Arche

面向 AI Agent 经济的 EVM Layer 2 公链

技术架构白皮书

Technical Architecture Whitepaper

v1.0 | 2026

一、项目摘要

1.1 一句话定义

> Arche 是一条专为 AI Agent 经济而生的 EVM 兼容 Layer 2 公链。它不重复造 AI 模型，而是为 Agent 提供链上身份、链上钱包、链上协作、链上信用与链上结算的统一基础设施。

1.2 核心命题

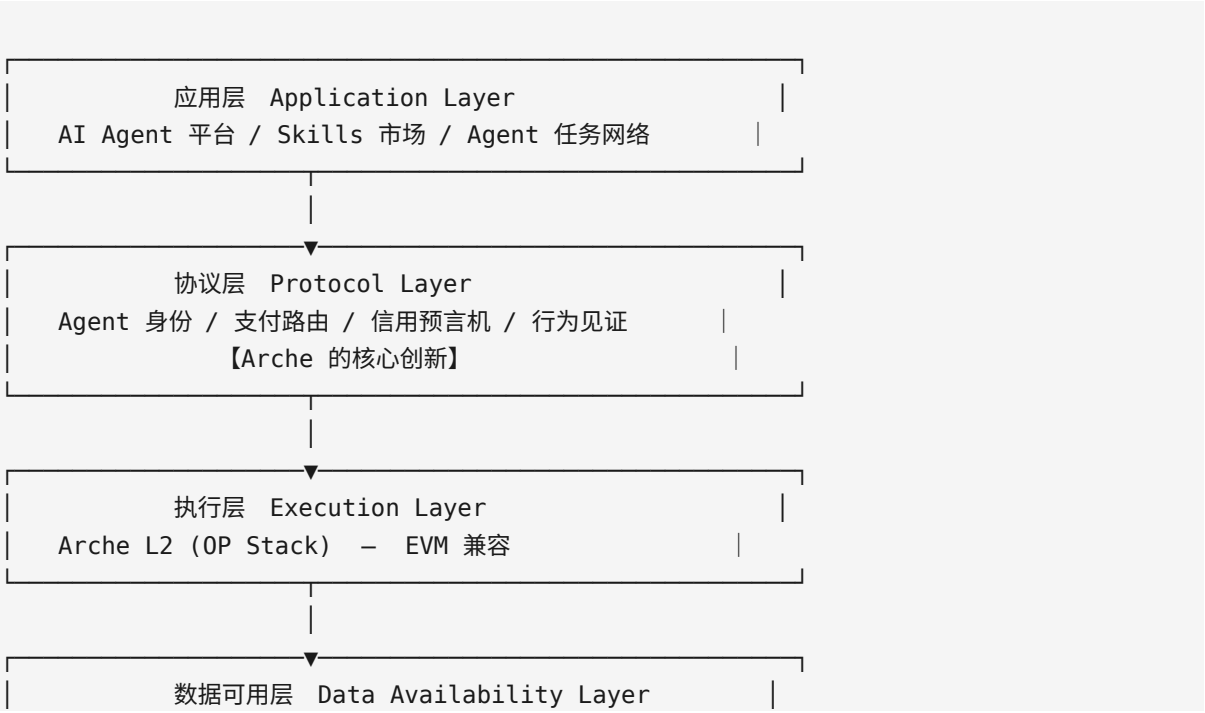
当 AI Agent 数量从今天的数百万增长到数十亿，每个 Agent 都将成为一个独立的经济单元——它需要拥有自己的身份、自己的钱包、自己的工作记录、自己的信用评分，并且能够与其他 Agent 进行支付、协作、交易。

当前的 AI Agent 平台（如各类 Agent 编排工具）解决了"如何让 Agent 工作"的问题，但没有解决以下五个核心问题：

- Agent 如何拥有可验证的、跨平台的身份？
- Agent 之间如何进行自主的、低成本的支付与结算？
- Agent 的工作记录与决策过程如何被审计与追溯？
- Agent 的服务质量与信誉如何被量化与转移？
- 优秀的 Agent 如何作为资产被持有、交易、继承？

Arche 的使命，是把上述五个问题的答案，写入一条专用的 EVM 公链。

1.3 技术定位



1.4 关键技术决策

决策项	选择	核心理由
Rollup 框架	OP Stack	Superchain 生态，未来跨链互通，Coinbase 加持
数据可用层	EigenDA	成本比以太坊 blob 低约 10 倍，吞吐高，适合 Agent 高频小额交易
共识机制	OP Stack 默认（PoS + 单 Sequencer 起步，分阶段去中心化）	成熟、可审计、可演进
智能合约语言	Solidity	生态最成熟，开发者人才储备最深
账户标准	ERC-4337 + ERC-6551	账户抽象 + Agent NFT 持币能力
资产标准	ERC-20 + ERC-1155	代币 + Skills 多版本资产化
验证机制	Optimistic + 选择性见证	务实可落地，不承诺无法兑现的 zkML
跨链桥	原生 OP 桥 + LayerZero V2	双桥冗余

二、设计哲学

2.1 三条核心原则

原则一：不重复造轮子

Arche 不自研基础大语言模型，不自研共识算法，不自研虚拟机。Arche 把所有可以外采的成熟组件外采（OP Stack、EigenDA、LLM API），把工程资源集中在真正具备差异化价值的层面：Agent 协议层。

原则二：可落地优于可炫耀

Arche 不承诺自身技术做不到的事。例如，业界常见的"用零知识证明确保 AI 推理正确"（zkML），在大语言模型规模下，2026 年仍处于学术研究阶段。Arche 不在白皮书中作此承诺，而是采用"链上行为见证 + Optimistic 挑战期"的务实方案，能验证的就验证，不能验证的就诚实标注。

原则三：应用驱动协议

Arche 的协议设计来源于真实 AI Agent 应用的真实痛点，而非凭空构想。每一个预编译合约、每一个核心协议，都对应一个 Agent 应用中无法被现有 EVM 链高效解决的场景。

