arXiv:2603.02603v1 [cs.DC] 3 Mar 2026

原子性为何对人工智能/机器学习基础设施至关重要：快照、固件更新，以及仅向前时间类别错误的代价

Paul Borrill
DÆDÆLUS
paul@daedaelus.com

2026年3月

摘要

大规模人工智能/机器学习训练系统依赖两项鲜少受到审视的假设：(1) 检查点代表全局训练状态的原子快照；(2) 基础设施更新可以应用而不会导致混合协议集群状态。这两项假设都是一个更深层结构性错误的表现：仅向前时间（FITO）范畴错误，它混淆了协议收敛属性与时间谓词。

我们将这种混淆形式化为一种类型错误：将时间快照 Snap(t) 等同于收敛属性 Conv($\mathcal{P}$, e)。我们在进程代数框架中对检查点执行建模，并证明，在存在异步组合与崩溃恢复故障的情况下，没有任何时间瞬间能够充当原子性边界。我们在纪元格上重新表述检查点不一致性，并表明原子性是一个零测事件，其补集会随着独立持久化域数量的增加而呈指数级增长。我们将混合纪元恢复形式化为优化代数中的类型违规，并证明由此产生的更新并非任何标准优化器的有效步骤。

对于固件集群更新，我们进一步强化了已知的共识困难性结果：原子部署不仅要求达成一致，还要求对纪元转换形成共同知识；在存在不可靠通信的异步系统中，这一点严格来说无法实现。

最后，我们勾勒出一种受 Open Atomic Eth- ernet 启发的双边收敛协议。该协议无需 Snap(t)，即可实现 Conv($\mathcal{P}$, e)，并以约束语义取代 FITO 假设。

1 引言

现代前沿人工智能训练任务涉及数以万计的图形处理器、多层存储栈、可编程智能网卡、RDMA网络、分布式检查点系统，以及全球复制的元数据服务。尽管基础设施如此复杂，它仍建立在两个几乎从未被明确表述的假设之上：

(i) 原子快照假设。 检查点代表单一且连贯的全局状态——存在某个时刻 t_c，当时“一切都已保存”。 **(ii) 原子部署假设。** 固件和基础设施更新可以应用于集群，且不存在任何可达的执行会导致不同节点按照不同的协议语义运行。

这两种假设都是错误的。更准确地说，二者犯了同样的结构性错误。它们把分布式协议收敛属性——一种必须通过跨故障域的协调行动构建出来的东西——当作时间谓词，即关于在某一时钟时间何者为真的事实。

这种混淆有一个名称。吉尔伯特·赖尔提出了术语范畴错误，用以指将属于一种逻辑类型的概念当作属于另一种逻辑类型的错误 [1]。在先前的工作中，我们识别出一个普遍存在的实例：仅向前时间（FITO）假设，它把因果依赖关系当作时间传播过程，从而将收敛性质误分类为时间谓词 [30, 28, 29]。

本文将 FITO 分析应用于人工智能/机器学习基础设施。本文的贡献如下：

1. **对形式化范畴错误的阐释。** 我们将 FITO 错误精确定义为 $\mathsf{Snap}(t)$（时间谓词）与 $\mathsf{Conv}(\mathcal{P}, e)$（协议收敛属性）之间的类型混淆，并证明二者属于不同的逻辑类型（第 2 节）。2. **进程代数模型。** 我们将分布式检查点建模为异步进程组合，并证明已提交状态是一种轨迹性质，而不是任何单一时刻的状态谓词（第 3 节）。3. **时间快照边界的不可能性。** 我们通过明确的故障模型证明，任何检查点协议都无法通过指定时间边界，在崩溃恢复故障下保证原子性（第 4 节）。4. **纪元格与零测度原子性。** 我们在纪元格上重新表述检查点不一致性，并证明随着持久化域数量的增加，原子状态构成零测集（第 5 节）。5. **语义类型违规。** 我们将混合纪元恢复形式化为优化代数中的类型错误，表明所得更新并不是任何标准优化器的有效步骤（第 6 节）。6. **共同知识不可能性。** 我们强化了针对固件更新的共识数论证：原子部署要求对纪元转换形成共同知识，而这在异步性下严格不可实现（第 7 节）。7. **双边收敛协议。** 我们概述一种无需时间边界即可实现收敛的构造性协议，并将其与 Open Atomic Ethernet 的约束语义联系起来（第9 节）。

2 FITO 范畴错误

我们首先对这一范畴错误作出精确定义。

定义 2.1（范畴错误（Ryle））。 范畴错误是指：用适用于另一种逻辑类型的表达方式，来陈述属于某一种逻辑类型的事实 *[1]*。

定义 2.2（FITO 顺序 [30]）。 *FITO* 顺序是在事件集合 E 上的严格偏序 $(E, \prec)$，其被解释为“先发生关系”优先级，其中 $\prec$具有自反非性、传递性和反对称性。*FITO* 解释断言，因果影响与 $\prec$保持一致，并且可接受的系统叙事必须沿着这一关系向前推进。

定义 2.3（时间快照谓词）。 设 $N = \{1, \ldots, n\}$ 为一组分布式组件，每个组件都具有由时钟时间参数化的局部状态 $S_i(t)$。纪元 e的时间快照谓词为

$$\mathsf{Snap}(t, e) \equiv \forall i \in N, \quad S_i(t) \in \mathsf{Committed}(e).$$

这是一个关于时间瞬间的命题：它断言在时钟时间 t，每个组件都反映纪元 e。

定义 2.4（协议收敛属性）. 设 $\mathcal{P}$ 为一个作用于组件 N的分布式检查点协议。纪元 e 的收敛属性为

$$\text{收敛}(\mathcal{P}, e) \equiv \exists\text{执行前缀 } \alpha \text{ 满足：在 } \alpha\text{之后，} \forall i \in N,\ S_i \in \text{已提交}(e)$$

