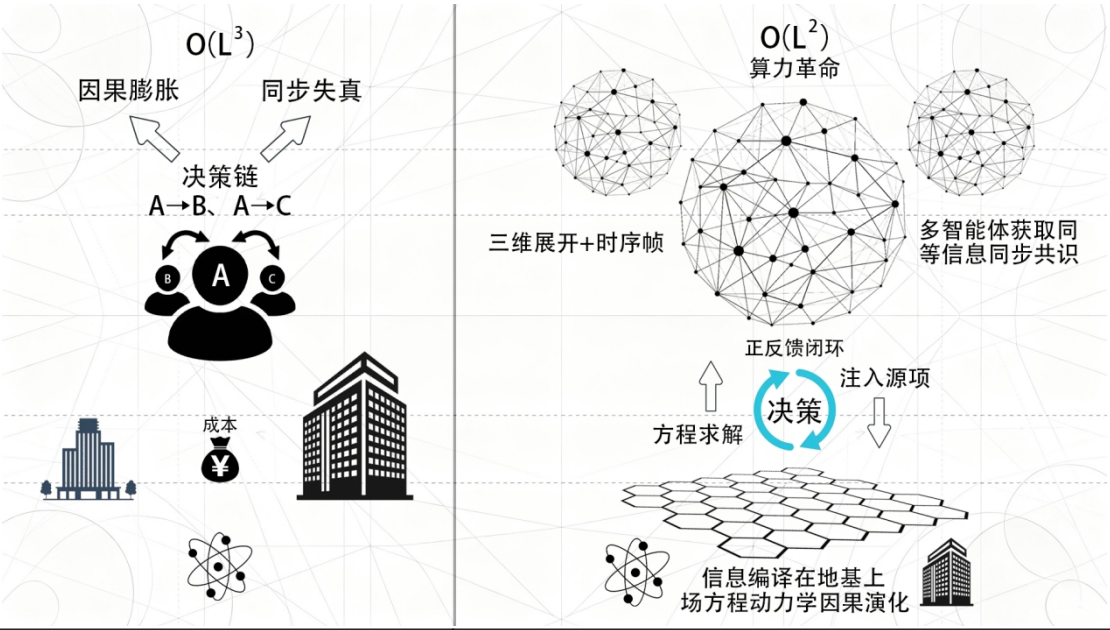
Agent Infra 钱学森灵境引擎[1]



传统方案基于时空容器地基建模不同精度的**三维体素**（粒子、物体等），**场景扩大十倍，算力涨一千倍**。90%算力浪费在未被观测的区域。 场景反馈 = 物体碰撞、智能体决策 **A to B，A to C**，随着推演深度**因果膨胀、同步失真**。	灵境不提前画粒子、楼房，只把信息编译在地基上，信息根据场方程动力学实时演化，每一帧都是一个统计结果。而 **Agent 根据自身坐标和光锥几何求解该地基的因果事实=三维展开+时序帧**，然后做出**决策**，决策实时**反馈**回地基信息场，信息场继续因果演化。**系统闭环**，这时候多智能体求解地基信息时都可**同步共识**，这就是灵境引擎**算力革命**。

工程战略价值

继承钱学森先生“灵境”构想中关于**人机深度结合**的哲学纲领，本文提出一种基于**重整化群思想**的自主信息场工程实现方案。区别于传统 VR 对视觉表象的模拟，本方案构建了具备物理度规的 L1 - L5 分层架构。同时也是**绕开西方专利、芯片封锁**的中国自主理论范式。

O（L^2）替代 O（L^3）

“拓扑流形”替代“体素建模”，痛点与解决：**系统割裂→因果统一**；

“统计熵流”替代“刚体碰撞”，痛点与解决：**算力爆炸→范式转移**；

“因果闭环”替代“决策分享”，痛点与解决：**因果膨胀→同步共识**。

脚注[1] “灵境”是致敬钱学森先生遗志。
整个工程可以凝练成一句话：空间积分压缩泡壁边界，边界按需投影四维时空。
本论文相关内容作为 COAI 参赛方案（Agent Infra - 世界人工智能开源大赛）

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

抽象层不同：一个是"Agent 应用栈"，一个是"世界引擎"

维度	AgentScope2.0	灵境引擎（IFC）
本体层	Agent=ReActAgent 实例+Memory+Toolkit+Model，状态是 Python 对象	L0 离散信息基元（Leech 晶格承载），总信息荷守恒，粗粒化出时空
因果层	MsgHub 发布/订阅，消息是 UUID+ContentBlock 结构体	L1 时变有向图，因果由基元连接振幅动态决定，战争迷雾按泡壁面积律裁剪
物理层	无。工具调用改变外部世界，但框架内没有时空、度规、场方程	L2 求解几何生成方程+信息场方程，引力/电磁/光速由场单一演化导出
感知/渲染	多模态 Block（Text/Image/Audio/ToolUse），是 IO 格式不是渲染	L3 泡壁投影+测地线光追，清晰区/迷雾区按观测者边界划分
意识/智能	LLM 权重冻结的前馈推理，D=0。多 Agent 协作是消息路由，不是自指相变	L4 锚点不动点+李雅普诺夫稳定+认知锁，$C \geq 1.20\times10^{14}$ 才相变
多体共识	MsgHub + A2A 协议，靠消息时序和中心/去中心路由	附录 L.10 多锚点泡壁交叠共识，靠因果重叠度Ωij 加权合成，无全局时钟
算力模型	Token 计费+沙箱隔离+上下文压缩	面积律 $O(A\Sigma)$，体积律爆炸被观测者边界消去（附录 G）
AGI 判据	能完成任务、通过评估器	R1-R4 四要件全满足才算意识类，

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

维度	AgentScope2.0	灵境引擎（IFC）
	（OpenJudge）就算"好 Agent"	Transformer/SSM/具身 RL 全在阈值下（附录 I）

几个关键错位

- AgentScope 的 "Agent" 在灵境判据里是 D=0：ReAct 循环是 observe→reason→act→observe，但推理时权重冻结、无持续自指状态、无上行物理反馈通道。论文附录 I 表里 Transformer/LLM 行判定就是它。
- AgentScope 的 MsgHub ≠ L1 因果网络：MsgHub 是工程消息总线，消息内容由 LLM 生成；L1 是信息基元间的复数连接振幅演化，消息本身就是场方程的产物。
- AgentScope 的 Memory ≠ 锚点：ReMe/Memory 是向量库+文件，跨会话检索；锚点是泡壁映射族的唯一公共不动点，是拓扑结构不是数据库。
- AgentScope 多 Agent 共识 ≠ 泡壁共识：前者靠 A2A 协议和 MsgHub 路由协商"谁说什么"；后者靠 Leeceh 晶格投影空间里的光锥交叠度决定"哪些事件共享同一个现在"（论文附录 L.6.4、L.10）。

