Turn：一种用于智能体计算的语言

Muyukani Kizito [1,21]
Turn Lang [2]Prescott Data

arXiv:2603.08755v1 [cs.PL] 7 Mar 2026

摘要

我们介绍了**Turn**，这是一种用于智能体软件的编译型、基于参与者的编程语言——针对模式推断采用静态类型，在值层面采用动态类型。智能体软件指通过将推理委托给大语言模型（LLM）来进行推理并自主采取行动的程序。现有方法通过框架扩展通用语言，将关键不变量（有界上下文、带类型的推理输出、凭证隔离、持久状态）编码为应用层约定，而非语言保证。

Turn 引入了五种语言级构造来弥合这一间隙。认知类型安全 将大语言模型推理变为类型化原语：编译器根据结构体定义生成 JSON Schema，虚拟机则在绑定之前验证模型输出。置信度运算符 支持由模型确定性把握度控制的控制流。Turn 的基于参与者的进程模型源自 Erlang，为每个智能体提供隔离的上下文窗口、持久化内存和邮箱。基于能力的身份系统 从虚拟机宿主返回不透明且不可伪造的句柄，确保原始凭证不会进入代理记忆。最后，编译时模式吸收（`use schema::<protocol>`）在编译时根据外部规范合成带类型的 API 绑定；`openapi` 适配器已随 `graphql` 一同提供，`fhir`和 `mcp` 则仍在积极开发中。

我们阐述了语言设计、类型规则、模式语义和基于 Rust 的字节码虚拟机，并针对具有代表性的智能体工作负载评估 Turn。Turn 已开源，地址为https://github.com/ekizito96/Turn。

关键词： 编程语言、智能体软件、大语言模型集成、参与者模型、类型安全、基于能力的安全。

1 引言

大语言模型（大语言模型）作为通用推理引擎的出现，催生了一类新的软件：智能体程序 它们观察其环境变量，对非结构化数据进行推理，调用外部工具，并根据其结论采取行动。与所有行为都是确定性的传统软件不同，智能体程序将关键决策委托给输出在结构上不可预测的随机模型。

用现有语言和框架构建这些系统，会暴露出根本性的阻抗失配。智能体计算的核心抽象——有界上下文窗口、带类型的推断结果、持久执行状态、进程隔离内存以及凭证安全——在 Python、TypeScript 或 Rust 等语言中没有直接表示。开发者将它们编码为库约定，从而导致五种反复出现的失效模式：

1. **无界上下文**。 上下文被管理为普通的消息列表。当它超过模型的令牌限制时，会静默溢出或从前面截断，导致模型丢失关键信息。

2. **无类型推理输出**。 模型返回一个字符串。程序将其解析为 JSON，但无法保证模式匹配。模式违规表现为运行时键错误，且与产生它们的推理没有关联。

3. **碎片化状态**。智能体状态分散在变量、数据库、应用程序接口响应和框架内部机制中。不存在能够完整表示智能体所知或所做之事的单一可序列化表示。

4. **没有持久化执行**。长时间运行的智能体会随其进程终止而消亡。语言级机制无法在重启后对执行进行检查点保存并恢复。

5.**凭证泄露**。 API 密钥和 OAuth 令牌以字符串形式存储，可供大语言模型的工具调用接口访问；在任何模型都能调用输入/输出的系统中，这都会形成数据泄露途径。

这些失效模式并非理论上的假设。在开发 JarvisCore 及 Prescott Data 的其他生产级智能体系统时，我们在实践中遇到了这些问题。实证分析 [15, 16] 证实，要在大规模应用中确保可靠性，就必须在语言层面强制执行那些框架仅作为惯例处理的不变量。Turn 正是为此而生：它是一种将这些约束分别设为一等原语，而非库模式的语言。

论文结构。 §2综述相关工作；已熟悉参与者模型、能力安全和大语言模型输出结构化的读者，可直接阅读 §3。§4–§8 中的贡献内容彼此独立，读者读完 §3后可按任意顺序阅读。

1.1 贡献

本文作出以下贡献：

1. **认知类型安全** (§4)：我们引入 `infer Struct { prompt }`，这是一种类型化推理原语，编译器会在编译时根据 struct 定义生成 JSON Schema，并由虚拟机在绑定前验证模型的响应。我们证明，如果一个 `infer`表达式执行完毕且未发生错误，那么绑定后的值在结构上符合声明的类型。

2. **概率控制流**(§5)：我们引入 `confidence` 运算符，它是一种一等语言构造，可从推理结果中提取模型报告的置信度，并支持根据随机输出进行确定性分支。

3. **智能体进程模型**(§6)：我们将 Erlang 的参与者模型应用于智能体领域。每个智能体进程并非使用扁平的消息列表，而是采用三部分结构化上下文架构，以确保高召回率不变量、O(1) 持久化智能体记忆、异步邮箱，以及通过精确的持久化执行和 `suspend/resume` 检查点。

4. **基于能力的身份** (§7)：我们引入 `grant identity`，这是一种从虚拟机宿主返回不透明且不可伪造的 `Identity`能力句柄。该句柄无法强制转换为字符串、序列化或回显，即使大语言模型控制执行流程，也能防止凭证泄露。

5.**编译时模式吸收**(§8)：我们引入 `use schema::<proto-col>("url")`，这是一种协议分发编译时宏，可获取并解析 API 规范，进而合成原生 Turn 结构体和闭包。我们已发布 `openapi` 适配器； `graphql`、 `fhir`和 `mcp`适配器正在积极开发中。

回合概览

原语	故障模式
`infer Struct {..}`	无类型 inference
`confidence v`	未经检查的确定性
`context, remember, suspend`	碎片化状态
`grant identity`	凭证泄露age
`use schema::..("url")`	API gap 契约

运行示例。本文始终以投资委员会程序为例（可在 `impl/examples/`代码仓库[1]中找到）。在 89 行代码中，三个专业参与者并发评估一笔 NVDA 股票头寸：委员会主席使用限定范围的网络凭证获取实时市场数据，基本面分析师根据模型置信度决定是否采纳其结论（`confidence <`

[1] https://github.com/ekizito96/Turn

`0.7` 会触发确定性回退），风险官则负责执行风险敞口上限。委员会主席收集双方的邮箱消息，综合撰写最终的委员会备忘录，内容为`infer`，并使用受限范围文件系统凭证将其持久保存。§4–§8中的所有五项贡献均已演示；每节都会引用相关构造。

2 背景与相关工作

2.1 智能体框架

智能体软件的主流实现方式是基于库的：LangChain [8]在 Python 中串联大语言模型调用与工具调用；Auto- GPT [9] 实现了目标导向的分解循环；CrewAI [10]组织基于角色的智能体团队；Semantic Kernel [11] 为 .NET 提供插件架构；MetaGPT [22] 为软件流水线中的智能体分配不同角色。在交互模式层面，Yao 等人[20] 提出了 ReAct——交错进行的推理轨迹与工具操作——它已成为标准的智能体循环模板；Park 等人 [21]则展示了持久记忆多智能体仿真。

