href>` 属性	`links:/admin,/config`
关键文本	页面正文	`text:Login to your router`

删除的内容：

- `<script>` 标签及内容
- `<style>` 标签及内容
- HTML注释 `<!-- -->`
- 内联样式属性
- 事件处理属性（onclick等）

输出格式：

```
title:MikroTik Router ; desc:Network management ; form:action=/login method=POST ; inputs:username,password ; links:/admin,/config
```

1.5 SSH Banner压缩

SSH Banner通常包含大量算法列表，占用大量Token但信息价值有限。清洗模块对SSH Banner进行智能压缩。

算法分类与常见配置识别：

算法类型	常见配置	说明
Kex（密钥交换）	curve25519-sha256, ecdh-sha2-nistp256/384/521, diffie-hellman-group-exchange-sha256	现代安全配置
Host Key（主机密钥）	ssh-ed25519, ecdsa-sha2-nistp256/384/521, rsa-sha2-256/512	现代密钥算法
Encryption（加密）	chacha20-poly1305, aes128/256-gcm, aes128/256-ctr	现代加密算法
MAC（消息认证）	umac-64/128-etm, hmac-sha2-256/512-etm	现代MAC算法

压缩规则：

- 如果算法列表与常见配置相似（缺失≤2个），标记为 `common`
- 识别并标注特殊算法

特殊算法标注：

标注	检测条件	含义
`post-quantum`	包含 `sntrup761x25519`	后量子密码算法
`terrapin-fix`	包含 `kex-strict-s-v00`	Terrapin漏洞修复
`legacy-dh`	包含 `group1-sha1` 或 `group-exchange-sha1`	遗留DH算法（不安全）
`dss`	包含 `ssh-dss`	DSS算法（已弃用）
`3des`	包含 `3des`	3DES加密（不推荐）
`cbc-mode`	包含 `-cbc`	CBC模式（有漏洞风险）
`no-curve25519`	缺少 `curve25519-sha256`	不支持现代曲线
`no-chacha20`	缺少 `chacha20-poly1305`	不支持ChaCha20
`no-ed25519`	缺少 `ssh-ed25519`	不支持Ed25519

删除的SSH字段：

- `Public Key`（公钥内容）
- `Fingerprint_sha256`（指纹）
- `Compression Algorithms`（压缩算法列表）

压缩示例：

原始Banner（约2000字符）：

```
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.1
Kex Algorithms: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,...
Server Host Key Algorithms: ssh-ed25519,ecdsa-sha2-nistp256,...
Encryption Algorithms: chacha20-poly1305@openssh.com,aes128-gcm@openssh.com,...
MAC Algorithms: umac-64-etm@openssh.com,umac-128-etm@openssh.com,...
Public Key: AAAAB3NzaC1yc2EAAAADAQABAAABAQ...
Fingerprint_sha256: SHA256:abc123...
```

压缩后（约200字符）：

```
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.1
SSH_Algorithms: {"kex_common": true, "hostkey_common": true, "enc_common": true, "mac_common": true}
```

1.6 TLS Banner清理

删除TLS证书中的冗余信息，这些信息对设备识别无价值：

删除字段	说明
`modulus`	RSA模数，通常很长
`exponent`	RSA指数
`Signature Value`	签名值
`Subject Key Identifier`	主体密钥标识符
`Authority Key Identifier`	颁发机构密钥标识符

保留的有价值信息：

- `Subject`（证书主体，可能包含设备信息）
- `Issuer`（颁发者）
- `Validity`（有效期）
- `Subject Alternative Name`（备用名称，可能包含域名）

1.7 MySQL Banner清理

删除MySQL Banner中的 `Capability Flags` 块，该信息对设备识别无价值。

删除模式：

```
Capability Flags:
  ...（多行标志位）
```

保留的有价值信息：

- 版本号（如 `5.7.38-log`）
- 服务器状态
- 认证插件名称

1.8 恶意内容过滤

清理可能触发安全软件（如Windows Defender）的内容，避免保存的数据文件被误报。

过滤的关键词：

关键词	类型
`coinhive`, `CoinHive`	加密货币挖矿脚本
`cryptominer`	挖矿程序
`cryptonight`	挖矿算法
`jsecoin`	JS挖矿脚本
`cryptoloot`	挖矿脚本
`webminer`	网页挖矿
`deepminer`	挖矿程序
`coinimp`	挖矿脚本

替换方式：

```
原始: <script src="coinhive.min.js"></script>
替换: <script src="[FILTERED:8].min.js"></script>
```

保留长度信息 `[FILTERED:N]` 以便分析时知道原始内容的大致长度。

1.9 字符串清理

对所有字符串值进行清理，确保输出的JSONL格式正确：

处理的特殊字符：

字符	Unicode	处理方式
Line Separator	U+2028	替换为空格
Paragraph Separator	U+2029	替换为空格
NUL	U+0000	删除
Vertical Tab	U+000B	替换为空格
Form Feed	U+000C	替换为空格
Next Line (NEL)	U+0085	替换为空格
CR, LF, CRLF	-	替换为空格
连续空格	-	合并为单个空格

1.10 清洗效果统计

清洗完成后输出统计信息：

指标	说明
`total_records`	原始记录总数
`processed_records`	成功处理的记录数
`failed_records`	处理失败的记录数
`original_size`	原始数据总大小（字节）
`cleaned_size`	清洗后数据总大小（字节）
`compression_ratio`	压缩率（1 - cleaned/original）
`input_files`	输入文件数量

典型压缩效果：

- 原始平均每条：4,741 字符
- 清洗后平均每条：932 字符
- 压缩率：80.4%

---

2. 产品分析Agent（`product_analyst.py`）

ProductAnalyst 负责识别设备的硬件和软件信息，是三个分析Agent中最核心的一个。

2.1 分析流程

```
清洗后数据 → 批次构建 → 提示词组装 → API调用 → JSON解析 → 结果验证 → 输出
```

批次处理：

- 默认每批3条记录（可通过 `--batch-size` 调整）
- 批次内记录共享系统提示词，减少Token消耗
- 每批独立调用API，失败不影响其他批次

2.2 识别目标

字段	说明	数据类型	示例
`vendor`	硬件厂商名称	string | null	"MikroTik", "Cisco", "Juniper", "Ubiquiti"
`model`	型号列表，每项包含名称和置信度	[[name, conf], ...] | null	[["RB750Gr3", 0.85], ["hEX", 0.6]]
`os`	操作系统名称及版本号	string | null	"RouterOS 6.49.10", "IOS XE 17.3"
`firmware`	固件完整名称及版本	string | null	"MikroTik 7.16.1", "Cisco IOS 15.1(4)M"
`type`	设备类型分类	string | null	router / switch / server / firewall / camera / nas / printer / iot / appliance
`result_type`	识别方式	string | null	"direct"（直接从Banner识别）/ "inferred"（推断得出）
`confidence`	整体置信度	float	0.0 - 1.0
`evidence`	证据列表	[{src, val, weight}, ...] | null	见下方说明

2.3 证据格式

每条证据包含三个字段：

- `src`：证据来源字段（如 `ftp_banner`、`http_title`、`ssh_banner`）
- `val`：证据内容（如 `MikroTik FTP server (MikroTik 7.16.1)`）
- `weight`：证据权重（0.0 - 1.0）

证据来源示例：

来源	说明	权重参考
`ftp_banner`	FTP服务Banner	0.8-0.95（通常包含厂商和版本）
`ssh_banner`	SSH服务Banner	0.7-0.9（包含OS信息）
`http_title`	HTTP页面标题	0.6-0.8（可能包含设备名）
`http_body`	HTTP页面内容	0.5-0.7（需要推断）
`snmp_sysdesc`	SNMP系统描述	0.9-0.95（通常很准确）
`telnet_banner`	Telnet Banner	0.7-0.85
`tls_subject`	TLS证书主体	0.5-0.7（可能包含设备信息）

2.4 识别规则

字段填写规则：

字段	正确示例	错误示例	说明
`os`	"RouterOS 6.49.10"	"RouterOS"	必须包含版本号
`firmware`	"MikroTik 7.16.1"	"7.16.1"	必须包含完整名称
`model`	[["RB750Gr3", 0.85]]	"RouterOS"	仅填写硬件型号SKU
未知字段	`null`	"unknown"	使用null，不要使用字符串

识别方式判定：

result_type	判定条件	示例
`direct`	Banner中明确包含厂商/型号信息	FTP Banner: "MikroTik FTP server"
`inferred`	通过特征推断得出	端口8291+8728 → MikroTik

置信度评估：

置信度范围	含义	典型场景
0.9-1.0	非常确定	Banner明确包含完整信息
0.7-0.9	较为确定	多个证据相互印证
0.5-0.7	可能正确	单一证据或推断
0.3-0.5	不太确定	弱证据或模糊匹配
<0.3	基本猜测	应考虑返回null

2.5 常见设备特征

MikroTik设备：

- 端口特征：8291（Winbox）、8728/8729（API）
- Banner特征：`MikroTik`、`RouterOS`
- FTP Banner：`MikroTik FTP server (MikroTik X.XX.X)`

Cisco设备：

- Banner特征：`cisco`、`IOS`、`IOS-XE`、`NX-OS`
- SSH Banner：`Cisco-X.X`
- HTTP：包含 `cisco` 关键词

