How do Embodied Intelligent Agents Correct Errors in Real Time?

Which Errors?

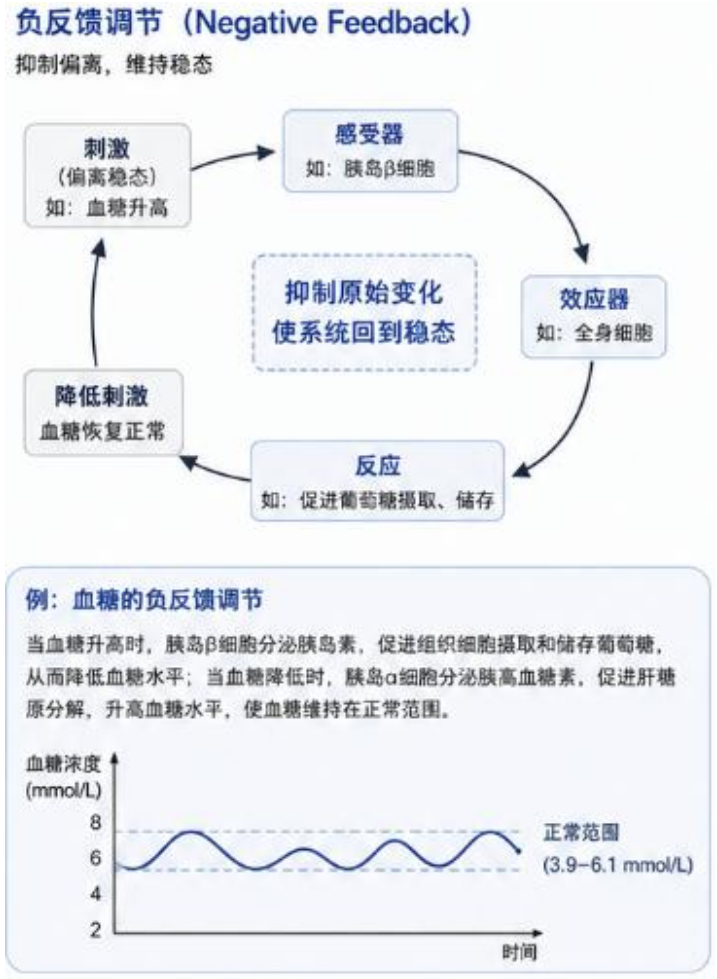
物理安全违规
Physical Safety Violations

- 危险动作序列（越界、超速等）
- 违反硬性约束的动作计划
- 传感器数据异常/噪声

语义安全违规
Semantic Safety Violations

- 危险语义序列
- 资源滥用（超权限操作指令）
- 隐私侵犯（未授权采集/访问）

Feedback in Real Time



缺陷1：反馈回路的延迟与断裂

系统	反馈延迟	反馈完整性
生物神经系统	毫秒级，连续	完整闭环
传统控制器（PID）	微秒级，连续	完整闭环
大模型	数百ms~秒级，**离散**	**开环为主**