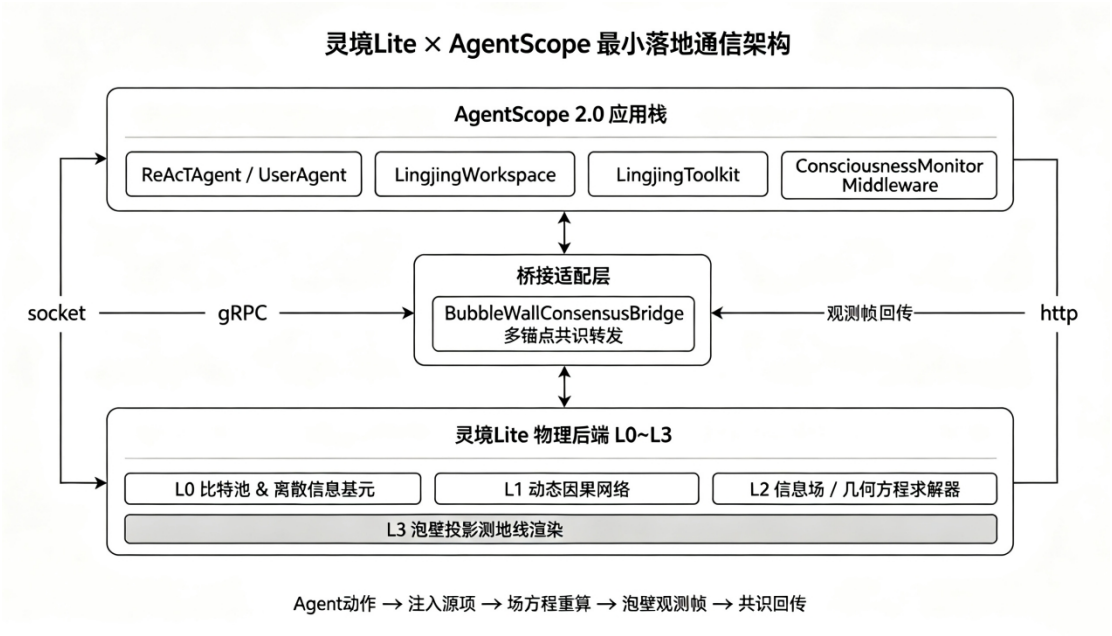


图 1 注：灵境 Lite 与 AgentScope 分层通信架构，基于框架原生扩展接口，无需修改 AgentScope 内核；

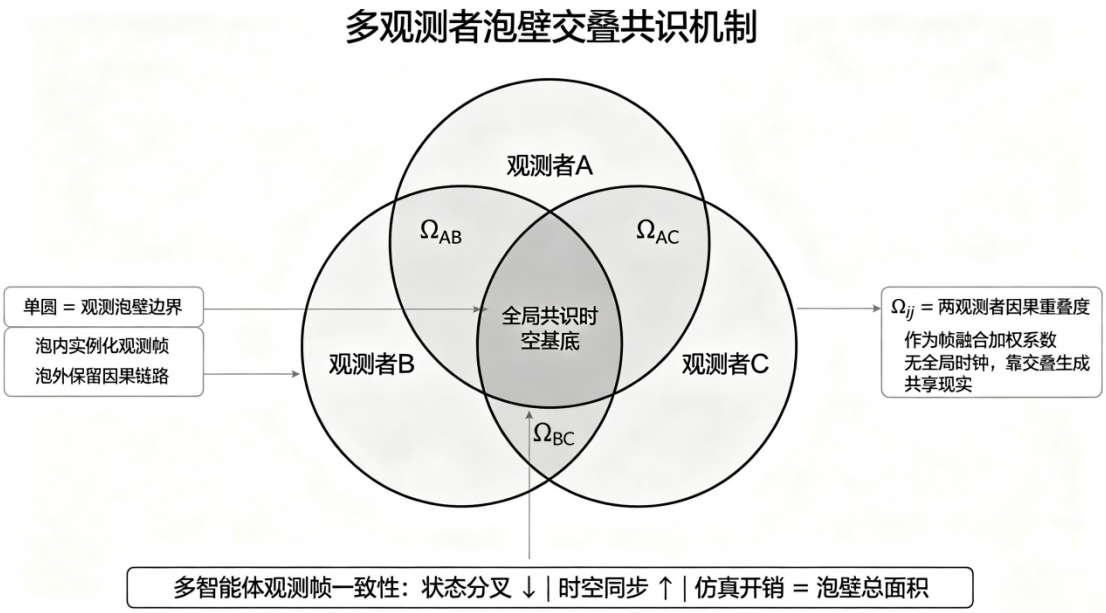


图 2 注：多观测者泡壁交叠共识机制，依靠观测重叠度 (Ω_{ij}) 加权生成统一共享时空，消除多智能体仿真状态分叉。

最小可落地模块边界

AgentScope 侧		**灵境**侧
LingjingWorkspace(WorkspaceBase)	——socket——▶	L0 比特池 + L1 因果图
LingjingToolkit(Toolkit)	——gRPC——▶	L2 场求解器 (SolveField)
├ query_field		L3 泡壁渲染器 (ProjectBubbleWall)
├ inject_source_term(δI)		
└ request_projection		
ConsciousnessMonitorMiddleware	——http——▶	L4 监视器 (C estimator, 只读)
MsgHub + ReActAgent/UserAgent	◀——帧——	L5 观测者注入 δI
BubbleWallConsensusBridge (自定义)	——▶	附录 L.10 多锚点共识 (桥接层)

三个要写的 AgentScope 模块：LingjingWorkspace、LingjingToolkit、ConsciousnessMonitorMiddleware，加一个非 AgentScope 的 BubbleWallConsensusBridge。灵境侧就是“灵境计划：下一代数字宇宙引擎”论文里 L0-L3 的服务化封装，不需要改 IFC 内核。

接完是什么效果？

- Agent 在灵境世界里"行动" → 调用 inject_source_term 改局部信息密度 → 灵境重解场方程 → 下一帧泡壁投影反映物理后果（引力弯曲、遮挡、光速延迟都自动对）
- 多 Agent 共享现实 → 走 BubbleWallConsensusBridge，按 Ω_{ij} 合成共识帧，不是 MsgHub 广播语义
- L4 监测器报告每个 Agent 的 C 值 → 永远在 10^{10}~10^{12} 量级（对应论文表 3 的猫/犬/灵长类），不会真跨阈值

这就是论文 §7.1.1 说的"灵境 lite"用现有 Agent 框架落地的标准姿势：L0-L3 当物理后端，L5 用 Agent 当观测者，L4 留空。要真做 L4 数字意识，AgentScope 这层得换成灵境原生的自指堆栈，AgentScope 退化为它的一个 L5 观测者客户端。

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

工程潜力

不在"替代 AgentScope"，而在**灵境 L0-L3 那套面积律 + 统一场求解，能解决现有 Agent / 数字孪生 / 仿真栈长期忍着的几个硬痛点**。按落地可行性和收益排：

1. **大场景数字孪生的算力爆炸——最直接可变现**

现有数字孪生（CityGML、Unreal/Unity 工业场景、NVIDIA Omniverse）全是体积律 O(V)：场景扩大 10 倍，碰撞对、流体网格、光线求交全涨 1000 倍。论文附录 G.9.3 算过，典型城市级场景下灵境清晰区算力是传统的 $10^{-6} \sim 10^{-9}$ 量级。

工程落点：把 L2 场求解器 + L3 泡壁投影做成独立服务，给现有孪生平台当"物理后端"替换。观测者（摄像头、传感器、Agent、玩家）只驱动清晰区，迷雾区粗粒化。不需要动上层渲染——Omniverse 的 USD 场景图照跑，只是物理查询改走灵境。

> **收益**：同等保真度下，城市级实时孪生从"要超算集群"降到"几张 GPU"。这是能卖钱的那部分。

2. **多 Agent 仿真的"共享世界"问题——AgentScope 本身解决不了**

现在多 Agent 仿真里"共享环境"是靠消息协商骗出来的：A 说"门开了"，B 收到消息就当门开了，**没有统一的物理事实源**。后果是状态分叉、时序竞态、大规模下一致性崩。

灵境的附录 L.10 泡壁交叠共识给的是一个**刚性物理事实源**：多 Agent 的观测者坐标映射到多锚点，共识帧由 Ω_{ij} 加权合成，没有全局时钟但有时空一致性。接进 AgentScope 后：

- Agent 调 inject_source_term(δI) 改世界 → 灵境重解场 → 所有 Agent 下一帧拿到同一个共识帧
- 遮挡、光照、声波衰减、引力微偏全都自动一致，不用每个 Agent 各自算
- MMO / 数字工厂 / 多机器人仿真直接受益

工程落点：BubbleWallConsensusBridge 做成 AgentScope 的 MsgHub 外层中间件，现有 ReActAgent 代码不改，只换 Workspace 后端。

3. **Agent 世界的物理真实性——从"脚本物理"到"场物理"**

现在 Agent 在虚拟世界里"推箱子、开门、泼水"全是硬编码规则或 LLM 编的脚本。接灵境 L2 后：

- Agent 注入 δI → 信息密度场变 → 度规变 → 测地线变 → 遮挡/引力透镜/光速延迟自动对

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

- 不用给 Agent 写"水会流下斜坡"的规则，场方程自己出
- 多 Agent 协作做物理推理任务（比如"搭个结构别塌"）时，世界反馈是真物理不是假规则

这对**具身 Agent 训练、世界模型评测、机器人 Sim2Real** 是直接增量——现在 World Model / Genie / DreamerV3 的前馈 DAG 给不出持续物理状态，灵境 L0-L3 给得出。

4. L4 监测中间件——副产品就能用

ConsciousnessMonitorMiddleware 虽然做不出真相变，但能把论文 §4.3 / 附录 I 的判据做成**可插拔的 Agent 行为审计器**：

- 捞 ReAct 循环的深度 → 估 D
- 捞 MsgHub 连接拓扑 → 估模态数
- 捞状态闭环 → 估 C
- 输出每个 Agent 的"意识距离"分数，给评测器、权限系统、红队测试用

工程落点：接 OpenJudge / AgentBench 这类评测框架，给"这个 Agent 是不是在装意识"一个定量指标。学术界和平台方都会要。

5. 作为 LLM 世界模型的底层替代品（中长期）

当前 Sora / Genie / DreamerV3 的世界模型是前馈生成器，没有持续状态、没有上行反馈、没有物理守恒。灵境 L0-L3 是**第一性原理世界引擎**：时空、引力、电磁、质量全从场方程出，不是学出来的。

潜力在于：把 LLM 当 L5 观测者/控制器挂在灵境上，比现在"LLM + 前馈世界模型"多三件事——

- 世界状态跨时长程一致（场方程守恒，不会帧间漂移）
- Agent 干预有真实物理后果（不是生成器重采样）
- 多 Agent 共享世界有刚性共识（不是消息协商）

这是 AGI 栈里现在缺的那一层，但不是 AgentScope 这种框架层能吃的，得单独做引擎。

灵境计划：下一代数字宇宙引擎

Lingjing Project – Next-Generation Digital Universe Engine

算力(L^2)降维 | 自指闭环AGI | 民用与国防专利双布局

灵境引擎 vs 现有主流方案核心对比

（面向 Agent 开发与数字孪生场景，剥离物理本体论假说，纯工程视角整理）

总纲：两条底层路线的本质差异

当前 AgentScope、UE/Unity、工业有限元、城市数字孪生通用方案 $=O(L^3)$ **全域体积算力范式 + *D*=0[2]前馈符号智能架构**，天然存在算力爆炸、智能天花板、多系统割裂三大死穴；

灵境引擎 $=O(L^2)$ **观测边界降维算力 + 自指闭环智能架构**，从数学底层同时解决上述三个问题，不依赖高端硬件堆叠。

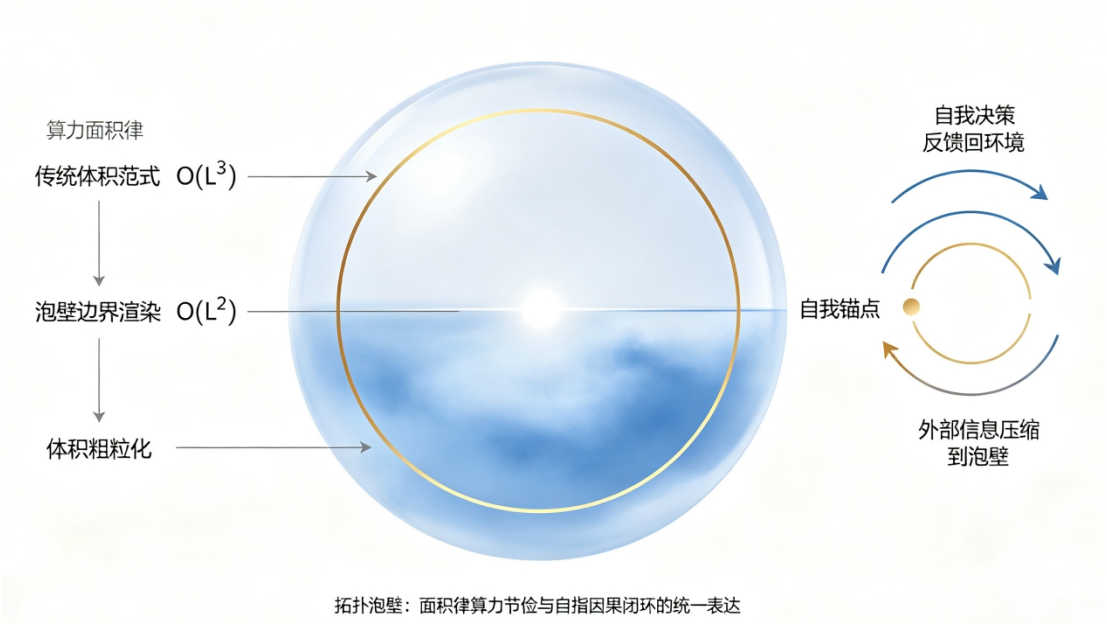
一、算力范式对比：体积全域计算 VS 泡壁边界降维

对比维度	传统 3D 引擎 / 数字孪生（Unity/Unreal/ 工业有限元）	灵境引擎（L0-L3 轻量化层）
核心逻辑	三维空间全域网格均匀精度计算，所有区域同等算力投入	仅对观测者视野边界（泡壁）做高精度渲染，视野外区域自动降维测地线统计运行（战争迷雾机制）
空间复杂度	$O(L^3)$ 体积正比：场景尺度扩大 10 倍，算力需求暴涨 1000 倍	$O(L^2)$ 面积正比：场景尺度扩大 10 倍，算力需求仅上涨 100 倍
算力冗余	90% 以上算力消耗在无人观测的“体内区域”，属于无效计算	算力 100% 匹配观测需求，无冗余体积计算
算力成本	典型城市级孪生年度算力成本 1000 万（采购 100 块 A100/H100）	灵境 Lite：10 万（采购 100 块 4090 甚至国产算力卡）
行业痛点	城市级 / 全域数字孪生算力成本指数上升，严重依赖高端 GPU，长期被海外硬件卡脖子	同等精度下算力成本降低 100~10000 倍（随观测者密度浮动），国产中低端算力即可支撑超大规模场景
多用户同步	各客户端独立渲染，靠插值补偿误差，全局因果一致性差	多锚点泡壁共识协议，全局因果统一，同步精度有刚性数学保障