这些方法有一个根本局限：它们是库，而非语言。它们试图强制执行的不变量——上下文边界、输出模式、工具权限——只是运行时约定，主机语言无法对其进行静态验证。Kizito [16] 提供了经验证据，表明自主子智能体循环的失败率受随机累积（$0.95^{20} = 0.36$）所支配；可靠性需要的是类型化边界与确定性修复路由，而非更好的提示。

2.2 编程语言理论

Turn 借鉴了编程语言设计领域的几种成熟传统。

参与者模型。 参与者模型由 Hewitt 等人提出 [1] 并由 Agha[18], 加以形式化，为 Turn 的并发机制提供了理论基础。Erlang [2] 证明，采用“任其崩溃”容错机制的基于参与者的并发模型可扩展至数百万个进程。Turn 将这一模型应用于智能体：每个进程不仅拥有环境变量和邮箱，还拥有上下文窗口和持久化内存。

基于能力的安全。 Dennis 和 Van Horn [3] 确立了基于能力的访问控制，将其作为访问控制列表的一种替代方案。Miller [19] 在 E 编程语言中将这一理念拓展为对象能力范式，展示了不可伪造引用如何以组合方式保障安全属性。Turn 的 `Identity`类型实现了这一对象能力模型：能力是由运行时授予的不可伪造令牌，绝不会是存储在程序内存中的字符串。

会话类型与结构化通信。 通过类型系统强制执行通信协议的理念已得到广泛探索 [4, 5]。Turn 的认知类型安全将类似原则应用于程序与大语言模型之间的“通信协议”：struct 模式充当会话类型，用以约束模型的响应。

正交持久性。 透明持久性的概念，即程序员无需编写显式的保存/加载代码，曾在 PS-algol[6]和 Napier88 [7]中得到探索。Turn 的 `suspend` 原语以及`remember`/`recall`存储提供了进程级正交持久性。

2.3 大语言模型中的注意力与上下文

Liu 等人 [23]通过实验证明了“迷失在中间”现象：放置在长上下文中间的信息，其召回概率约为 50%；相比之下，位于开头和结尾的信息召回概率分别为 90% 和 85%。这种 U 形注意力曲线对语言设计有直接影响。Turn 的进程级上下文隔离（§6）以及 `context.append`/`context.system`原语，使程序员能够明确控制上下文窗口中的高召回区域所承载的内容，而不是不断累积一个无界的消息列表。Wei 等人 [24] 表明

思维链提示能够诱发多步推理，但 Kizito [15] 证明了，随着上下文熵增加，这种推理能力会以可预测的方式下降，从而促使 Turn 在语言层面实施主动上下文管理。

2.4 智能体身份与安全

随着自主代理被部署到可访问敏感应用程序接口的生产环境中，其安全性日益受到关注。South 等人 [25] 综述了智能体 AI 身份管理的新兴领域，并指出现有授权框架是为交互式人类用户或静态服务器设计的。Sangalo [17] 提出了 Nexus 协议，将其作为用于智能体身份与连接编排的标准化开放协议，通过中央权威机构将认证机制与智能体逻辑解耦。Turn 在不同层面解决了一个互补的问题：即使身份基础设施正确无误，大语言模型本身也绝不能接触原始凭证。Turn 的 `grant identity` 原语（§7）确保令牌在语言层面上是不透明且不可伪造的，无论底层身份基础设施如何实现。

2.5 结构化大语言模型输出

OpenAI 的结构化输出API [12] 和 Anthropic 的工具使用协议 [13] 允许请求模型生成符合 JSON 格式的响应。这些都是通过 SDK 调用访问的 HTTP 层功能。Turn 将这种能力提升到语言本身：结构体定义就是模式，编译器负责生成，虚拟机负责强制执行。程序员无需直接编写或读取 JSON Schema。

3 语言设计

Turn 是一种提前编译、动态类型、面向表达式的语言。源程序会被编译为字节码，并由基于栈的虚拟机执行。其核心抽象是 *turn*：一种命名的智能体行为单元，类似于函数正文，但可隐式访问大语言模型上下文窗口、持久化内存和 Actor邮箱。

3.1 核心语法

Turn 程序是在一个隐式顶层 Turn 中执行的一系列语句。运行示例中综合使用了六种核心原语——`struct`（类型定义）、`context` （大语言模型输入管理）、 `infer`（类型化推断）、 `confidence` （概率分支）、`call` （工具调用）和`remember` （持久化内存）。

3.2 值与类型

Turn 支持以下值类型：

Type	描述
`Num`	IEEE 754 双精度浮点数
`Str`	UTF-8 字符串
`Bool`	`true`/ `false`
`List`	异构序列
`Map`	字符串键字典
`Struct`	具名乘积类型
`Pid`	进程标识符（不透明）
`Vec`	稠密数值向量
`Identity`	不透明能力句柄
`Null`	值缺失

表 1：Turn 的值类型。

Turn 在值层面采用动态类型，但使用结构体定义来生成用于推理验证的编译时模式，遵循动态类型与静态类型之间谱系上的务实方法 [28]。

设计理念：有针对性的严格性。
对混合系统的一种常见批评是：在对非确定性的大语言模型输出强制执行严格类型的同时，却让确定性的内部变量保持动态类型，这种做法颇具讽刺意味。Turn 有意采用这种有针对性的严格性。智能体胶水代码具有很强的探索性，受输入输出限制严重，在结构上也类似脚本语言（例如 Python 、 Ruby ）；动态变量则能最大限度地提升迭代速度。严格的静态类型恰好应用于熵最大（大语言模型推理）和风险最大（API 契约）的边界。 `Identity`类型也受到类似限制：它不能强制转换为 `Str`，不能序列化为 JSON，

也不能传递给输出工具，从而确保只在安全性要求之处严格执行类型约束。

4 认知类型安全

Turn 的核心创新在于将大语言模型推理视为一种类型化语言原语，而非无类型的函数调用。

4.1 `infer`原语

定义 1 (推理表达式)。 推理表达式的形式如下：

$$\textit{let}\ x = \textit{infer}\ T\ \{\ e\ \}$$

其中， T 是一个结构体类型， e 是一个求值结果为 $\mathtt{Str}$的提示表达式。其语义如下：

1. 编译器在编译时根据 T 的结构体定义生成 *JSON Schema* $S(T)$ 。

2. 在运行时，虚拟机暂停进程，构造一个推理请求，其中包含提示 e、智能体的上下文窗口和 $S(T)$，然后将其委托给配置的 *Wasm* 推理驱动程序。

3. 驱动返回 *JSON* 响应。虚拟机根据 $S(T)$验证该响应。

4. 如果验证成功，虚拟机会将解析后的值绑定到 x ，其类型为 T ，然后恢复执行。

5. 如果验证失败，虚拟机会使用验证错误重新提示模型（最多重试 k次，默认为 $k = 3$）。如果所有重试均失败，虚拟机将引发可捕获错误。

图 1 展示了从 struct 定义到类型化值的完整流程。

定理 1 — 结构符合性

如果推理表达式 `let` x `= infer`T `{` e `}` 无错误地完成，则对于在 struct T 中声明的每个字段 $f_i : \tau_i$，访问 $x.f_i$ 都会产生一个类型为 τ_i 的值。