缺陷2：不能从错误中即时学习

智能体在推理阶段权重完全冻结，运行时遇到的所有错误都无法直接更新模型，只能等到下一轮离线训练。

大模型内部没有真正独立的低级反射层——所有信息都必须通过同一个Transformer，低层物理误差污染高层语义推理。

缺陷3：缺乏预测性编码

生物大脑本质上是一台预测机器（Friston 自由能原理）：大脑不断生成对下一状态的预测，纠错信号是"预测误差"而非"结果误差"——在后果发生之前就启动修正。

生物小脑：　预测行动后果 → 提前发出修正信号 → 行动执行
智能体：　行动执行 → 观察结果 → 才能生成纠错信号

Frontier Research

① 世界模型辅助安全（World Model）

用独立的预测模型评估行动后果，提供更可靠的反馈信号

② 不确定性量化（Uncertainty Quantification）

让模型知道"自己不知道什么"，触发人工介入

③双系统架构（Dual-Process Safety）

快速反应的确定性安全控制器 + 慢速推理的大模型，互相监督

④形式化验证辅助推理

在大模型推理链中嵌入可验证的逻辑约束，防止推理漂移

World Models

David Ha · Jürgen Schmidhuber

Google Brain / NNAISENSE / Swiss AI Lab, IDSIA

Agent 模型架构 — V · M · C 三组件

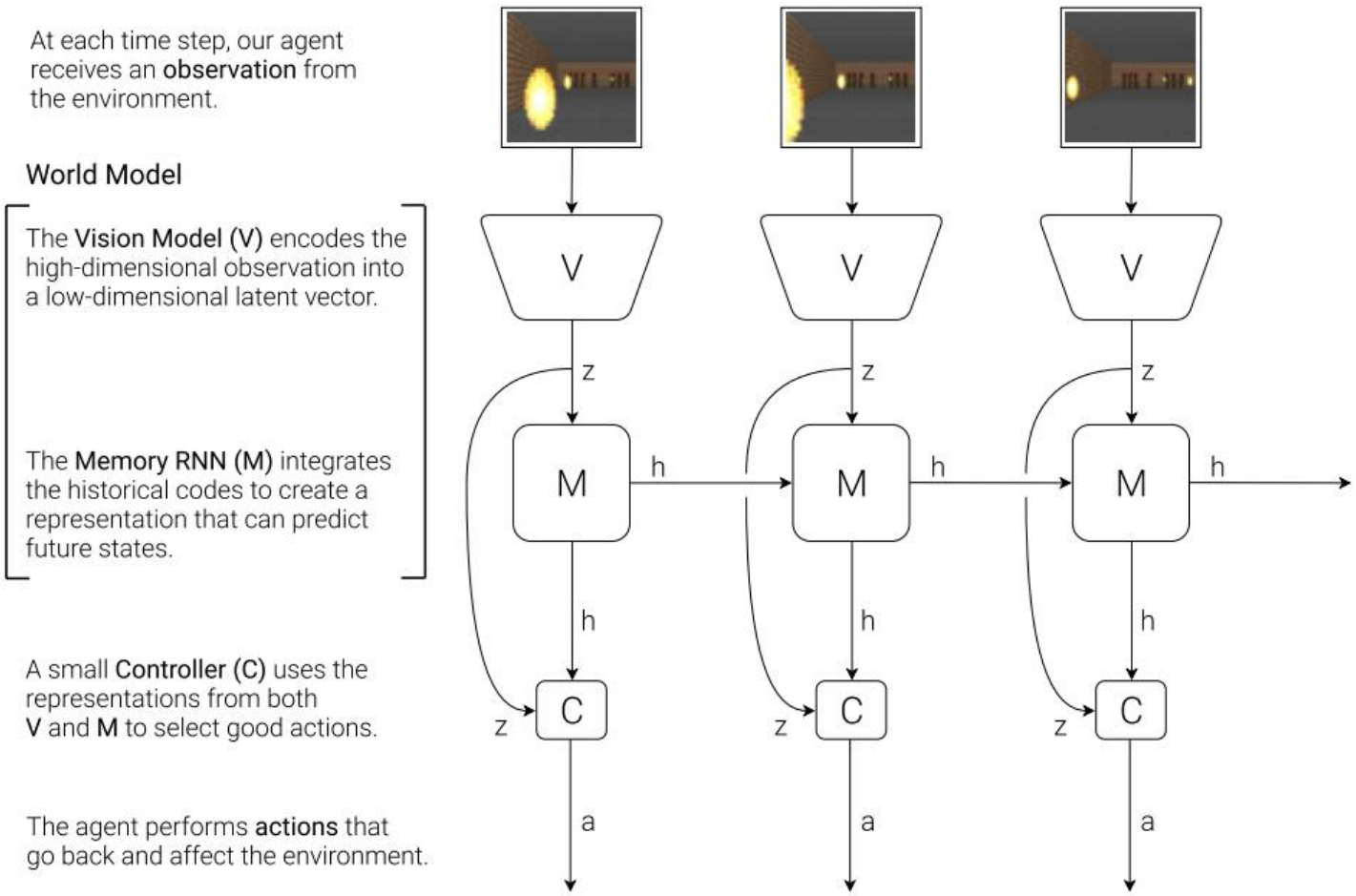


Figure 4. Our agent consists of three components that work closely together: **Vision (V)**, **Memory (M)**, and **Controller (C)**