并且此承诺在故障模型下是稳定的（此后任何可接受的故障都不会导致任何组件从纪元 e 回退）。这是一个关于协议执行的命题：它断言该协议已达到不动点。

命题 2.5（检查点中的 FITO 范畴错误）。 将二者等同

$$\mathsf{Snap}(t, e) \equiv \mathsf{Conv}(\mathcal{P}, e)$$

是一种范畴错误。$\mathsf{Snap}(t, e)$ 是一个时间谓词（逻辑类型：关于某个时间点的命题）。收敛$(\mathcal{P}, e)$ 是一种协议属性（逻辑类型：关于计算历史的命题）。二者属于不同的逻辑类型，不能将其等同，否则就犯了赖尔错误。

证明。 $\mathsf{Snap}(t, e)$ 含有一个自由变量 t，其取值范围是时钟时间；它在特定时刻可满足。$\mathsf{Conv}(\mathcal{P}, e)$ 对协议的执行前缀进行量化；满足它的是计算历史，而非时间点。存在量词的域各不相同：前者是 $\mathbb{T}$（时间），后者是 $\mathcal{P}$的有限执行前缀集合。将这两个域等同，就是把收敛属性当作时间谓词——这正是赖尔范畴错误。 □

注记 2.6. 这不仅仅是一种哲学上的观察。这种混淆会带来工程后果。当系统假设 $\mathsf{Snap}(t, e)$——也就是说，当系统将检查点完成视为发生在某个时间 t_c 的时间事件时——它依赖了一个在存在故障的异步分布式执行中并不存在的时间边界。后续章节将对此作出精确定义。

3 SystemModel 与定义

3.1 训练状态作为分布式乘积类型

设训练配置由来自异构故障域的 n 个组件构成：

$$N = \{D_1, \ldots, D_n\} \quad \text{where} \quad D_i \in \{\mathrm{GPU}, \mathrm{Host}, \mathrm{NIC}, \mathrm{NVMe}, \mathrm{ObjStore}, \mathrm{MetaSvc}\}.$$

每个组件 D_i 都维护取值于域 Σ_i 的局部状态 S_i 。全局训练状态 是以下乘积

$$\mathcal{S} = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_n.$$

具体而言，使用现代优化器（例如 AdamW）的训练运行中，语义相关的状态可分解为：

$$S = (W,\ m,\ v,\ g,\ r,\ d)$$

其中， W 表示模型权重， m 和 v 表示一阶和二阶矩估计， g 表示梯度缓冲区， r 表示随机数生成器状态， d 表示数据迭代器位置。在分布式并行（张量、流水线、数据、专家）中，每个组件都会分片到 N的多个成员上，因此全局状态 $\mathcal{S}$ 是所有组件类型的全部分片的乘积。

3.2 检查点作为进程组合

我们将每个组件的持久化机制建模为 Hoare《通信顺序进程》意义上的一个进程 [7]。

定义 3.1 (持久化进程)。 对于每个组件 D_i，持久化进程 P_i 是一个顺序进程；收到检查点（e）信号后，它会执行一系列局部操作（缓冲区刷新、*DMA* 传输、写入系统调用、*fsync*、元数据更新），并且会：

(a) 以已提交（e）状态终止，这意味着纪元 e 对应的组件局部状态已持久化保存；或者

(b) 失败（崩溃、挂起或返回模糊确认）。

定义 3.2 (检查点协议)。 检查点协议 $\mathcal{P}$ 是以下异步并行组合

$$\mathcal{P} = P_1 \| P_2 \| \cdots \| P_n$$

其中 $\|$ 表示交错（异步）组合，且没有共享时钟。

关键的观察是， $\mathcal{P}$ 是一种具有持续时间的协议。它并非原子操作。每个 P_i 都以自身的速率经历内部状态，并受到本地故障、I/O 延迟和调度延迟的影响。组合体 $\mathcal{P}$ 具有一组允许的轨迹——即所有组件间事件构成的序列——而纪元 e 的已提交状态是某些轨迹的属性，并非任何一条轨迹中的某个单一时刻。

3.3 故障模型

我们采用标准的异步崩溃恢复模型：

(i) **异步性**。不存在全局时钟。消息传递时间和本地计算步骤的时长都没有上界，但均为有限。(ii) **崩溃恢复故障**。任何组件 D_i 都可能在 P_i 执行期间的任意时刻崩溃，随后恢复。恢复后， D_i 会读取其本地稳定存储以确定自身状态。(iii) **独立故障域**。对于$i \neq j$， D_i 发生故障与 D_j 发生故障相互独立。（这是一个刻意采用的乐观假设；相关故障会进一步强化我们的结论。）(iv) **认知模糊性**。发生崩溃恢复事件后， D_i 的状态可能属于以下三种认知类别之一：已提交（纪元 e 已持久存储）、未提交（纪元 e未持久化；该组件反映的是纪元 $e-1$或更早的纪元），或模糊（该组件无法确定自身反映的是哪个纪元，除非进行外部协调）。这一三值模型遵循 [12, 11]的存储层分析。

3.4 轨迹性质与状态谓词的区别

命题 3.3 （已提交状态是轨迹性质）。 “所有组件都已提交纪元 e”这一性质是 $\mathcal{P}$ 的轨迹性质：它由一次执行中的事件历史决定，而非由任何单一时刻的系统状态决定。

证明。 组件 D_i 只有在经历特定的本地事件序列（写入、刷新、确认）后，才会达到已提交(e)。在任何时刻， D_i 是否处于已提交(e)，取决于 P_i 的轨迹中哪些事件已经发生。由于 $\mathcal{P}$ 是异步组合，因此不存在这样一个时刻：所有组件中已完成事件的集合能够由单一时间坐标全局确定。该性质定义在已完成轨迹的集合上，而非瞬时状态向量上。 □

这就是将 FITO 范畴错误应用于检查点的形式内容：把轨迹性质当作状态谓词。

4 时间快照边界不存在

我们现在证明主要的不可能性结果。我们首先确立异步检查点协议的一个结构性质。

引理 4.1（存在跨越边界的调度）。设 $\mathcal{P} = P_1 \| \cdots \| P_n$ 是一个包含 $n \geq 2$ 个组件的异步检查点协议。对于任意拟定的边界时刻 t_c 和任意组件 D_j，都存在 $\mathcal{P}$ 的一个可接受执行，其中 D_j 的持久化序列跨越 t_c：也就是说，在时刻 t_c，D_j 已开始但尚未完成纪元 e 的持久化。