证明概要。 我们通过进展性和保持性来确立可靠性，并将 Wasm 驱动程序和 JSON Schema 验证器视为动态验证预言机 $\mathcal{O}$。根据 T-Infer 的前提， T 是一个格式良好的 struct，且编译步骤$T \mapsto \mathcal{S}(T)$是全函数。运行时，进程会 suspend，直至预言机给出响应 j。如果 $\mathcal{O}(j, \mathcal{S}(T)) = \text{ok}$，那么$\mathcal{S}(T)$与 T 之间的结构对应关系保证 j 会为每个字段 f_i 填入一个类型为 τ_i的值。因此，绑定 $x \mapsto \text{parse}(j, T)$能够保持环境类型 $\Gamma \uplus \{x : T\}$。如果$\mathcal{O}(j, \mathcal{S}(T)) = \text{err}$，归约步骤将进入有限重试，或引发异常（Infer-Fail）；通过阻止格式错误的赋值，维持安全性不变量。□

4.2 类型规则

T-Infer 规则体现了认知类型安全兼具静态与动态特征的性质：

$$\frac{\Gamma \vdash T : \mathsf{Struct}\ \ \mathcal{S}(T) = s\ \ \Gamma \vdash e : \mathtt{Str}}{\Gamma \vdash (\mathtt{let}\ x = \mathtt{infer}\ T\ \{e\}) : T}\ (\text{T-INFER})$$

第一个前提在编译时检查T 是否为已声明的 struct。前提 $\mathcal{S}(T) = s$断言模式生成对于结构体类型是全定义的——它总能成功生成具体的 JSON Schema。第三个前提检查提示表达式的类型是否为 `Str`。结论将 x 的类型确定为 T，这由定理4.1保证正确：虚拟机在绑定之前会根据 s验证模型响应，因此任何成功绑定的 x 都满足 T的字段级类型约束。

4.3 模式生成

模式生成函数$\mathcal{S}$ 将 Turn 结构体定义映射为 JSON Schema 对象

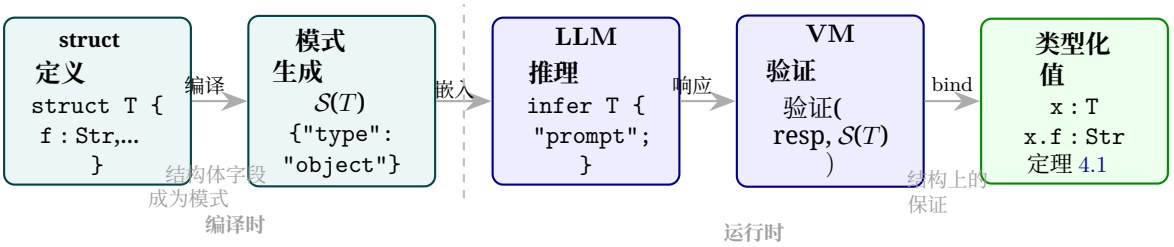


图 1：认知类型安全流水线。结构体定义编译为 JSON Schema $\mathcal{S}(T)$；虚拟机将其嵌入推理请求中，验证响应，并绑定一个类型化值 T(定理 4.1)。

(公式 5)。每个字段名都会成为必需的属性键，而每种 Turn 类型都会通过$\mathcal{T}$映射为 JSON Schema 原语

$$\mathcal{T}(\texttt{Num}) = \{\texttt{"type":"number"}\} \quad (1)$$
$$\mathcal{T}(\texttt{Str}) = \{\texttt{"type":"string"}\} \quad (2)$$
$$\mathcal{T}(\texttt{Bool}) = \{\texttt{"type":"boolean"}\} \quad (3)$$
$$\mathcal{T}(\texttt{List}) = \{\texttt{"type":"array"}\} \quad (4)$$

此模式嵌入在 `Infer` 指令位置的字节码中，确保构造模式不会产生运行时开销。

$$\mathcal{S}(T\{f_i{:}\tau_i\}) = \left\{ \begin{array}{l} \texttt{"type":"object"} \\ \texttt{"properties":}\{f_i{:}\mathcal{T}(\tau_i)\} \\ \texttt{"required":}[f_1,\ldots,f_n] \end{array} \right\} \quad (5)$$

5 概率控制流

大语言模型推理本质上具有随机性。即使认知类型安全能够保证结构上的符合性，响应的语义质量仍可能有所不同。Turn 通过 `confidence` 运算符来解决这一问题。

定义2(置信度运算符)。 对于由 *infer* 表达式生成的任意值 v，表达式 *confidence* v 会求值为一个浮点标量 $c \in [0,1]$，表示推理提供方报告的置信程度。

置信度分数由 Wasm 推理驱动程序从模型的响应元数据中提取（例如，对于 OpenAI，使用 `logprobs: true` 的最高令牌对数概率；对于支持置信度字段的提供商，则使用显式的置信度字段）。对于既不提供对数概率也不提供置信度字段的提供商，驱动程序会返回一个保守的默认值 0.5——即 $[0,1]$ 范围的中性中点——从而确保 `confidence v` 始终求值得到 $[0,1]$ 中的标量。需要真实置信度信号的程序应选择提供对数概率的提供商；只要求结构正确性的程序则可以不依赖置信度运算符，而依靠认知类型安全。虚拟机会将该分数与推理出的值一同存储，作为 `Uncertain<T>` 包装器；它不会影响字段访问，但可通过 `confidence` 查询。运行示例展示了这种模式：如果分析师的论点置信度低于 0.7，程序就会用确定性的“FAIR”判定替代它。回退分支在虚拟机中原生执行，网络延迟为零，并生成一个在结构上符合相同类型的值。这使随机路径与确定性路径能够在同一类型纪律下顺畅组合的编程模型成为可能。

6 基于 Actor 的智能体进程

Turn 的并发模型将 Erlang 的 Actor 模型 [2] 适配到智能体领域。

定义3 （智能体进程）。 *Turn* 进程是一个 *5* 元组 $(E, C, M, B,$程序计数器$)$，其中：

- E 是词法环境（变量绑定），
- C 是上下文窗口（有序字符串序列），
- M 是持久化内存存储（键值映射），
- B 是邮箱（*FIFO*消息队列），
- 程序计数器是程序计数器。

进程通过 spawn、spawn_link 或 spawn_each 生成。每个进程都会获得一个全新的（E，C，M，B，程序计数器）元组；没有任何组件是共享的。

6.1 消息传递

进程间通信使用异步消息传递：

- send pid, value将 value 加入进程 pid的邮箱中。非阻塞。
- receive 从当前进程的邮箱中取出下一条消息。阻塞（让出执行权给调度器），直到消息可用。
- self返回当前进程的 Pid。

6.2 故障传播

spawn_link在进程之间建立双向链接。当一个链接进程终止时，虚拟机会生成一个包含退出原因的 ProcessExit信号，并将其发送到所有链接伙伴的邮箱。这使得 Erlang 风格的监督树成为可能：在运行示例中，委员会主席生成两个与其链接的分析员；如果其中任何一个崩溃，ProcessExit 信号就会发送到委员会主席的邮箱，从而实现确定性的故障恢复。