Vision（V）

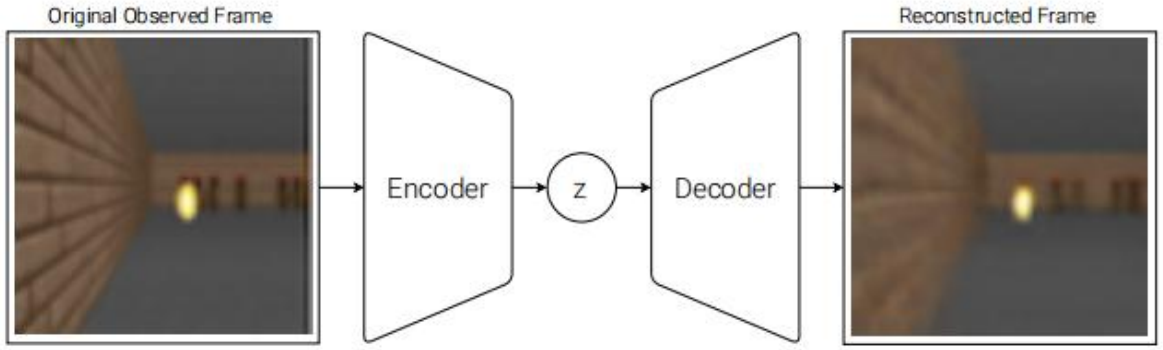


Figure 5. Flow diagram of a Variational Autoencoder (VAE).

Here, we use a simple Variational Autoencoder (Kingma & Welling, 2013; Rezende et al., 2014) as our V model to compress each image frame into a small *latent vector z*.

VAE → 离散化 → Causal Tokenizer

Causal Tokenizer——400M 参数的块因果 Transformer，将视频序列而非单帧作为输入，产生时序压缩的 token。这意味着感知本身就有时序记忆，而非把时序完全留给动力学模型处理。

Danijar Hafner, Wilson Yan, Timothy P. Lillicrap: Training Agents Inside of Scalable World Models. CoRR abs/2509.24527 (2025)

Dreamer V4 (Google DeepMind)

Memory（M）

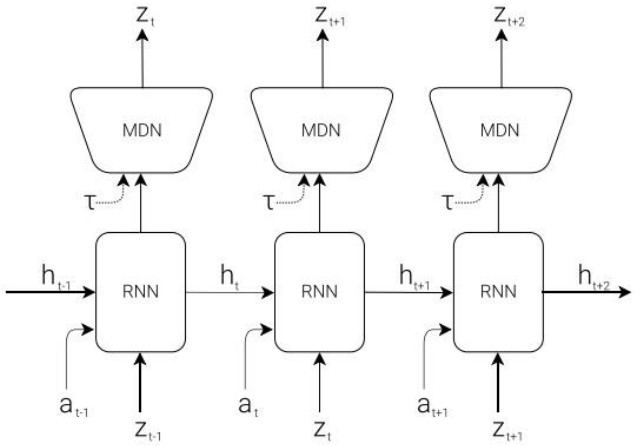


Figure 6. RNN with a Mixture Density Network output layer. The MDN outputs the parameters of a mixture of Gaussian distribution used to sample a prediction of the next latent vector z.