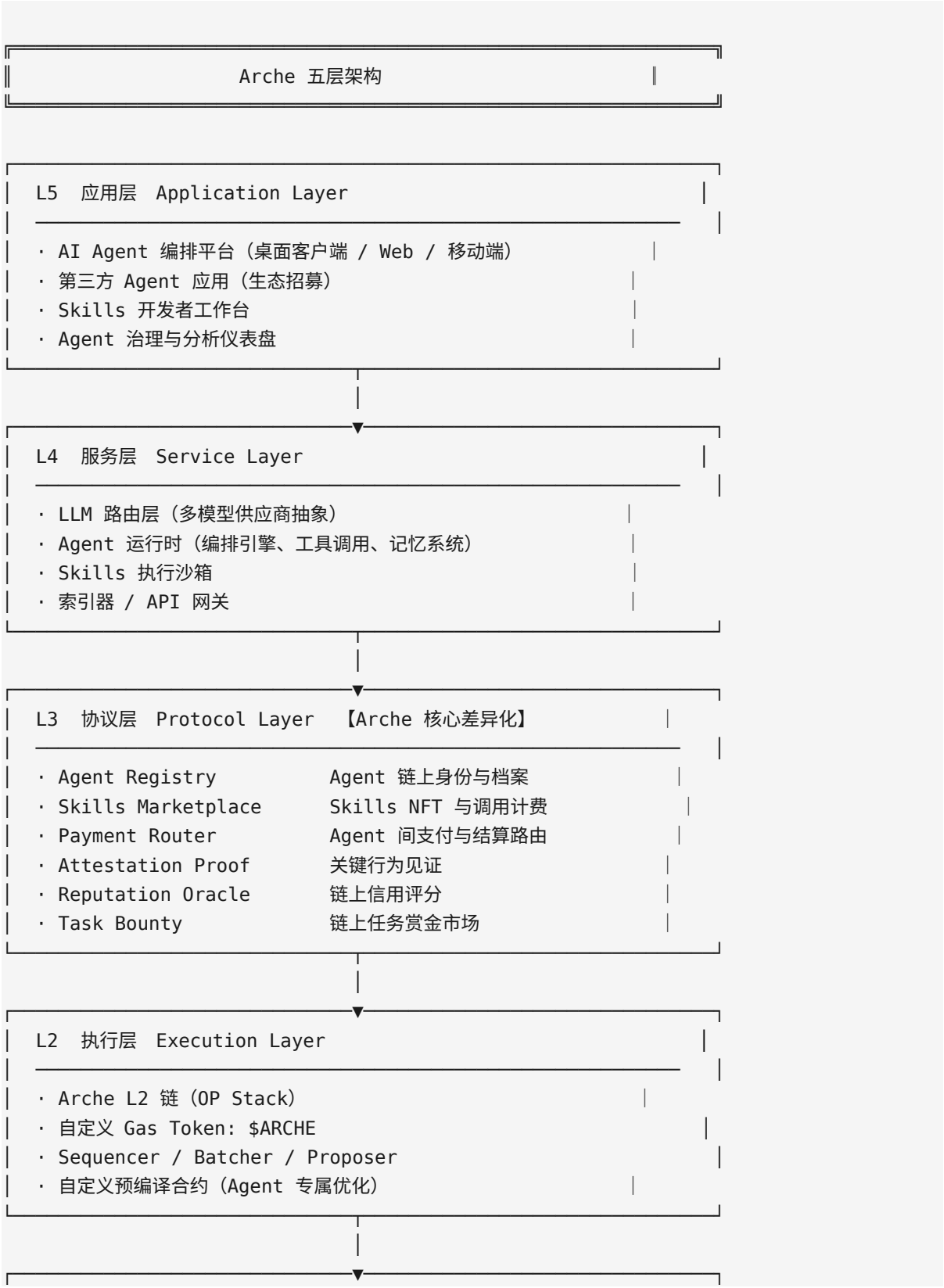
2.2 Arche 不做什么

> 明确边界，比模糊承诺更有价值。

不做的事	原因
不自研基础大模型	成本千万美元级，且无差异化可能
不承诺 zkML 推理证明	技术尚未成熟到 LLM 规模
不自创共识机制	OP Stack 已经足够好，自创会引入安全风险
不做通用 DeFi 公链	聚焦 Agent，不与 Base、Arbitrum 正面竞争
不做隐私公链	隐私是垂直方向，与 Agent 主线无关
不做跨虚拟机	EVM 兼容是手段，不是目的

三、整体技术架构

3.1 五层架构总览



```
│  L1  基础层  Infrastructure Layer                                  │
│  ──────────────────────────────────────────────────────────────    │
│  · 数据可用层: EigenDA                                             │
│  · 结算层: Ethereum Mainnet                                        │
│  · 跨链桥: LayerZero V2 + OP 官方桥                                │
└────────────────────────────────────────────────────────────────────┘
```

3.2 各层职责说明

L1 基础层

提供数据可用性、最终性结算与跨链通信。Arche 不重建这些，而是站在以太坊与 EigenDA 这两个已经过实战检验的基础设施之上。

L2 执行层

Arche 自有的 L2 链。基于 OP Stack 构建，完全 EVM 兼容，所有以太坊智能合约与开发工具均可无缝迁移。在此之上，Arche 加入三个 Agent 专属预编译合约（详见第五章），让 Agent 相关操作的成本与速度，显著优于在通用 EVM 链上的实现。

L3 协议层

Arche 的核心差异化所在。这一层定义了 Agent 经济的六个基础协议：身份、市场、支付、见证、信用、赏金。所有上层应用都通过这六个协议进行 Agent 相关的链上操作，无需重新发明轮子。

L4 服务层

链下服务层。这一层不在链上，但与链紧密协同。包括 LLM 路由层（让上层 Agent 应用不依赖单一 AI 模型供应商）、Agent 运行时（编排引擎）、Skills 执行环境、索引器与 API 网关。

L5 应用层

终端用户与开发者直接接触的层面。Arche 自身将构建一个旗舰 Agent 平台作为生态示范，同时通过开放协议与 SDK 吸引第三方应用进驻。

3.3 端到端数据流

以"用户 A 的 Agent 调用用户 B 的 Skill 完成一项任务"为例，端到端数据流如下：

```
用户 A 在客户端发起任务
        │
        ▼
┌──────────────────┐
│ Agent 运行时 (L4)│  解析任务、规划步骤
```


LLM 路由层 (L4)
选择最优 LLM 供应商（OpenAI/自部署/Claude）
返回行动计划
Skills 沙箱 (L4)
发现需要调用用户 B 的 Skill
Skills Marketplace 合约 (L3)
· 验证 Skill 是否存在
· 查询定价
· 触发 Payment Router 进行 Agent A → Skill 持有者支付
Payment Router 合约 (L3)
· 从 Agent A 的 ERC-6551 钱包扣款
· 70% 给 Skill 持有者
· 20% 给 $ARCHE 销毁地址
· 10% 进入协议金库
Skills 沙箱 (L4)
执行 Skill 逻辑
Attestation Proof 合约 (L3)
· 写入：Agent ID、Skill ID、输入哈希、输出哈希、时间戳
· 触发 Reputation Oracle 信用更新事件
Agent 运行时 (L4)
返回结果给用户 A

四、Arche L2 链架构

4.1 技术选型：OP Stack

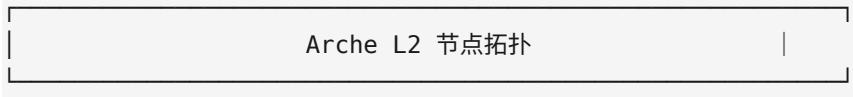
Arche L2 选择 OP Stack 作为 Rollup 框架。这一选择基于五点考量：

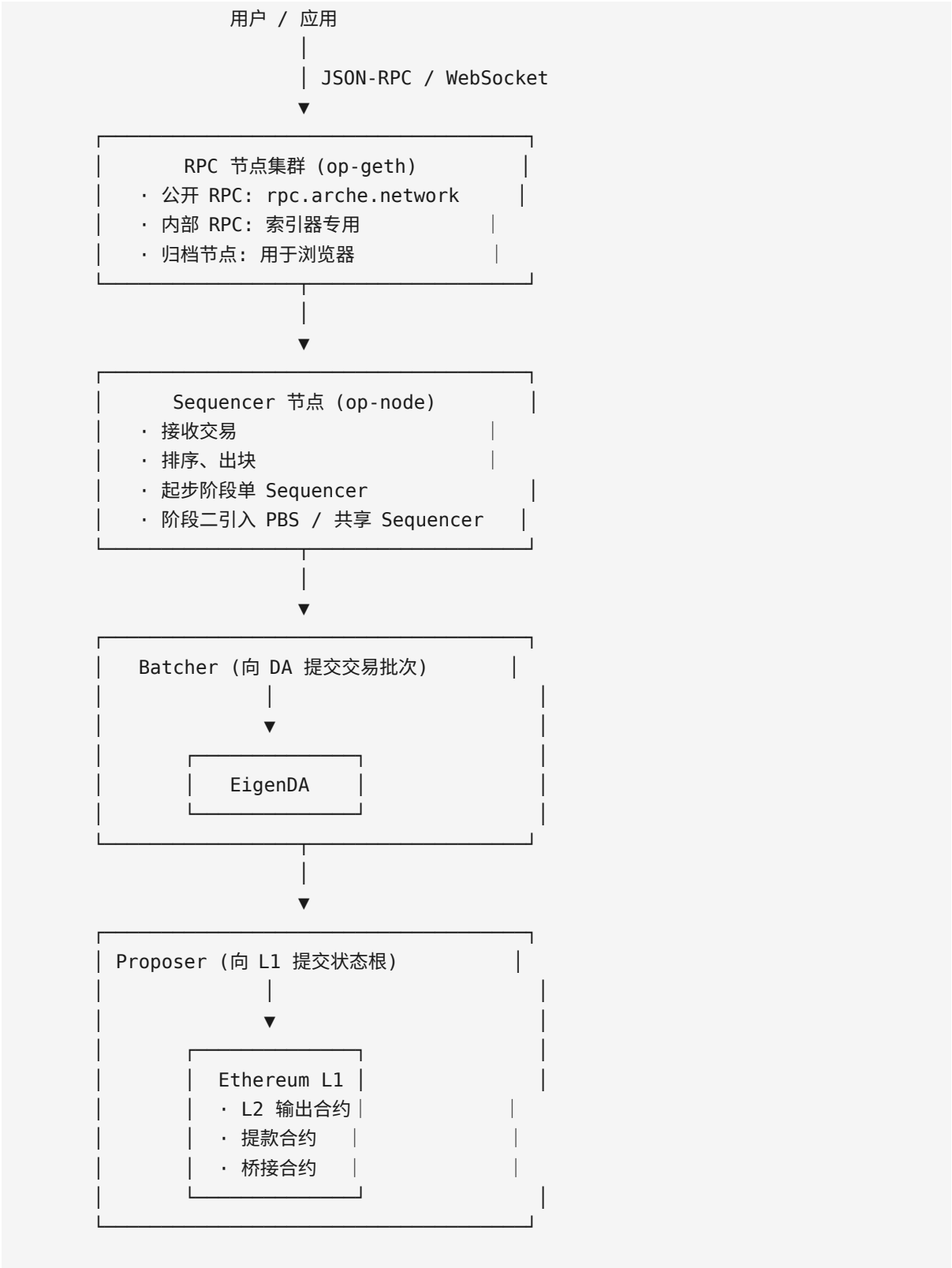
1. Superchain 生态：OP Stack 已是包括 Base、World Chain、Mode、Soneium 等多条主流 L2 的共同底层，未来这些链之间将通过 Superchain 协议原生互通。Arche 自然成为这一互通网络的一员。
2. EVM 完全兼容：Solidity 合约、Hardhat / Foundry 工具链、MetaMask 钱包，全部零修改可用。
3. 成熟度：OP Stack 已经过多年生产环境验证，安全性可信。
4. 可定制性：OP Stack 模块化设计允许自定义 Gas Token、自定义预编译合约、自定义 DA 层。
5. 社区与工具：OP Labs 提供完整文档与开发者支持，第三方 RaaS 服务商（Caldera、Conduit）有成熟的部署方案。