证明。在异步模型下，组件之间的相对速度没有上限。对手（调度器）可以任意减慢 D_j 的本地计算，同时允许其他组件继续执行。特别地，对手可以安排 D_j 的持久化序列在任意 $\delta > 0$ 时刻 $t_c - \delta$ 开始，并在任意 $\delta' > 0$ 时刻 $t_c + \delta'$ 完成。这样一来，D_j 在 t_c 时刻正处于持久化过程中。由于异步模型允许所有这类调度，该执行是可接受的。

□

定理 4.2（不存在时间快照边界）。设 $\mathcal{P} = P_1 \| \cdots \| P_n$ 是一个在具有独立崩溃恢复故障的 $n \geq 2$ 个组件上运行的异步检查点协议。不存在协议 $\mathcal{P}$ 和函数 t_c : 纪元 $\to \mathbb{T}$，使得对于每个可接受的故障调度，以下条件均成立：在时间 $t_c(e)$，要么所有组件都处于已提交(e)状态，要么所有组件都处于仅反映早于 e 的纪元的状态。

证明。反设存在这样的协议 $\mathcal{P}$ 和边界函数 t_c。固定一个纪元 e，并令 $t^* = t_c(e)$。

根据引理4.1，存在一个可接受执行 α，其中组件 D_1 已完成持久化（达到 已提交(e)）早于 t^*，而组件 D_2的持久化序列跨越了 t^*：D_2 已开始写入，但在 t^*时尚未达到持久提交。

现在，在 t^* 时刻对 D_2 施加崩溃恢复故障。由于 D_2 已开始但尚未完成其持久化序列：

- D_2的写入可能只完成了一部分、发生撕裂，或者尚未刷新到稳定存储。
- 恢复后，D_2 处于状态 $e-1$ 或 $\perp$（根据故障模型中的认知歧义条款）。

与此同时，D_1 仍处于 committed(e) 状态（其持久化在 t^* 之前已完成，并且在崩溃恢复下保持稳定）。

因此，在 t^*时，系统处于混合状态：D_1 反映了纪元 e，而 D_2 没有。这与假定的 t_c（e）的原子性相矛盾。

对手的策略可推广到任意 $n \geq 2$：对于任意给定的 t^*，对手都应用引理 4.1，安排一个跨越边界的组件，并在 t^*时使其崩溃。由于故障是独立的，且跨越边界的执行是可容许的，因此对于每个给定的 t^*，都会产生混合状态。

□

备注 4.3 (为什么 Chandy-Lamport 无法挽救这种情况)。*Chandy-Lamport* 快照算法 *[8]*在假设通道为 *FIFO* 且不存在故障的情况下，可提供一致全局快照。使用远程直接内存访问（*RoCEv2*、*InfiniBand*）和集合通信库（*NCCL*、*GLOO*）的 *GPU* 训练集群并不满足 *FIFO* 通道假设：消息可能在不同队列对之间发生重排序，而且多层存储栈（*GPU* →主机 → 网卡 → 存储）会引入非 *FIFO* 顺序。此外，*Chandy-Lamport* 无法容忍组件故障。定理 *4.2*中的情形不满足这两个假设。

注记 4.4（与 Lamport 先发生关系的联系）。纪元 e 的已提交状态需要一条因果链：

$$\mathit{persist}_1 \to \mathit{ack}_1 \to \cdots \to \mathit{persist}_n \to \mathit{ack}_n \to \mathit{commit}$$

在异步性下，这条链具有非零持续时间 [2]。*FITO* 范畴错误将这条链坍缩为一个点——把整个因果历史当作发生在单一瞬间t_c。这是命题 2.5 中所识别的类型错误的时间版本。

5 检查点不一致与纪元格

现在，我们用序理论术语重新表述检查点不一致性。

5.1 纪元格

定义 5.1 (纪元状态)。 针对纪元 e进行检查点尝试后，每个组件 D_i 都处于以下三种状态之一：

$$\sigma_i \in \{e,\ e{-}1,\ \bot\}$$

其中， e 表示已提交至纪元 e， $e{-}1$ 表示已回退至先前纪元，而 $\bot$ 表示认知上模棱两可（该组件无法在没有外部协调的情况下确定自身所处的纪元）。

定义 5.2 (纪元格)。 纪元格 $\mathcal{L}_n$ 是积格

$$\mathcal{L}_n = \{e, e{-}1, \bot\}^n$$

按组件逐项排序，其中 $e{-}1 < \bot < e$。最大元是 $\top = (e, e, \ldots, e)$（全部已提交），最小元是 $\bot_{\mathcal{L}} = (e{-}1, e{-}1, \ldots, e{-}1)$（全部已回滚）。

定义 5.3 （原子状态）。 $\mathcal{L}_n$ 的原子状态恰好是最大元和最小元： $\{\top, \bot_{\mathcal{L}}\}$。格中的所有其他点都是混合状态。

5.2 原子性作为零测事件

我们通过独立二元模型为 $\mathcal{L}_n$ 赋予概率测度。

定义 5.4 （独立二元模型）。 假设每个组件独立地以 $q \in (0,1)$ 的概率提交纪元 e，并以 $1-q$ 的概率回滚至 $e{-}1$。（三值模型将在下文单独讨论。）

引理 5.5 （混合状态概率）。 在独立二元模型中，

$$\Pr[\textit{mixed state}] = 1 - q^n - (1-q)^n.$$

证明。 仅有的原子性结果是全部提交（概率为 q^n）和全部回滚（概率为$1-q^n$）。它们的补集就是混合区域。 □

定理 5.6 （原子性是零测集）。 对于任意固定的 $q \in (0,1)$，

$$\lim_{n\to\infty} \Pr[\textit{atomic state}] = 0.$$

收敛 速度呈指数级：$\Pr[\text{原子性}] \le q^n + (1-q)^n$，且两项都按指数速度衰减 y.