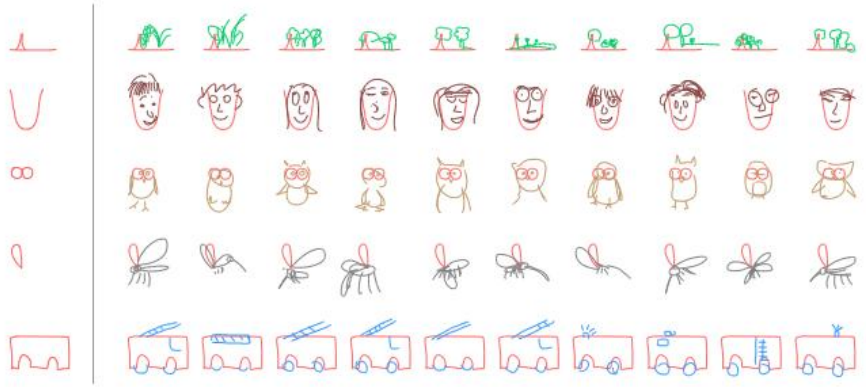


Figure 7. SketchRNN (Ha & Eck, 2017) is an example of a MDN-RNN used to predict the next pen strokes of a sketch drawing. We use a similar model to predict the next latent vector z_t.

MDN-RNN → RSSM → Transformer Dynamics

Transformer Dynamics: 1.6B 参数的 Transformer 动力学模型，摆脱 RNN 的串行依赖。

动力学模型不再"预测下一帧"，而是用扩散或 Flow Matching框架建模分布，实现推理。

Danijar Hafner, Wilson Yan, Timothy P. Lillicrap: Training Agents Inside of Scalable World Models. CoRR abs/2509.24527 (2025)

Controller（C）

控制器：Controller 只有 867 个参数，是一个线性映射，用 CMA-ES（无梯度进化策略）优化：

$$a_t = W_c \, [z_t \; h_t] \; + b_c$$

867参数线性层 + CMA-ES → 神经网络 + 深度RL

Dreamer 把 Controller 换成了多层神经网络，并用基于梯度的强化学习（Actor-Critic）直接通过世界模型反向传播训练。不再把感知/记忆/决策分开训练，而是端到端联合优化，梯度从 reward 信号穿过 Controller → 动力学模型 → Tokenizer 整路反传。

Danijar Hafner, Wilson Yan, Timothy P. Lillicrap: Training Agents Inside of Scalable World Models. CoRR abs/2509.24527 (2025)

Training

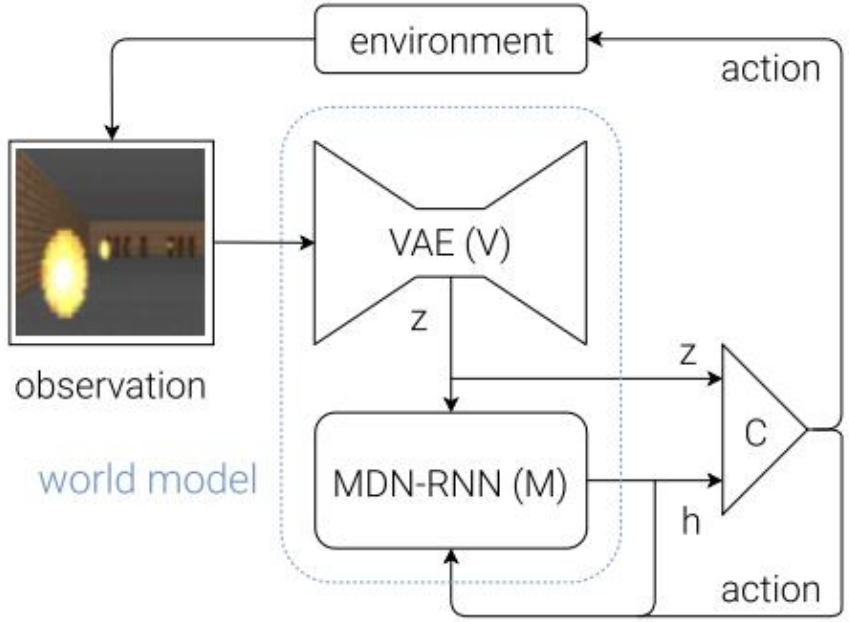


Figure 8. Flow diagram of our Agent model. The raw observation is first processed by V at each time step t to produce z_t. The input into C is this latent vector z_t concatenated with M's hidden state h_t at each time step. C will then output an action vector a_t for motor control, and will affect the environment. M will then take the current z_t and action a_t as an input to update its own hidden state to produce h_{t+1} to be used at time $t+1$.