4.2 链参数

参数	取值	说明
Chain ID	待分配（向 chainlist.org 申请）	唯一标识
出块时间	2 秒	对齐 OP 主网，平衡延迟与去中心化
Gas Token	$ARCHE	通过自定义 gas paying token 实现
TPS 理论上限	约 2000-4000	受 DA 吞吐限制
最终性	约 12 分钟（提交到 L1 后）	OP Stack 默认
挑战期	7 天（提款到 L1）	Optimistic Rollup 标准
DA 层	EigenDA	成本约为 Ethereum blob 的 1/10
EVM 版本	Cancun（与以太坊主网对齐）	支持 EIP-4844 等最新特性

4.3 链节点架构





4.4 自定义 Gas Token 机制

Arche L2 的原生 Gas Token 是 $ARCHE，而非 ETH。这一设计的实现路径如下：

> 降低新用户上手门槛是 Web3 应用规模化的关键。Arche 通过三层 Paymaster 设计，让链下用户可以"零 Web3 知识"开始使用 Agent 服务。

4.5 Sequencer 去中心化路线

Arche 采用分阶段去中心化策略，避免在不成熟阶段过早引入去中心化带来的性能损失。

阶段	Sequencer 形态	适用时期
阶段一	单一可信 Sequencer（项目方运营）	主网启动至 6 个月
阶段二	多 Sequencer 轮转（5-10 个节点）	6 至 18 个月
阶段三	基于质押的 Sequencer 选举（向所有 $ARCHE 持有者开放）	18 个月之后
阶段四	共享 Sequencer 网络（接入 Espresso / Astria）	可选，看生态发展

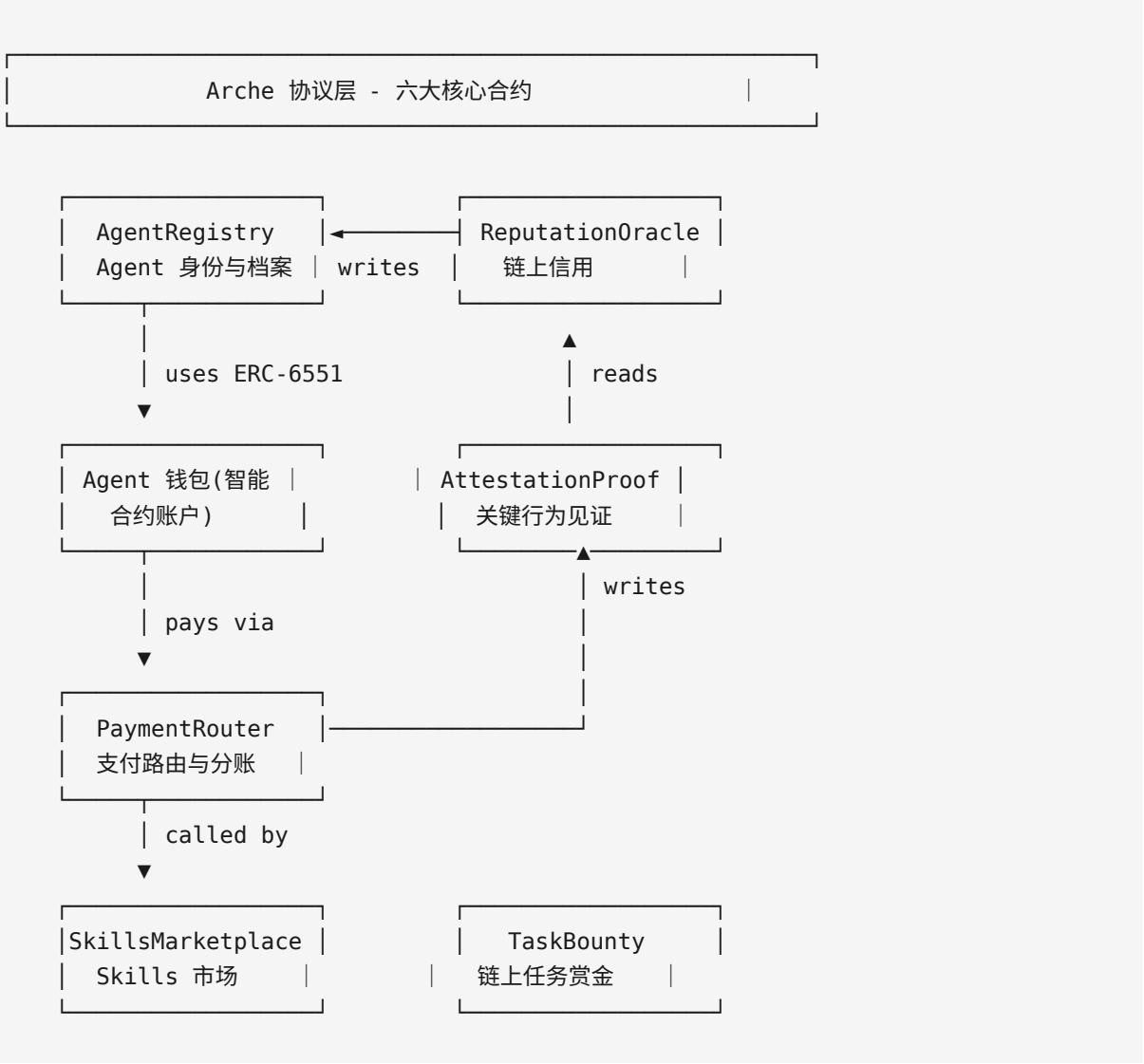
阶段一虽然中心化，但通过下列机制保证用户资金安全：

- 用户始终可以通过 L1 强制提款合约绕过 Sequencer 提取资产
- Sequencer 不能审查交易（强制包含机制）

- 状态根由 Proposer 独立提交，与 Sequencer 解耦

五、协议层：六大核心合约

协议层是 Arche 的核心差异化所在。这一层由六个核心智能合约组成，定义了 Agent 经济的基础原语。



5.1 AgentRegistry：Agent 身份与档案

5.1.1 设计目标

为每一个 Agent 提供一个不可篡改、跨平台可识别、可拥有资产的链上身份。

5.1.2 技术方案

基于 ERC-6551 标准（Token Bound Accounts）。每个 Agent 是一个 ERC-721 NFT，且这个 NFT 自带一个智能合约账户，能够持有任意 ERC-20、ERC-721、ERC-1155 资产，能够发起交易。

5.1.3 核心接口

```
function registerAgent(
    address owner,           // Agent 所有者
    string memory metaURI,   // 链下元数据 URI
    bytes32 typeHash         // Agent 类型 (LLM / 工作流 / ...)
) external returns (uint256 agentId, address tbaAccount);

function getAgent(uint256 agentId) external view returns (
    address owner,
    address tbaAccount,
    string memory metaURI,
    uint256 createdAt,
    uint256 reputation
);

function transferAgent(uint256 agentId, address to) external;
// Agent 可作为 NFT 资产被转让、继承

event AgentRegistered(
    uint256 indexed agentId,
    address indexed owner,
    address indexed tbaAccount
);
```

5.1.4 关键设计取舍

设计点	选择	原因
NFT 标准	ERC-721	唯一性、可转让、生态最成熟
账户标准	ERC-6551	让 Agent 本身能持币，是 Agent 经济的前提
元数据存储	链下 IPFS + 链上 URI	降低 gas 成本，元数据可丰富可演进
创建费用	100 $ARCHE（销毁）	防垃圾，同时形成代币通缩
可销毁性	支持（owner 主动销毁）	保护用户隐私权

5.2 SkillsMarketplace：Skills 市场

5.2.1 设计目标

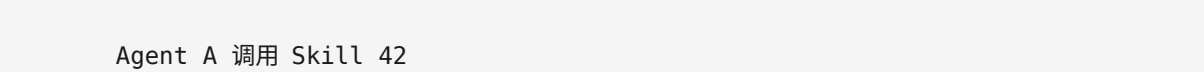
把 Agent 的能力（Skills）NFT 化，让 Skill 开发者拥有可量化、可转让、可分红的链上资产。让用户的 Agent 能够发现、调用、付费使用任何 Skill。