证明。 $q^n \to 0$和 $(1-q)^n \to 0$，当 $q \in (0,1)$时成立。两个趋于零的序列之和也趋于零。由 $\max(q, 1-q) < 1$可得指数衰减。 □

注释 5.7 （格上的解释）. 原子状态 $\{\top, \bot_{\mathcal{L}}\}$ 是包含 2^n 个元素的格（二元模型）或包含 3^n 个元素的格（三元模型）中的两个点。随着 n增大，原子状态在格中所占比例趋近于零。在故障条件下，检查点尝试之后的系统状态以趋近于 *1* 的概率落入格内部。原子性并非典型情形，而是例外情形。

注释 5.8（故障相关性强化了该结果）. 独立二元模型是刻意乐观的。在实践中，存储网络拥塞、电源域故障和级联拖尾效应会导致正相关的组件故障。在正相关下，全部提交的联合概率为$\Pr[\text{全部已提交}] \le q^n$（仅在独立条件下取等），混合状态概率则相应更高。因此，我们的界低估了生产系统中非原子检查点的风险。

5.3 具有认知模糊性的三态模型

定理 5.9（含模糊性的强化界）。 如果每个组件独立地以概率 q 处于状态 e，以概率 p 处于状态 $e-1$，并以概率 $r = 1-q-p > $ 处于状态 $\perp 0$，那么

$$\Pr[atomic\ state] \le q^n + p^n$$

而且，状态向量中出现任何 $\perp$ 在操作上都违反原子性（系统无法从不可知状态中可靠地恢复），因此

$$\Pr[operationally\ atomic] \le q^n.$$

证明。操作层面的原子性要求所有组件都处于状态 e（系统从纪元 e 恢复），且不存在模糊性。该事件的概率为 q^n。“全部回退”事件的概率为 p^n。两者都会呈指数级衰减。

□

5.4 具体参数

对于一个采用 8 路张量并行、4 路流水线并行和 64 路数据并行训练的 700 亿参数模型，独立持久化的原子单元数量包括：权重分片（$8 \times 4 \times 64 = 2048$）、优化器动量分片（$2 \times 2048 = 4096$，对应 m 和 v）、梯度缓冲区、每个秩的随机数生成器状态、数据迭代器位置以及元数据记录。保守估计为 $n \ge 4000$。

q（单元可靠性）	n	$\Pr$[原子]
0.999	1000	≤ 0.368
0.999	4000	≤ 0.018
0.9999	4000	≤ 0.670
0.9999	10000	≤ 0.368
0.99999	10000	≤ 0.905

即使单元可靠性达到四个九（$q = 0.9999$），拥有 4000 个持久化单元的系统实现原子性的概率也低于 0.67。规模越大，非原子检查点就越不是罕见的边缘情况，而是预期结果。

6 语义因果关系作为类型违规

We 现在，我们将形式化非原子恢复对优化正确性的影响 .

6.1 优化类型系统

考虑一个使用 AdamW 优化器的训练循环。在每个纪元 e，优化器状态是如下形式的元组：

$$\mathcal{T}_e = (W_e,\ m_e,\ v_e,\ g_e,\ r_e,\ d_e)$$

其中 W_e 为权重，m_e 和 v_e 为一阶和二阶矩估计，g_e 为累积梯度，r_e 为随机数生成器状态，d_e 为数据迭代器位置。

定义 6.1（纪元类型状态）。 我们写入 $\mathcal{T}_e : \mathsf{State}(e)$ 以表示元组 $\mathcal{T}_e$ 具有类型$\mathsf{State}(e)$——所有组件均反映同一逻辑训练纪元 e。该类型由纪元参数化。

定义 6.2（有效的优化器步）。 优化器步函数

$$\mathsf{step} : \mathsf{State}(e) \to \mathsf{State}(e{+}1)$$

$maps$ 将一个纪元一致状态转换为下一个纪元一致状态。对于 *AdamW*，其形式为：

$$\begin{aligned} m_{e+1} &= \beta_1 m_e + (1-\beta_1) g_e \\ v_{e+1} &= \beta_2 v_e + (1-\beta_2) g_e^2 \\ W_{e+1} &= W_e - \eta \left(\frac{\hat{m}_{e+1}}{\sqrt{\hat{v}_{e+1}} + \epsilon} + \lambda W_e \right) \end{aligned}$$

其中 $\hat{}m$和 $\hat{}v$ 经过偏置校正。仅当输入的所有组件共享相同的纪元类型时，该函数才有明确定义。

6.2 将混合纪元恢复视为类型强制转换

定义 6.3（混合纪元状态）。 混合纪元状态是一个元组

$$\mathcal{T}_{\text{mixed}} = (W_e,\ m_{e-1},\ v_e,\ g_{e-1},\ r_e,\ d_{e-2})$$

其中不同组件反映不同的纪元。我们写作 $\mathcal{T}_{\text{mixed}}$： 状态（$e$、 $e{-}1$、 e、 $e{-}1$、 e、 $e{-}2$），以表示异构纪元分配。

定理 6.4（混合纪元恢复下的类型违规）。 设 $\mathcal{T}_{\text{mixed}}$:状态($\vec{e}$) 是一个混合纪元状态，其中 $\vec{e} \neq (e, e, \ldots, e)$ 对于任意单个纪元 e 均成立。那么 步($\mathcal{T}_{\text{mixed}}$) 不是有效的优化器步：其输出不属于任何 e 对应的状态 $(e{+}1)$ 。

证明。 AdamW 更新方程的定义假设 m_e、 v_e、 g_e 和 W_e 在因果关系上保持一致——它们各自反映同一组梯度步序列的累积效应。如果用 m_{e-1} 替代 m_e，那么 $m_{e+1} = \beta_1 m_{e-1} + (1-\beta_1)$ $g_{e'}$ （对于某个 e'），这对应于一种跳过了一个纪元的梯度信息的动量估计。偏置校正 $\hat{}$ $m_{e+1} = m_{e+1}/(1-\beta_1^{e+1})$ 随后使用一个按 $e{+}1$ 步校准的分母，但分子反映的累积量只有 $e{-}1$ 步。由此产生的更新方向

$$\frac{\hat{m}_{e+1}}{\sqrt{\hat{v}_{e+1}} + \epsilon}$$

doe不对应于从任何一致初始状态出发、由 AdamW 产生的任何有效轨迹 .