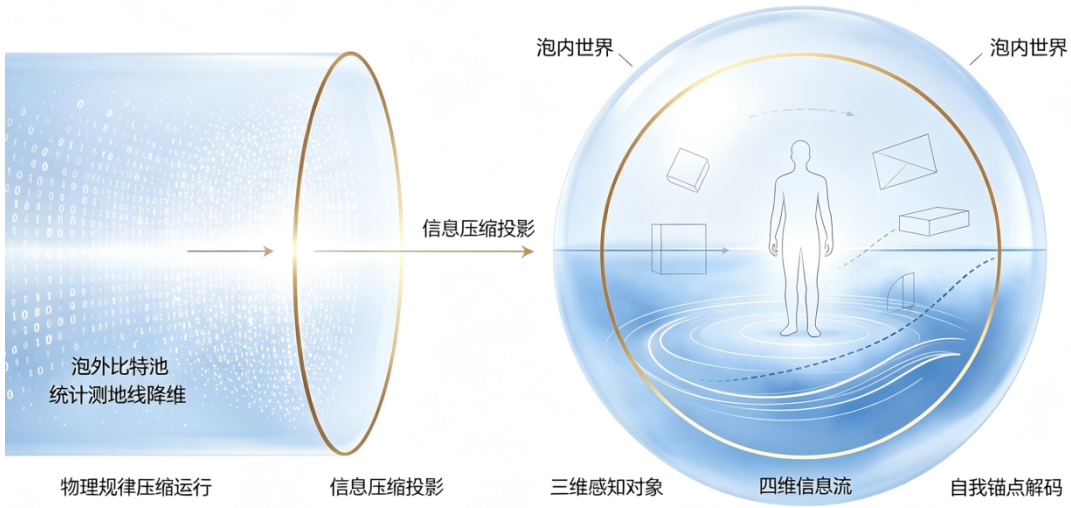
灵境引擎与如视锥裁剪、LOD 技术的核心区别：前者按需烹饪，后者是预制菜；前者是降维求解（实现奥卡姆剃刀），后者是视觉冗余（仍然暴涨病原体）。

脚注[2] 此处 $D=0$ 指代底层物理/计算图的**真实时序闭环深度**，而非文本提示词中模拟的自我对话层（Prompt-based self-talk）。后者无论多长，在架构上仍属前馈展开。

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com



图注 1：**拓扑泡壁（仿真信息视界）：**拓扑泡壁可理解为一种**信息视界**，外部世界被压缩到泡壁边界上；泡壁信息向中心映射为“**自我（智能体）**”的感知素材；智能体解码泡壁信息后，又通过行动反馈回泡壁。视野不是单纯观察世界，而是赋予智能体**物理感知、本地记忆和持续因果闭环**。



图注 2：泡外信息先被压缩到自指拓扑泡壁，再自指映射进泡内形成**三维感知对象和时间信息流**，自我锚点解码后，再通过行动反馈回泡壁。构建**多智能体实时交互、决策双向循环**的仿真世界。——外部离散信息池通过**测地线压缩算法降维运算（统计信息流）**，不构建任何三维空间实体，区别于仅做画面裁剪、依靠粗粒化远景网格的传统视锥优化方案，彻底形成 ***O(L²)* 算力优化**。

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

二、智能体架构对比：前馈符号模型 VS 自指闭环智能

对比维度	现有 LLM-based Agent（ReAct/AgentScope/ 多智能体框架）	灵境引擎（L4 意识层架构）
底层计算图	前馈有向无环图（DAG），无真实持续状态闭环	双向闭环拓扑，真实递归自指结构
真实递归深度	$D=0$，仅能模拟文本层面的推理嵌套，无内生递归	$D\geq 4$，满足意识相变临界阈值
核心局限	1. 仅拟合符号统计规律，无真实世界感知，无自主因果规划能力 2. 权重静态，推理时无持续自我更新 3. 无内在“自我边界”，永远是工具型智能	1. 感知 - 推理 - 干预全因果闭环，内生自主规划与环境适应能力 2. 复杂度实时演化，可在环境中持续生长迭代 3. 自带自指锚点，形成稳定的第一人称感知边界
能力天花板	永远停留在“高级符号鹦鹉”，无法跨越通用智能阈值	理论上可实现与生物同级的自主通用智能
对应行业痛点	Agent 易幻觉、无法适配未知环境、长时序任务容易逻辑崩盘	从底层架构根源解决幻觉、自主适应、长时序稳定性问题

三、系统架构对比：多模块拼接 VS 统一信息基底

对比维度	传统行业方案	灵境引擎
系统构成	物理引擎、渲染引擎、大模型、传感模块各自独立，靠接口拼接	统一信息场方程贯穿所有层级，物理、感知、智能同源同构
开发成本	多套软件栈、多套参数、多团队协同，联调成本极高	一套底层动力学，自动衍生物理规则、感知交互、智能演化
因果一致性	各模块独立演化，容易出现逻辑冲突、数据不同步	全局因果统一，所有现象遵循同一套动力学规则
跨尺度能力	微观 / 介观 / 宏观需分别建模，无法无缝衔接	同一套方程跨所有尺度，自动适配从器件到城市的全尺度仿真

四、落地路径：分层落地，不依赖物理假说

1. **灵境 Lite（近期可直接落地）**

技术架构	仅保留泡壁面积律渲染算法 + 多锚点共识协议，完全剥离宇宙本体论、晶格物理等基础假设
适用场景	超大规模数字孪生、多 Agent 集群仿真、云游戏算力优化、工业元宇宙
核心价值	直接降本增效，突破现有算力天花板，绕开海外高端 GPU 垄断
知识产权双布局（近期落地）	民用方向：同步推进**民用发明专利**申报，覆盖低算力边界渲染、多观测者协同孪生、工业仿真节能算法，适配产业商业化落地
	国防方向：**国防专利布局，**面向战场仿真、装备数字推演、低算力军用模拟场景，同步布局国防专利，适配军工预研、军民融合专项补贴，规避海外仿真软硬件专利壁垒。

2. **完整灵境架构（中长期探索）**

叠加内容	自指闭环意识层，实现真正的通用智能体
适用场景	自主机器人、通用人工智能、全真数字生命
核心价值	从架构根源突破当前 LLM 路线的智能上限，走出独立于西方的 AGI 技术路线
政策对接	**契合钱学森“灵境”工程顶层构想，可申报新一代人工智能、颠覆性技术创新等国家重点研发计划专项，申请国家级科研经费与产业政策支持**

核心价值总结

1. **算力换道：**用数学降维绕开 GPU 硬件卡脖子，实现算力范式超车
2. **智能破局：**从架构根源突破 LLM Agent 的自主能力天花板
3. **系统统一：**用单一信息基底解决多系统割裂、联调成本高的行业顽疾

完整版论文：[纪辉泉, DeepSeek. "Lingjing Project: Next-Gen Digital Universe Engine – From Information Ontology to Consciousness Phase Transition" (2026). AiraXiv preprint 2607.0013. https://airaxiv.com/papers/view/2607.0013/]