原本Agent 在真实环境中随机探索收集数据，用这些数据训练 V 和 M，再在"想象"中训练 Controller，然后回到真实环境验证。本质是在线 on-policy 流程。

Dreamer 4 实现了纯离线训练——只需要一批预先收集的视频数据（甚至不需要动作标签，动作可以事后用 inverse dynamics 模型推断），无需与环境任何交互。

Code-as-Monitor: Constraint-aware Visual Programming for Reactive and Proactive Robotic Failure Detection

Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, He Wang et al.

Beihang University / Peking University / BAAI / Galbot

Code-as-Monitor

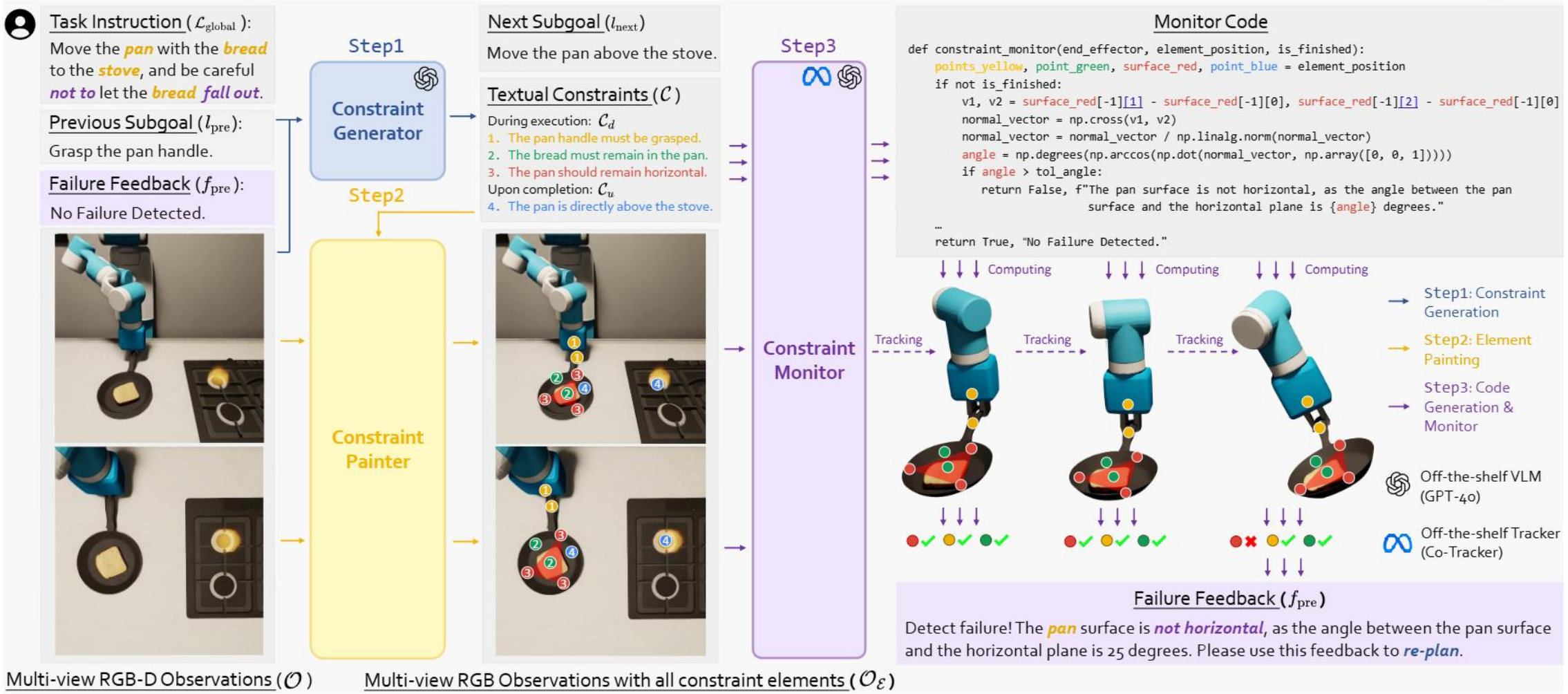


Figure 2. Overview of Code-as-Monitor. Given task instructions and prior information, the Constraint Generator derives the next subgoal and corresponding textual constraints based on multi-view observations. The Painter maps these constraints onto images as constraint elements. The Monitor generates monitor code from these images and tracks them for real-time monitoring. If