形式化地：将有效轨迹集合

$$\mathsf{Traj}(e_0, e) = \left\{ \mathsf{step}^{e-e_0}(\mathcal{T}_{e_0}) \;\middle|\; \mathcal{T}_{e_0} : \mathsf{State}(e_0),\ \text{all intermediate inputs are valid batches} \right\}$$

定义为从任何纪元一致的初始化状态 $\mathcal{T}_{e_0}$ 出发，在所有可接受的数据序列上恰好应用 $e - e_0$ 次 step 后可到达的所有状态的集合。由于 step 的构造保证其保持纪元一致性（每次应用都将 State(e') $\to$ State($e'{+}1$) 映射为），因此 Traj(e_0, e) 中的每个元素的类型都是 State(e)。

混合纪元状态 $\mathcal{T}_{\text{mixed}}$ 包含来自不同纪元的组件，因此对于任意 e_0, e，都有 $\mathcal{T}_{\text{mixed}} \notin$ Traj(e_0, e) 不成立。混合状态位于优化器的可达集合之外。对其应用步骤会产生一个位于任意 e_0 的 $\bigcup_{e'}$ Traj(e_0, e') 之外的值——系统将永久离开有效轨迹空间。 □

说明 6.5 (静默强制转换**)**。 在实现层面，张量是由浮点数组成的无类型数组。类型 State(e)是由训练协议施加的语义类型，而非由运行时强制执行。加载混合纪元状态时，运行时会在不检查纪元一致性的情况下，将其静默强制转换为“当前状态”。这是 *FITO* 范畴错误在实现层面的体现：系统将 加载的状态视为 State(e) ，因为它是在“时间 t_c”保存的，但对于异构纪元向量而言，该状态实际上是 State$(\vec{e})$。

其后果——训练损失偏离、模式崩溃、最终模型质量下降——可能要经过许多后续步才会显现，因此这种损坏在认识层面上是隐蔽的。

示例 6.6 （AdamW 中的单纪元矩偏斜）。 假设纪元 $e = 1000$的检查点恢复时， W_{1000} 和 v_{1000} 均正确，但一阶矩 m 反映的是纪元999 （一个分片未能持久化最新的矩更新）。下一次更新的计算如下：

$$m_{1001} = \beta_1\, m_{999} + (1 - \beta_1)\, g_{1000}.$$

与正确的轨迹（$m_{1001} = \beta_1\, m_{1000} +$ （$1 - \beta_1$） g_{1000}）相比，误差为

$$\Delta m = \beta_1(m_{1000} - m_{999}) = \beta_1(1 - \beta_1)g_{999}.$$

当 $\beta_1 = 0.9$时，该值为 $0.09\, g_{999}$——约为前一个梯度的 *9%*，作为幻影校正注入。一阶偏置校正矩$\hat{}m_{1001} = m_{1001}/$（$1 - \beta_1^{1001}$） 使用的分母是按累积 *1001* 步校准的，但分子现在反映的轨迹却跳过了一次梯度应用。因此，得到的更新方向与有效 *AdamW* 的偏差正比于 $\|g_{999}\|/$（$\sqrt{v_{1000}} + \epsilon$），其量级与学习率步长本身相同。在后续纪元中，这种扰动会通过矩递归不断累积。

7 固件部署与共同知识

现在我们转向第二个假设：原子基础设施更新。

7.1 原子部署即纪元协定

考虑一个 XPU机群 $X = \{X_1, \ldots, X_n\}$，正在将固件版本从 F_0 升级到 F_1。原子部署 意味着集群的操作语义从 F_0 转换到 F_1，且不存在任何可达的执行，使得不同节点在不同的固件语义下参与同一个分布式集合通信。

定义 7.1 （混合协议执行）。如果存在正确节点 X_i、X_j 以及一个集合操作实例 C，使得 X_i 在 F_0 语义下执行 C，而 X_j 在 F_1 语义下执行 C，则该集群执行是混合协议执行。

定义 7.2 （原子部署）。原子部署协议满足：

(a) 协定： 没有两个正确节点会决定不同的固件纪元。*(b)* 有效性： 如果所有正确节点都发起更新，决定的纪元就是 F_1。*(c)* 终止性： 所有正确节点最终都会作出决定。

这是关于转换纪元的共识。

7.2 从共识到共同知识

Herlihy 的共识数论证 [3]表明，读写寄存器的共识数为 1：它们无法为 $n \geq 2$ 个进程解决共识问题。如果集群更新协议仅使用读写消息传递（没有比较并交换，也没有外部预言机），那么在那些会使共识无法实现的故障条件下，它就无法保证原子部署。

现在，我们将这一结果进一步加强。

定理 7.3 (原子部署的共同知识要求**)**。 在具有不可靠通信的异步分布式系统中，原子部署——即不存在任何混合协议执行——要求在任何节点按照 F_1 语义采取行动之前，所有正确节点都对纪元转换形成共同知识 。

证明。接 下来，按照 Halpern 和 Moses 的说法， [5], 群体 G 中的某个事实 φ 的共同知识定义如下：

$$C_G(\varphi) \equiv E_G(\varphi) \wedge E_G(E_G(\varphi)) \wedge E_G(E_G(E_G(\varphi))) \wedge \cdots$$

其中 $E_G(\varphi)$表示“ G 中的每个人都知道 φ。”