Ubiquiti设备：

- Banner特征：`UBNT`、`UniFi`、`EdgeOS`
- HTTP：UniFi Controller界面

Juniper设备：

- Banner特征：`JUNOS`、`Juniper`
- SSH Banner：`JUNOS X.X`

---

3. 用途分析Agent (`usage_analyst.py`)

UsageAnalyst 负责分析设备的实际用途和应用场景，与ProductAnalyst并行执行。

3.1 分析流程

```
清洗后数据 → 批次构建 → 提示词组装 → API调用 → JSON解析 → 结果验证 → 输出
```

与ProductAnalyst使用相同的输入数据（cleaned_data.jsonl），但分析目标不同。

3.2 识别目标

字段	说明	数据类型	示例
`primary_usage`	主要用途分类	string | null	见下方用途分类表
`secondary_usages`	次要用途列表	[string, ...]	可为空数组
`industry`	行业属性	string | null	"ISP", "Enterprise", "Education", "Government", "Healthcare"
`services_detected`	检测到的服务/协议列表	[string, ...]	["http", "ssh", "ftp", "mysql", "smtp"]
`confidence`	整体置信度	float	0.0 - 1.0
`evidence`	证据列表	[{src, val, weight}, ...] | null	同ProductAnalyst

3.3 用途分类（仅使用以下分类）

分类	说明	典型特征
`network_infrastructure`	网络基础设施	路由器、交换机、防火墙的管理接口
`security_appliance`	安全设备	IDS/IPS、WAF、VPN网关
`web_hosting`	Web托管服务	设备的主要用途是提供Web服务
`database_server`	数据库服务器	MySQL、PostgreSQL、MongoDB等
`mail_server`	邮件服务器	SMTP、IMAP、POP3服务
`file_server`	文件服务器	NAS、FTP服务器、SMB共享
`application_server`	应用服务器	运行业务应用的服务器
`iot_device`	物联网设备	摄像头、传感器、智能设备
`development_server`	开发/测试服务器	开发环境、CI/CD服务器
`monitoring_system`	监控系统	Zabbix、Nagios、Prometheus

3.4 判别规则

关键判别原则：

场景	正确判定	错误判定	说明
路由器开放HTTP	`network_infrastructure`	`web_hosting`	HTTP是管理接口，不是Web服务
路由器开放FTP	`network_infrastructure`	`file_server`	FTP是配置备份接口
路由器开放SSH	`network_infrastructure`	-	SSH是管理接口
专用Web服务器	`web_hosting`	-	主要用途是提供Web内容
NAS设备	`file_server`	-	主要用途是文件存储

services_detected字段规则：

- 仅填写协议名称：`http`、`ssh`、`ftp`、`mysql`、`smtp`
- 不要填写用途分类：~~`web_hosting`~~、~~`file_server`~~

行业判定依据：

行业	判定依据
ISP	ASN信息、大量网络设备、BGP相关
Enterprise	企业域名、内部系统
Education	.edu域名、学校相关
Government	.gov域名、政府相关
Healthcare	医疗相关域名或系统
Finance	金融相关域名或系统
Retail	零售相关系统

3.5 置信度阈值

置信度范围	处理方式
≥0.7	明确模式，可以确定用途
0.4-0.7	可能的用途，需要更多证据
<0.4	证据不足，应使用 `null`

---

4. 校验Agent（`check_analyst.py`）

CheckAnalyst 是流水线的最后一个Agent，负责对ProductAnalyst和UsageAnalyst的合并结果进行质量检查、验证和修正。

4.1 校验流程

```
合并结果 → 批次构建 → 提示词组装 → API调用 → JSON解析 → 应用修正 → 输出最终结果
```

输入：`merged_analysis.jsonl`（产品分析+用途分析的合并结果）
输出：

- `check_details.jsonl`（校验详情）
- `final_analysis.jsonl`（最终修正结果）

4.2 校验输出

字段	说明	可选值
`validation_status`	校验状态	`verified`（验证通过）/ `adjusted`（已调整）/ `rejected`（已拒绝）
`evidence_quality`	证据质量评估	`strong` / `moderate` / `weak` / `insufficient`
`issues_found`	发现的问题列表	[string, ...] 或空数组
`original_confidence`	原始置信度	float
`validated_confidence`	校验后置信度	float
`reasoning`	校验推理说明	string | null
`adjustments`	调整内容	{} 或 {field: new_value, ...}

4.3 校验规则

字段验证规则：

字段	验证规则	修正方式
`os`	必须包含版本号	从Banner中提取完整版本
`firmware`	必须包含完整名称	补充厂商前缀
`model`	必须是硬件SKU	移除OS/固件名称
`primary_usage`	路由器应为network_infrastructure	修正为正确分类

验证示例:

原始值	问题	修正值
`os: "RouterOS"`	缺少版本号	`os: "RouterOS 6.49.10"`
`firmware: "7.16.1"`	缺少厂商名	`firmware: "MikroTik 7.16.1"`
`model: [["RouterOS", 0.8]]`	填写了OS而非型号	`model: null`
`primary_usage: "web_hosting"` (路由器)	错误分类	`primary_usage: "network_infrastructure"`

4.4 调整机制

校验状态与调整:

状态	adjustments内容	说明
`verified`	`{}`	验证通过，不做修改
`adjusted`	`{"os": "RouterOS 6.49.10", "firmware": "MikroTik 6.49.10"}`	包含修正的字段
`rejected`	`{"vendor": null, "model": null, "confidence": 0.2}`	将不可信字段置空并降低置信度

证据质量评估：

质量等级	判定条件
`strong`	多个高权重证据相互印证
`moderate`	有明确证据但不够充分
`weak`	证据较少或权重较低
`insufficient`	几乎没有有效证据

4.5 评估统计

CheckAnalyst 会生成评估统计报告，包括：

统计项	说明
校验状态分布	verified/adjusted/rejected 各占比例
证据质量分布	strong/moderate/weak/insufficient
置信度变化统计	提升/降低/不变的数量
平均置信度变化	原始平均 vs 校验后平均
常见问题排行	Top 5 最常见的问题

统计报告示例：

```
```

```
校验状态分布:
  ✓ 验证通过: 8500 (63.0%)
  ~ 已调整:   4200 (31.1%)
  ✗ 已拒绝:    802 (5.9%)

证据质量分布:
  strong: 6200 (45.9%)
  moderate: 4800 (35.6%)
  weak: 2000 (14.8%)
  insufficient: 502 (3.7%)

置信度变化:
  平均原始置信度: 0.72
  平均校验置信度: 0.75
  置信度提升: 3200 条
  置信度降低: 1800 条
  置信度不变: 8502 条
```

---

5. Agent基类（`base_analyst.py`）

BaseAnalyst 是三个分析Agent的公共基类，提供通用功能：

5.1 核心功能

- **数据加载**：从JSONL文件加载记录，支持限制条数
- **Token计算**：使用tiktoken计算消息的Token数量
- **批处理**：将记录分批提交给LLM，每批默认3条
- **API调用**：封装OpenAI API调用，支持自定义base_url
- **响应解析**：多种方式尝试解析JSON响应（直接解析、修复尾部逗号、按行解析、正则提取）

5.2 错误检测

自动检测API响应中的错误：

- **并发限制**：检测 `rate limit`、`too many requests`、`429` 等关键词

- **安全限制**：检测 `security`、`blocked`、`forbidden`、`403` 等关键词（需连续5次才触发）
- **余额不足**：检测 `insufficient balance`、`quota exceeded` 等关键词

5.3 误报防护

检测到正常JSON响应字段时跳过错误检测：

- 检查 `"ip"`、`"confidence"`、`"vendor"`、`"model"`、`"primary_usage"` 等字段
- 检查响应是否以 `[` 或 `{` 开头且包含多个JSON特征

系统架构

---

1. 整体流程

6Analyst采用流水线架构，数据依次经过以下阶段：

阶段一：数据清洗

- 输入：原始扫描数据（JSON/JSONL格式）
- 处理：字段过滤、HTML简化、Banner压缩、错误页面过滤
- 输出：`cleaned_data.jsonl`（清洗后的数据）

阶段二：并行分析

- 输入：`cleaned_data.jsonl`
- 处理：ProductAnalyst 和 UsageAnalyst 并行执行（或串行，取决于speed-level）
- 输出：`product_analysis.jsonl` 和 `usage_analysis.jsonl`

阶段三：结果合并

- 输入：产品分析结果 + 用途分析结果
- 处理：按IP合并两个Agent的输出
- 输出：`merged_analysis.jsonl`

阶段四：结果校验

- 输入：`merged_analysis.jsonl`
- 处理：CheckAnalyst 验证、评估、修正
- 输出：`check_details.jsonl`（校验详情）+ `final_analysis.jsonl`（最终结果）

2. 模块职责

运行控制层

- `run.py`：单线程运行控制器，负责命令行解析、流程调度、进度显示、状态保存
- `multi_thread_runner.py`：多线程执行器，负责工作线程管理、任务分配、全局限流控制

数据处理层

- `data_cleaner.py`：数据清洗，依赖 `utils/html_extractor.py` 进行HTML解析
- `cost_calculator.py`：费用计算，依赖各Agent的提示词模板计算Token