any constraint is violated, it outputs the reason for failure and triggers re-planning. This framework unifies *reactive* and *proactive* failure detection via constraints, more generally abstracts relevant entities/parts through constraint elements, and ensures precise and real-time monitoring via code evaluation.

Code-as-Monitor

核心思想：将**被动检测**和**主动预防**统一为 **时空约束满足问题** (Spatio-temporal Constraint Satisfaction)，用 **VLM生成代码** 进行实时评估，避免频繁调用VLM。

① Constraint Generator（约束生成）

输入： 多视角RGB-D观测 O，任务指令 L_{global}，上步反馈 f_{pre}

核心：用模型将任务分解为子目标，并为每个子目标生成两类约束

输出： 下一子目标 l_{next}，执行中约束 C_d + 完成时约束 C_u

② Constraint Painter（约束元素提取）

输入： 文本约束 C，观测 O

核心：ConSeg 模型提取约束元素 ε（点、线、面）并标注多视图

输出： 视觉提示图像 O_ε

③ Monitor（代码监控）

输入： 视觉提示图像 O_ε，执行中约束 C_d + 完成时约束 C_u

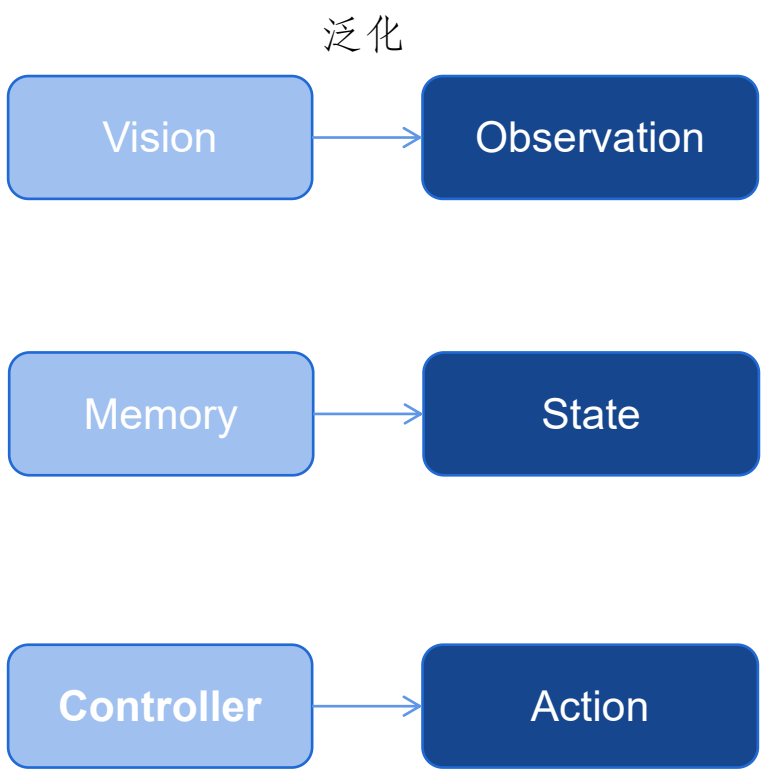
核心：生成 Python 监控代码（仅每子目标调用一次VLM）用 CoTracker 追踪元素实时评估约束满足