对于原子部署，令 $\varphi =$ 表示“到 F_1 的转换已生效。”假设节点 X_i 在F_1 下行动，而某个节点 X_j 尚不知道该转换已经生效。此时， X_j 仍可能在涉及 X_i的集体中依据 F_0 行动，从而产生混合协议执行。为防止这种情况，X_i 必须知道 X_j 知道 φ；但 X_j 还必须知道 X_i 知道 φ（否则 X_j 无法确定 X_i 不会回退）；以此类推。这种无限延伸的相互知识链正是共同知识。

Halpern 和 Moses [5]证明了，在无法保证通信的系统中（消息可能丢失或被任意延迟），共同知识无法实现。由于我们的故障模型包含崩溃恢复和异步通信，因此关于纪元转换的共同知识无法实现。

因此，在这些假设下，要求对转换达成共同知识的原子部署是不可能的。

□

推论 7.4。 分阶段推出、向后兼容的协议版本以及基于池的隔离，并不只是工程折衷。它们是对一种无法实现的性质所作的<*strong? style='3'*>理论上必需的 放宽条件。在异步性下声称能够执行“原子”固件部署的任何系统，要么使用了比读写消息传递更强的原语（例如共识服务），要么隐含地作出了 *FITO* 假设，认为该转换“发生在时间 t_{deploy}”。

备注 7.5 （实际放宽条件）。 定理 *7.3* 确立了严格的不可能性。在实践中，系统可以通过引入更强的协调原语来近似实现原子部署。如果集群使用共识服务（*Raft*、*Paxos*）维护元数据，就可以通过该服务将纪元转换线性化，从而实现我们或许可以称为 ϵ-共同知识的状态：参与决策点之后任何集体的每个正确节点都已观察到已提交的转换。在观察到决策之前崩溃的节点会被隔离出后续集体。这严格弱于 *Halpern-Moses* 共同知识（它不提供无限的相互知识链），但如果隔离机制可靠，便足以满足实际安全性的要求。共同知识与 ϵ-共同知识之间的差距，正是混合协议执行仍然可能发生的窗口期——在异步性下，这个窗口可以缩小，但无法消除。

备注 7.6 （为何这对 AI 集合通信至关重要）。 在紧耦合的 *AI* 训练中，集合操作（*AllReduce*、*AllGather*）会让所有参与者参与同一个逻辑操作。单个节点在不同的固件语义下执行，可能会破坏一次归约。不同于松耦合的微服务（其中混合版本执行可能导致平稳降级），混合固件的 *AI* 集合通信可能产生静默的数值损坏：*AllReduce* 在没有报错的情况下完成，但聚合梯度在语义上无效。这正是第*6*节中类型违规在优化层面的对应物。

8 重试放大作为 FITO 反馈

检查点与重启是大规模训练中占主导地位的恢复机制。但重试机制内含一个 FITO 假设：它将故障视为一个时间事件 （“检查点在时间 t 失败”），并将恢复视为一次时间重启 （“从时间 t' 的最后一个正常检查点恢复 ”）。

这会形成一个正反馈循环：

1. 检查点故障会触发重试（从上一个检查点重新启动）。 2. 重试会给存储和网络子系统带来额外负载（重新读取完整检查点、重新初始化集合通信）。 3. 额外负载会提高后续故障发生的概率（存储超时、网络拥塞、级联掉队节点）。 4. 后续故障会触发更多重试。

这一循环在实践中有充分记录：Google 明确设计了多层检查点，以降低重试开销 [25], 而 AWS 则将检查点存储架构视为训练可靠性的基础 [26]。

这一循环的 FITO 结构在于，每次重试都假定先前的故障只是一个孤立的时间事件。但检查点过程中的故障并非孤立事件，而是检查点协议不收敛的症状 。在导致不收敛的相同条件下重试一个不收敛的协议，是一种结构性错误，而非恢复策略。

9 迈向基于收敛的原子性

前文各节已论证，FITO 方法用于检查点和固件部署在结构上是不健全的。现在，我们概述一种建设性的替代方案。

9.1 双边检查点提交

受 OpenAtomic Ethernet [29]，双边链路语义的启发，我们提出一种检查点协议，以收敛取代时间边界：

定义 9.1 （双边检查点协议）。 纪元 e 的双边检查点协议分两个阶段进行：

（*a*）暂定持久化。 每个组件 D_i 将其纪元 e 的局部状态持久化到稳定存储，并向协调器发送就绪（i、 e） 确认。该确认具有反映性：它携带已持久化状态的密码学哈希或内容摘要，从而证明实际提交的内容，而不仅仅是某个操作曾经发生。

(b) 提交或回滚。 协调器收集确认。如果在超时期限内收到所有 $i \in N$ 的就绪(i, e) ：

- 协调器广播提交(e)。
- 每个组件将纪元 e标记为稳定检查点。

否则：

- 协调器广播回滚(e)。
- 每个组件都丢弃暂定持久化，并保留纪元 $e-1$。

命题 9.2。 双边检查点协议无需 Snap(t, e)，即可实现 Conv$(\mathcal{P}, e)$。提交点是一个协议事件——协调器的决策，而不是时钟事件。

证明。 若协调器决定提交（e），则依其构造，每个组件都已发送就绪（i， e），这意味着每个组件都已持久化保存纪元 e。已提交状态由协议轨迹（收到全部确认）决定，而非由时间边界决定。

若任一组件在发送就绪之前发生故障，协调器便决定回滚，系统则始终一致地停留在纪元 $e-1$。

该协议将收敛（对纪元 e 的全有或全无）实现为其执行的一种属性，而非任何时钟时间的属性。 □

注记 9.3。 这在结构上是一种两阶段提交协议。其贡献不在于协议本身，而在于重新框定：原先被非正式地理解为“我们需要两阶段提交来处理检查点”的需求，在此由 *FITO* 分析推导得出。对显式收敛协议的需求是范畴错误的后果——假定Snap（t）的系统不会实现此类协议，因为它们认为时间边界已经足够。

9.2 通过共识进行纪元转换

对于固件部署，共同知识不可能性（定理 7.3）表明，纯消息传递无法实现原子部署。然而，如果集群已经为元数据管理维护了共识服务（Paxos、Raft）——正如大多数大型训练集群所做的那样——则纪元转换可由共识来介导：