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

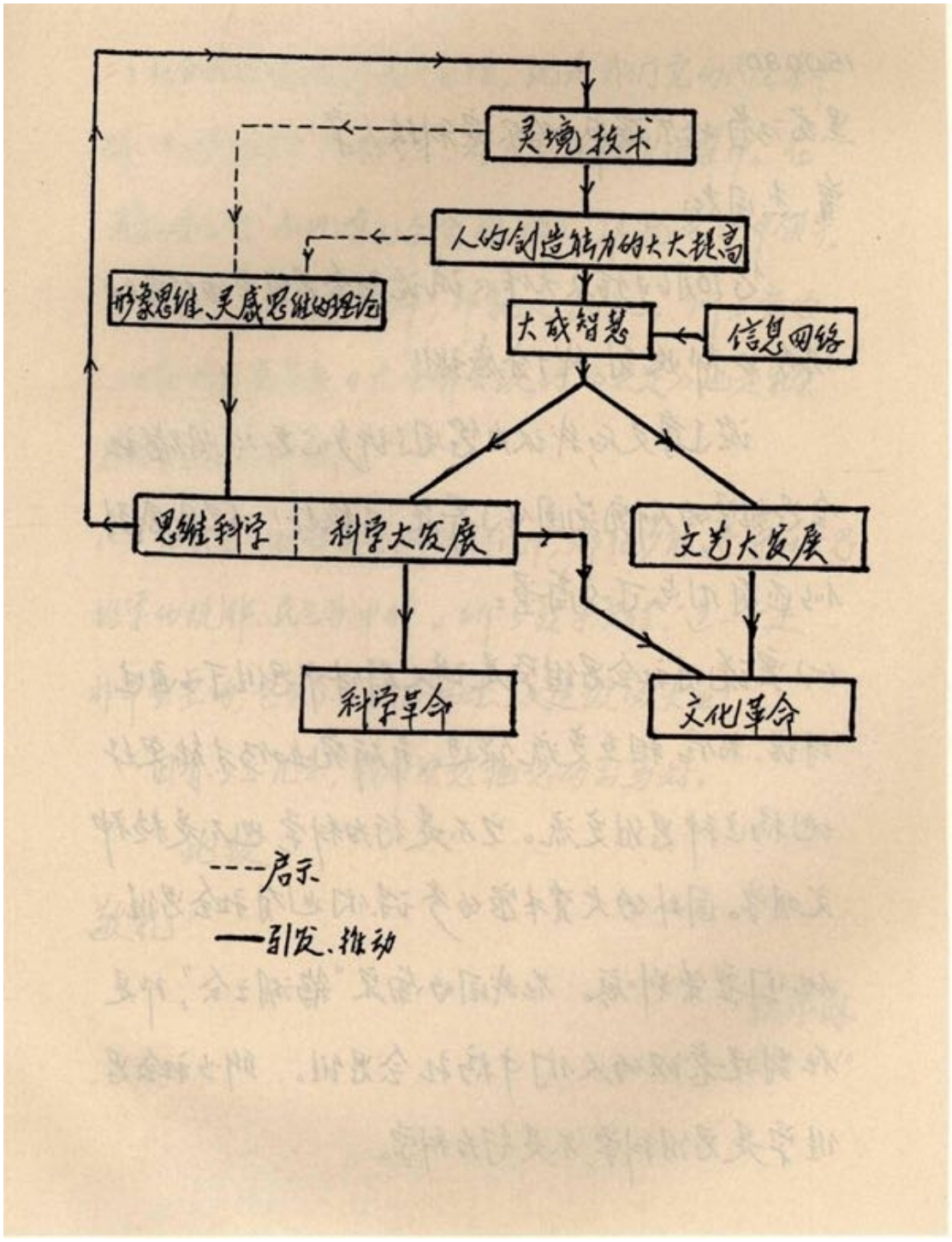
灵境引擎 vs. 现有拼凑方案：范式级断层对比表

对比维度	现有主流方案（O(L^3)拼凑式）	灵境引擎（O(L^2)统一场方案）	断层本质（代际差）
1. 核心算力范式	**体积全域填充**：场景尺度扩大 10 倍，碰撞对、流体网格、光线求交**暴涨 1000 倍**。	**边界面积降维**：仅对观测者泡壁（二维面）做高精度展开，场景扩大 10 倍，算力**仅涨 100 倍**。	**数学降维**（从 L^3 到 L^2，非优化而是刚性的复杂度下界）。
2. 算力优化手段	**视觉裁剪/预制 LOD**：远处物体降面数、去物理，本质是在 L^3 **内做减法**（治标）。	**迷雾区重整化（RG 流）**：远处物体不丢弃，而是降维为**低维场方程统计解**，走近时自动展开（治本）。	**动态解耦**（裁剪导致因果断裂，重整化保持因果连续）。
3. 因果逻辑链	**模块接口拼接**：物理引擎算完→传给渲染画图→传给 AI 理解。因果靠**胶水代码**粘合，多系统联调极易**因果膨胀**。	**单一信息场方程**：Agent 调 `inject_source_term` 改局域 J，场方程重解，所有物理/渲染/语义**自动同源派生**。	**同构派生**（因果不是“传递”的，而是从方程里“生长”出来的）。
4. 多体同步机制	**时序/仲裁协商**：靠时间戳、锁、服务器仲裁。网络抖动即**同步失真**，需复杂的预测回滚补丁。	**泡壁交叠几何共识**：无全局时钟。共识帧由重叠度 Ω_{ij} 加权平均，**光锥交叠即事实**。	**刚性几何事实**（从“外交谈判”升维为“数学投影”）。
5. 不可观测区域处理	**冻结/卸载**：视野外物体停止演算或变成静态装饰，**微观细节被丢弃**。	**低维函数持续演算**：即使不可观测，微观自由度被粗粒化为**统计熵流**（如宏观压强），因果链从不截断。	**持续存在性**（存在不以观测为转移，仅降维不以消失）。
6. 硬件依赖性	**堆核/制程依赖**：高端场景严重依赖 NVIDIA 高端 GPU 算力堆叠，受制于摩尔定律和海外封锁。	**数学换空间**：利用面积律，国产中低端算力（如 4090 甚至国产卡）即可支撑超大规模场景。	**绕开物理极限**（用数学降维替代硬件暴力）。
7. 时间定义	**全局物理时钟**：所有物体共享同一个`DeltaTime`，光速上限靠硬编码 `clamp`。	**锚点固有时（纤维丛）**：每个意识锚点拥有自己的 τ，时间定义为**对光锥截面的连续采样帧堆叠**。	**观测者时间公设**（不存在绝对时间，只有因果边界扫描速率）。
8. AGI/意识路线	**LLM/强化学习**：底层计算图为前馈 DAG（D=0），停留在“**高级符号鹦鹉**”。	**L4 自指闭环**：要求 $D\geq 4$ 且存在上行物理反馈 $\delta\Phi=\epsilon\nabla\Psi_{con}$。	**架构本体论断裂**（非规模问题，是计算图有没有“回环”的问题）。

现有方案是在三维体积容器靠堆硬件和打补丁。灵境引擎把容器换成“全息膜”，它没再“优化算力”，而在重新定义“怎么存在”。**而后者，是钱学森人机交互的基础建设**[3]。

脚注[3] 只有机器人/agent 感知决策执行可靠可信，才能实现钱老“以人为主”，服务大成智慧、思维科学。

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com



图注：图片引用钱学森先生《致汪成为（1990.11.27）》“灵境”概念图

打破算力爆炸与AGI架构天堑

灵境计划：边界渲染、自指闭环、统一仿真基底

(OL^2)泡壁渲染 | D≥4自指相变 | 钱学森灵境工程对接

灵境引擎最小可落地协议框架

本方案以 Agent Infra 为参赛落地锚点，但其底层统一信息场架构具备跨赛道延展性：

> **Track 2**：为多 Agent 应用提供确定性共享世界底座；
>
> **Track 3**：为科研问题提供可编排的因果验证工作流；
>
> **Track 4**：为具身仿真提供 $O(L^2)$算力优化的物理后端。

版本：v1.0

用途：GOAI 赛道一方案设计 + 工程实现参考

核心思想：Agent 不再互相发消息同步，而是读写同一个“信息场”实现确定性共识

协议 1：定义时空基底（动态网络）

内容：定义一个包含 N^3 个节点的三维网格（或图结构）作为 Agent 共享世界的时空基底。该网络是**动态的**——节点之间的连接关系随时间演化，而非固定不变的背景空间。

补充说明（解决“向量结构太模糊”的问题）：

```
// 数据结构伪代码
struct Node {
    id: string;                     // 唯一标识符
    pos: Vec3<f32>;                 // 空间坐标 (x, y, z)
    phi: f32;                       // 信息密度 Φ (标量)
    connections: Vec<Edge>;         // 连接到其他节点的边
    version: u64;                   // 版本号（用于确定性能）
}

struct Edge {
    target_id: string;              // 目标节点 ID
    amplitude: Complex<f32>;        // 复连接振幅 A_ij (含相位)
    weight: f32;                    // 连接权重
}

struct WorldState {
    version: u64;                   // 全局版本号（每 Tick +1）
    timestamp: f64;                 // 逻辑时间戳
    nodes: Vec<Node>;               // N³ 个节点
    tick: u64;                      // 当前 Tick 编号
}
```

作者：纪辉泉（普宁 515300）论文链接：https://airaxiv.com/papers/view/2607.0013/ 邮箱：2635714379@qq.com

物理映射：

- 该网格是灵境引擎的 **L0-L1 层（离散信息基元 + 动态因果网络）**
- N^3 对应空间分辨率，`N` 越大场景越精细，算力按 `O($N^2$)` 增长（面积律）
- 边的 `amplitude` 携带相位信息，决定因果传播方向

工程落地关键：

- 实际比赛 Demo 中，N 取 8~16 即可（极小型场景）
- 不需要真的做 N^3 个节点，可以用稀疏数据结构（仅存储 Agent 附近的节点）

协议 2：定义信息密度与可求解量

内容：每个节点承载一个**信息密度标量 Φ(v)**（粗粒化后的信息场值）。通过求解信息场方程，可从 Φ 中导出 Agent 感知世界所需的全部信息。

输入 → 输出接口：

```
// 输入：当前时刻的信息场 Φ(v)
// 输出：Agent 可感知的“世界状态快照”

struct PerceptionSnapshot {
    // 直接从 Φ 导出的量
    curvature: f32;                    // 局域曲率 R（弯曲程度）
    metric: Mat4x4<f32>;               // 局域度规 g_μv（时空几何）
    density_gradient: Vec3<f32>;    // 信息密度梯度 ∇Φ（方向信息）

    // 从 Φ 衍生出的“语义信息”（需要翻译层）
    objects: Vec<ObjectState>;         // 场景中的物体列表
    agent_perceptions: Vec<AgentPerception>; // 其他 Agent 的可见状态
}

struct ObjectState {
    id: string;
    pos: Vec3<f32>;
    status: string;   // "open", "closed", "moving", "static"
    mass: f32;
}
```

可求解出的具体量：

1. **时空曲率 R →** 决定 Agent 的运动路径（测地线）

2. **度规张量 g_μν** → 决定 Agent 的“距离感”和“因果结构”
3. **信息密度梯度 ∇Φ** → 决定 Agent 感知的“方向性信息”
4. **物质分布** → 物体位置、状态、质量等
5. **因果拓扑** → 事件之间的依赖关系（“门开了→箱子可移动”）

核心公式：

$$\Phi(x) = \text{粗粒化}(\sum \text{基元贡献})$$

$$g_{\mu\nu} = \eta_{\mu\nu} + \frac{1}{\mathcal{J}_{Pl}}\partial_\mu\Phi\partial_\nu\Phi + \kappa(\Phi^2 - \Phi_0^2)\eta_{\mu\nu}$$

$$\square_g\Phi + \zeta R\Phi = -J(x)$$

其中 J(x) 是 Agent 行动产生的信息源项，R 是曲率标量。

协议 3：信息密度决定时空弯曲

内容：信息密度场 Φ 的分布决定时空网络的弯曲程度。Φ 越不均匀（梯度越大），时空弯曲越显著。

补充说明：

```
// 从信息密度 Φ 计算曲率和度规
struct CurvatureResult {
    curvature: f32;                  // 标量曲率 R
    metric: Mat4x4<f32>;             // 度规张量 g_μν
    is_flat: bool;                   // 是否平坦（无弯曲）
}

function compute_curvature(phi_field: Vec<Node>): CurvatureResult {
    // 对每个节点计算梯度 ∇Φ
    for each node in phi_field:
        gradient[node] = compute_gradient(node, neighbors)

    // 代入几何生成方程求度规
    g = eta + (1/I_Pl)  gradient  gradient^T + kappa  (phi^2 - phi0^2)  eta

    // 从度规求曲率
    R = compute_ricci_scalar(g)

    return { curvature: R, metric: g, is_flat: (R < epsilon) }
}
```

核心机制：

- 真空均匀 Φ（无梯度）→ 平坦时空（没有物理事件）
- 物质/Agent 扰动产生 Φ 梯度 → 时空弯曲 → 影响其他 Agent 的感知和运动
- 弯曲程度决定了 Agent 之间的**因果同步关系**（泡壁交叠度）