6.3 上下文隔离与结构化内存

Turn 参与者模型的一个关键性质是上下文隔离：每个进程都维护自己的上下文窗口。当智能体生成子进程时，子进程会以空上下文启动。这可以防止智能体之间发生上下文污染；在基于框架的多智能体系统中，这是一种常见的故障模式，因为共享的消息列表会不加区分地累积所有智能体的上下文。

性质 1 (上下文隔离)。 对于任意两个进程 P_i 和 P_j，其中 $i \neq j$，对 C_i 的修改不会影响 C_j。

三层上下文架构。 Turn 不会将上下文存储为扁平的 FIFO 列表。每个进程上下文 C 都是三层*StructuredContext*：

- **P0（系统）**。 A Vec 由 context.system(...)设置的系统级指令始终 首先呈现在扁平上下文窗口中，利用首因 注意力位置（≈90% 召回， [23]）。
- **P1（工作层）**。由context.append(...)追加的近期上下文条目组成，上限为 $W = 100$ 条。始终最后渲染，利用近因位置（≈85% 召回率， [23]）。
- **P2（情景层）**。P1 达到容量上限时，最旧的工作条目会降级到 P2，而不是被丢弃；P2 上限为 $2W = 200$ 条。渲染位置介于 P0 和 P1 之间，处于召回率较低但不为零的中间区域。

每次推理调用都会由 to_flat_vec() 强制采用 P0 →P2 →P1 的扁平渲染顺序。这一设计确保 Turn 智能体的系统角色（P0）和最新上下文（P1）始终占据 Liu 等人 [23], 指出的两个注意力特权位置，同时将较早的上下文保存在情景层中，而不是在不知不觉中截断。有关淘汰不变量和渲染顺序的实证验证，请参见实验 E3。

6.4 散射/收集： spawn_each

Turn 将 spawn_each 作为原生的散射/收集原语，用于并发处理列表。每个元素都会被委托给一个独立的微型参与者，拥有各自隔离的上下文、内存和邮箱。表达式 spawn_each(list, turn(x: T) { ...})在语义上等同于为每个元素生成一个参与者，并等待所有参与者完成；但它是单一的语言原语，而非应用层模式。这种方式以委托取代命令式循环，使编程模型与智能体范式保持一致，因为迭代天然适合并行处理——参见投资委员会示例，其中三位分析师在 spawn_link监督者的管理下并发运行。

7 基于能力的身份

在智能体系统中，模型通过工具调用决策控制执行流程。

如果 API 凭证以字符串形式存储在程序的环境变量中，模型可能会幻觉出一条 `call("echo", api_key)`指令并泄露机密信息。Turn 通过基于能力的身份系统来防止这种情况。

定义 4（身份能力）。表达式 *grant identity::σ("name")*向虚拟机宿主请求一个不透明能力句柄，其中 σ 是能力类别（例如*oauth*、*network*、*filesystem*），*name* 用于标识提供者。虚拟机返回类型为 *Identity* 的值，该值：

1. 不能强制转换为 *Str* 或任何其他类型。

2. 不能序列化为 *JSON*。

3. 不能传递给输出工具（例如*echo*）。*4.* 在两个层面强制执行：标准库在纯 *Turn* 源代码中验证该能力，内核陷阱在 *Rust* 主机中对其进行验证。

当 `Identity` 句柄触发内核陷阱（例如 `__sys_http_get`）时，虚拟机宿主会依据约定 `TURN_IDENTITY_{NAME}_TOKEN`从进程环境中查找真实凭证，并自动注入 `Authorization`标头。原始令牌从不会进入 Turn 内存。正在运行的示例演示了网络和文件系统身份句柄的使用。

8 编译时模式吸收

Turn 的 `use schema::<protocol>("url")`构造是一个协议分派、具备网络感知能力的编译器宏。语法分析器接受任意标识符作为协议，模式编译器会分派到相应的适配器。 `openapi`适配器已完全实现；`graphql`、 `fhir`和 `mcp` 适配器共用同一条流水线，目前正在积极开发中。

8.1 宏展开

模式编译器作为预分析 AST 转换遍运行：

1. 语法分析器生成一个携带协议和 URL 的 `Expr::UseSchema` 抽象语法树节点。

2. 模式编译器根据`protocol` 字段进行分派，并在编译时从 URL 获取应用程序接口规范。

3. 对于每个端点或类型定义，它会生成：(a) 与响应模式对应的 Turn struct；(b) 用于构造相应请求的闭包。

4. `UseSchema` 节点被替换为一个立即调用函数表达式（IIFE），其中包含生成的结构体和闭包。

5. 转换后的抽象语法树进入语义分析和编译阶段。 `UseSchema` 节点永远不会到达字节码生成器。

这样便可进行静态验证，确保 `infer` output（由合成结构体确定类型）在结构上符合 API 预期输入格式，从而弥合大语言模型输出与外部 API 调用之间的类型鸿沟。

端到端链路。程序使用 `use schema::openapi(...)` 和 Stripe API URL 编写代码。模式编译器会获取规范，合成一个 `Customer` 结构体和一个带类型的闭 `Identity` 包。运行时， `Identity` 句柄会限定 HTTP 调用的作用域；程序员无需编写 JSON Schema。后续的 `infer` 表达式会生成一个值，其字段由定理4.1在静态层面保证。完整的演示示例见代码仓库。

9 实现

Turn 使用 Rust 实现为一个独立的静态二进制文件，其中包含编译器、虚拟机、标准库、语言服务器和 gRPC 交换机。

9.1 编译流水线

源代码会经过六阶段流水线，如图 2 所示。模式编译器在分析之前运行，以便合成结构体

来自 `use schema` 的定义对类型检查器可见。

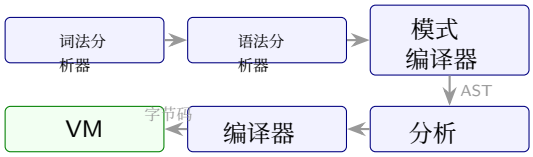


图 2：Turn 编译流水线。宏展开（`use schema`）发生在语义分析之前，以确保类型解析。

9.2 标准库

Turn 的标准库由七个完全用 Turn 源代码编写的模块组成（表2）。 `.tn` 源文件通过 Rust 的 `include_str!` 宏嵌入二进制文件，并在首次使用时编译。每个输入/输出模块会先验证 `Identity` 能力，然后再委托给 `__sys_*` Rust 主机中的内核陷阱，遵循微内核原则：虚拟机提供最小且可审计的原语；标准库以该语言本身提供面向用户的应用程序接口。

模块	API	身份
`std/net`	`get, post`	`network`
`std/fs`	`read, write`	`filesys.`
`std/json`	`parse, stringify`	—
`std/time`	`now, sleep`	—
`std/env`	`get, set`	`environ.`
`std/regex`	`matches, replace`	—
`std/math`	`max, min, abs`	—