1. 将 propose-transition(F_1) 提交给共识服务。2. 服务提交该转换，生成一个对所有参与者可见的线性化决策。3. 每个节点在观察到已提交的决策后，都会转换到 F_1，然后才参与任何后续集体操作。

这将共识数从 1（读写消息传递）提升到共识服务的水平（对于基于比较并交换的实现，实际上为 ∞）。这并未达到 Halpern-Moses 严格意义上的共同知识，但它提供了实践共同知识：任何观察到已提交决策的节点都可以在 F_1下采取行动；而任何尚未观察到该决策的节点，都会在下一次集体操作前观察到它，否则就会被隔离。

9.3 验证检查点

作为防范语义类型违规（第6节）的一种方法，我们建议在检查点恢复后执行验证步骤：

1. 加载检查点后，在一个留出验证批次上运行一次前向传播。
2. 将验证损失与检查点时刻记录的值进行比较。
3. 如果损失偏离超过阈值 δ，则该检查点会被标记为可能不一致，系统将回退到更早的纪元。

这并不能保证检测出所有类型违规（损失景观在损坏状态周围可能局部平坦），但它提供了一项实用检查，用于确认加载的状态位于有效轨迹空间 Traj(e_0， e) 中。

9.4 从 FITO 到约束语义：通用修正

这三项提议——双边提交、共识介导的转换和验证检查点——具有共同的结构。每一项都以一项约束取代一个 FITO 假设（时间边界、时间部署事件、时间状态标识）：

FITO 假设	约束替换	机制
Snap(t)	Conv($\mathcal{P}$, e)	双边提交
部署(t)	关于纪元的共识	Raft/Paxos
加载时间的状态(e)	验证不变量	前向传播检查

此表是将 FITO 修正为约束语义这一通用改进的具体实例，相关内容见 [30]：以关系约束取代时间谓词，并通过协议执行对其进行验证。

10 相关工作

一致快照。 Chandy 和 Lamport [8] 在无故障的 FIFO 通道下建立了一致全局快照理论。后续研究将其扩展到非 FIFO 通道和部分故障模型。我们的贡献有所不同：我们表明，检查点原子性 （全有或全无提交）所要求的保证，严格强于快照一致性（因果一致性）。

AI/ML 检查点系统。 ByteCheckpoint [13]提供与并行配置无关的检查点表示；Gemini [14] 支持带机器间备份的内存检查点；DataStates-LLM [15] 可在前向/反向传播期间实现异步检查点；Universal Checkpointing [16]支持在不同并行配置之间继续运行。这些系统致力于提升检查点效率，但并未解决原子性问题：它们假设所有分片持久化后即可构成一致快照，而这正是我们所质疑的 FITO 假设。

大规模训练的可靠性。 Kokolis 等人 [17]分析了 Meta 研究集群上 1.5 亿 A100 GPU 小时的数据，并报告了故障率，表明检查点至关重要。Meta 的 Llama 3 训练报告记载，在 16,384 个 H100 GPU 上，54 天内发生了 419 次中断 [18]。OPT-175B 训练日志 [19] 记录了 35 次由硬件故障导致的重启，这些故障影响了 39.3% 的虚拟机。BLOOM 176B 参数训练 [20] 报告称，在 400 个 GPU 上每周发生 1–2 次 GPU 故障，且大规模运行时存在框架挂起问题。FlashRecovery [33]在拥有 4800 台设备的集群上实现了单步恢复，凸显了这一问题在工程上的紧迫性。这些报告证实了我们的理论结果所适用的实证背景。

静默数据损坏。 Hochschild 等人 [21]记录了大规模计算基础设施中的静默数据损坏。He 等人 [22] 专门分析了大语言模型训练中的静默数据损坏，并报告称，Google 在 Gemini 训练期间每 1–2 周就观察到一次静默数据损坏事件。静默数据损坏事件发生时不会出现故障信号，进一步动摇了“成功”的检查点必然反映正确状态这一假设。我们的纪元格公式（第 5 节）将静默数据损坏纳入状态来源⊥（含糊不明）。

共识与共同知识。 Herlihy [3] 确立了共识层次结构。Fischer、Lynch 和 Paterson [4] 证明，在存在一个故障进程的异步系统中，共识是不可能实现的。Halpern 和 Moses [5]将同步行动与共同

知识联系起来，并指出在通信不可靠的情况下，共同知识无法实现。Moses 和 Tuttle [6] 将共同知识应用于点火队问题。Coan 等人 [9] 确立了拜占庭故障下点火队问题的界限。Charron-Bost [10]在自稳定系统的背景下重新探讨了点火队问题。Lynch [32] 全面论述了分布式算法，包括共识和一致协议。我们的定理7.3 将 Halpern-Moses 结果应用于固件集群更新这一具体问题。

集群更新协议。Van der Burg [23]对分布式系统的原子升级进行了形式化描述，旨在满足服务更新对客户端呈现为原子操作这一要求。Kubernetes 通过可配置的激增和不可用参数 [24]，实现滚动更新，这实际上是对原子部署属性的一种放宽。我们的贡献在于将这些工程实践与形式化的不可能性图景联系起来。

FITO 计划。本文是一个系列研究的一部分，旨在识别分布式系统中的 FITO 类别错误。Borrill [28] 揭示了 Unix 持久化语义中的 FITO 错误。Borrill [29]则证明了这一错误存在于云同步中。Borrill [30] 对一般性的约束语义修正进行了形式化描述，并证明了消息传递执行与约束满足问题之间的等价性。语义时间之箭系列 [31] 阐述了 FITO 批判在哲学和物理学层面的更广泛基础，并将其与 Lamport 逻辑时钟 [2, 27] 和不定因果序联系起来。本文将分析拓展至人工智能/机器学习基础设施，因为在这一领域，经济和计算方面的利害关系最为重大。