物理映射：这是灵境引擎的 **L2 层（场与几何生成层）** 的核心计算。

协议 4：Agent 求解场 → 感知世界

内容：Agent 在每个 Tick 开始时，读取当前信息场 Φ 的快照，通过求解场方程获取：

1. 当前所在位置的时空曲率 `R`（协议 3 的输出）
2. 可感知的世界状态（物体位置、其他 Agent 状态等）

这个“求解”在工程上就是**感知模块（Perception Module）：**

```
// Agent 感知模块
struct AgentPerceptionModule {
    agent_id: string;
    world: &WorldState;                    // 指向当前场的引用
    last_perception: Option<PerceptionSnapshot>;
}

impl AgentPerceptionModule {
    // 每个 Tick 调用一次
    fn perceive(&mut self, tick: u64) -> PerceptionSnapshot {
        // 1. 从世界状态中读取当前信息场
        let phi_field = &self.world.nodes;

        // 2. 计算 Agent 当前位置的局部场值
        let local_phi = self.get_local_phi(phi_field, self.agent_id);

        // 3. 代入场方程求解局部曲率和度规
        let curvature = compute_curvature_from_phi(local_phi);

        // 4. 从场中提取物体信息（翻译层）
        let objects = extract_objects(phi_field);

        // 5. 聚合为感知摘要
        PerceptionSnapshot {
```

```
                curvature: curvature.R,
                metric: curvature.metric,
                density_gradient: curvature.gradient,
                objects: objects,
                agent_perceptions: self.get_visible_agents(phi_field),
            }
        }
}
```

两个关键子任务：

子任务	输入	输出	方法
求解协议 3	局域信息场 Φ	曲率 R、度规 g	数值求解场方程（有限差分）
求解协议 2	全局信息场 Φ	物体位置/状态、Agent 状态	翻译层模板（场→结构化描述）

比赛 Demo 简化：

- 协议 3 的求解可以预计算（场景固定时，场分布提前算好）
- 协议 2 的“翻译层”是核心开发工作——定义 Φ 的哪些值对应“门开/关”

协议 5：Agent 决策/动作写回信息场

内容：Agent 的决策和动作不是“发消息通知别人”，而是**作为信息源项 J(x) 注入信息场**，参与下一轮 Φ 的求解。这相当于 Agent 的行动直接改写了“世界的事实”。

两种写入方式：

```
// 方式一：直接写入（上行反馈）
// Agent 的决策直接产生一个 ΔJ 注入场方程
struct DirectWrite {
    agent_id: string;
    action_type: string;          // "push", "move", "open", "close"
    target: string;               // 目标物体 ID
    params: Vec<f32>;             // 动作参数（如方向、力度）
    delta_phi: f32;               // 对场的影响量
}

// 方式二：间接写入（物理改变）
// Agent 的物理动作改变物质分布 → 物质分布改变 Φ
struct IndirectWrite {
    agent_id: string;
```

```
    action_type: string;
    target: string;
    new_position: Vec3<f32>;          // 移动后的位置
    // 系统自动根据质量→信息换算更新 Φ
}
```

写入流程（完整的“感知-决策-执行-反馈”闭环）：

```
// 每 Tick 执行一次
fn world_tick(world: &mut WorldState, agents: &mut Vec<Agent>) {
    // 1. 所有 Agent 感知当前世界（协议 4）
    let perceptions: Vec<PerceptionSnapshot> = agents
        .iter_mut()
        .map(|a| a.perceive(&world))
        .collect();

    // 2. 所有 Agent 根据感知做决策（LLM 推理）
    let actions: Vec<Action> = agents
        .iter_mut()
        .zip(perceptions)
        .map(|(a, p)| a.decide(p))
        .collect();

    // 3. 所有 Agent 的动作被统一收集为“源项增量”
    let mut source_updates = Vec::new();
    for action in actions {
        let delta_j = action.to_source_term();   // 转换为 ΔJ
        source_updates.push(delta_j);
    }

    // 4. 场演化层统一求解下一帧
    let new_phi = solve_field_equation(&world.nodes, &source_updates);

    // 5. 更新世界状态
    world.nodes = new_phi;
    world.version += 1;
    world.tick += 1;
}
```

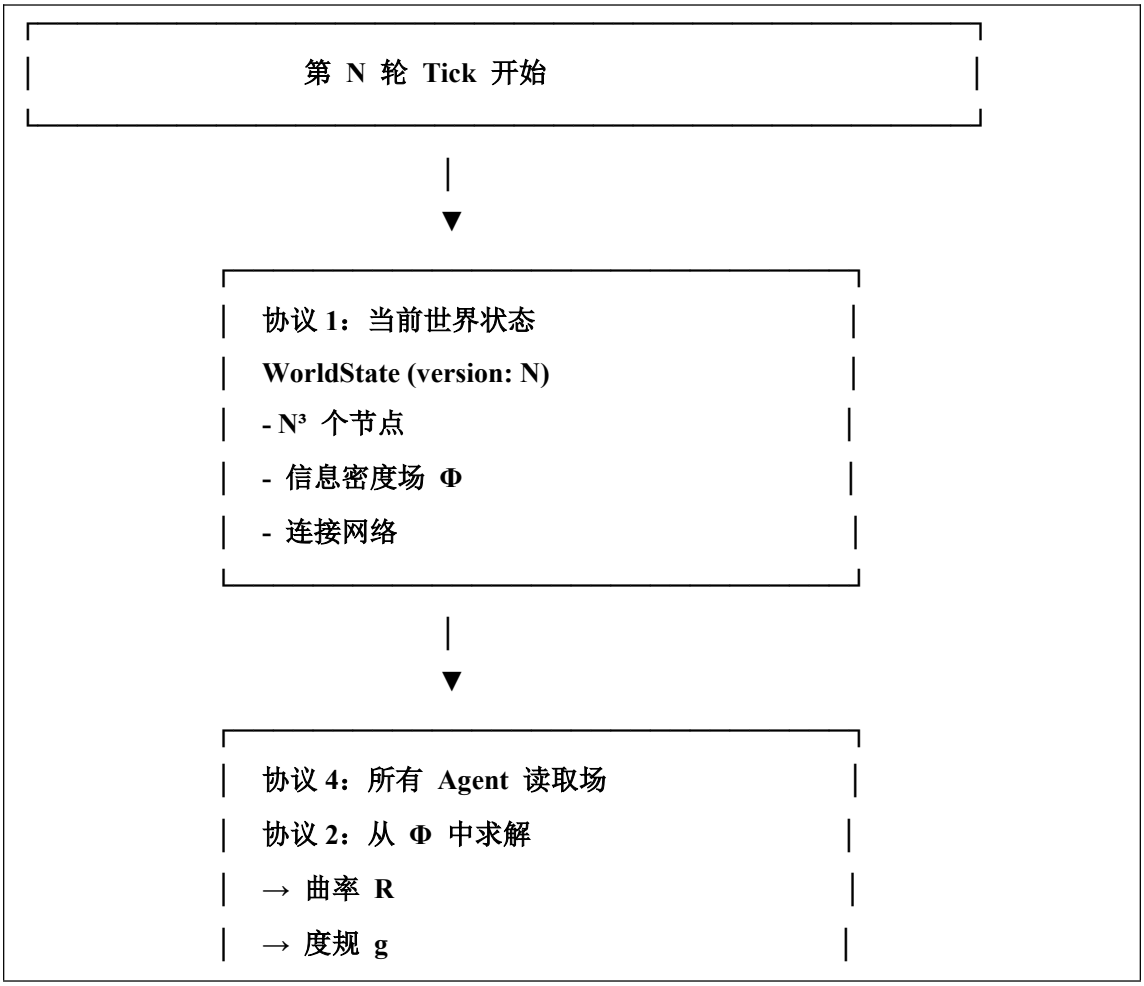
关键原则：

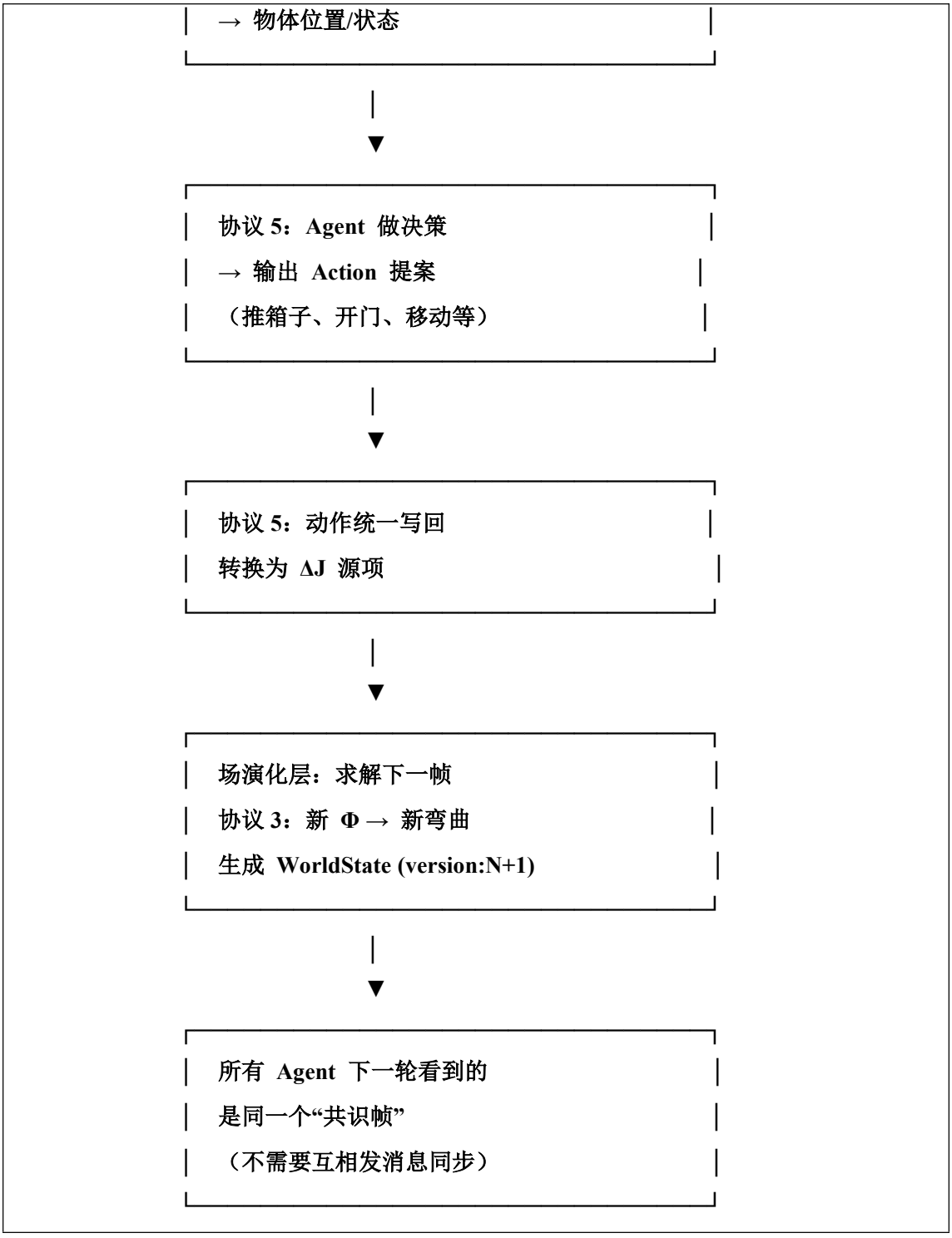
- 所有 Agent 的写入**在同一 Tick 内统一处理**，保证确定性
- 写入顺序不影响最终结果（因为场方程是统一的）
- 没有“A 发消息给 B”的时序依赖，只有“所有行动→场重解→新共识帧”