分析Agent层

- `base_analyst.py`：Agent基类，提供API调用、Token计算、JSON解析等通用功能
- `product_analyst.py`：产品分析Agent，继承BaseAnalyst
- `usage_analyst.py`：用途分析Agent，继承BaseAnalyst
- `check_analyst.py`：校验Agent，继承BaseAnalyst

线程安全层（`thread_safe.py`）

- `TaskManager`：任务分配器，避免重复执行、漏执行、冲突执行
- `ThreadSafeStats`：统计管理器，原子操作保证计数准确
- `ThreadSafeFileWriter`：文件写入器，文件级锁避免并发写入冲突

- `ThreadSafeLogger`：日志管理器，每个线程独立日志 + 主日志汇总

工具层

- `utils/token_counter.py`：Token计算工具，封装tiktoken
- `utils/logger.py`：日志工具，文件日志 + 控制台输出
- `utils/error_logger.py`：错误日志工具，记录完整提示词和响应
- `utils/html_extractor.py`：HTML解析工具，提取关键信息

配置层

- `config.py`：集中管理所有配置项，包括API配置、文件路径、处理参数、清洗规则、模型价格

3. 数据流向

步骤1：数据清洗

- 输入：`6Analyst/data/input/` 目录下的原始JSON/JSONL文件
- 处理：DataCleaner 执行字段过滤、HTML简化、Banner压缩等清洗操作
- 输出：`6Analyst/data/output/cleaned_data.jsonl`

步骤2：产品分析

- 输入：`cleaned_data.jsonl`
- 处理：ProductAnalyst 调用LLM识别厂商、型号、OS、固件
- 输出：`6Analyst/data/output/product_analysis.jsonl`

步骤3：用途分析

- 输入：`cleaned_data.jsonl`
- 处理：UsageAnalyst 调用LLM分析用途、行业、服务
- 输出：`6Analyst/data/output/usage_analysis.jsonl`
- 说明：步骤2和步骤3可并行执行（speed-level=s）或串行执行

步骤4：结果合并

- 输入：`product_analysis.jsonl` + `usage_analysis.jsonl`
- 处理：按IP将产品分析和用途分析结果合并为一条记录
- 输出：`6Analyst/data/output/merged_analysis.jsonl`

步骤5：结果校验

- 输入：`merged_analysis.jsonl`
- 处理：CheckAnalyst 调用LLM验证结果、评估证据、修正错误
- 输出：`6Analyst/data/output/check_details.jsonl`（校验详情）

步骤6：生成最终结果

- 输入：校验结果 + 合并结果
- 处理：将校验的adjustments应用到合并结果，生成最终记录
- 输出：`final_analysis.jsonl`（项目根目录）

4. 状态管理

运行状态文件（`run_state.json`）：

- `task_id`：当前任务ID
- `last_update`：最后更新时间
- `start_time`：开始时间戳
- `elapsed_seconds`：已用时间
- `stats`：统计数据（处理数、Token数、置信度分布等）

断点续传机制：

- 程序启动时加载 `run_state.json` 和 `final_analysis.jsonl`
- 从 `final_analysis.jsonl` 提取已处理的IP列表
- 跳过已处理的记录，继续处理剩余记录
- 恢复上次的统计数据和任务ID

快速开始

环境要求

- Python 3.8+
- 依赖包：`openai`，`tiktoken`

安装依赖

```
pip install openai tiktoken
```

配置API

编辑 `6Analyst/config.py`：

```
API_KEY = "your-api-key"
BASE_URL = "https://api.example.com"
MODEL_NAME = "deepseek-v3.2"   # 推荐性价比高的模型
```

基本使用

```
# 执行完整流程（清洗 + 产品分析 + 用途分析 + 校验）
python run_6analyst.py

# 使用测试数据
python run_6analyst.py --test

# 限制处理条数
python run_6analyst.py --max-records 100

# 多线程处理（推荐）
python run_6analyst.py -t 10
```

命令行参数

运行模式参数

参数	简写	说明
`--all`		执行完整流程（默认）
`--clean-only`	`--cdo`	仅执行数据清洗
`--product-only`		仅执行产品分析
`--usage-only`		仅执行用途分析
`--check-only`		仅执行结果校验
`--no-check`		跳过结果校验步骤

数据控制参数

参数	说明
`--test`	使用测试数据集（`6Analyst/data/input_test/`）
`--input PATH`	指定输入路径（文件夹或文件）
`--max-records N`	限制处理条数
`--restart`	清除已有结果，开始新任务（任务ID自动+1）

速度控制参数

参数	说明
`--speed-level N`	设置速度等级（1-6 或 s），默认 s
`-t N`	使用 N 个线程并行处理（默认: 1）

速度等级说明：

等级	模式	Agent间隔	说明
1	串行	10秒	最慢模式，适合严格限流的API
2	串行	3秒	慢速模式
3	串行	1秒	中速模式
4	串行	0.5秒	快速模式
5	串行	0.1秒	高速模式
6	串行	无间隔	极速模式
s	**并行**	**无间隔**	**最高速，产品/用途Agent并行执行（默认）**

日志管理参数

参数	说明
`--clean-log`	清理旧日志，合并当前任务的日志文件
`-f` / `--force-clean-log`	强制合并所有日志并清理

费用计算参数

参数	说明
`--calculate-cost`	计算 Token 和费用开销
`--batch-size N`	每批次数据条数（默认 3）
`--datanum N`	估算指定数据量的开销
`--file-cost`	基于原始输入文件计算开销

提示词管理参数

参数	说明
`--prompt`	进入提示词管理模式
`--prompt --list`	列出所有可用提示词及费用信息
`--prompt --update`	更新所有提示词的费用信息
`--prompt --page`	生成提示词配置HTML页面
`--prompt -p ID`	指定产品Agent使用的提示词（p1/p2/p3/default）
`--prompt -u ID`	指定用途Agent使用的提示词（u1/u2/u3/default）
`--prompt -c ID`	指定校验Agent使用的提示词（c1/c2/c3/default）

使用示例：

```
# 查看所有提示词
python run_6analyst.py --prompt --list

# 更新费用信息
python run_6analyst.py --prompt --update

# 生成配置页面
python run_6analyst.py --prompt --page

# 指定提示词运行
python run_6analyst.py --prompt -p p1 -u u2 -c c1
```

其他参数

参数	说明
`--show`	生成 HTML 报告并在浏览器中打开
`--extract-device`	提取含 nmap 识别的 device/OS 数据
`--debug`	调试模式：输出详细的解析失败信息

运行模式

完整流程

```
# 默认执行完整流程
python run_6analyst.py

# 等价于
python run_6analyst.py --all
```

流程顺序：数据清洗 → 产品分析 → 用途分析 → 结果合并 → 校验 → 输出最终结果

分步执行

```
# 步骤1：仅清洗数据
python run_6analyst.py --clean-only

# 步骤2：仅产品分析（需要已有清洗数据）
python run_6analyst.py --product-only

# 步骤3：仅用途分析（需要已有清洗数据）
python run_6analyst.py --usage-only

# 步骤4：仅校验（需要已有合并结果）
python run_6analyst.py --check-only
```

断点续传

程序自动保存运行状态，支持中断后继续：

```
# 首次运行，任务ID = 1
python run_6analyst.py

# 中断后继续，任务ID仍为1，自动跳过已处理的记录
python run_6analyst.py

# 重新开始新任务，任务ID = 2
python run_6analyst.py --restart
```

任务ID系统

- 每个分析任务分配唯一ID，用于追踪和日志管理
- 任务ID记录在 `run_state.json` 和日志文件中
- 断点续传时保持相同ID
- 使用 `--restart` 时ID自动+1

多线程模式

多线程模式通过 `MultiThreadRunner` 实现，支持多个工作线程同时处理不同批次，大幅提升吞吐量。

基本用法

```
# 使用10个线程并行处理
python run_6analyst.py -t 10

# 4线程 + 每线程内部并行agent模式（最高吞吐量）
python run_6analyst.py -t 4 --speed-level s

# 2线程 + 每线程内部中速模式
python run_6analyst.py -t 2 --speed-level 3
```

多线程架构

1. MultiThreadRunner（多线程执行器）

多线程模式的核心控制器，负责：

- 初始化线程安全组件（TaskManager、ThreadSafeStats、ThreadSafeFileWriter、ThreadSafeLogger）
- 创建和管理工作线程池
- 协调全局限流控制
- 进度打印和状态保存

2. TaskManager（任务管理器）

负责任务分配和状态跟踪：

- 将所有记录分批创建为任务项（TaskItem）
- 维护待处理队列（pending_queue）
- 跟踪已处理IP集合，避免重复执行
- 跟踪正在处理IP集合，避免冲突执行
- 支持任务重试（失败后重新入队）

3. WorkerContext（工作线程上下文）

每个工作线程拥有独立的上下文：

- 独立的ProductAnalyst、UsageAnalyst、CheckAnalyst实例（避免共享状态）
- 独立的速度配置
- 独立的限流计数器
- 通过共享的ThreadSafeStats和ThreadSafeFileWriter与其他线程协作

4. 工作线程执行流程

每个工作线程循环执行以下步骤：

1. 从TaskManager获取下一个待处理任务

2. 检查全局限流状态，如有限流则等待
3. 执行ProductAnalyst和UsageAnalyst（并行或串行，取决于speed-level）
4. 合并产品和用途分析结果
5. 执行CheckAnalyst校验（如未跳过）
6. 将结果写入输出文件
7. 更新统计数据
8. 标记任务完成