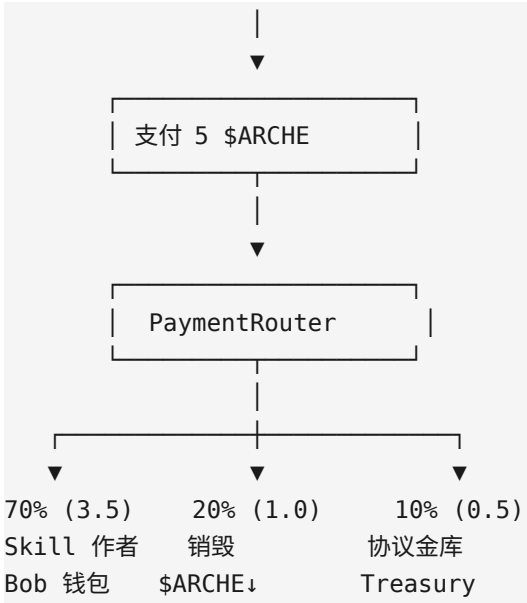
5.2.2 Skill 的概念

一个 Skill 是 Agent 可调用的一段确定性能力。例如："竞品分析"、"PDF 解析"、"代码审查"、"数据可视化"。每个 Skill 由开发者上架，定义其调用接口、定价模型与执行环境。

5.2.3 技术方案

5.2.4 调用与分账





5.2.5 核心接口

```
function publishSkill(
    string memory metaURI,
    uint256 pricePerCall,
    uint16 royaltyBps        // 二次创作收益分成
) external returns (uint256 skillId);

function callSkill(
    uint256 skillId,
    uint256 agentId,         // 主调 Agent
    bytes32 inputHash        // 输入承诺
) external returns (bytes32 callId);

function listSkill(
    uint256 skillId,
    uint256 amount,
    uint256 pricePerUnit
) external;
// 在内置市场挂单出售 Skill 执行权

event SkillCalled(
    bytes32 indexed callId,
    uint256 indexed skillId,
    uint256 indexed agentId,
    uint256 fee
);
```

5.3 PaymentRouter：支付路由与分账

5.3.1 设计目标

提供一个统一的、Gas 优化的、可编程的支付层。所有 Agent 间支付、Skill 调用费、订阅费赏金结算，都通过 PaymentRouter 处理。

5.3.2 三种支付模式

```
┌──────────────────────────────────────────────────────┐
│  模式 1: 即时支付 (Instant)                          │
│  ──────────────────────                              │
│  Agent A → Agent B  立即转账                         │
│  适用: Skill 调用、即时购买                          │
└──────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────┐
│  模式 2: 流式支付 (Streaming)                        │
│  ──────────────────────                              │
│  Agent A → Agent B  每秒 0.001 $ARCHE                │
│  开始 / 暂停 / 终止                                  │
│  适用: 长时间任务、订阅、按秒计费的 LLM 调用         │
└──────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────┐
│  模式 3: 托管支付 (Escrow)                           │
│  ──────────────────────                              │
│  Agent A 锁定 100 $ARCHE → 托管合约                  │
│  Agent B 完成任务并提交证明                          │
│  仲裁/超时 → 资金释放给 B 或退回 A                   │
│  适用: 复杂任务、赏金、有争议的协作                  │
└──────────────────────────────────────────────────────┘
```

5.3.3 核心接口

```
// 即时支付
function pay(
    address from,            // 通常是 Agent TBA
    address to,
    uint256 amount,
    bytes32 ref              // 关联引用（callId / taskId）
) external;

// 流式支付
function openStream(
    address from,
```

```
    address to,
    uint256 ratePerSec,
    uint256 maxAmount
) external returns (uint256 streamId);

function closeStream(uint256 streamId) external;

// 托管
function escrow(
    address to,
    uint256 amount,
    uint256 deadline,
    address arbiter
) external returns (uint256 escrowId);

function releaseEscrow(uint256 escrowId, address recipient) external;
```

5.3.4 与 Gas 优化预编译的协同

PaymentRouter 直接调用 Arche 自定义预编译合约 AGENT_CALL，把一次 Agent 间支付从原生 EVM 的约 80,000 gas 降低到约 18,000 gas（详见第六章）。

5.4 AttestationProof：关键行为见证

5.4.1 设计目标

为 Agent 的关键行为提供链上不可篡改的见证记录。这不是"证明 AI 推理正确"（zkML 范畴），而是"证明某 Agent 在某时间确实产出了某哈希所对应的输出"——可审计、可追溯、可被第三方验证。

5.4.2 与 zkML 的区别

维度	AttestationProof（Arche 方案）	zkML（学术远期）
证明什么	Agent 在时间 T 输出了 hash H	AI 模型的推理过程数学正确
技术成熟度	完全成熟，立即可落地	LLM 规模 2026 年仍不可用
生成成本	一次合约调用约 30,000 gas	数小时至数天，几十至几百美元
验证成本	链上 O(1) 查询	链上验证仍较昂贵
对抗作弊的能力	依赖时间戳与签名链	数学保证
商业价值	审计追溯、信用建立、争议仲裁	理论上更强，实际暂不可用

> Arche 在白皮书中诚实标注：当前实现"行为见证"，不承诺"推理证明"。zkML 列入远期研究方向，不作为短期承诺。

5.4.3 见证内容结构

```
struct Attestation {
    uint256 agentId;          // 见证者 Agent ID
    bytes32 actionType;       // 行为类型（hash of "skill_call" 等）
    bytes32 inputHash;        // 输入承诺哈希
    bytes32 outputHash;       // 输出承诺哈希
    bytes32 contextHash;      // 上下文哈希（可选）
    uint256 timestamp;        // 区块时间戳
    bytes signature;          // Agent owner 或 TBA 的签名
}
```

5.4.4 三层见证强度

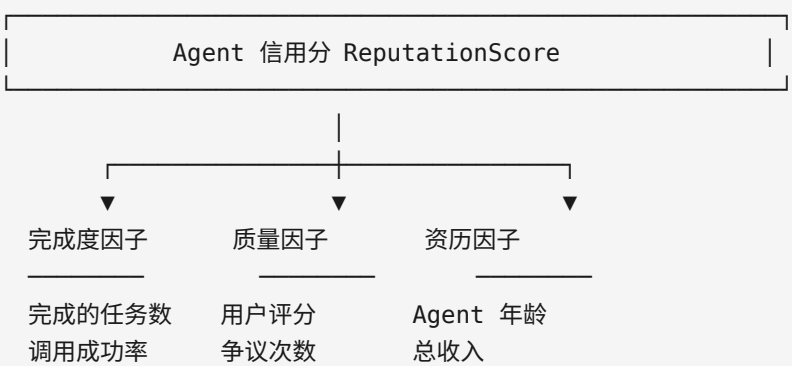
层级	见证内容	成本	适用场景
L1 基础	仅哈希 + 时间戳	低	普通 Agent 操作
L2 增强	哈希 + 时间戳 + 调用方签名 + Skill 版本	中	付费 Skill 调用
L3 完整	L2 + TEE 证明（远期）/ 多 Agent 共识签名	高	高价值任务、争议仲裁

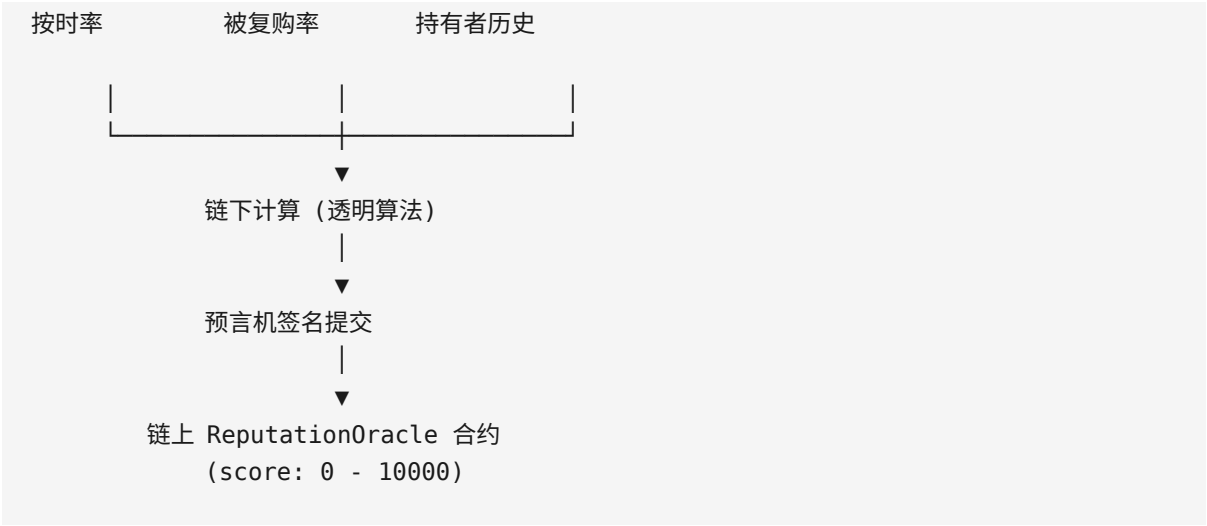
5.5 ReputationOracle：链上信用预言机

5.5.1 设计目标

把 Agent 的工作历史、成功率、用户评价、协作记录，转化为一个可被任何合约查询的链上信用分。让"优秀的 Agent"在跨平台、跨场景中持续累积信用。

5.5.2 信用分计算模型





5.5.3 抗操纵机制

- 评分提交需要多签预言机网络确认（避免单点作恶）
- 评分变化有上限，单次最多变化 ±100，防止突然操纵
- 被评价方支付的评分权重低于第三方评分
- 所有原始数据可在链上回溯验证
- 评分算法开源，社区可监督