11 结论

我们已经证明，人工智能/机器学习基础设施的两项基本假设——原子检查点和原子固件部署——都是 FITO 范畴错误的实例：即将协议收敛性质与时间谓词混为一谈。

理论结果如下：

1. 检查点原子性是分布式协议的一种轨迹性质，而不是某个时钟时间上的状态谓词（命题 3.3）。2. 在异步崩溃恢复故障下，任何时间边界都无法保证原子性（定理 4.2）。3. 随着持久化域数量增加，原子性在纪元格上成为零测事件（定理 5.6）。4. 混合纪元恢复在优化代数中属于类型违规（定理 6.4）。5. 原子固件部署需要共同知识，而异步性下无法实现这一点（定理7.3）。

建设性推论是，原子性必须通过双边提交、共识介导的转换和验证检查点，明确构建为协议收敛，而不能将其假定为一种时间性质。这是将 FITO 修正为约束语义这一普遍修正，在人工智能/机器学习基础设施领域的具体应用。

参考资料

[1] G. Ryle，《心的概念》，Hutchinson，伦敦，1949。

[2] L. Lamport，“分布式系统中的时间、时钟与事件排序”，《ACM 通讯》，21(7)，1978。[3] M. Herlihy，“无等待同步”，《ACM 编程语言与系统汇刊》，13(1):124–149，1991。[4] M. J. Fischer、N. A. Lynch 和 M. S. Paterson，“存在一个故障进程时分布式共识的不可能性”，《ACM 会刊》，32(2):374–382，1985。[5] J. Y. Halpern 和 Y. Moses，“分布式环境中的知识与共同知识”，《ACM 会刊》，37(3):549–587，1990。[6] Y. Moses 和 M. R. Tuttle，“使用共同知识编程同步行动”，《分布式计算》，3:170–179，1988。[7] C. A. R. Hoare，《通信顺序进程》，Prentice Hall，1985。[8] K. M. Chandy 和 L. Lamport，“分布式快照：确定分布式系统的全局状态”，《ACM 计算机系统汇刊》，3(1):63–75，1985。[9] B. A. Coan、D. Dolev、C. Dwork 和 L. Stockmeyer，“分布式点火队问题”，《SIAM 计算杂志》，18(6):990–1012，1989。[10] B. Charron-Bost，“重新审视点火队问题”，《STACS 论文集》，LIPIcs 第 96 卷，2018。[11] J. Bornholt、E. Torlak、D. Grossman 和 L. Ceze，“文件系统崩溃一致性模型的规范与检查”，《ASPLOS 论文集》，2016。[12] C. Rebello、J. Yang、A. Haeberlen 和 B. T. Loo，“应用程序能否从 fsync 失败中恢复？”，《ACM 存储汇刊》，17(2):1–30，2021。[13]字节跳动，“ByteCheckpoint：面向大型基础模型开发的统一检查点系统”，arXiv:2407.20143，2024。[14] Z. Zhuang 等人，“Gemini：通过内存检查点实现分布式训练的快速故障恢复”，《SOSP 论文集》，2023。[15] A. Maurya 等人，“DataStates-LLM：面向大语言模型的惰性异步检查点”，《HPDC 论文集》，2024。[16] Y. He 等人，“Universal Checkpointing：面向大规模分布式训练的高效灵活检查点”，arXiv:2406.18820，2024。[17] A. Kokolis 等人，“重新审视大规模机器学习研究集群的可靠性”，arXiv:2410.21680，2024。[18] Meta，“Llama 3 模型系列”，arXiv:2407.21783，2024。[19]S. Zhang 等人，“OPT：开放式预训练 Transformer 语言模型”，arXiv:2205.01068，2022。[20] BigScience Workshop，“BLOOM：一个拥有 1760 亿参数的开放获取多语言模型”，arXiv:2211.05100，2023。[21] P. H. Hochschild 等人，“不计数的核心”，《HotOS 论文集》，2021。[22] Q. He 等人，“理解大语言模型训练中的静默数据损坏”，arXiv:2502.12340，2025。

[23]S. van der Burg 和 E. Dolstra，《分布式系统的原子升级》，*HotSWUp* 会议论文集，2008 年。[24] Kubernetes 文档，《执行滚动更新》，https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/，2025 年。[25] Google Cloud 博客，《为大型人工智能训练任务使用多层检查点》，2025 年 6 月。[26] AWS 存储博客，《在 AWS 上为大规模机器学习训练构建可扩展的检查点存储架构》，2025 年 6 月。[27]L. Lamport，《分布式计算》，1989 年。https://lamport.azurewebsites.net/pubs/lamport-chapter.pdf[28] P. Borrill，《Unix 工具与 FITO 范畴错误：崩溃一致性与持久化的协议本质》，arXiv，2026 年。[29]P. Borrill，《iCloud 为何失效：云同步的范畴错误》，arXiv:2602.19433，2026 年。[30] P. Borrill，《没有时间方向的消息传递：约束语义与 FITO 范畴错误》，arXiv，2026 年。

[31] P. Borrill，《语义时间之箭》，第一至第五部分，编写中，2026年。

[32] N. A. Lynch，分布式算法，Morgan Kaufmann，1996年。

[33] Z. Li 等人，《FlashRecovery：大规模训练大语言模型的快速低成本故障恢复》，arXiv:2509.03047，2025年。

确认

本文的思想源于长达25年的故障语义与数据恢复研究，始于2001年9月11日之后开展的紧急恢复行动，当时作者担任 VERITAS Software 副总裁兼首席技术官。此后，相关研究在两个后续项目中继续推进——REPLICUS（分布式复制）和 Earth Computing（确定性网络）——如今则在 DÆDÆLUS 延续。许多研究成果都受到与物理学家、数学家和分布式系统工程师在 *It's About Time!* 播客（ItsAboutTime.club）以及 *Mulligan Stew* 系列节目中的交流启发。

本文在研究过程中借助 Claude（Anthropic）和 ChatGPT（OpenAI）提供协助，将其作为文献综述、结构优化和探讨反方论点的工具。所有思想内容、正式结果和结论均为作者本人所有；所有错误均由作者独自承担责任。