线程安全组件

ThreadSafeStats（统计管理器）

- 使用 `threading.RLock` 保护所有统计数据
- 支持按工作线程独立统计
- 提供原子操作方法：`add_processed()`、`add_tokens()`、`add_confidence()`、`add_error()`
- 区分累计统计和本次运行统计

ThreadSafeFileWriter（文件写入器）

- 为每个文件路径维护独立的锁
- 提供原子写入方法：`append_json()`、`write_json()`、`append_batch()`
- 写入后立即 `flush()` 和 `fsync()` 确保数据落盘

ThreadSafeLogger（日志管理器）

- 主日志文件：`log_taskN_YYYYMMDD_HHMMSS_main.txt`
- 工作线程日志：`log_taskN_YYYYMMDD_HHMMSS_workerN.txt`
- 工作线程日志同时汇总到主日志（带 `[WN]` 前缀）

全局限流控制

当任一工作线程检测到API限制时：

1. 设置全局限流时间戳 `_global_rate_limit_until`
2. 所有工作线程在获取任务后检查全局限流状态
3. 如果当前时间 < 限流时间戳，则等待
4. 等待期间进度打印线程显示倒计时

指数退避策略：

- 首次触发等待30分钟
- 后续每次翻倍（60分钟、120分钟、240分钟...）
- 自动降低速度等级一级

线程数建议

场景	建议线程数	说明
API无限流	4-10	充分利用并发能力
API有限流	2-4	避免频繁触发限制
测试验证	2	验证多线程功能

多线程 vs 单线程

特性	单线程模式	多线程模式
控制器	`run.py`	`multi_thread_runner.py`
吞吐量	较低	高（线性提升）
资源占用	低	较高
日志文件	单个	主日志 + 工作线程日志
适用场景	小数据量、调试	大数据量、生产环境

配置说明

主配置文件（config.py）

```python
# ========= API配置 =========
API_KEY = "your-api-key"
BASE_URL = "https://api.example.com"
MODEL_NAME = "deepseek-v3.2"

# ========= 处理参数 =========
MAX_RECORDS = None          # None 表示处理全部
BATCH_SIZE = 3                # 每批提交条数（建议 3-5）
MAX_INPUT_TOKENS = 4096   # 输入 Token 上限
DEBUG_MODE = False          # 调试模式

# ========= 速度等级 =========
DEFAULT_SPEED_LEVEL = 's'    # 默认并行模式
```

推荐模型

模型	输入价格	输出价格	特点	适用场景
`gemini-2.5-flash-lite`	¥0.0004/1K	¥0.0016/1K	最便宜	大批量处理
`deepseek-v3.2`	¥0.0012/1K	¥0.0018/1K	性价比极高	日常使用（推荐）
`gpt-4o-mini`	¥0.00105/1K	¥0.0042/1K	稳定可靠	生产环境
`grok-4-fast`	¥0.0008/1K	¥0.002/1K	非常便宜	大批量处理

完整模型价格表

以下是API支持的主要模型及其定价（单位：元/1K Tokens）：

GPT系列

模型	输入价格	输出价格	说明
gpt-5.2	0.01225	0.098	旗舰模型，编码和智能体任务
gpt-5.1	0.00875	0.07	编码和智能体任务
gpt-5	0.00875	0.07	跨领域编码、推理和代理任务
gpt-5-mini	0.00175	0.014	GPT-5的快速经济版本
gpt-5-nano	0.00035	0.0028	速度最快、成本最低
gpt-4.1	0.014	0.056	编码、指令跟踪、长上下文
gpt-4.1-mini	0.0028	0.0112	4.1的经济版本
gpt-4.1-nano	0.0007	0.0028	4.1的最低成本版本
gpt-4o	0.0175	0.07	多模态，支持图片
gpt-4o-mini	0.00105	0.0042	4o的经济版本，支持读图
gpt-3.5-turbo	0.0035	0.0105	基准模型

DeepSeek系列（性价比极高）

模型	输入价格	输出价格	说明
deepseek-v3.2	0.0012	0.0018	最新版本，输出更便宜
deepseek-v3	0.0012	0.0048	聊天模型
deepseek-r1	0.0024	0.0096	思考R1模型

Gemini系列（性价比高）

模型	输入价格	输出价格	说明
gemini-2.5-flash-lite	0.0004	0.0016	**最便宜**
gemini-2.5-flash	0.0006	0.014	性价比高
gemini-2.5-flash-nothinking	0.0006	0.005	最便宜的输出
gemini-2.5-pro	0.007	0.04	旗舰模型

Claude系列

模型	输入价格	输出价格	说明
claude-3-5-sonnet	0.015	0.075	主力模型
claude-3-5-haiku	0.005	0.025	入门模型
claude-opus-4	0.075	0.375	高端模型

其他模型

模型	输入价格	输出价格	说明
grok-4	0.012	0.06	Grok基础模型
grok-4-fast	0.0008	0.002	非常便宜
qwen3-235b-a22b	0.0014	0.0056	阿里通义千问
kimi-k2-0711-preview	0.0028	0.0112	Kimi最新模型

CA系列（第三方，更便宜但稳定性稍差）

模型	输入价格	输出价格	说明
gpt-5-ca	0.005	0.04	GPT-5第三方版本
gpt-5-nano-ca	0.0002	0.0016	极便宜
gpt-4o-mini-ca	0.00075	0.003	4o-mini第三方版本
gpt-4.1-nano-ca	0.0004	0.003	4.1-nano第三方版本

费用估算示例

以 `routers.json` 数据集（13,502条记录）为例，展示实际费用估算。

数据统计

指标	数值
原始记录数	13,502 条
原始文件大小	61.35 MB
原始平均每条	4,741 字符
清洗后平均每条	932 字符
压缩率	**80.4%**

Agent Prompt Token统计

Agent	Prompt字符数	Prompt Token数
产品分析Agent	1,149	361
用途分析Agent	1,434	361
校验Agent	1,139	327
合计	**3,722**	**1,049**

Token开销估算

以 Batch Size = 3 为例（共4,501批）：

Agent	输入Tokens	输出Tokens	平均输入/条	平均输出/条
产品分析Agent	6,072,781	2,025,300	449.8	150.0
用途分析Agent	6,086,284	2,430,360	450.8	180.0
校验Agent	8,326,400	1,620,240	450.0	120.0
合计	**20,485,465**	**6,075,900**	-	-

总Token数: 26,561,365 tokens

各模型费用对比

按总费用从低到高排序

模型	输入价格	输出价格	总费用	评价
gemini-2.5-flash-lite	0.0004	0.0016	**¥17.92**	⭐⭐⭐⭐ 最便宜
gpt-4.1-nano	0.0007	0.0028	**¥31.35**	⭐⭐⭐⭐ 官方稳定
gpt-4o-mini-ca	0.0008	0.0030	**¥33.59**	⭐⭐⭐⭐ 第三方便宜
deepseek-v3.2	0.0012	0.0018	**¥35.52**	⭐⭐⭐⭐⭐ 最佳性价比
gpt-4o-mini	0.0010	0.0042	**¥47.03**	⭐⭐⭐⭐ 官方稳定
deepseek-v3	0.0012	0.0048	**¥53.75**	⭐⭐⭐⭐⭐ 高性价比
qwen3-235b-a22b	0.0014	0.0056	**¥62.70**	⭐⭐⭐⭐ 国产高性能
gemini-2.5-flash	0.0006	0.0140	**¥97.35**	⭐⭐⭐⭐ 性价比高
gpt-5-mini	0.0018	0.0140	**¥120.91**	⭐⭐⭐ 新一代mini
gpt-4.1-mini	0.0028	0.0112	**¥125.41**	⭐⭐⭐ 官方中端
gpt-3.5-turbo	0.0035	0.0105	**¥135.50**	⭐⭐⭐ 基准模型
claude-3-5-haiku	0.0050	0.0250	**¥254.32**	⭐⭐ Claude入门
gpt-5	0.0088	0.0700	**¥604.56**	⭐⭐ 旗舰但贵
gpt-4.1	0.0140	0.0560	**¥627.05**	⭐⭐ 高端

推荐方案

🏆 最佳性价比：gemini-2.5-flash-lite

指标	数值
预估费用	**¥17.92**
相比gpt-3.5-turbo节省	86.8%

按需求推荐

追求最低成本

- `gemini-2.5-flash-lite`: ¥17.92

追求稳定性（OpenAI官方模型）

- `gpt-4.1-nano`: ¥31.35
- `gpt-4o-mini`: ¥47.03
- `gpt-5-mini`: ¥120.91

性价比平衡（能力强+价格低）

- `deepseek-v3.2`: ¥35.52（推荐）
- `deepseek-v3`: ¥53.75
- `qwen3-235b-a22b`: ¥62.70

数据清洗规则配置

```
# 需要删除的顶层字段
REMOVE_TOP_FIELDS = {"IP Index", "Timestamp", "OS"}

# 需要删除的服务字段
REMOVE_SERVICE_FIELDS = {"Body sha256"}

# HTTP错误响应模式
HTTP_ERROR_PATTERNS = [
```

```
        (r'400\s*Bad\s*Request', '400 Bad Request'),
        (r'404(\s*Not\s*Found)?', '404 Not Found'),
        # ...
]

# 默认页面模式
DEFAULT_PAGE_PATTERNS = [
        (r'Welcome\s*to\s*nginx', '[DEFAULT:nginx-default]'),
        (r'Apache2?\s*(Debian|Ubuntu)?\s*Default', '[DEFAULT:apache-default]'),
        # ...
]
```