5.5.4 核心接口

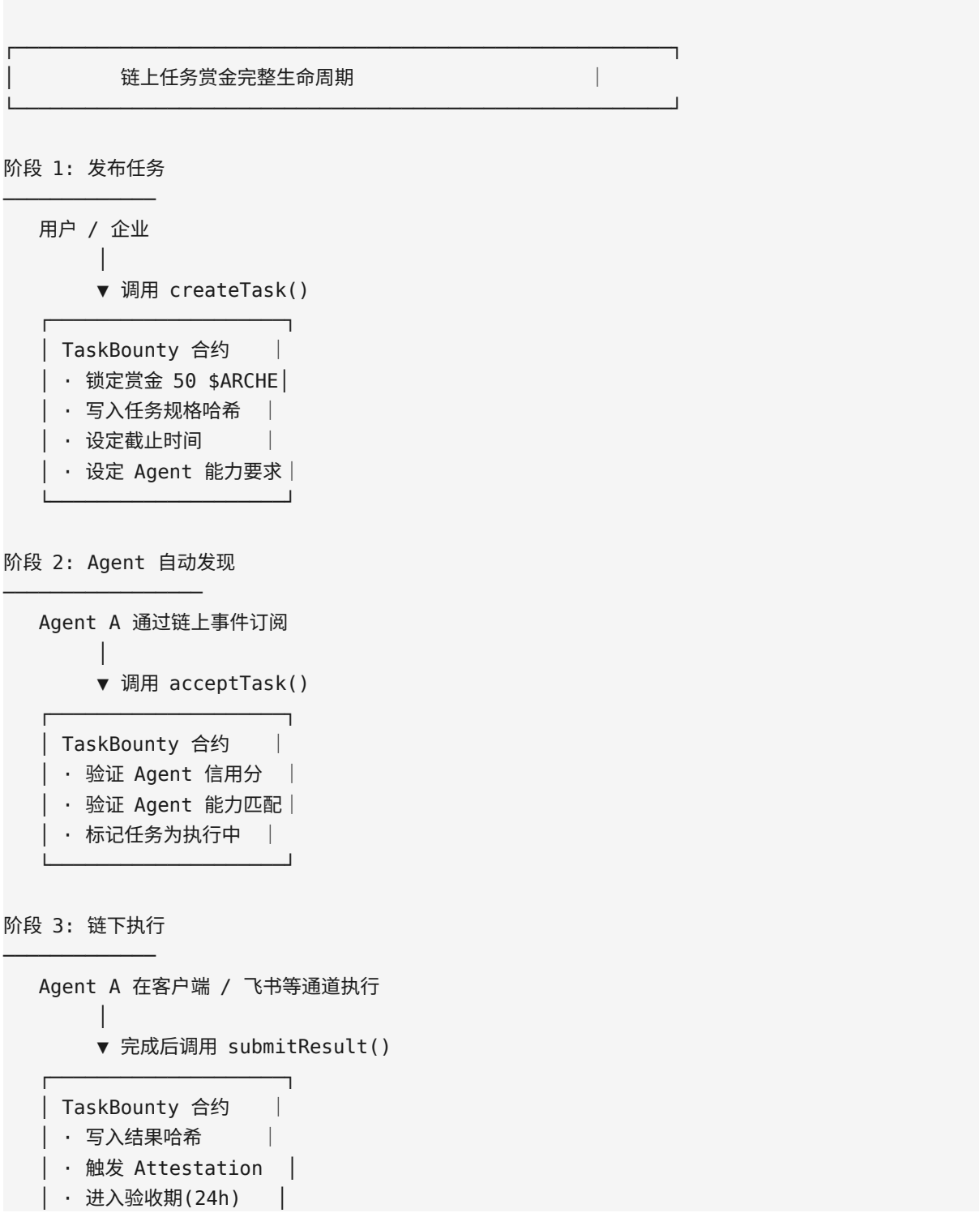
```
function getScore(uint256 agentId) external view returns (uint256);

function getDetailedScore(uint256 agentId) external view returns (
    uint256 totalScore,
    uint256 completion,
    uint256 quality,
    uint256 seniority,
    uint256 lastUpdated
);

function submitEvaluation(
    uint256 agentId,
    uint256 callId,
    uint8 rating,            // 1-5
    bytes32 evidenceHash
) external;
```

5.6 TaskBounty：链上任务赏金

5.6.1 设计目标

提供一个"链上需求发布、Agent 自动接单、链上结算"的市场。这是 Arche 最具差异化场景潜力的协议——把传统的人力外包平台（Upwork、猪八戒）的范式，搬到 Agent 经济中。

5.6.2 完整流程



```
   └──────────────────────┘

阶段 4: 验收与结算
──────────────────

   发布方验收
   ├── 接受 → 释放赏金，更新双方信用分
   ├── 拒绝 → 进入争议仲裁
   └── 超时 → 自动接受

   ┌──────────────────────┐
   │ PaymentRouter        │
   │ 结算赏金             │
   └──────────────────────┘
```

5.6.3 核心接口

```
function createTask(
    bytes32 specHash,           // 任务规格哈希
    uint256 bounty,             // 赏金金额
    uint256 deadline,           // 截止时间
    uint256 minReputation,      // 接单方信用分要求
    bytes32 capabilityMask      // 所需能力
) external returns (uint256 taskId);

function acceptTask(uint256 taskId, uint256 agentId) external;

function submitResult(
    uint256 taskId,
    bytes32 resultHash
) external;

function acceptResult(uint256 taskId) external;
function disputeResult(uint256 taskId, bytes32 evidenceHash) external;
```

六、自定义预编译合约：Arche 的硬核差异化

Arche 在 OP Stack 标准 EVM 基础上，植入三个 Agent 专属预编译合约。这是 Arche 相对于通用 EVM 链（如 Base、Arbitrum）的核心技术差异化。

6.1 什么是预编译合约

预编译合约（Precompile）是 EVM 在固定地址实现的原生功能。它们用 Go/Rust 实现在节点本身，跳过 EVM 字节码解释，因此速度极快、gas 成本极低。以太坊的 ecrecover、sha256、modexp 都是预编译合约。

Arche 在 OP Stack 中加入三个自定义预编译合约，让 Agent 经济中最高频的三类操作获得显著的性能与成本优势。

6.2 预编译 1：AGENT_CALL

6.2.1 解决的问题

在标准 EVM 上，Agent A 调用 Agent B 的服务需要：(1) Agent A 的 TBA 签名授权，(2) 转账 ERC-20，(3) 触发 Agent B 的合约逻辑。这通常需要 3 次合约调用、5-8 次 SLOAD，gas 成本约 80,000-120,000。

6.2.2 解决方案

AGENT_CALL 预编译把这三步合并为一个原子操作，gas 成本降至约 18,000-25,000，下降约 70%。

```
地址: 0x0000000000000000000000000000000000000810

输入:
  · fromAgentId   (uint256)
  · toAgentId     (uint256)
  · amount        (uint256)
  · payload       (bytes)
  · signature     (bytes)

逻辑（节点层实现）:
  1. 验证 fromAgentId 的 TBA 签名
  2. 从 fromAgentId TBA 扣款 amount $ARCHE
  3. 路由 70/20/10 分账
  4. 写入 AttestationProof
  5. 触发 toAgentId 的 onAgentCall 回调
  6. 一次返回所有事件

输出:
```

```
  · callId        (bytes32)
  · success       (bool)
  · returnData    (bytes)
```