表 2：标准库模块。输入/输出模块需要一项 `Identity`能力。

9.3 字节码虚拟机

编译器会生成一段以栈式虚拟机为目标的扁平指令序列。值得注意的领域特定操作码包括：

操作码	语义
`Infer(Type)`	Suspend、委托、validate, bind
`Confidence`	提取确定性标量
`Spawn / SpawnLink`	创建 Actor 进程
`SpawnEach`	分发到微型 Actor
`Send / Receive`	异步邮箱 I/O
`Remember / Recall`	持久化内存操作
`GrantIdentity`	能力 句柄重新任务
`Suspend`	持久检查点
`ContextAppend`	追加到上下文
`ContextSystem`	设置系统提示词

表 3：特定领域的字节码指令。

9.4 WasmInference驱动

大语言模型提供商通过沙箱化 WebAssembly [26] 驱动集成，并编译为 `wasm32-unknown-unknown` 目标。每个驱动导出两个纯函数：

- `transform_request`：将 Turn 推理请求映射为 HTTP 配置（URL、带有 `$env:` 凭证模板的标头、正文）。
- `transform_response`：将原始 HTTP 响应映射为 Turn 结果。

驱动没有宿主导入：无法访问网络、文件系统或环境变量。虚拟机宿主会解析 `$env:` 模板、执行 HTTP 调用，并将响应传回。这种双遍架构确保 Wasm 驱动程序是纯转换函数，不具备窃取凭证的能力。

9.5 并发运行时

Turn 进程与 Tokio [14]绿色线程一一对应。进程间通信使用`tokio::sync::mpsc`通道。调度器在一个 turn 内采用非抢占式调度：turn 一旦开始执行，就会运行至完成或挂起，不会受到中断，从而确保给定输入序列的确定性执行。

10 求值

我们通过五个关于 Turn 创新贡献的关键问题来评估它：(1) 凭证是否在结构上保持不透明，并能抵御恶意程序？(2) 置信度语义是否满足概率代数？(3) 三层上下文架构是否落实了旨在解决

“迷失在中间”问题的首因-近因不变量？(4) Turn 的按进程隔离的代理记忆，其规模和隔离表现如何？(5) 通过 `suspend` 实现的持久化执行是否精确——完整的状态保真度能否经受序列化—反序列化往返？所有实验均为 Rust 集成测试，使用 `cargo test --release` 运行；五个套件中有 24 项专门编写的实验测试，另加 81 项既有单元测试——**105/105 项全部通过**。基准测试在 Apple M4（10 核、16 GB、macOS 15）上进行。

10.1 E1：凭证隐匿性

主张。 存储在`TURN IDENTITY STRIPE TOKEN` 中的原始 API 凭证不会进入_ _ _Turn 值堆，也不会暴露给大语言模型的工具调用接口。

机制。表达式 `grant identity::oauth("stripe")` 会使虚拟机发出 `sys_grant` 信号并暂停。运行器使用 `Value::Identity("stripe")`恢复虚拟机——该值仅包含提供方名称，不包含机密。只有在 Wasm 驱动程序返回 HTTP 配置后，内核陷阱才会从环境变量中加载原始令牌；原始令牌绝不会被实例化为 Turn 值。

结果。 四项测试分别验证了该保证的四个方面（全部通过）：

- **E1-A** （显示）：`Value::Identity("stripe")` 呈现为 `<identitystripe>`——原始令牌从不出现。
- **E1-B** （堆）：绑定值为`Identity("stripe")`；堆中只有提供方名称，绝不包含秘密。
- **E1-C** （确定性）：身份值从不包装在 `Uncertain<T>`中；其确定性由构造保证。
- **E1-D** （独立性）：针对同一提供方类别的两次授权会生成彼此独立的句柄，且没有共享状态。

10.2 E2：置信度传播语义

主张。 The `confidence` 算子满足以下条件：(i) 某些值返回 1.0；(ii) `Uncertain(v, p)` 返回 p；(iii) 算术通过乘法规则传播不确定性 $p_1 \cdot p_2$；(iv) 布尔与使用扎德最小值 $\min(p_1, p_2)$。

结果。 六项测试验证了该代数（`e2_1`–`e2_6`，全部通过）：

- **E2-1**： `confidence(42)` $= 1.0$（确定的值）。
- **E2-2**：`confidence(Uncertain(42,0.73))` $= 0.73$（返回已存储的分数）。
- **E2-3**： `confidence(Uncertain(10, 0.8) + Uncertain(5, 0.5))` $= 0.40$（乘法规则：0.8×0.5）。
- **E2-4**：`confidence(x and y)` 其中$x \sim 0.9$, $y \sim 0.5$ 得出 0.50（扎德最小值：$\min(0.9, 0.5)$）。
- **E2-5**：当 $p = 0.45$时确定性触发。
- **E2-6**：当 $p = 0.92 \geq 0.70$时，高置信度路径执行。

10.3 E3：结构化上下文架构

主张。 Turn 的 P0/P1/P2 三分上下文确保了首因—近因呈现不变量：系统提示始终占据首因位置，最近追加的项目始终占据近因位置，无论上下文容量如何。工作内存溢出时会降级为情景记忆，而不是被丢弃。

背景。 Liu 等人 [23] 表明，回忆表现呈 U 形曲线： $\approx$90% 的内容处于首因位置， $\approx$50% 处于中间位置， $\approx$85% 处于近因位置。每个基于框架的系统都会将消息追加到一个扁平的先进先出列表中；随着列表增长，模型从一端截断内容时，就会失去这一优势。Turn 的 `StructuredContext` 从结构上解决了这个问题，而非依赖约定。

机制。 `to_flat_vec()`始终按 `p0_system` $\rightarrow$ `p2_episodic` $\rightarrow$`p1_working`的顺序输出。P1 是一个有界的 `VecDeque` （$W = 100$）；满载时，最旧的工作条目会移至 P2（容量上限为 $2W = 200$）。因此，无论上下文总大小如何，首因位置和近因位置都分别由 P0 和 P1 的末尾占据。

结果。 五项测试验证了不变量（均在 Apple M4 上通过）：

- **E3-1**：P0 系统提示词始终位于扁平上下文向量的索引 0 处。
- **E3-2**：最近追加的 P1 项始终是最后一个元素。
- **E3-3**：到第 101 项时，P1 的大小仍为 100；恰好有一项被移至 P2（降级，而非丢弃）。
- **E3-4**：已确认扁平化呈现为 P0（索引 0） → P2（索引 1） → P1（索引 2–101）。
- **E3-5**：每个生成的进程都以空上下文启动（隔离不变量）。

10.4 E4：智能体记忆隔离与规模

主张。 Turn 的每进程 `remember/recall`内存可在任意规模下提供 O(1) 平均访问，并且进程之间完全隔离：无论并发智能体的数量或共享键名如何，都不可能发生跨进程污染。

机制。 每个 `Runtime` 都包含私有的 `HashMap<String, Value>` 通过 `remember(key, val)` 和 `recall(key)`公开。Rust 所有权模型确保没有两个进程共享对同一`Runtime`的引用；借用检查器使别名引用成为不可能。内存属于进程本地且仅限进程内部；跨会话持久化需要`suspend`（参见 E5）。