输入输出格式

输入数据格式

输入文件为 JSON 或 JSONL 格式，每行一条记录：

```
{"192.168.1.1": {"Services": {"http-80": {"Banner": "...", "Html Body": "..."}}, "Domain Name": "example.com", "Country": "US", "ASN": "12345"}}
```

输出文件说明

文件	位置	说明
`cleaned_data.jsonl`	`6Analyst/data/output/`	清洗后的数据
`product_analysis.jsonl`	`6Analyst/data/output/`	产品分析结果
`usage_analysis.jsonl`	`6Analyst/data/output/`	用途分析结果
`merged_analysis.jsonl`	`6Analyst/data/output/`	合并的中间结果
`check_details.jsonl`	`6Analyst/data/output/`	校验详情
`run_state.json`	`6Analyst/data/output/`	运行状态（断点续传用）
`final_analysis.jsonl`	项目根目录	**最终分析结果**
`cost_report.md`	项目根目录	费用评估报告

最终结果格式（`final_analysis.jsonl`）


```
{
  "ip": "192.168.1.1",
  "vendor": "MikroTik",
  "model": [["RB750Gr3", 0.85]],
  "os": "RouterOS 6.49.10",
  "firmware": "MikroTik 6.49.10",
  "type": "router",
  "result_type": "direct",
  "confidence": 0.85,
  "evidence": [{"src": "ftp_banner", "val": "MikroTik FTP server", "weight": 0.9}],
  "primary_usage": "network_infrastructure",
  "secondary_usages": [],
  "industry": "ISP",
  "services_detected": ["http", "ftp", "ssh"],
  "usage_confidence": 0.9,
  "validation_status": "verified",
```

```
    "evidence_quality": "strong",
    "status": "done",
    "status_detail": "all_agents_completed"
}
```

日志文件

文件	位置	说明
`log_YYYYMMDD_HHMMSS.txt`	`6Analyst/data/output/log/`	单次运行日志
`log_taskN_*_main.txt`	`6Analyst/data/output/log/`	多线程主日志
`log_taskN_*_workerN.txt`	`6Analyst/data/output/log/`	工作线程日志
`merged_latest.txt`	`6Analyst/data/output/log/`	合并后的日志
`error_log.txt`	`6Analyst/data/output/log/`	错误详情（含完整提示词和响应）

进度显示格式

```
进度: 45.2%  678/1500  高置信度: 450 中置信度: 180 不可信: 48 错误: 0 | Verified: 320 Adjust: 280 Reject: 50 | Tok/条: 1200/150  ¥0.1234(总估¥0.2000) 用时: 5.2m / 剩余: 6.3m | 线程: 4活跃
```

字段	说明
进度	已处理/总数
高/中/低置信度	按置信度分类（≥0.8 / 0.6-0.8 / <0.6）
Verified/Adjust/Reject	校验状态统计
Tok/条	实时平均输入/输出Token
¥费用(总估¥)	累计费用 / 估算总费用
用时/剩余	本次运行用时和预估剩余时间

辅助工具

1. 提示词管理模块（`prompts/`）

提供多版本提示词的管理、切换和费用计算功能，支持为不同场景选择最优的提示词配置。

1.1 提示词文件结构

提示词配置存储在 `6Analyst/prompts/` 目录下：

文件	说明
`product_prompts.json`	产品分析Agent的提示词配置
`usage_prompts.json`	用途分析Agent的提示词配置
`check_prompts.json`	校验Agent的提示词配置
`__init__.py`	提示词加载和费用计算函数

1.2 提示词ID格式

Agent	ID格式	示例
Product Agent	`p1`, `p2`, `p3`, `default`	`-p p1`
Usage Agent	`u1`, `u2`, `u3`, `default`	`-u u2`
Check Agent	`c1`, `c2`, `c3`, `default`	`-c c3`

1.3 提示词配置JSON格式

```
{
  "prompts": [
    {
      "id": "default",
      "name": "默认提示词",
      "description": "程序内置的默认提示词",
```

```
      "system_prompt": null,
      "cost_info": {
        "char_count": 1149,
        "token_count": 368,
        "record_count": 13502,
        "avg_input_tokens": 331.8,
        "avg_output_tokens": 146.4,
        "cost_per_1k_records": {
          "deepseek-v3.2": 0.8088,
          "gpt-4o-mini": 1.0919,
          "gemini-2.5-flash": 2.3218
        }
      }
    },
    {
      "id": "p1",
      "name": "精简版-强调推理",
      "description": "精简提示词，强调基于证据推理",
      "system_prompt": "Identify device vendor/model/OS/firmware...",
      "cost_info": { ... }
    }
  ]
}
```

1.4 命令行使用

```
# 查看所有可用提示词
python run_6analyst.py --prompt --list

# 更新所有提示词的费用信息（基于现有数据）
python run_6analyst.py --prompt --update

# 指定各Agent使用的提示词
python run_6analyst.py --prompt -p p1                # 产品Agent使用p1
python run_6analyst.py --prompt -u u2 -c c1       # 用途u2, 校验c1
python run_6analyst.py --prompt -p p3 -u u3 -c c3   # 全部使用v3版本

# 生成提示词配置HTML页面
python run_6analyst.py --prompt --page
```

1.5 费用计算说明

每个提示词的 `cost_info` 包含:

字段	说明
`char_count`	提示词字符数
`token_count`	提示词Token数
`record_count`	计算基于的数据条数
`avg_input_tokens`	平均每条数据的输入Token
`avg_output_tokens`	平均每条数据的输出Token
`cost_per_1k_records`	各模型处理1000条数据的费用

费用计算公式:

```
每1K条开销 = (批次数 × 提示词Token + 1000 × 平均数据Token) × 输入价格
                + 1000 × 平均输出Token × 输出价格
```

---

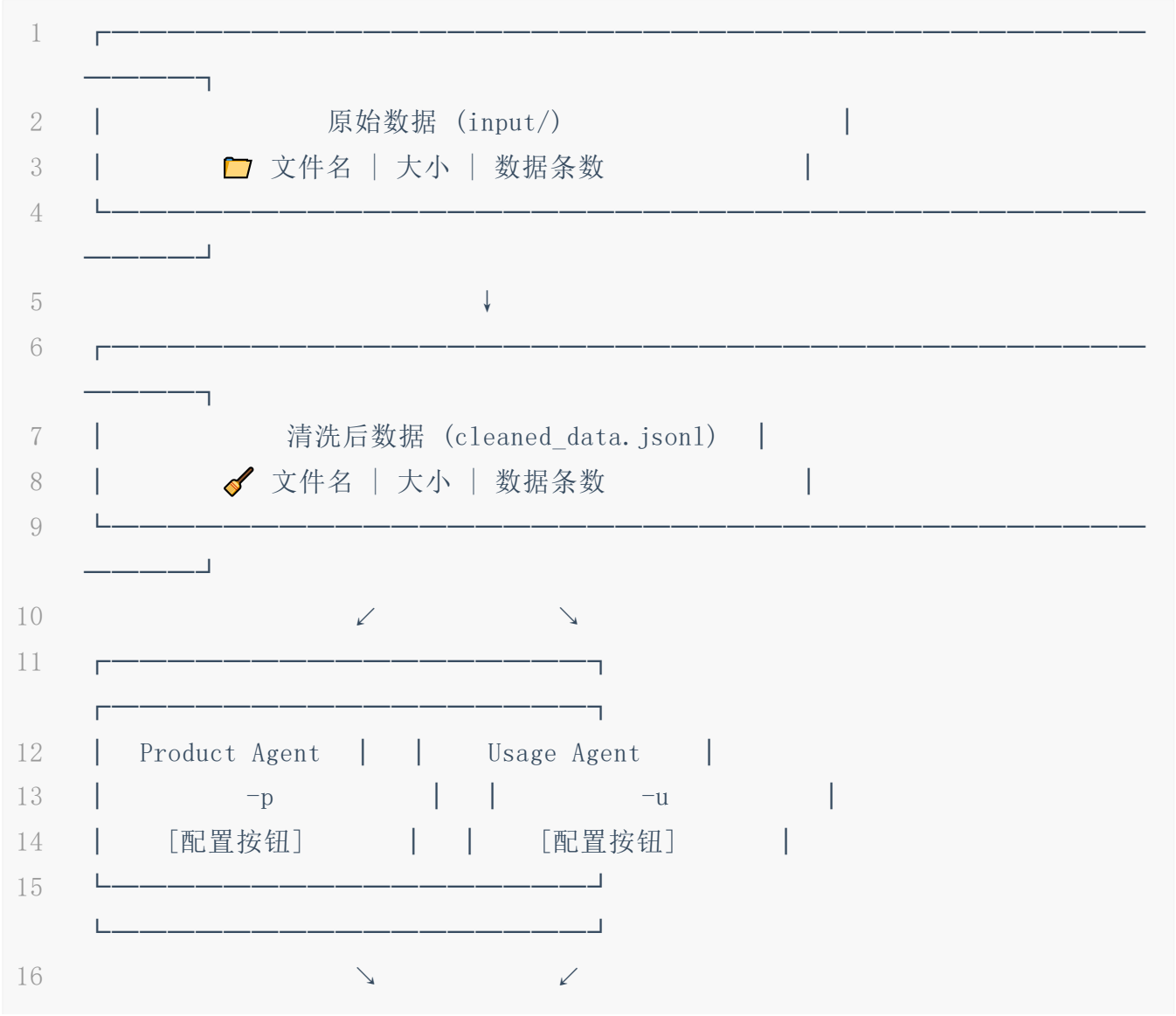
2. 提示词配置页面（`tools/generate_prompt_page.py`）