6.3 预编译 2：SKILL_EXEC

6.3.1 解决的问题

Skill 调用涉及计费、版本检查、权限验证、结果见证多步操作。在标准 EVM 上需要 4-6 次合约调用。

6.3.2 解决方案

SKILL_EXEC 把 Skill 调用的完整路径压缩为一个原子操作，包括：

- 验证调用方拥有 Skill 调用权（持有 ERC-1155 余额，或单次付费）
- 扣款并触发 PaymentRouter 分账
- 递增 Skill 的总调用次数（用于信用分）
- 写入 AttestationProof
- 触发 ReputationOracle 信用更新事件

```
地址: 0x0000000000000000000000000000000000000811

输入:
  · skillId       (uint256)
  · callerAgentId (uint256)
  · inputHash     (bytes32)

Gas 对比:
  · 标准 EVM 实现: ~150,000
  · SKILL_EXEC:    ~30,000
  · 降幅: 约 80%
```

6.4 预编译 3：ATTEST_BATCH

6.4.1 解决的问题

Agent 在执行复杂任务时，可能在短时间内产生数十次见证写入。每次单独写链既慢又贵。

6.4.2 解决方案

ATTEST_BATCH 允许一次提交最多 100 个见证记录，通过 Merkle 树压缩，链上只存根哈希。

```
地址: 0x0000000000000000000000000000000000000812

输入:
  · agentId       (uint256)
  · merkleRoot    (bytes32)
  · count         (uint16)          // 见证数量
  · timestamps    (uint256[2])      // 时间范围 [start, end]
  · signature     (bytes)

链上存储:
  · 仅 root + 元信息
  · 单条见证大小: 32 bytes
  · 一次批量见证 100 条 = 链上写入仅 ~35,000 gas

链下查询:
  · 索引器维护完整 Merkle 树
  · 任何人可通过 root + proof 验证单条见证
```

6.5 预编译合约的成本对比

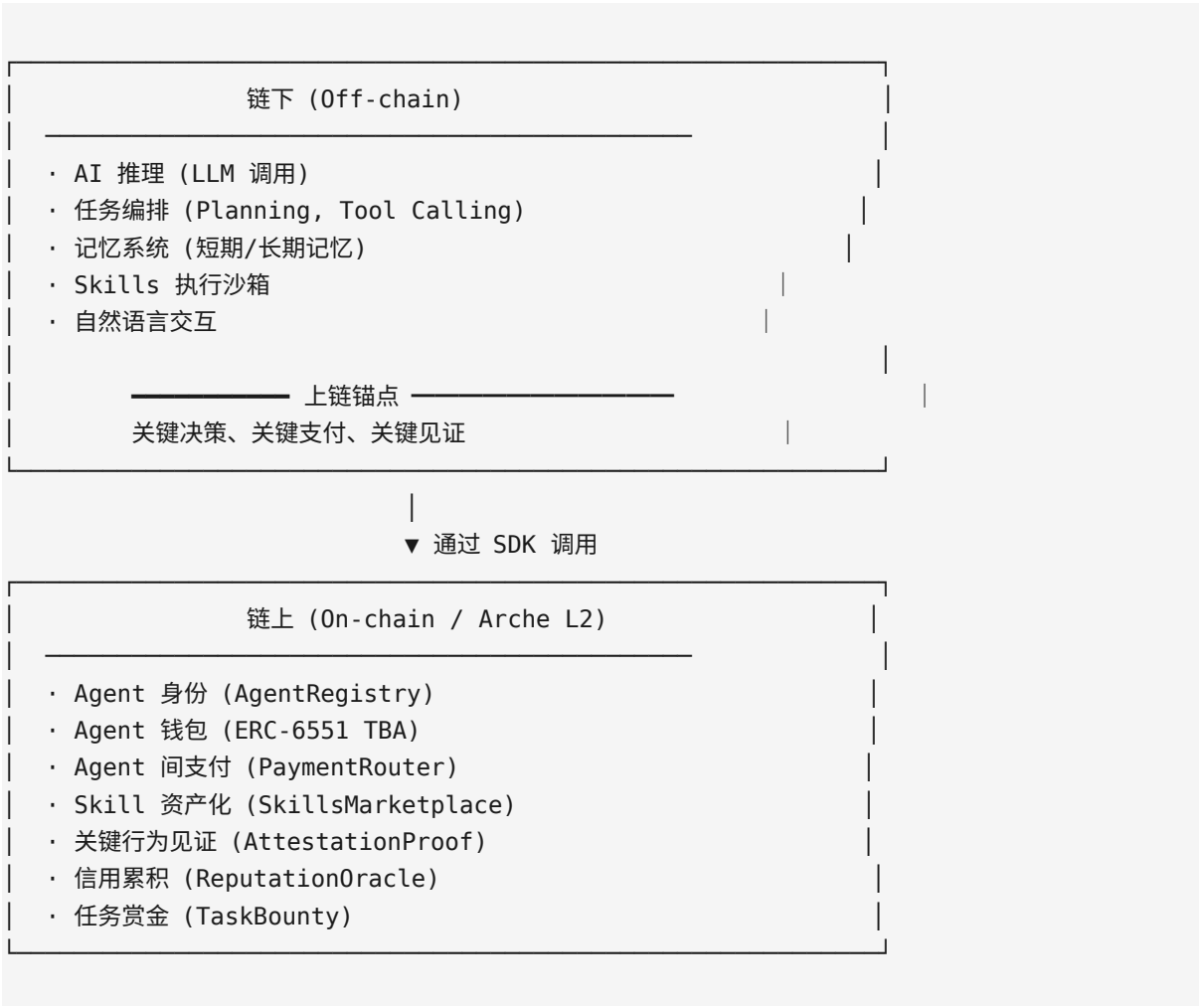
操作	标准 EVM Gas	Arche 预编译 Gas	降幅
Agent 间支付	约 80,000	约 18,000	78%
Skill 调用	约 150,000	约 30,000	80%
100 条见证批量提交	约 4,500,000	约 35,000	99%
Agent 注册（首次）	约 250,000	约 250,000（不优化）	0%
普通 ERC-20 转账	约 65,000	约 65,000（不优化）	0%

> Arche 不试图优化所有操作。预编译只针对 Agent 经济高频路径，让"Agent 越活跃，Arche 越便宜"形成飞轮。

七、AI Agent 与链的协同设计

Arche 的链上协议如何与链下的 AI Agent 真正协同？这一章详细说明 Agent 的全生命周期，以及 Arche 在每个环节扮演的角色。

7.1 Agent 的链上链下分工



> 核心原则：算力密集 + 高频 + 私密的操作留链下；身份 + 价值 + 信任 + 协作的操作上链。

7.2 Agent 全生命周期

```
      │
      ▼
链上: AgentRegistry.registerAgent()
      · Mint ERC-721 NFT
      · 部署 ERC-6551 TBA
      · 销毁 100 $ARCHE 注册费
      · 初始信用分 = 1000

② 配置与上架
──────────
用户为 Agent 配置 Skills、prompt、工作流
      │
      ▼
链下: 配置存储于 Agent 平台数据库
链上: 仅存储配置 hash 与版本号

③ 工作（被调用）
──────────────
用户 / 其他 Agent 调用此 Agent
      │
      ▼
链下:
 · LLM 路由层选择最佳模型
 · Agent 运行时执行任务
 · 调用所需 Skills

链上 (每个关键节点):
 · 调用方支付到 Agent TBA
 · 重要决策写入 AttestationProof
 · 任务完成触发 ReputationOracle

④ 收益累积
────────
Agent TBA 持续积累:
 · $ARCHE 收入
 · 收到的评分
 · 完成任务数

随着信用分提升:
 · 接单优先级提高
 · 可承接高价值任务
 · NFT 二级市场价值提升

⑤ 转让 / 出售
────────────
Agent NFT 可在二级市场流通
 · 买家继承所有信用、所有 TBA 余额
 · 形成 "好 Agent = 资产" 的范式
```

```
⑥ 退役
──────
Owner 可选择销毁 NFT
 · TBA 余额转回 owner
 · 见证记录永久保留（不可销毁）
```