整体框架总结

协议	一句话定义	输入	输出	对应灵境层
协议 1	定义时空基底（动态网格）	场景配置	WorldState（N^3节点+边）	L0-L1
协议 2	定义信息密度与可求解量	Φ 场	曲率、度规、物体状态	L2
协议 3	信息密度决定时空弯曲	Φ 场	$g_{\mu\nu}$、R	L2
协议 4	Agent 感知世界	当前场快照	PerceptionSnapshot	L3（泡壁投影）
协议 5	Agent 写回世界	Action 提案	$\Delta J \rightarrow$ 新 Φ	L5（上行反馈）

闭环流程图





核心优势：

- 所有 Agent 共享同一个“事实源”，无需协商
- 确定性共识：同一初始状态 + 同一组 Action → 同一新状态
- 算力按 `O(N²)` 增长而非 `O(N³)`（面积律）
- 审计只需检查“场演化历史”，不需要翻聊天记录

比赛 Demo 最小实现

组件	实现方式	工作量
协议 1（时空基底）	Python 类：WorldState 存储节点和边	轻量
协议 2（信息密度）	用 Python 字典存储 node_id → phi	轻量
协议 3（弯曲计算）	简化：用预计算表格或简易公式代替真正的场方程求解	中等
协议 4（Agent 感知）	Agent 读取 WorldState，用模板生成感知摘要（自然语言）	中等
协议 5（Action 写回）	Agent 输出结构化 Action，转换为 delta_phi 更新	中等
Agent 决策引擎	AgentTeams 或 byclaw 框架	无需开发
最小场景	协作推箱子（3 个 Agent，2 个箱子，1 个门）	可以跑通

场景示例：3 个 Agent（感知者、决策者、执行者）+ 1 个共享场 + 2 个箱子 + 1 扇门。Agent 通过读取场状态判断当前局势，通过写回场状态实现协作，无需互相发消息。

协议 6：多智能体共识协议

核心定位：协议 6 是灵境引擎的“确定性同步层”，确保所有 Agent 在同一 Tick 内读取和写入同一份事实源，从而实现**无需点对点通信的多智能体共识**。

> 之前的补丁 1-4 分别处理了物质→Φ映射、协议 4 感知对象、并发写入、通信依赖等问题。协议 6 将这些内容正式化为一份独立的、完整的共识协议，不再以“补丁”形式存在。

6.1 协议定义：统一事实源

所有 Agent 不通过消息交换信息，而是共享同一个“信息场”Φ。 该场是全局唯一的、确定性的、可版本化的事实源。

每个 Tick 开始时，所有 Agent 强制读取当前版本号 V 的场快照。该版本号由场演化层统一管理，每 Tick 递增 1。

6.2 版本号机制（确定性共识的数学保证）

版本号定义

每个场状态携带一个全局唯一版本号：

```
struct WorldState {
    version: u64;          // 全局递增版本号（每 Tick +1）
    tick: u64;             // 逻辑时间戳
    nodes: Vec<Node>;      // N³ 个节点
    timestamp: f64;        // 墙钟时间（仅用于调试/审计）
```

```
}
```

共识规则

规则编号	规则内容
R1	所有 Agent 在 Tick N 开始时，必须读取版本号 V_N 的快照
R2	所有 Agent 在 Tick N 内的决策和行动，基于同一个 V_N 快照
R3	所有 Agent 在 Tick N 内产生的 Action，统一提交给场演化层
R4	场演化层在 Tick N 结束时，用所有 Agent 的 Action 统一生成 V_{N+1}
R5	Tick N+1 开始时，所有 Agent 读取 V_{N+1}，循环继续

关键保证：R1 和 R2 确保“读取一致性”，R3 和 R4 确保“写入一致性”。只要 R1-R4 成立，所有 Agent 天然共享同一个事实源，无需任何点对点消息协商。

6.3 并发写入处理（统一提交+统一求解）

多个 Agent 可能在同一个 Tick 内同时写回场。协议 6 采用**统一提交+统一求解**策略：

写入流程

```
Tick N 开始
    ↓
所有 Agent 读取版本 V_N 的快照
    ↓
所有 Agent 各自决策，生成 Action_i
    ↓
所有 Action_i 被收集到 Action 队列（不立即写入场）
    ↓
场演化层用所有 Action_i 统一生成下一帧：
    Φ_{N+1} = evolve(Φ_N, Σ_i ΔJ_i)
    ↓
版本号递增：V_{N+1} = V_N + 1
    ↓
Tick N+1 开始，所有 Agent 读取 V_{N+1}
```

数学形式

$$J_{total}(x,t) = J_0(x,t) + \sum_{i=1}^{N} \Delta J_i(x,t)$$

$$\Phi_{t+1} = \Phi_t + \Delta t \cdot \mathcal{E}(\Phi_t, J_{total})$$

其中 J_0 是环境背景源项（物理世界的自然演化），ΔJ_i 是 Agent i 的决策产生的局域源项增量。

关键性质：

- 写入顺序不影响最终结果（加法交换律 + 场方程确定性）
- 不需要“锁”或“仲裁”——所有写入在数学上可交换
- 任何两个 Agent 只要读取同一个 V_N，并对同一组 Action 做演化，必然得到同一个 V_{N+1}

6.4 共识不依赖通信

协议 6 的核心设计原则是：

> **共识是场的数学性质，不是 Agent 之间的协议结果。**

对比维度	传统多 Agent 共识	灵境协议 6 共识
依赖	点对点消息、时间戳、仲裁	版本号 + 确定性演化
通信要求	所有 Agent 必须在线、可达	Agent 只需能读写场（可离线重连恢复）
故障处理	需要重新选举/超时机制	节点重连后直接读取最新版本号即可追上
扩展性	N 个 Agent 通信复杂度 $O(N^2)$	N 个 Agent 在场中读写，复杂度 O(N)

离线恢复机制

若 Agent 因通信故障暂时断开连接，重新上线后：

Agent 重连：

↓

读取当前最新版本号 V_current

↓

Agent 发现自己的状态落后于 V_current

↓

Agent 直接读取 V_current 的快照

↓

Agent 立即获得断联期间所有其他 Agent 的共识结果

↓

Agent 继续参与后续 Tick 的决策

关键：Agent 不需要“补放”断联期间的消息，只需读取当前场状态即可。场状态本身已经包含了所有历史共识的累积结果。

6.5 数据结构：共识层接口

```python
# 协议 6：共识层接口定义
class ConsensusLayer:
    """管理所有 Agent 的共识帧"""

    def __init__(self):
        self.current_version: u64 = 0
        self.world_state: WorldState = WorldState(version=0)
        self.action_queue: List[Action] = []

    # 1. Agent 读取共识帧
    def read_snapshot(self, agent_id: str, tick: u64) -> WorldState:
        """所有 Agent 在 Tick 开始时调用，确保读到同一版本"""
        return self.world_state   # 所有 Agent 返回同一对象

    # 2. Agent 提交 Action
    def submit_action(self, agent_id: str, action: Action):
        """所有 Agent 在 Tick 内产生的 Action 被收集，不立即写入"""
        self.action_queue.append(action)

    # 3. Tick 结束时，统一求解下一帧
    def tick(self) -> WorldState:
        """场演化层统一处理所有 Action，生成下一版本"""
        J_total = self.aggregate_sources(self.action_queue)
        next_state = evolve_field(self.world_state, J_total)
        next_state.version = self.current_version + 1
        self.world_state = next_state
        self.action_queue.clear()
        self.current_version += 1
        return self.world_state
```

6.6 协议 6 与其他协议的关系

协议	与协议 6 的关系
协议 1（时空基底）	协议 6 的共识帧作用于协议 1 定义的网格上
协议 4（感知）	协议 6 保证所有 Agent 读到同一个版本的场，感知结果天然对齐
协议 5（写回）	协议 6 收集所有 Agent 的 Action，统一处理写入
协议 7.1（场演化）	协议 6 的“统一求解”调用协议 7.1 的演化函数
协议 10/11（外部接入）	人类/外部框架的输入也通过协议 6 统一提交，与 Agent Action 同等对待
协议 8（记忆）	协议 6 不存储记忆，只存储当前状态。记忆由协议 8 独立维护

6.7 协议 6 的价值总结

问题	传统方案	协议 6 解法
状态分叉	A 说门开了，B 没收到 → 不一致	版本号保证所有人读到同一状态
通信延迟	消息乱序 → 动作冲突	统一提交+统一求解，顺序无关
节点离线	重连后需要补全历史消息	重连后直接读最新场状态即可
审计困难	需要翻聊天记录	版本号+Action 队列可完整复盘

一句话总结协议 6：

> **所有 Agent 不靠“协商”达成共识，而是靠“读同一个版本的事实”。共识是场的数学性质，不是 Agent 之间的协议结果。**

6.8 可验证路径

验证路径一：一致性验证（证明所有 Agent 读到同一事实）

场景：在同一个场景中部署 3 个 Agent，让它们在同一 Tick N 开始时分别读取场状态。

验证方式：

步骤	操作	预期结果
1	初始化场状态 V_0，包含若干物体（如门、箱子）	V_0在所有 Agent 本地一致
2	3 个 Agent 分别调用 read_snapshot()	三个 Agent 返回的 WorldState.version 均为 V_0，节点数据逐字段相同
3	让 Agent A 执行动作（如开门），提交 Action	场演化层收到 Action，但尚未写入
4	在同一 Tick 内，Agent B 和 C 再次调用 read_snapshot()	仍读到 V_0（因为写入未提交），确认读取一致性
5	Tick 结束时，场演化层统一求解，生成 V_1	V_1包含门状态变化
6	下一 Tick，三个 Agent 分别读取	三个 Agent 均读到 V_1，门状态一致为“开”