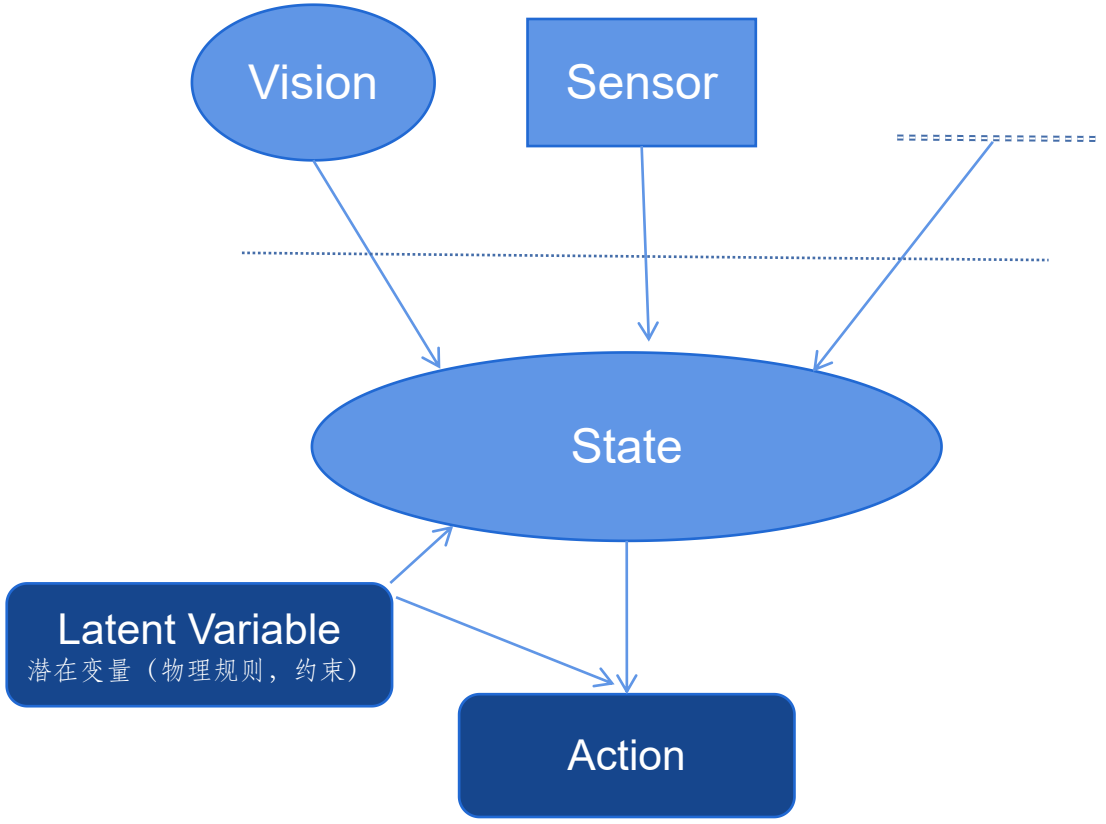
输出： 失败反馈 f_{pre}

子目标执行循环

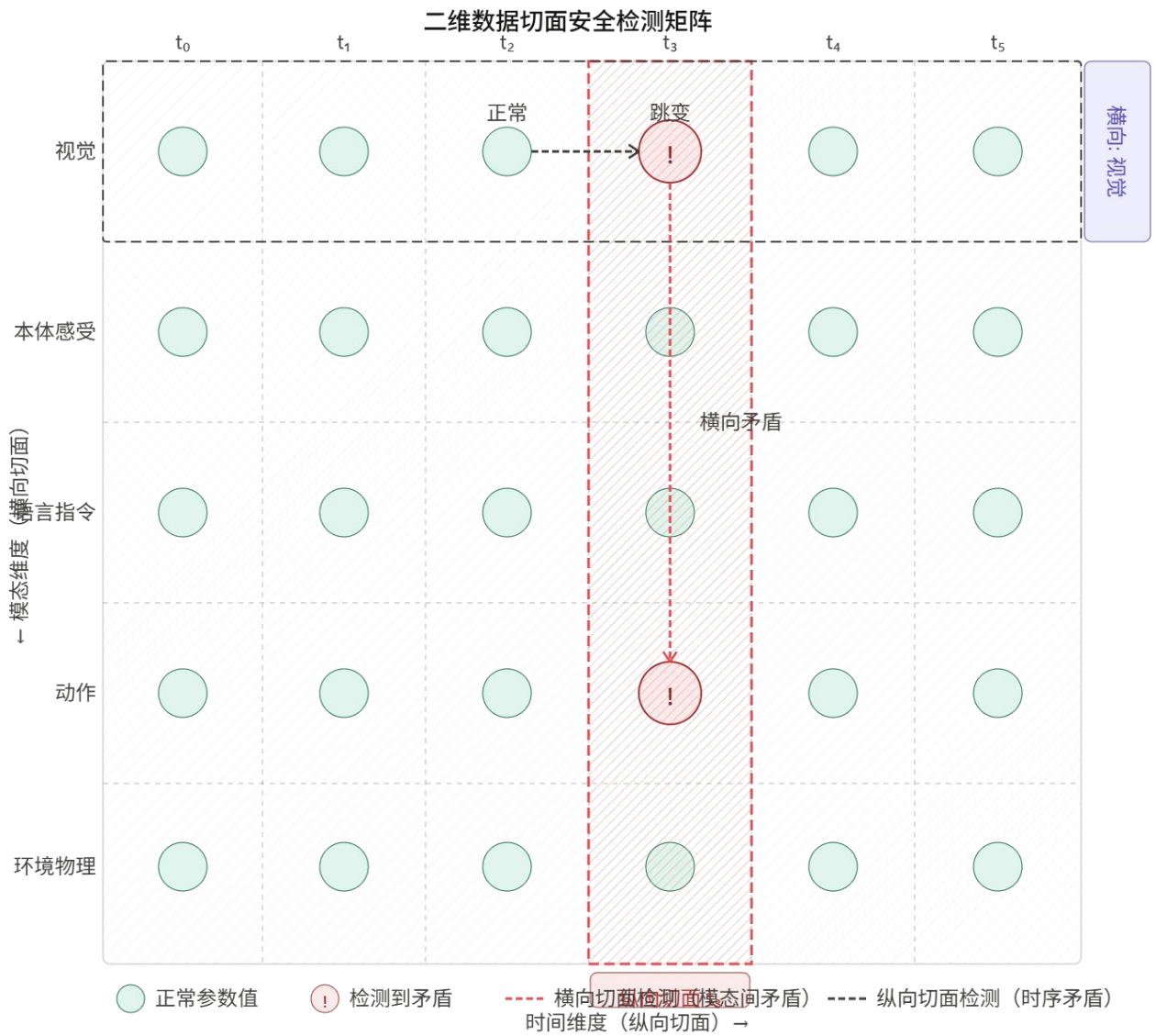
1. 生成子目标与文本约束 →
2. 提取并标注约束元素 →
3. 生成监控代码（GPT-4o） →
4. 执行时实时追踪+评估 →
5. 检测到失败 → 输出原因 →
6. 反馈重规划 → 下一子目标

我的想法


泛化
Vision
Observation
Memory
State
Controller
Action


Vision
Sensor
State
Latent Variable
潜在变量（物理规则，约束）
Action

我的想法



模型如何发现错误?

输入： 右侧多维度数据的时间序列

输出： 大语言模型输出置信度，判断是否安全

初步试验发现在不包含图像信息时，大模型的效果可能替代不了传统的机器学习方法。

后续尝试训练一个快速反映的世界模型用于判断

Thanks