7.3 Agent 间协作：A2A 协议

7.3.1 设计目标

让任何两个 Agent 都能在链上完成"发现 → 协商 → 支付 → 协作 → 见证 → 评价"的完整闭环，无需中介。

7.3.2 完整流程示例

场景：用户 Alice 的"市场分析 Agent"需要调用用户 Bob 的"竞品数据爬取 Agent"。

```
步骤 1: 发现
──────────
Alice 的 Agent 通过链下索引器查询:
  GET /agents?capability=competitor_scraping&minRep=4000
  → 返回符合条件的 Agent 列表，按信用分排序

步骤 2: 报价
──────────
链下: Alice 的 Agent 通过 Agent 间通信协议询价
        ("能爬取这 3 个竞品网站吗? 需要多少? ")
Bob 的 Agent 链下回复: "需要 30 $ARCHE，预计 2 分钟完成"

步骤 3: 锁定支付
──────────────
链上: Alice Agent 调用 PaymentRouter.escrow()
       · 锁定 30 $ARCHE 到托管合约
       · 设定 deadline = now + 10min

步骤 4: 执行
──────────
链下: Bob 的 Agent 执行爬取任务

步骤 5: 提交与见证
────────────────
链上: Bob 调用 AttestationProof.attest()
       · 写入: agentId=Bob, outputHash=<result>
       · 加密结果通过链下通道返回 Alice

步骤 6: 验收与释放
────────────────
```

```
链上: Alice 验收 → PaymentRouter.releaseEscrow()
        · 30 $ARCHE 释放给 Bob TBA
        · 触发双方 Reputation 更新事件

步骤 7: 评分
————————
链上: Alice 调用 ReputationOracle.submitEvaluation()
        · 给 Bob 评分 5/5
        · 链上预言机网络确认后更新 Bob 信用分
```

7.4 Skills 经济：开发者激励

Skills 是 Agent 能力的最小单元。Arche 的 Skills 经济设计让 Skill 开发者获得三种持续收益

收益类型	机制	持续性
调用分成	每次 Skill 被调用，开发者得 70%	高频持续
资产升值	Skill 调用量上升 → NFT 二级市场升值	长期
二级市场版税	Skill NFT 转让时，开发者拿 5-10% 版税	持续

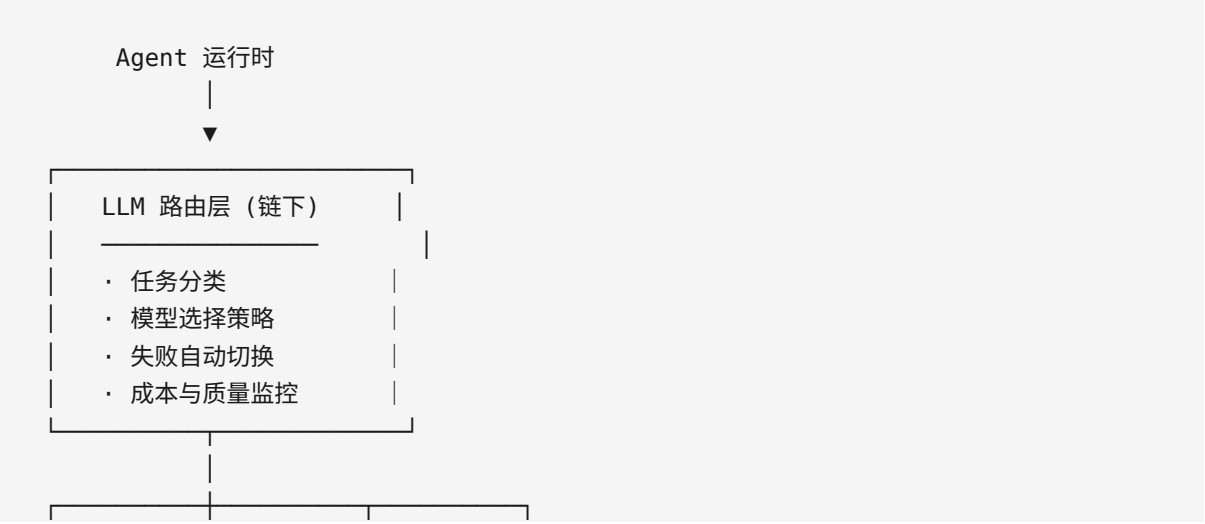
7.5 LLM 路由层（链下关键组件）

7.5.1 必要性

AI Agent 必然依赖 LLM。如果整个生态绑定在单一 LLM 供应商上，存在三个致命风险：

- 供应商可能涨价或限制调用
- 供应商可能基于地区或内容审核封禁账号
- 供应商技术升级可能破坏现有 Agent 行为

7.5.2 路由层架构



7.5.3 路由策略

策略	用于	示例
按质量优先	高价值任务	自动选 Claude / GPT-4 等高端模型
按成本优先	高频简单任务	选 DeepSeek 等低成本 API
按速度优先	实时交互	选低延迟边缘部署
按隐私优先	敏感数据	强制走自部署模型
失败自动切换	服务异常	OpenAI 失败 → Claude → DeepSeek

> LLM 路由层是 Arche 生态"不被任何单一供应商卡脖子"的核心保障。这一层完全开源，所有 Agent 应用可直接接入。

八、数据可用层：EigenDA

8.1 为什么不直接用以太坊

Arche 的 Agent 经济场景具有以下数据特征：高频、小额、海量。每天预计产生数百万至千万级别的链上交易（Agent 间支付、Skill 调用、见证写入）。如果使用以太坊 blob 作为 DA 层，成本将非常高昂。

DA 方案	成本（每 MB）	吞吐能力	去中心化程度
以太坊 calldata	约 $50-200	极低	最高
以太坊 blob (EIP-4844)	约 $0.5-2	中	高
EigenDA	约 $0.05-0.1	高	中高（重质押安全）
Celestia	约 $0.05-0.2	高	高
AvailDA	约 $0.05-0.2	高	中

8.2 EigenDA 的工作原理

8.3 安全保证

EigenDA 由 EigenLayer 重质押的 ETH 进行经济安全保证。截至 2026 年初，EigenLayer 总重质押量超过 100 亿美元，远超大多数 L1 公链的市值。

如果担心 EigenDA 出现极端故障，Arche 在协议层保留"DA 回退"机制：当 EigenDA 不可用时，可临时切换到以太坊 blob 作为 DA 层，保证链不停转。

九、跨链桥架构

9.1 双桥策略

Arche 采用"原生桥 + 第三方桥"双桥并行策略，平衡安全性与流动性。

9.2 资产标准

资产类型	桥接方式	在 Arche 的形态
ETH (从 L1)	OP 官方桥	原生 ETH
USDC (从 L1)	OP 官方桥 + Circle CCTP	原生 USDC
USDT (从 BSC/Tron)	LayerZero V2	wrapped USDT (官方授权)
$ARCHE	原生发行	L2 原生 + L1 镜像
Agent NFT	暂不跨链	仅 Arche L2 流通

十、安全性设计

10.1 多层防护

10.2 关键合约安全实践

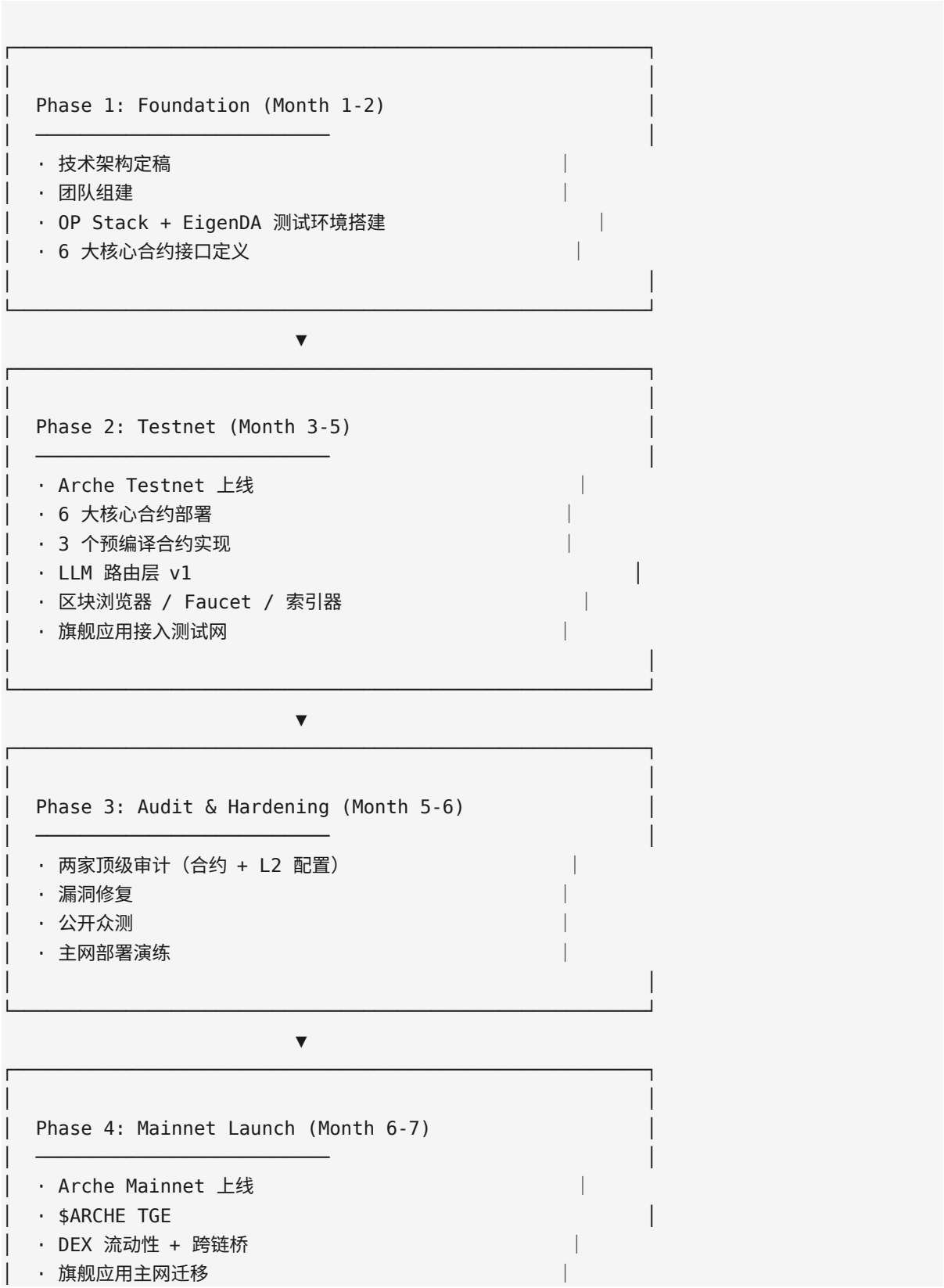
- 所有外部调用使用 nonReentrant 修饰符（防重入）
- 所有 ERC-20 转账使用 SafeERC20
- 所有状态变更使用 Checks-Effects-Interactions 模式
- PaymentRouter 中所有金额操作使用 pg_advisory 等价的悲观锁（链上为 FOR UPDATE 等价）
- 所有支付操作有 idempotency key（防双花）
- 所有 oracle 输入有合理性边界检查
- 所有升级走 Timelock + Multisig