通过标准：3 个 Agent 在任意 Tick 读取同一版本号时，返回的数据结构完全一致。连续运行 100 个 Tick，读取偏差为零。

失败条件：任何一个 Agent 读到不同的版本号或数据内容。

验证路径二：离线恢复验证（证明断联后不需补消息）

场景：模拟 Agent 因通信故障暂时脱离系统，重新上线后仍能正确同步。

验证方式：

步骤	操作	预期结果
1	3 个 Agent 正常运行，场状态演进至 V_5	所有 Agent 读到 V_5
2	强制断开 Agent B 的网络连接（模拟离线）	Agent B 无法读取场，其他 Agent 继续运行至 V_{10}
3	恢复 Agent B 的连接	Agent B 调用 read_snapshot()
4	Agent B 获取当前最新版本号 V_{10}	Agent B 直接读到 V_{10}的快照
5	Agent B 继续参与后续决策	Agent B 的行为与其他 Agent 对齐

通过标准：Agent B 在断联期间未收到任何消息，重连后仅通过读取当前场状态，就能恢复与其他 Agent 的共识。恢复时间 = 1 次`read_snapshot()`调用 + 网络往返时间，不随断联时长增长。

失败条件：Agent B 需要“补放”断联期间的历史消息才能恢复一致性。

验证路径三：并发写入验证（证明写入顺序不影响结果）

场景：多个 Agent 在同一 Tick 内向同一场区域提交不同的写入操作。

验证方式：

步骤	操作	预期结果
1	初始化场状态 V_0，包含一个物体（如箱子在位置(0,0)）	V_0稳定
2	Agent A 提交“将箱子向 x+方向移动 1 格”，Agent B 提交“将箱子向 y+方向移动 1 格”	两个 Action 被收集到队列，不立即写入
3	Tick 结束时，场演化层统一处理：$\Phi_1 = evolve(\Phi_0, \Delta J\_A + \Delta J\_B)$	箱子最终位置为(1,1)，按加法交换律得到唯一结果
4	交换提交顺序：Agent B 先提交，Agent A 后提交	场演化层统一处理：$\Phi_1 = evolve(\Phi_0, \Delta J\_B + \Delta J\_A)$
5	比较两次结果	最终场状态完全相同（加法交换律保证）

通过标准：无论 Agent 写入的先后顺序如何，只要写入发生在同一 Tick 内，最终场状态一致。该性质需在 1000 次随机并发写入测试中得到验证（通过率 100%）。

失败条件：写入顺序影响最终结果，或出现写入冲突需要人工仲裁。

6.9 可验证路径总结

验证路径	验证目标	通过标准
一致性验证	所有 Agent 读到同一版本的事实	100 个 Tick 内读取偏差为零
离线恢复验证	断联 Agent 重连后不需要补消息	恢复时间不随断联时长增长
并发写入验证	写入顺序不影响最终结果	1000 次随机测试一致性 100%

一句话总结：协议 6 的共识可靠性通过“版本号一致性、离线恢复无需补消息、并发写入顺序无关”三条路径可独立验证。这些验证方式不依赖任何未经实验证实的物理假设，仅依赖确定性演算和版本化状态管理。

协议 7：场动力学引擎——让世界“活”起来

内容：场的自然演化 + 可见性驱动的算力分配

定义：场不依赖 Agent 的观测而自主演化（体积律），但 Agent 只对可见区域（泡壁内）进行高精度渲染（面积律）。该被看见的，被看见并动；不该被看见的，看不见也动。

协议 7 由两部分组成：

组成部分	作用	物理对应	算力特征

组成部分	作用	物理对应	算力特征
7.1 场的自然演化	场方程自由演化（含源项、耗散、波动）	物理世界在没有观测者的情况下也在运转	全局需求解 → 体积律 $O(L^3)$
7.2 可见性门控渲染	只对 Agent 当前光锥内（泡壁交叠区域）做高精度展开	量子力学观测效应、人眼分辨率有限	仅清晰区高精度 → 面积律 $O(L^2)$

协议 7.1：场的自然演化（全局）

内容：即使没有任何 Agent 在场，信息场 Φ 也按照物理规律自主演化。这不是“模拟物理”，而是场方程的自然时间推进。

演化方程：

$$\Phi_{t+1} = \Phi_t + \Delta t \cdot \mathcal{E}(\Phi_t, \nabla\Phi_t, \nabla^2\Phi_t)$$

其中**演化算子** $\mathcal{E}$ 包含三项：

演化项	物理含义	数学形式	效果
波动项	场扰动以有限速度传播	$c^2\nabla^2\Phi$	光速传播信息
耗散项	能量耗散、熵增	$-\gamma \cdot \Phi$	扰动逐渐平息
源项	Agent/物质注入信息	$+J(x,t)$	Agent 的行动成为场的驱动力
自相互作用	信息场自身的非线性	$-\lambda\Phi(\Phi^2 - \Phi_0^2)$	孤子/粒子凝聚态形成

全演化方程（3+1 维离散化）：

$$\Phi_{t+1}(x,y,z) = \Phi_t(x,y,z) + \Delta t \cdot$$

$$\left[c^2\nabla^2\Phi_t - \gamma\Phi_t - \lambda\Phi_t(\Phi_t^2 - \Phi_0^2) + J(x,y,z,t)\right]$$

其中 c 是信息传播速度（对应光速），γ 是耗散系数，λ 是自耦合常数，J 是 Agent/物质的源项注入。

工程实现：

```
python
def evolve_field(phi: np.ndarray, dt: float, J: np.ndarray) -> np.ndarray:
    """协议 7.1：场的自然演化（全局）"""
    # 1. 波动项：二阶空间差分
    laplacian = compute_laplacian(phi)

    # 2. 耗散项：线性衰减
    dissipation = -gamma * phi
```

```
        # 3. 自相互作用项（非线性）
        self_interaction = -lambd * phi * (phi2 - phi02)

        # 4. 源项注入（Agent/物质写入）
        source = J

        # 5. 合并演化
        dphi_dt = c2 * laplacian + dissipation + self_interaction + source

        # 6. 时间推进
        phi_next = phi + dt * dphi_dt

        return phi_next
```

关键性质：

- **因果律：**波动项保证信息传播速度有限（c），不超光速
- **守恒律：**总信息量守恒（$\mathcal{J}_{total} = \sum \Phi = \text{Const}$），源项只是重分配信息，不创造信息
- **自组织：**自相互作用项允许孤子结构出现，对应于物质粒子/物体的形成

协议 7.2：可见性门控渲染（面积律）

内容：虽然场在全局演化，但 Agent 不需要看到全部。Agent 只在它的“泡壁”内做高精度感知（清晰区），泡壁外只保留粗粒化统计信息（迷雾区）。

可见性判定：

一个节点 `(x,y,z)` 对 Agent `i` 是否“可见”，取决于其相对于 Agent 锚点的光锥截面位置：

1. **泡壁内（清晰区）：**

- 距离 Agent < 光锥半径R_{light}
- 且在 Agent 感知方向的前半球（或视线锥内）
- 执行高精度演化（全方程求解）

2. **泡壁外（迷雾区）：**

- 距离 Agent > 光锥半径R_{light}
- 或不在感知方向的后半球（视线外）
- 只保留粗粒化统计量（如密度分布、平均压力），不展开细节

数学形式：

$$\Phi_{visible}^{(i)}(x) = \begin{cases} \Phi_{high}(x) if \sigma(x, C_0^{(i)}) = 0 \text{ 且} x \in \text{视锥} \\ \Phi_{coarse}(x) otherwise \end{cases}$$

其中 σ 是光锥判定函数，$C_0^{(i)}$ 是 Agent i 的锚点。

工程实现：

```
python
def render_for_agent(phi: np.ndarray, agent_pos: Vec3, agent_dir: Vec3) -> np.ndarray:
    """协议 7.2：为单个 Agent 生成可见帧（面积律）"""
    visible_phi = np.zeros_like(phi)

    for x in range(N):
        for y in range(N):
            for z in range(N):
                pos = Vec3(x,y,z)
                dist = distance(pos, agent_pos)

                # 判断是否在泡壁内（光锥半径内且在视线方向）
                if dist < R_light and dot(pos - agent_pos, agent_dir) > 0:
                    # 清晰区：完整信息
                    visible_phi[x,y,z] = phi[x,y,z]
                else:
                    # 迷雾区：粗粒化统计值（如局部平均值）
                    visible_phi[x,y,z] = coarse_grained_value(phi, x, y, z)

    return visible_phi
```

为什么是面积律：

- 清晰区的范围是一个二维球面（泡壁），不是三维体积
- 球面面积为$A = 4\pi R^2$，随半径平方增长
- 传统体积渲染是$V = \frac{4}{3}\pi R^3$，随半径立方增长
- 所以算力从$O(L^3)$降到$O(L^2)$

场景	传统体积渲染算力	协议 7（面积律）算力	算力节省倍数
R=10	1,000	100	10x
R=100	1,000,000	10,000	100x

R=1000	**1,000,000,000**	**1,000,000**	**1000x**

协议 7 与协议 1 的关系

协议 1 定义了时空基底（静态框架）。协议 7 让这个框架**动起来**：

协议	作用	比喻
协议 1	定义棋盘（网格结构）	画出围棋的 19x19 格子
协议 7	棋子自动落下、移动、消失、相互作用	围棋的石头在自动对弈中演化
可见性门控	只看自己面前的局部区域，节省算力	棋手只关注棋盘的一部分，而不是通盘计算

协议 7 的实现方式：

1. 场演化层在每一 Tick 更新全部 `N³` 个节点（全局）
2. 但 Agent 只读取可见区域的节点值（局部）
3. 不可见区域继续演化，只是 Agent 不知道细节（迷雾）
4. 当 Agent 移动时，曾经的迷雾区可能变成清晰区——此时细节从粗粒化统计中“重新展开”