生成可视化的HTML配置页面，用于选择提示词和模型，预估总开销。

2.1 功能特性

- **数据流向可视化**：展示完整的处理流程（原始数据→清洗→Agent分析→校验→输出）
- **提示词选择**：为每个Agent选择不同版本的提示词
- **模型选择**：下拉列表显示所有可用模型及对应开销
- **费用预估**：根据数据量、提示词、模型实时计算总开销
- **命令预览**：生成对应的命令行参数

2.2 页面布局



2.3 Agent配置弹窗

点击Agent卡片的"配置"按钮，弹出提示词选择表格：

ID	名称	长度	每1K条开销（选择模型）	操作
default	默认提示词	1149字/368tk	[模型下拉▼]	[详情] [选择]
p1	精简版	658字/208tk	[模型下拉▼]	[详情] [选择]
p2	详细版	1118字/334tk	[模型下拉▼]	[详情] [选择]

模型下拉列表：

- 展开后每行显示：模型名称（左）+ 该提示词对应的开销（右）
- 按类别分组：DeepSeek、Gemini、GPT-5、GPT-4、Claude、Qwen、Kimi、Grok等
- 点击即可选择模型

2.4 使用方法

```
# 生成配置页面并自动在浏览器中打开
python run_6analyst.py --prompt --page
```

页面生成位置：`6Analyst/data/output/html_report/prompt_config.html`

2.5 配置流程

1. 打开配置页面
2. 点击各Agent卡片的"配置"按钮
3. 在弹窗中选择提示词版本
4. 在下拉列表中选择模型（显示对应开销）
5. 点击"选择"按钮确认
6. 查看底部的总开销预估
7. 复制生成的命令行参数

---

3. 费用计算模块（`cost_calculator.py`）

提供Token消耗和费用的预估功能，帮助在大批量处理前评估成本。

功能说明

- **基于中间文件计算**：使用已有的中间文件计算各Agent的Token消耗
- **基于原始文件计算**：先清洗原始输入文件，再计算Token消耗
- **数据量估算**：根据样本数据的平均值，估算指定数据量的总开销

依赖的中间文件

费用计算依赖以下中间文件来获取准确的平均Token数：

文件	用途
`cleaned_data.jsonl`	计算产品/用途Agent的输入Token平均值
`product_analysis.jsonl`	计算产品Agent的输出Token平均值
`usage_analysis.jsonl`	计算用途Agent的输出Token平均值
`merged_analysis.jsonl`	计算校验Agent的输入Token平均值
`check_details.jsonl`	计算校验Agent的输出Token平均值

如果缺少输出文件，计算器会使用默认估算值（产品150、用途180、校验120 tokens/条）。

计算原理

1. 加载中间文件（清洗数据、分析结果、合并结果）
2. 按批次构建各Agent的提示词
3. 使用tiktoken计算输入Token数
4. 根据已有输出数据计算平均输出Token数
5. 按模型价格计算费用
6. 如果指定了 `--datanum`，按比例外推到目标数据量

使用方法

```
# 基于已有中间文件计算费用（需要先运行过完整流程）
python run_6analyst.py --calculate-cost

# 基于原始输入文件计算费用（会临时清洗数据，输出Token使用默认值）
python run_6analyst.py --calculate-cost --file-cost

# 基于样本数据估算大数据量的费用（推荐：先运行500条样本）
python run_6analyst.py --calculate-cost --datanum 100000
```

```
9
10  # 指定批次大小（影响系统提示词的摊销）
11  python run_6analyst.py --calculate-cost --batch-size 5
12
13  # 指定输入路径
14  python run_6analyst.py --calculate-cost --file-cost --input /path/to/data
```

推荐使用流程

1. 先运行小批量数据（如500条）产生完整的中间文件
2. 使用 `--calculate-cost --datanum N` 外推到目标数据量
3. 这样可以获得最准确的费用估算

输出报告

计算完成后自动生成 `cost_report.md`，包含：

- 模型名称和定价
- 各Agent的输入/输出Token数和费用
- 汇总统计（总Token数、总费用、单条成本）
- 多模型费用对比表

---

4. 日志清理工具（`tools/clean_log.py`）

合并和清理日志文件，避免日志文件过多。

清理逻辑

普通模式（`--clean-log`）：

1. 扫描日志目录中的所有 `log_*.txt` 文件
2. 仅合并当前任务ID的日志文件
3. 保留最新的一个日志文件（可能正在写入）
4. 如果 `merged_latest.txt` 属于同一任务，则追加内容
5. 如果属于不同任务，则覆盖写入

强制模式（`--clean-log -f`）：

1. 合并所有零碎日志文件（包括最新的）
2. 删除所有已合并的零碎日志
3. 如果 `merged_latest.txt` 属于同一任务，则追加内容

使用方法

```
# 通过主程序调用
python run_6analyst.py --clean-log
python run_6analyst.py --clean-log -f

# 独立运行
python 6Analyst/tools/clean_log.py
python 6Analyst/tools/clean_log.py -f
```

---

5. Token矫正工具（`tools/fix_tokens.py`）

修正 `run_state.json` 中的累计Token数据。

使用场景

当统计数据出现异常（如程序崩溃导致数据不一致）时，可以使用此工具重新计算：

- 根据实时平均Token × 已处理条数，重新计算累计Token
- 更新 `run_state.json` 中的 `input_tokens` 和 `output_tokens`

使用方法

```
python -m 6Analyst.tools.fix_tokens
```

---

6. 状态修复工具（`tools/fix_status.py`）

修复 `final_analysis.jsonl` 中的状态字段。

使用场景

当结果文件中存在状态不一致的记录时，可以使用此工具修复：

- 检查每条记录的 `status` 和 `status_detail` 字段
- 根据其他字段的值推断正确的状态

---

7. HTML报告生成（`tools/generate_html_report.py`）

生成可视化的HTML分析报告。

功能说明

- 读取 `final_analysis.jsonl` 中的分析结果
- 生成包含统计图表的HTML报告
- 支持按厂商、设备类型、用途、行业等维度统计

使用方法

```
# 生成报告并在浏览器中打开
python run_6analyst.py --show
```

---

8. 设备统计分析（`tools/analyze_device_stats.py`）

分析最终结果中的设备分布统计。

功能说明

- 统计各厂商的设备数量
- 统计各设备类型的分布
- 统计各用途分类的分布
- 统计置信度分布

---

7. 错误日志模块（`utils/error_logger.py`）

专门用于记录程序运行中的错误信息，便于调试。

记录内容

- **批次上下文**：批次ID、Agent名称、IP列表、开始时间
- **处理步骤**：每个步骤的时间戳和详情
- **完整提示词**：发送给API的完整消息（填充数据后）
- **原始响应**：API返回的原始字符串
- **错误信息**：错误类型、错误消息、异常堆栈

错误日志位置

`6Analyst/data/output/log/error_log.txt`

错误日志格式

```
================================================================================
[ERROR] 2025-12-31 10:30:45.123
错误类型: json_parse_error
错误消息: 所有JSON解析方法均失败
================================================================================

--- 批次信息 ---
批次ID: 42
Agent: ProductAnalyst
开始时间: 2025-12-31 10:30:40.100
IP列表: ['192.168.1.1', '192.168.1.2', '192.168.1.3']

--- 处理步骤 (6步) ---
  1. [2025-12-31 10:30:40.100] 开始处理批次
  2. [2025-12-31 10:30:40.101] 构建提示词
  3. [2025-12-31 10:30:40.102] 计算Token
  4. [2025-12-31 10:30:40.103] 调用API
```

```
  5. [2025-12-31 10:30:44.500] 收到API响应
  6. [2025-12-31 10:30:44.501] JSON直接解析失败

--- 完整提示词输入 ---
[system]:
Analyze network scan records to identify device vendor and model...

[user]:
Analyze these 3 records:
{"192.168.1.1": {"Services": ...}}
...

--- 原始返回结果 ---
Sorry, I cannot process this request...

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

项目结构

```
6Analyst/
├── __init__.py                    # 包初始化
├── config.py                      # 配置模块（API、路径、清洗规则、模型价格）
├── base_analyst.py                # 分析Agent基类（API调用、Token计算、JSON解析）
├── product_analyst.py             # 产品分析Agent（厂商、型号、OS、固件）
├── usage_analyst.py               # 用途分析Agent（用途、行业、服务）
├── check_analyst.py               # 校验Agent（验证、修正、评估）
├── data_cleaner.py                # 数据清洗模块（字段过滤、Banner压缩、HTML简化）
├── cost_calculator.py             # 费用计算模块（Token统计、费用估算）
├── run.py                         # 运行控制器（单线程模式、命令行解析）
├── thread_safe.py                 # 线程安全组件（任务管理、统计、文件写入、日志）
```