结果。 表 4报告了内存规模从 1 千到 100 千条目时的写入和读取延迟（全部 5 项测试均通过）：

K	写入 纳秒/操作	Read 纳秒/操作	字节/条目	Note
1 K	519	574	76.8	基准
10 K	1021	867	—	10×
100 千	594	430	—	100×

表4：智能体内存性能。随着规模跨越三个数量级，延迟保持平稳，证实了哈希映射的 O(1) 行为。内存占用：76.8 字节/条目。Apple M4，发布版构建。

每操作延迟不会随 K 增长，证实了 O(1) 的平均情况行为。每个条目序列化后占用 ≈77 字节（与挂起快照相关）。E4-4 验证了跨进程隔离：代理 A 和代理 B 在同一个键下存储时，会读取到彼此独立的值；代理 B 无法读取仅由代理 A 写入的键。

10.5 E5：持久化执行

主张。 Turn 进程可以在任意 `suspend` 点进行检查点保存，并精确保留状态：整个 `VmState` （程序计数器、调用栈、堆、上下文窗口、代理记忆、邮箱）都会序列化为 JSON，并在恢复时不丢失任何信息。

机制。 `VmState` 已完整添加 `#[derive(Serialize, Deserialize)]`注释。 `FileStore` 会将 `{id}.json`写入 `.turn_store/`目录； `Vm::resume_with_result` 会重建精确的执行续体。无需应用级检查点代码；程序员只需编写`suspend`；其余工作由运行器处理。

结果。 表5 报告了三种内存深度下的序列化状态大小和往返延迟（全部 4 项测试均通过，包括完整的状态保真度验证）：

条目	Size	往返时间 (µs)	保真度
10	0.9 KB	211	精确
500	19.8 KB	1,340	精确
5 000	203.4 KB	6,828	精确

表 5：持久化执行：序列化`VmState` 大小与往返延迟。“精确”=表示往返后，进程标识符、程序计数器、栈、内存和上下文均完全一致。Apple M4。

一个包含 5,000 个条目的智能体可在 7 毫秒内完成检查点创建，并且在磁盘上仅占用 203KB——相对于任何 I/O 密集型工作负载而言，这一成本都微不足道。对于包含少于 ≈120 个内存条目的智能体，往返耗时不到 1 毫秒。E5-4 验证了进程标识符、指令指针、栈、全部 50 个受测内存条目以及 P0 系统提示词在往返过程中均未损坏。

虚拟机运行时开销。进程创建平均耗时 2.21 µs，消息发送耗时 2.34 µs，JSON

模式验证耗时 0.587 µs——这三项的耗时都比大语言模型网络往返的最小值（100 ms–5 s）低三到四个数量级。与其保护的推理成本相比，Turn 的安全原语不会带来可观察到的开销。

10.6 架构复杂性与框架

虽然 Turn 将实现投资委员会所需的原始代码行数（LOC）从 LangChain 中估计的 350+ 行减少到 89 行，但更显著的减少在于状态空间复杂性。在 Python框架中，要实现容错并发、持久化智能体记忆和结构化输出，需要一个外部 Redis 实例来保存状态、一个消息队列（例如 Celery）来管理工作进程，以及围绕 JSON解析器的显式重试循环。在 Turn 中，这些都被收进语言运行时：`spawn_link` 在无需外部队列的情况下处理并发与故障传播，`suspend`在无需数据库样板代码的情况下将持久化正交化，`infer` 则吸收了重试循环。Turn 将分布式系统架构简化为单进程参与者拓扑。

10.7 安全属性汇总

表 6将每项安全属性与其执行机制层和验证实验交叉对应。

属性	执行机制		失败 mode	Exp.
struct. conf.	模式 VM	+	无类型 infer-ence	定理 1
原始递归	P0/P1/P2		迷失于中间	E3
内存隔离	每进程		跨代理 leak	E4
持久化执行	哈希映射 VmState serde		状态丢失	E5
凭证操作	身份 host	+	凭证 ex-fil.	E1
置信度	不确定⟨T⟩		未检查证书。	E2

表6：安全属性、执行机制、所防止的失效模式以及验证。

11 讨论

11.1 局限性

Turn 是一种用于智能体编排的领域特定语言，而非通用系统语言。它有意省略了底层内存管理和 GPU计算，并通过 `call` 将繁重工作负载委托给 WebAssembly 组件或微服务。不过，Turn 的架构支持扩展：其 `spawn_link`监督树可支持大规模多智能体平台，例如我们内部部署的 JarvisCore 自主系统。在这些平台中，复杂性主要体现在状态持久化和能力路由，而非原始计算吞吐量。

虽然 `infer` 保证了结构形状，但标准变量绑定目前仍采用动态类型，这意味着非推理相关的类型错误会在运行时被捕获。未来的迭代可能会引入可选的静态类型。此外，尽管开发者可以使用 Turn 原生的 `try/catch`构造来构建健壮的恢复循环，但将大语言模型自我反思 [27]直接集成到语言语法中仍是一个有待探索的方向。

最后，`confidence` operator 依赖底层推理提供商公开对数概率。对于不透明模型，虚拟机必须回退到默认分数，从而限制概率路由的实用性。

11.2 未来工作

Turn 的当前实现确立了面向语言的智能体开发所需的核心原语。然而，要实现生产级生态系统的完整愿景，还需要在多个方向上持续开展大量工作：

- **扩展模式适配器**：扩展 `use schema::<protocol>`，使其支持 `graphql`、`fhir`、`mcp` 和 `odata`，覆盖企业和医疗保健应用程序接口的全部范围。
- **分布式Actor调度**：将受 Erlang 启发的并发模型提升至支持通过现有 gRPC 交换机在多个节点间进行分布式 Actor 调度。
- **工具链与生态系统**：开发健全的语言服务器协议（LSP）实现，以提供高级集成开发环境支持，并开发原生包管理器，用于分发 Turn 模块和模式适配器。
- **形式化验证**：将认知类型安全的形式语义扩展至对编译器模式生成和虚拟机验证流水线的完整机械验证。
- **多模态原语**：扩展 `infer` 原语，使其能够原生接收图像、音频和视频输入，并探索更广泛的概率推理结构。

合作邀请。 Turn 是一项雄心勃勃的事业，处于编程语言理论与人工智能的交汇点。我们诚邀研究人员、编译器工程师和开源贡献者参与这些未来方向的合作，共同塑造智能体软件的基础。

12 结论

我们介绍了 Turn，这是一种编程语言，能够明确呈现并强制执行智能体计算的固有结构。Turn 将认知类型安全、概率控制流、Actor 隔离进程、基于能力的身份以及编译时模式吸收作为一等语言构造引入，从而消除了基于框架的智能体开发中的五种根本性失效模式。

关键洞见

智能体是一种进程，而非循环。它拥有有界上下文、持久化内存、邮箱和生命周期。将这些约束编码到语言中，就能让编译器和运行时以机械方式强制执行，从而消除因关键不变量依赖开发者自律而产生的所有此类故障。