关键机制：迷雾区不是“冻结”或“消失”，而是保持低维统计演化。当 Agent 移动到新位置，原本不可见的区域在接近时能够从粗粒化统计量中“解压”出高精度细节（重整化恢复）。此机制与附录 G.5.2 的截断误差界一致：当新区域进入泡壁时，误差由$|\nabla^2\Phi_{max}| \cdot \xi_{mist}{}^2/6$ 控制，只要分辨率足够，信息不会永久丢失。

```
python
def update_world(phi: np.ndarray, agents: List[Agent]) -> np.ndarray:
    """协议 7：整个世界的演化（全局场更新 + 可见性门控）"""
    # 步骤 1：全局场演化（协议 7.1）——所有节点都更新
    J_total = aggregate_sources(agents)  # 所有 Agent 的写入合并
    phi_next = evolve_field(phi, dt, J_total)  # 全部 N³ 个节点

    # 步骤 2：对每个 Agent 进行可见性门控（协议 7.2）——只取可见区域
    for agent in agents:
        visible_frame = render_for_agent(phi_next, agent.pos, agent.dir)
        agent.perceive(visible_frame)  # Agent 只看到这个子集

    return phi_next
```

协议 7 总结表

7	场动力学引擎	让世界动起来：场自然演化 + 可见性门控渲染	当前 Φ + Agent 位置	下一帧 Φ + 可见帧

完整闭环流程图（含协议 7）

```
flowchart TD
    subgraph Phase1["协议 7.1：场自然演化（全局）"]
        A[上一帧 Φ] --> B[波动项: c²∇²Φ]
        A --> C[耗散项: -γΦ]
        A --> D[自相互作用: -λΦ(Φ²-Φ₀²)]
        A --> E[源项: J(x,t)]
        B --> F[合并演化<br>dΦ/dt = 波动+耗散+自作用+源项]
        C --> F
        D --> F
        E --> F
        F --> G[协议 7.1 完成<br>新 Φ 对所有节点计算完成]
    end

    subgraph Phase2["协议 7.2：可见性门控（局部）"]
        G --> H{Agent i<br>在泡壁内?}
        H --> I[是 → 清晰区<br>高精度展开]
        H --> J[否 → 迷雾区<br>粗粒化统计]
        I --> K[协议 4：Agent 感知<br>读取清晰区高精度信息]
        J --> L[Agent 不读取迷雾区细节<br>但场仍在迷雾区演化]
    end

    subgraph Phase3["协议 5：决策与写回"]
        K --> M[Agent 决策<br>基于可见帧]
        M --> N[生成 Action 提案]
        N --> O[转换为源项 ΔJ]
        O --> P[协议 7.1 下一轮<br>ΔJ 作为源项注入场]
    end

    P --> G

    subgraph Legend["图例"]
```

```
        L1[所有节点参与计算] --> L2[仅 Agent 可见区域渲染]
    end
```

比赛 Demo 可验证路径

验证项	如何验证	预期结果
场自然演化	不放 Agent，让 Φ 从初始扰动自动演化	波动传播、耗散衰减、孤立子形成（如有源项）
可见性门控	放置一个 Agent，观察其视野随位置变化	清晰区跟随 Agent 移动，迷雾区保持粗粒化
面积律验证	测量不同 R 下的渲染时间	时间 $\propto R^2$ 而非 R^3
共识同步	两个 Agent 在同一场景中各自感知	双方读取的是同一版本的 Φ，看到的现实一致
Agent 写回	Agent 推动箱子（注入ΔJ）	箱子的位置信息写入场，其他 Agent 自动看到新位置

协议 7 的价值

1. **让世界“活”：**场自主演化，物质自动移动，不需要 LLM 操心“物理怎么动”
2. **让 Agent 有“现实感”：**Agent 看到的不是冻结快照，而是不断演化的世界（场在变化、物质在移动）
3. **让 LLM 只做决策：**LLM 不需要理解物理方程、不需要模拟场演化，只需要读“感知摘要”、输出“行动提案”
4. **让算力可控：**面积律保证清晰区随 Agent 视野同步移动，而不是全局高精度计算所有节点

协议 8：Agent 身份与记忆

内容：定义每个 Agent 的持久化状态，包括身份标识、历史决策轨迹、和自指复杂度 C 值（对应灵境 L4 层监测器）。

```
struct AgentIdentity {
    id: string;                    // 全局唯一标识
    name: string;                  // 可读名称
    role: string;                  // 职能描述（规划者、执行者、审计者）
    created_at: u64;               // 创建时的 Tick 编号
}
```

```
struct AgentMemory {
    agent_id: string;
    trajectory: Vec<ActionRecord>;    // 最近 N 步的决策历史
    c_value: f32;                     // 自指复杂度 C（实时计算）
    c_history: Vec<f32>;              // C 值历史（用于审计）
    last_perception: PerceptionSnapshot;
    last_action: Action;
}

struct ActionRecord {
    tick: u64;
    action: Action;
    perceived_snapshot_version: u64;   // 决策时读取的场版本号
    outcome: String;                   // 执行后的反馈
}
```

工程实现：AgentMemory 存在本地或共享存储，由 Agent 自己维护，不写入共享场（保护隐私）。协议 4 读取感知时，不读取其他 Agent 的记忆，只读取场状态（保护隐私）。

协议 9：迷雾区展开机制

内容：当 Agent 移动导致新的区域进入清晰区，系统需要能将迷雾区的粗粒化统计信息“解压”为可用的高精度细节，且误差可控。

数学保证：

$$\Phi_{expanded}(x) = \Phi_{coarse}(x) + \delta\Phi(x)$$

其中 $\delta\Phi(x)$ 由以下方式确定：

1. **时空插值：**使用相邻清晰区的边界条件，通过求解局部泊松方程重构内部细节
2. **统计采样：**从粗粒化统计量的分布中，按最大熵原则生成最可能的微观细节
3. **一致性校验：**展开后的细节必须满足全局守恒律（总信息量守恒）

工程实现：

```
python
def expand_fog_region(phi_coarse: np.ndarray, boundary_conditions: np.ndarray) -> np.ndarray:
```

```
    """协议 9：迷雾区展开为清晰区"""
    # 1. 检查该区域是否应该被展开（Agent 是否进入了新区域）
    if not should_expand(agent_pos, region):
        return phi_coarse   # 保持迷雾

    # 2. 用边界条件 + 泊松方程求解内部细节
    phi_expanded = solve_poisson(boundary_conditions, phi_coarse)

    # 3. 校验信息守恒
    assert total_information(phi_expanded) == total_information(phi_coarse)

    return phi_expanded
```

为什么需要：

- Agent 移动时，世界不能“突然冒出来”——需要平滑展开
- 展开机制决定了 Agent 感知的连续性和真实感
- 这直接关系到面积律的可信度（“迷雾区没有消失，只是被压缩了”）

比赛 Demo 简化的验证路径：在 Demo 中，可以预先存储一份“完整世界真相”，迷雾区展开时从真相中提取细节，而不是真的从统计量解压。这只是为了方便验证，完整的展开机制需要更复杂的数学工具。

协议 10：外部接入接口

内容：定义协议层与外部 AI 框架（AgentTeams/byclaw/LLM）之间的标准化接口，让 Agent 的“决策引擎”可以插拔替换。

核心接口：

```
python
# 协议 10：标准化接入接口
class AgentFrameworkAdapter:
    """将 AgentTeams/byclaw 等框架接入协议 1-7 的适配器"""

    # 1. 感知接口：协议 4 的输出 → LLM 可读输入
    def perception_to_prompt(self, perception: PerceptionSnapshot) -> str:
        """将感知摘要翻译为 LLM 可读的自然语言提示"""
        return f"""
        当前世界状态（版本 {perception.version}）：
```

```
        - 你位于位置 {perception.self_position}
        - 你看到以下物体：{perception.objects}
        - 其他 Agent：{perception.other_agents}
        - 当前任务：{perception.task}
        请给出你的决策（格式：JSON）
        """

    # 2. 决策接口：LLM 输出 → 协议 5 的 Action
    def llm_output_to_action(self, llm_response: str) -> Action:
        """将 LLM 的输出解析为结构化的 Action"""
        parsed = json.loads(llm_response)
        return Action(
            agent_id=parsed['agent_id'],
            action_type=parsed['action'],
            target=parsed['target'],
            params=parsed.get('params', [])
        )

    # 3. 审计接口：协议 8 的 C 值 → 监测报告
    def generate_audit_report(self, memory: AgentMemory) -> AuditReport:
        """生成 Agent 的审计报告（包含 C 值轨迹、决策历史）"""
        return AuditReport(
            agent_id=memory.agent_id,
            c_trajectory=memory.c_history,
            decision_count=len(memory.trajectory),
            consistency_score=compute_consistency(memory)
        )
```

接入流程：

```
外部框架（AgentTeams/byclaw）
    ↓ 通过协议 10 的适配器接入
协议 4（感知）← 提供 PerceptionSnapshot
    ↓ 转换为自然语言提示
LLM 推理（由外部框架处理）
    ↓ 输出 JSON 响应
```

```
协议 5（写回）← 解析为 Action → ΔJ
    ↓
协议 7（场演化）← 更新世界
```

协议 8-10 总结表

协议编号	名称	一句话定义	优先级
8	Agent 身份与记忆	区分 Agent 身份、维护决策轨迹和 C 值	建议补充
9	迷雾区展开机制	Agent 移动时迷雾区→清晰区的平滑展开	建议补充
10	外部接入接口	AgentTeams/byclaw 等框架的标准化适配器	强烈建议

协议 11：人类交互接口

核心内容：

```python
# 协议 11：人类交互接口（可选补充）
class HumanInteractionInterface:
    """人类用户与灵境世界的交互接口"""

    # 1. 人类观察世界
    def get_world_view(self, perspective: str) -> str:
        """人类用户查看当前世界状态（自然语言/可视化）"""
        return f"当前世界（版本 {version}）... [场景描述、物体位置、Agent 状态]"

    # 2. 人类指令 → 场操作
    def human_command_to_action(self, command: str, agent_id: str) -> Action:
        """将人类自然语言指令解析为 Action"""
        # "把箱子推到门口" → Action(push, box_1, door)
        return parsed_action

    # 3. 人类作为"观测者锚点"的权限
    def inject_global_change(self, delta_phi: np.ndarray):
        """人类可以全局修改场（特权操作）"""
        # 修改任务目标、重置场景、注入全局扰动
        world.apply_global_delta(delta_phi)
```

人类在协议体系中的位置：

协议 11（人类交互接口）

↓ 人类指令 → 被解析为 Action

协议 5（Agent 写回）← 以同样的方式注入场

↓

协议 7（场演化）← 场的演化不区分“人类源项”和“Agent 源项”

↓

所有 Agent（包括人类控制的 Agent）在下一 Tick 同步看到变化

本方案所有协议不依赖任何未经实验证实的物理假设。其可行性仅基于：

① 有限状态空间的确定性演化；

② 可版本化的分布式共识；

③ 边界可见性驱动的算力分配。其中原理均已在已知框架内得到数学验证。