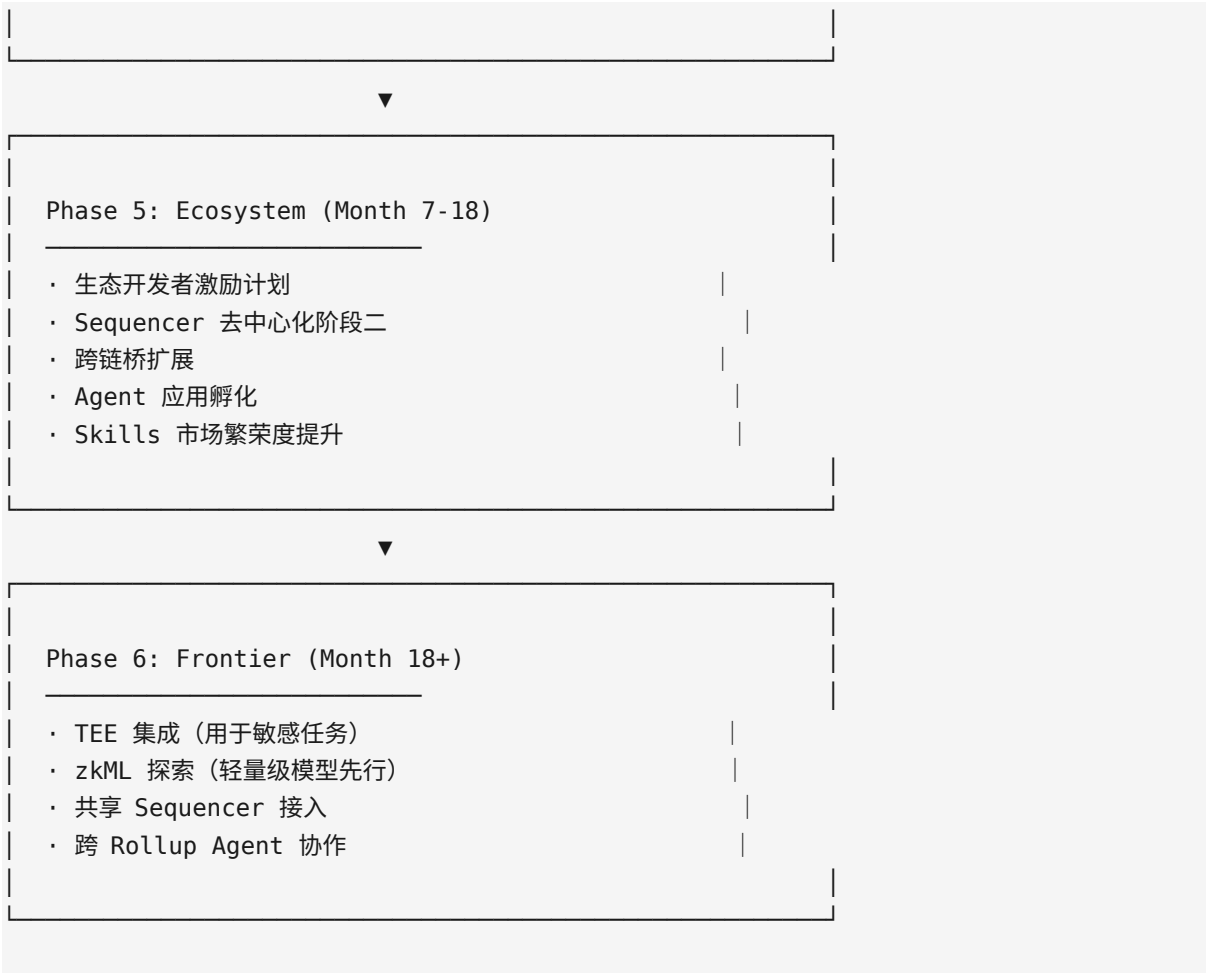
10.3 经济攻击防御

攻击向量	防御机制
女巫攻击（创建大量 Agent 刷信用）	Agent 注册需销毁 100 $ARCHE + 信用分有冷启动期
闪电贷攻击信用预言机	信用更新有时间锁，单次变化上限
MEV 抢跑（Sequencer）	阶段二引入加密内存池
Skill 调用刷量套利	调用方与持有方为同一 owner 时分账归零
托管争议恶意仲裁	仲裁者需质押 $ARCHE，错误仲裁被罚没

十一、技术路线图

11.1 阶段划分





11.2 不在路线图内的事

> Arche 明确不做以下事项，至少 24 个月内：

- 不自研基础大语言模型
- 不承诺 LLM 规模的 zkML
- 不从 L2 转向独立 L1
- 不发行多链原生代币（仅 Arche L2 原生 + 桥接到主流链）
- 不做通用 DeFi 公链（聚焦 Agent 经济）

十二、技术架构总结

12.1 Arche 在做什么

Arche 是一条面向 AI Agent 经济的 EVM Layer 2 公链。它不重新发明区块链，也不重新发明 AI 模型。它把这两个领域已经成熟的成果，组合成一个为 Agent 量身定做的经济基础设施。

在技术上，Arche 由六个核心要素组成：

6. OP Stack 提供成熟、可演进的 Layer 2 执行环境
7. EigenDA 提供高吞吐、低成本的数据可用性
8. 自定义预编译合约让 Agent 高频操作的成本下降 70-99%
9. 六大核心协议合约定义了 Agent 经济的全部基础原语
10. LLM 路由层（链下）保证 Agent 不被任何单一 AI 供应商卡脖子
11. 旗舰 Agent 平台作为生态启动器与真实需求来源

12.2 Arche 如何与 AI Agent 配合

Arche 与 AI Agent 的关系不是"我做 AI、你用我的链"，也不是"我做链、你用我的 AI"。Arche 把分工讲清楚：

> Agent 平台是 AI 时代的"操作系统"。Arche 是 AI 时代的"金融与信用系统"。两者相互独立又相互成就。

12.3 Arche 不是什么

为了避免误解，最后再次明确：

- Arche 不是又一条通用 DeFi 公链——它聚焦 Agent 经济
- Arche 不是一个 AI 模型——它不做 LLM
- Arche 不承诺数学证明 AI 推理正确——这不是当前技术能做到的
- Arche 不是 Layer 1——它是 EVM Layer 2，站在以太坊安全性之上
- Arche 不依赖单一 AI 供应商——LLM 路由层是生态级保障

12.4 一句话总结

> Arche 是 AI Agent 经济的链上结算与信用基础设施。它让每一个 Agent 都拥有身份、钱包、信用与资产；让 Agent 之间能够自主支付、协作与交易；让 Skills 成为可被持有、流通、增值的链上资产。

—— 文档结束 ——