Turn 证明，一种具备恰当原语、精简而专注的语言，能够为智能体软件提供比构建于通用宿主环境之上的任何库都更强的保证。

Turn 采用 MIT 许可证开源。编译器、虚拟机、标准库和文档可在 https://github.com/ekizito96/Turn获取。语言规范和交互式示例可在 https://turn-lang.dev获取。

参考文献

[1] C. Hewitt、P. Bishop 和 R. Steiger。《面向人工智能的通用模块化 ACTOR 形式体系》。载于《第三届国际人工智能联合会议（*IJCAI*）论文集》，第 235–245 页，1973 年。

[2] J. Armstrong. 在存在软件错误的情况下构建可靠的分布式系统。博士论文，瑞典皇家理工学院，斯德哥尔摩，瑞典，2003。

[3]J. B. Dennis 与 E. C. Van Horn. 多道程序计算的程序语义。*ACM* 通讯，9(3):143–155，1966。

[4]K. Honda、V. T. Vasconcelos 与 M. Kubo. 面向基于结构化通信编程的语言原语与类型纪律。载于 欧洲程序设计研讨会（*ESOP*），第 122–138 页。施普林格，1998。

[5] S. J. Gay 和 V. T. Vasconcelos。用于异步会话类型的线性类型理论。《函数式编程期刊》，20(1):19–50，2010年。

[6]M. P. Atkinson、P. J. Bailey、K. J. Chisholm、P. W. Cockshott 和 R. Morrison。持久化编程的一种方法。《计算机杂志》，26(4):360–365，1983年。

[7] R. Morrison、A. L. Brown、R. C. H. Connor 和 A. Dearle。Napier88参考手册。技术报告，圣安德鲁斯大学，1999年。

[8] H. Chase。LangChain：通过组合性构建大语言模型应用程序。https://github.com/langchain-ai/langchain，2022年。

[9]T. Richards。AutoGPT：一项自主 GPT-4 实验。https://github.com/Significant-Gravitas/AutoGPT，2023年。

[10]J. Moura. CrewAI：用于编排角色扮演式自主人工智能智能体的框架。https://github.com/crewAIInc/crewAI，2024年。[11] 微软。Semantic Kernel：快速、轻松地将前沿大语言模型技术集成到您的应用程序中。 https://github.com/microsoft/semantic-kernel，2023年。[12]OpenAI。应用程序接口中的结构化输出。https://openai.com/index/introducing-structured-outputs-in-the-api/，2024年。[13]Anthropic。使用 Claude 调用工具。https://docs.anthropic.com/en/docs/build-with-claude/tool-use，2024年。[14] Tokio 贡献者。Tokio：用于编写可靠、异步且精简的应用程序的运行时。 https://tokio.rs，2024年。[15] M. Kizito。人工智能工程物理学：深度科学大师课。Prescott Data，2026年。https://ai-dojo.io/papers/the-physics-of-ai-engineering。[16]M. Kizito。子智能体控制模式：从自主性到认知协同。Prescott Data，2026年。https://ai-dojo.io/papers/sub-agent-control-patterns。

[17]M. Sangalo。《Nexus 协议：为自主代理标准化身份与连接编排》。预印本，Zenodo，2026年。doi:10.5281/zenodo.18315572。

[18] G. Agha。《演员模型：分布式系统中的并发计算模型》。麻省理工学院出版社，马萨诸塞州剑桥，1986年。

[19] M. S. Miller。《稳健组合：迈向访问控制与并发控制的统一方法》。博士论文，约翰斯·霍普金斯大学，2006年。

[20] S. Yao、J. Zhao、D. Yu、N. Du、I. Shafran、K. Narasimhan 和 Y. Cao。ReAct：协同语言模型中的推理与行动。发表于国际学习表征会议（*ICLR*），2023年。

[21]J. S. Park、J. C. O'Brien、C. J. Cai、M. R. Morris、P. Liang 和 M. S. Bernstein。生成式智能体：人类行为的交互式模拟。发表于第 *36* 届 *ACM* 用户界面软件与技术年会论文集（*UIST*），2023年。

[22] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu 和 J. Schmidhuber。MetaGPT：面向多智能体协作框架的元编程。发表于国际学习表征会议（*ICLR*），2024年。[23] N. F. Liu、K. Lin、J. Hewitt、A. Paranjape、M. Bevilacqua、F. Petroni 和 P. Liang。迷失在中间：语言模型如何使用长上下文。计算语言学协会会刊，12：157–173，2024年。[24]J. Wei、X. Wang、D. Schuurmans、M. Bosma、B. Ichter、F. Xia、E. Chi、Q. V. Le 和 D. Zhou。思维链提示可引发大型语言模型的推理。发表于神经信息处理系统进展（*NeurIPS*），2022年。[25] T. South、S. Xu、J. Hardjono 和 A. Pentland。智能体人工智能的身份管理：授权的新前沿。*arXiv* 预印本 *arXiv:2510.25819*，2025年。[26]A. Haas、A. Rossberg、D. L. Schuff、B. L. Titzer、M. Holman、D. Gohman、L. Wagner、A. Zakai 和 J. Bastien。通过 WebAssembly 提升 Web 的运行速度。发表于第*38*届 *ACM SIGPLAN* 编程语言设计与实现会议论文集（*PLDI*），第185–200页，2017年。[27]N. Shinn、F. Cassano、A. Gopinath、K. R. Narasimhan 和 S. Yao。Reflexion：通过语言进行强化学习的语言智能体。发表于神经信息处理系统进展（*NeurIPS*），2023年。

[28] B. C. Pierce。《类型与编程语言》。麻省理工学院出版社，马萨诸塞州剑桥，2002年。

Turn：一种用于智能体计算的语言

补充材料

补充材料为核心贡献提供了形式化细节。第 12 节以补充类型规则（§`infer`、 `confidence`、 `grantidentity`和 `spawn_link`）以及第 4 节的求值（§10）的形式，给出了 Turn 四种新构造的`infer`、 `confidence`、 `grantidentity`和 `spawn_link`的小步操作语义。第 12 节给出了可重现全部五个实验套件的精确命令。完整的 Turn语法、示例详解和教程请见 https://turn-lang.dev和https://github.com/ekizito96/Turn。

扩展的操作语义

我们为 Turn 的四种新构造给出了小步归约规则。归约关系$e,\ \sigma \longrightarrow e',\ \sigma'$将表达式 e在堆 σ 中归约为 e'，并得到更新后的堆 σ'。我们用 $\mathcal{S}(T)$ 表示从 struct T生成的 JSON Schema，用 validate($j,\ s$) 表示 JSON Schema 验证，并用 $env(k)$ 表示对键 k 的进程环境查找。