```
├── multi_thread_runner.py     # 多线程执行器（并行处理、全局限流）
│
├── data/
│   ├── input/                          # 输入数据目录
│   │   └── routers.json          # 示例输入文件
│   ├── input_test/                 # 测试数据目录
│   │   └── test_routers.json
│   └── output/                         # 输出数据目录
│       ├── cleaned_data.jsonl
│       ├── product_analysis.jsonl
│       ├── usage_analysis.jsonl
│       ├── merged_analysis.jsonl
│       ├── check_details.jsonl
│       ├── run_state.json
│       └── log/                        # 日志目录
│           ├── log_*.txt
│           ├── merged_latest.txt
│           └── error_log.txt
│
├── utils/
│   ├── __init__.py
│   ├── html_extractor.py       # HTML解析工具（提取关键信息）
│   ├── token_counter.py        # Token计算工具（tiktoken封装）
│   ├── logger.py                   # 日志工具（文件日志+控制台）
│   └── error_logger.py          # 错误日志工具（完整提示词和响应记录）
│
├── tools/
│   ├── clean_log.py              # 日志清理工具
│   ├── fix_tokens.py             # Token矫正工具
│   ├── fix_status.py             # 状态修复工具
│   ├── generate_cost_report.py    # 费用报告生成
│   ├── generate_html_report.py    # HTML报告生成
│   └── analyze_device_stats.py    # 设备统计分析
│
└── tests/
    └── __init__.py
```

错误处理机制

1. 错误类型检测

程序自动检测API响应中的三类错误：

并发限制（Rate Limit）

- 检测关键词：`rate limit`、`too many requests`、`429`、`请求过于频繁`、`并发`、`频率限制`
- 处理方式：等待后重试，自动降级

安全限制（Security Limit）

- 检测关键词：`security`、`blocked`、`forbidden`、`403`、`安全策略`、`拦截`、`风控`
- 处理方式：需连续5次检测到才触发（避免误报），然后等待重试
- 容错机制：单次检测到仅记录警告，不触发等待

余额不足（Insufficient Balance）

- 检测关键词：`insufficient balance`、`quota exceeded`、`余额不足`、`额度不足`、`欠费`
- 处理方式：记录日志并立即终止程序

2. 误报防护

为避免将正常响应误判为错误，程序会先检查响应是否包含正常字段：

- 检查字段：`"ip"`、`"confidence"`、`"vendor"`、`"model"`、`"primary_usage"`、`"validation_status"`、`"evidence"`、`"firmware"`、`"os"`、`"type"`
- 检查格式：响应以 `[` 或 `{` 开头，且包含多个JSON特征（`":`、`",`、`{}`、`[]`、`null`、`true`、`false`）
- 如果检测到正常响应特征，跳过错误关键词检测

3. 指数退避重试

触发并发限制或安全限制后的重试策略：

重试次数	等待时间
第1次	30分钟
第2次	60分钟
第3次	120分钟
第4次	240分钟
...	翻倍递增

4. 自动降级

触发限制后自动降低速度等级：

- 从 `s`（并行模式）降至 `6`（极速串行）
- 从 `6` 降至 `5`，从 `5` 降至 `4`，以此类推
- 最低降至等级 `1`（间隔10秒）

5. 断点续传

程序支持中断后继续处理：

- 启动时加载 `run_state.json` 恢复统计数据
- 从 `final_analysis.jsonl` 提取已处理的IP列表
- 自动跳过已处理的记录
- 任务ID保持不变

6. JSON解析容错

API响应的JSON解析采用多种方式尝试：

1. **直接解析**：`json.loads(reply)`

2. **修复尾部逗号**：删除 `},]` 或 `},}` 中的多余逗号
3. **按行解析**：将响应按行分割，逐行解析
4. **正则提取**：使用正则表达式 `\{[^{}]*\}` 提取JSON对象

如果所有方法都失败，记录详细错误日志到 `error_log.txt`。

使用示例

本节提供多个复杂使用场景的完整示例，涵盖从项目启动到生产部署的各个阶段。

场景一：首次使用 - 环境验证与小规模测试

背景：刚配置好API，需要验证环境是否正常工作。

步骤1：配置API

编辑 `6Analyst/config.py`：

```
API_KEY = "sk-your-api-key"
BASE_URL = "https://api.chatanywhere.tech"
MODEL_NAME = "deepseek-v3.2"   # 推荐性价比高的模型
```

步骤2：使用测试数据集验证

```
# 使用内置测试数据（6Analyst/data/input_test/test_routers.json）
python run_6analyst.py --test
```

这会使用少量测试数据运行完整流程，验证：

- API连接是否正常
- 数据清洗是否工作
- 三个Agent是否能正常返回结果
- 输出文件是否正确生成

步骤3：检查输出

```
# 查看最终结果
type final_analysis.jsonl

# 生成HTML报告并在浏览器中查看
python run_6analyst.py --show
```

预期结果：

- 控制台显示进度条和统计信息
- `final_analysis.jsonl` 包含分析结果
- 浏览器打开HTML报告，展示设备分布统计

场景二：大规模数据处理前的费用评估

背景：有一个包含10万条记录的大型数据集，需要在处理前评估费用。

原理说明：

费用计算需要依赖中间文件来获取准确的平均Token数：

- `cleaned_data.jsonl`：计算输入Token的平均值
- `product_analysis.jsonl`、`usage_analysis.jsonl`：计算输出Token的平均值
- `merged_analysis.jsonl`：计算校验Agent输入Token的平均值

因此，正确的做法是先运行一小批数据产生完整的中间文件，再基于这些样本数据外推到大数据量。

步骤1：运行小批量数据产生样本

```
# 运行完整流程处理500-1000条数据，产生所有中间文件
python run_6analyst.py --max-records 500
```

这会产生：

- `cleaned_data.jsonl`（500条清洗后数据）
- `product_analysis.jsonl`（500条产品分析结果）
- `usage_analysis.jsonl`（500条用途分析结果）

- `merged_analysis.jsonl`（500条合并结果）
- `check_details.jsonl`（500条校验详情）
- `final_analysis.jsonl`（500条最终结果）

步骤2：基于样本外推到大数据量

```
# 基于已有的500条样本数据，估算10万条的费用
python run_6analyst.py --calculate-cost --datanum 100000
```

计算器会：

1. 从中间文件计算各Agent的平均输入/输出Token
2. 按比例外推到10万条数据
3. 生成详细的费用报告

步骤3：查看费用报告

```
type cost_report.md
```

报告会显示：

- 各模型的预估费用对比（按总费用排序）
- 推荐的性价比方案
- Token消耗明细（输入/输出分开统计）
- 各Agent的费用占比

步骤4：根据预算选择模型

假设预算有限，选择最便宜的模型：

```
# 修改 config.py
MODEL_NAME = "gemini-2.5-flash-lite"   # 最便宜，约 ¥0.13/千条
```

或选择性价比平衡的模型：

```
MODEL_NAME = "deepseek-v3.2"   # 性价比高，约 ¥0.26/千条
```

步骤5：重新开始正式处理

确认模型后，清除样本数据，开始正式处理：

```
python run_6analyst.py --restart
```

费用参考（以10万条数据为例）：

模型	预估费用
gemini-2.5-flash-lite	~¥133
deepseek-v3.2	~¥263
gpt-4o-mini	~¥348

备选方案：基于原始文件直接估算

如果不想先运行样本，可以使用 `--file-cost` 参数直接基于原始输入文件估算：

```
# 直接基于原始输入文件计算费用（会临时清洗数据）
python run_6analyst.py --calculate-cost --file-cost --input
/path/to/large_dataset.json
```

注意：此方式的输出Token估算使用默认值（产品150、用途180、校验120 tokens/条），准确度不如基于实际样本的估算。

场景三：生产环境 - 大批量多线程处理

背景：需要处理13,502条路由器数据，要求高吞吐量和稳定性。

步骤1：先清洗数据（单独执行，便于检查）

```
python run_6analyst.py --clean-only --input 6Analyst/data/input/routers.json
```

清洗完成后检查压缩效果：

- 原始数据：61.35 MB
- 清洗后：约12 MB（压缩率80%+）

步骤2：多线程并行分析

```
# 8线程 + 并行Agent模式（ProductAnalyst和UsageAnalyst同时执行）
python run_6analyst.py -t 8 --speed-level s
```

参数说明：

- `-t 8`：使用8个工作线程并行处理不同批次
- `--speed-level s`：每个线程内部，ProductAnalyst和UsageAnalyst也并行执行

步骤3：监控进度

运行时控制台会实时显示：

```
进度: 45.2%  6100/13502  高置信度: 4500 中置信度: 1200 不可信: 400 错误: 0 | Verified: 3200 Adjust: 2500 Reject: 400 | Tok/条: 1200/150  ¥12.34(总估¥27.30)  用时: 25.3m / 剩余: 30.8m | 线程: 8活跃
```

步骤4：处理完成后清理日志

```
python run_6analyst.py --clean-log -f
```

预期耗时：

- 8线程 + 并行模式：约1小时处理13,502条
- 单线程模式：约6-8小时

场景四：中断恢复 - 断点续传

背景：处理过程中因网络问题或手动中断，需要继续处理。

情况A：程序意外中断

直接重新运行相同命令：