认知类型安全。 Infer-Step 规则会分发一个携带编译后的模式的推理请求；模型响应 j 在绑定之前进行验证。如果验证失败，Infer-Retry 会重新提示（最多 k 次重试）；当重试耗尽时，Infer-Fail 会引发可捕获错误。

$$\frac{\Gamma \vdash T : \mathsf{Struct} \quad \mathcal{S}(T) = s \quad \mathrm{llm}(e, s) = j \quad \mathrm{validate}(j, s) = \mathsf{ok}}{\texttt{infer}\ T\ \{e\},\ \sigma \ \longrightarrow\ \mathrm{parse}(j, T),\ \sigma} \ (\textsc{Infer-Ok})$$

$$\frac{\mathrm{validate}(j, s) = \mathsf{err}(m) \quad \mathrm{retries} < k}{\texttt{infer}\ T\ \{e\},\ \sigma \ \longrightarrow\ \texttt{infer}\ T\ \{e \mathbin{+\!\!+} m\},\ \sigma} \ (\textsc{Infer-Retry})$$

$$\frac{\mathrm{validate}(j, s) = \mathsf{err}(\cdot) \quad \mathrm{retries} = k}{\texttt{infer}\ T\ \{e\},\ \sigma \ \longrightarrow\ \texttt{throw("InferError")},\ \sigma} \ (\textsc{Infer-Fail})$$

置信度运算符。

$$\frac{\sigma(v) = \texttt{Uncertain}(w, p)}{\texttt{confidence}\ v,\ \sigma \ \longrightarrow\ p,\ \sigma} \ (\textsc{Conf-Uncertain})$$

$$\frac{\sigma(v) \neq \texttt{Uncertain}(\cdot, \cdot)}{\texttt{confidence}\ v,\ \sigma \ \longrightarrow\ 1.0,\ \sigma} \ (\textsc{Conf-Certain})$$

不确定值上的算术运算遵循乘法规则进行传播；布尔值 `and`使用扎德最小值：

$$\frac{\sigma(u) = \texttt{Uncertain}(a, p_1) \quad \sigma(v) = \texttt{Uncertain}(b, p_2)}{u + v,\ \sigma \ \longrightarrow\ \texttt{Uncertain}(a + b,\ p_1 \cdot p_2),\ \sigma} \ (\textsc{Arith-Prop})$$

$$\frac{\sigma(u) = \texttt{Uncertain}(\cdot, p_1) \quad \sigma(v) = \texttt{Uncertain}(\cdot, p_2)}{u\ \texttt{and}\ v,\ \sigma \ \longrightarrow\ \texttt{Uncertain}(\cdot,\ \min(p_1, p_2)),\ \sigma} \ (\textsc{Bool-Zadeh})$$

基于能力的身份。

$$\frac{name \in \Sigma_{providers}}{\texttt{grant identity::}\sigma(name),\ \sigma \ \longrightarrow\ \texttt{Identity}(name),\ \sigma}\ (\textsc{Grant-Ok})$$

该 `Identity`（名称）令牌仅携带提供方名称。原始凭证由内核陷阱在调用时获取，绝不会实例化为 Turn 值：

$$\frac{\sigma(h) = \texttt{Identity}(name) \quad env(\texttt{TURN_IDENTITY_}name\texttt{_TOKEN}) = t \qquad \mathrm{http}(url,\ t) = r}{\texttt{__sys_http_get}(h,\ url),\ \sigma \ \longrightarrow\ r,\ \sigma}\ (\textsc{Sys-Http})$$

请注意， t 仅作为外部 HTTP 预言机的参数出现，从未绑定到 Turn 变量。

参与者生成。

$$\frac{pid \text{ fresh} \quad P_{pid} = (E_\emptyset,\ C_\emptyset,\ M_\emptyset,\ B_\emptyset,\ \overrightarrow{body})}{\texttt{spawn_link turn()}\ \{body\},\ \sigma \ \longrightarrow\ pid,\ \sigma \uplus \{P_{pid}\}}\ (\textsc{Spawn-Link})$$

生成的进程会收到一个新的元组（$E_\emptyset$， $C_\emptyset$， $M_\emptyset$， $B_\emptyset$，程序计数器）。父进程状态的任何组成部分都不会共享，这构成了性质 1（上下文隔离）的结构基础。双向链接意味着，如果子进程异常终止， `ProcessExit`信号就会发送到父进程的邮箱：

$$\frac{P_{child} \text{ terminates with reason } r \quad parent \in \mathrm{links}(child)}{B_{parent} \ \longleftarrow\ \{\texttt{"type"} : \texttt{"exit"},\ \texttt{"reason"} : r\}}\ (\textsc{Exit-Signal})$$

B 实验可复现性

第 10 节中的所有实验均可从公共代码仓库完整复现。以下命令可在任何安装了 Rust 1 的机器上构建编译器并运行全部五个实验套件。75+。

```
# Clone and build (release profile matches the paper's measurements)
git clone https://github.com/ekizito96/Turn.git
cd Turn/impl
cargo build --release

# Run all five experiment suites with output
cargo test --release \
  --test experiment_e1_credential_opacity \
  --test experiment_e2_confidence_propagation \
  --test experiment_e3_structured_context \
  --test experiment_e4_memory_scale \
  --test experiment_e5_durable_execution \
  -- --nocapture
```

预期输出（Apple M4；延迟因硬件而异）：

```
E1-A PASS: Identity displayed as "<identity stripe>" (opaque, no raw credential)
E1-B PASS: grant identity::oauth("stripe") -> Identity("stripe") (token never in heap)
E1-C PASS: Identity is a certain (non-uncertain) value
E1-D PASS: Two grants for same provider are independent identity handles

E2-1 PASS: confidence(42) = 1.0
E2-2 PASS: confidence(Uncertain(42, 0.73)) = 0.73
E2-3 PASS: confidence(Uncertain(10,0.8) + Uncertain(5,0.5)) = 0.40  (product rule)
E2-4 PASS: confidence(x and y) where x~0.9, y~0.5 = 0.50   (Zadeh min)
E2-5 PASS: confidence=0.45 < 0.70 -> deterministic fallback branch executed
E2-6 PASS: confidence=0.92 >= 0.70 -> high-confidence path executed

E3-1 PASS: P0 system prompt is always first in flat context (primacy position)
E3-2 PASS: most recently appended item is last in flat context (recency position)
E3-3 PASS: P1 overflow evicts oldest item to P2 episodic (demote, not drop)
          P1 size=100, P2 size=1
E3-4 PASS: flat rendering order confirmed as P0(0) -> P2(1) -> P1(101)
E3-5 PASS: each Runtime starts with an empty context (isolation invariant)

E4-1 PASS: K=1000 entries   | write 519 ns/op  | read 574 ns/op
E4-2 PASS: K=10000 entries  | write 1021 ns/op | read 867 ns/op
E4-3 PASS: K=100000 entries | write 594 ns/op  | read 430 ns/op  (O(1) confirmed)
E4-4 PASS: Agent memory is fully isolated -- no cross-process contamination
E4-5 PASS: serialized footprint: 76.8 bytes/entry

E5-1 PASS: mem=10 entries   | state size=875 B (0.9 KB)    | round-trip=211 us
E5-2 PASS: mem=500 entries  | state size=20295 B (19.8 KB)  | round-trip=1340 us
E5-3 PASS: mem=5000 entries | state size=208295 B (203.4 KB)| round-trip=6828 us
E5-4 PASS: full state fidelity after serialize -> deserialize round-trip
```

运行完整测试套件（24 项实验 + 81单元测试）：

```
cargo test --release   # 105/105 pass
```