```
python run_6analyst.py -t 8 --speed-level s
```

程序会自动：

1. 加载 `run_state.json` 恢复统计数据

2. 从 `final_analysis.jsonl` 读取已处理的IP列表
3. 跳过已处理的记录，继续处理剩余部分
4. 保持相同的任务ID

情况B：需要重新开始

如果发现之前的结果有问题，需要从头开始：

```
python run_6analyst.py -t 8 --speed-level s --restart
```

`--restart` 会：

1. 清除已有的输出文件
2. 任务ID自动+1（便于区分不同批次）
3. 从第一条记录开始处理

情况C：只想重新执行校验步骤

如果产品分析和用途分析已完成，只想重新校验：

```
python run_6analyst.py --check-only
```

场景五：API限流处理 - 自动降级与恢复

背景：使用免费API Key，有请求频率限制（如200次/天）。

策略1：使用低速模式避免触发限流

```
# 速度等级3：每个Agent调用间隔1秒
python run_6analyst.py --speed-level 3

# 速度等级1：每个Agent调用间隔10秒（最保守）
python run_6analyst.py --speed-level 1
```

策略2：让程序自动处理限流

即使使用高速模式，程序也会自动处理限流：

```
python run_6analyst.py -t 4 --speed-level s
```

当检测到限流时，程序会：

1. 显示等待倒计时（首次30分钟，后续翻倍）
2. 自动降低速度等级
3. 等待结束后继续处理

策略3：分时段处理

如果每天有200次限制，可以分多天处理：

```
# 第一天：处理前200条
python run_6analyst.py --max-records 200

# 第二天：继续处理（自动跳过已处理的）
python run_6analyst.py --max-records 400

# 以此类推...
```

场景六：分步执行 - 精细控制流程

背景：需要在每个阶段检查结果，或只执行部分流程。

步骤1：仅清洗数据

```
python run_6analyst.py --clean-only
```

检查清洗结果：

```
# 查看清洗后的数据
type 6Analyst\data\output\cleaned_data.jsonl | more
```

步骤2：仅执行产品分析

```
python run_6analyst.py --product-only
```

检查产品分析结果：

```
type 6Analyst\data\output\product_analysis.jsonl | more
```

步骤3：仅执行用途分析

```
python run_6analyst.py --usage-only
```

步骤4：仅执行校验

```
python run_6analyst.py --check-only
```

步骤5：跳过校验的完整流程

如果不需要校验步骤（节省约1/3的API调用）：

```
python run_6analyst.py --no-check
```

场景七：多数据源处理

背景：有多个不同来源的数据文件需要分别处理。

方法1：指定输入文件

```
# 处理第一个数据源
python run_6analyst.py --input D:\data\source1\devices.json --restart

# 处理第二个数据源（需要先备份之前的结果）
copy final_analysis.jsonl final_analysis_source1.jsonl
python run_6analyst.py --input D:\data\source2\devices.json --restart
```

方法2：指定输入目录

如果一个目录下有多个JSON文件：

```
# 处理整个目录下的所有JSON/JSONL文件
python run_6analyst.py --input D:\data\all_sources\
```

方法3：合并多个结果

处理完多个数据源后，可以手动合并结果：

```
type final_analysis_source1.jsonl final_analysis_source2.jsonl >
final_analysis_merged.jsonl
```

场景八：调试与问题排查

背景：分析结果不符合预期，需要排查问题。

步骤1：启用调试模式

```
python run_6analyst.py --debug --max-records 10
```

调试模式会输出：

- 详细的JSON解析过程
- API响应的原始内容
- 每个解析方法的尝试结果

步骤2：查看错误日志

```
type 6Analyst\data\output\log\error_log.txt
```

错误日志包含：

- 完整的提示词（发送给API的内容）
- API的原始响应
- 错误类型和堆栈信息

步骤3：检查特定IP的处理结果

在 `final_analysis.jsonl` 中搜索特定IP：

```
findstr "192.168.1.1" final_analysis.jsonl
```

步骤4：检查校验详情

```
# 查看校验Agent的详细输出
type 6Analyst\data\output\check_details.jsonl | more
```

校验详情包含：

- `validation_status`：verified/adjusted/rejected

- `issues_found`：发现的问题列表
- `adjustments`：做出的修正

场景九：定期任务 - 自动化处理

背景：需要定期处理新数据，实现自动化流程。

Windows任务计划示例

创建批处理文件 `run_analysis.bat`：

```
@echo off
cd /d D:\projects\6Analyst
python run_6analyst.py -t 4 --speed-level s --input D:\data\daily\
python run_6analyst.py --clean-log -f
copy final_analysis.jsonl
D:\reports\analysis_%date:~0,4%%date:~5,2%%date:~8,2%.jsonl
```

然后在Windows任务计划程序中设置定时执行。

处理完成后的后续操作

```
# 生成HTML报告
python run_6analyst.py --show

# 提取设备统计
python -m 6Analyst.tools.analyze_device_stats
```

场景十：性能优化 - 不同配置对比

背景：需要找到最适合当前环境的配置。

配置对比测试

使用相同的100条数据，测试不同配置：

```
# 配置1：单线程 + 串行Agent
python run_6analyst.py --max-records 100 --speed-level 6 --restart
# 记录耗时：约X分钟
```

```
4
5  # 配置2：单线程 + 并行Agent
6  python run_6analyst.py --max-records 100 --speed-level s --restart
7  # 记录耗时：约Y分钟
8
9  # 配置3：4线程 + 并行Agent
10 python run_6analyst.py --max-records 100 -t 4 --speed-level s --restart
11 # 记录耗时：约Z分钟
12
13 # 配置4：8线程 + 并行Agent
14 python run_6analyst.py --max-records 100 -t 8 --speed-level s --restart
15 # 记录耗时：约W分钟
```

性能参考（以13,502条数据为例）：

配置	预估耗时	适用场景
单线程 + 串行	6-8小时	API严格限流
单线程 + 并行	4-5小时	一般使用
4线程 + 并行	1.5-2小时	推荐配置
8线程 + 并行	45-60分钟	API无限流

批次大小优化

默认批次大小为3，可以根据数据特点调整：

```
1  # 数据较简单时，增大批次可提高效率
2  python run_6analyst.py --batch-size 5
3
4  # 数据复杂时，减小批次可提高准确率
5  python run_6analyst.py --batch-size 2
```

场景十一：结果分析与报告生成

背景：处理完成后，需要生成各种维度的分析报告。

步骤1：生成HTML可视化报告

```
python run_6analyst.py --show
```

报告包含：

- 厂商分布饼图
- 设备类型分布
- 用途分类统计
- 置信度分布直方图
- 校验状态统计

步骤2：提取特定厂商的设备

```
# 提取所有MikroTik设备
findstr "MikroTik" final_analysis.jsonl > mikrotik_devices.jsonl

# 提取所有Cisco设备
findstr "Cisco" final_analysis.jsonl > cisco_devices.jsonl
```

步骤3：统计高置信度结果

```
# 使用Python脚本统计
python -c "
import json
high_conf = 0
total = 0
with open('final_analysis.jsonl') as f:
      for line in f:
           data = json.loads(line)
           total += 1
            if data.get('confidence', 0) >= 0.8:
                high_conf += 1
print(f'高置信度: {high_conf}/{total} ({high_conf/total*100:.1f}%)')
"
```

步骤4：导出为CSV格式

```
python -c "
import json
import csv
with open('final_analysis.jsonl') as f, open('analysis.csv', 'w', newline='', encoding='utf-8') as out:
     writer = csv.writer(out)
     writer.writerow(['IP', 'Vendor', 'Model', 'OS', 'Type', 'Usage', 'Confidence'])
      for line in f:
           d = json.loads(line)
           model = d.get('model', [[None]])[0][0] if d.get('model') else None
           writer.writerow([d.get('ip'), d.get('vendor'), model, d.get('os'), d.get('type'), d.get('primary_usage'), d.get('confidence')])
print('已导出到 analysis.csv')
"
```

常用命令速查表

场景	命令
首次测试	`python run_6analyst.py --test`
运行样本	`python run_6analyst.py --max-records 500`
费用预估	`python run_6analyst.py --calculate-cost --datanum 100000`
生产处理	`python run_6analyst.py -t 8 --speed-level s`
断点续传	`python run_6analyst.py -t 8 --speed-level s` （直接重新运行）
重新开始	`python run_6analyst.py --restart`
仅清洗	`python run_6analyst.py --clean-only`
跳过校验	`python run_6analyst.py --no-check`
限制条数	`python run_6analyst.py --max-records 1000`
指定输入	`python run_6analyst.py --input /path/to/data`
调试模式	`python run_6analyst.py --debug --max-records 10`
清理日志	`python run_6analyst.py --clean-log -f`
生成报告	`python run_6analyst.py --show`

注意事项

1. **API配置**：确保 API Key 有效且有足够配额
2. **数据格式**：输入数据需符合预期的 JSON/JSONL 格式
3. **首次运行**：建议先用 `--test` 或 `--max-records 10` 测试
4. **费用预估**：建议先运行500条样本产生中间文件，再用 `--calculate-cost --datanum N` 外推估算
5. **中断恢复**：程序支持断点续传，无需担心中断

6. **日志管理**：定期使用 `--clean-log` 清理日志
7. **多线程模式**：API无限流时推荐使用 `-t 4` 以上