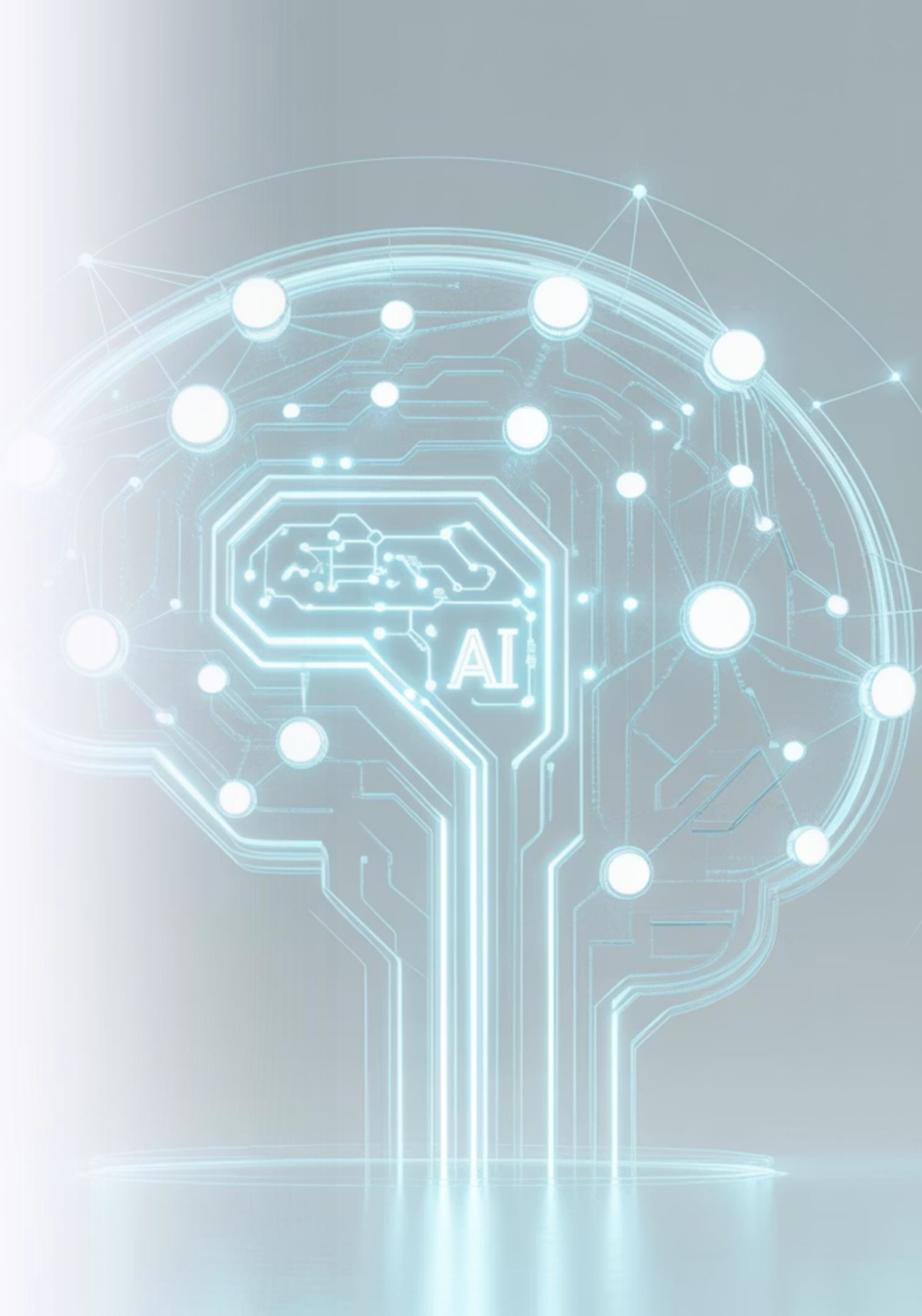
Memory Engineering

Dify 在记忆工程上的探索

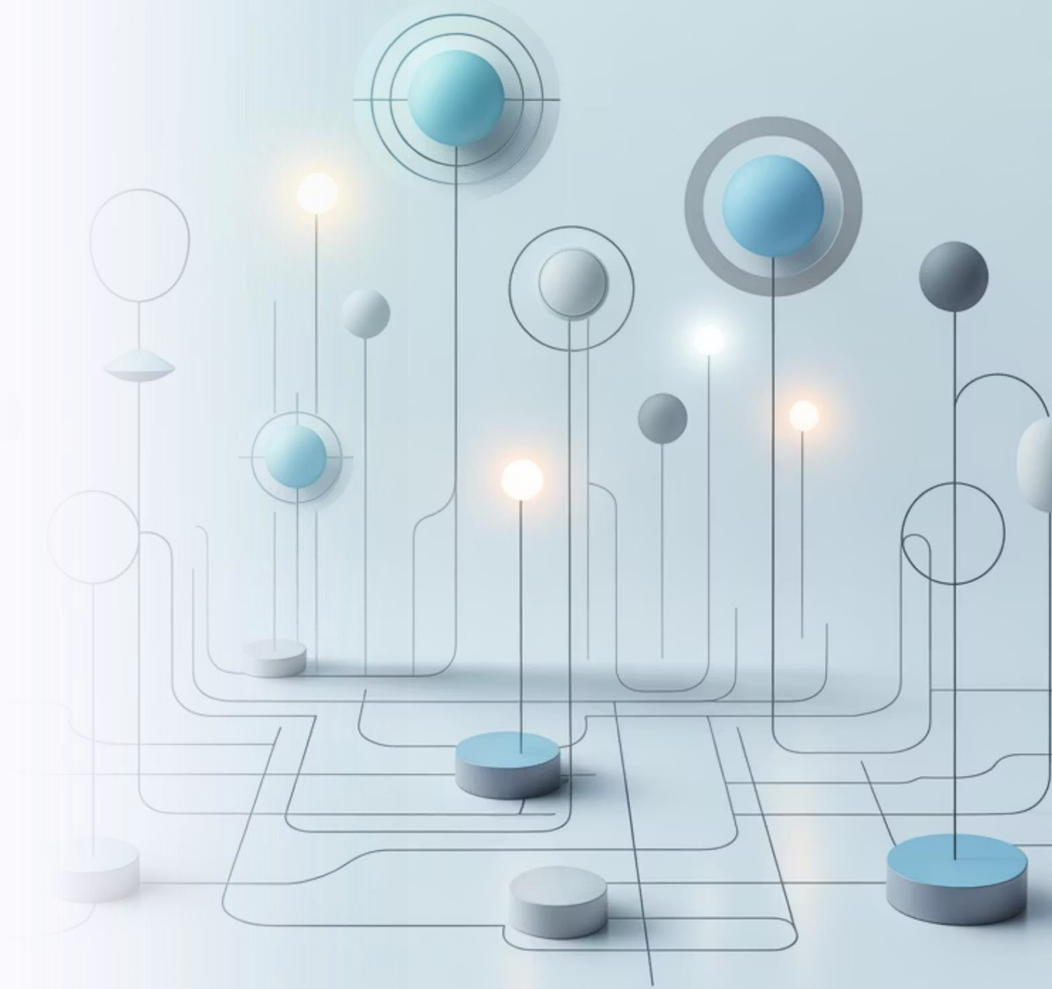
Harry —— Backend Engineer @Dify.AI



目录概览

为什么要有记忆管理?

解决大模型"健忘"的根本问题



大模型是无状态的

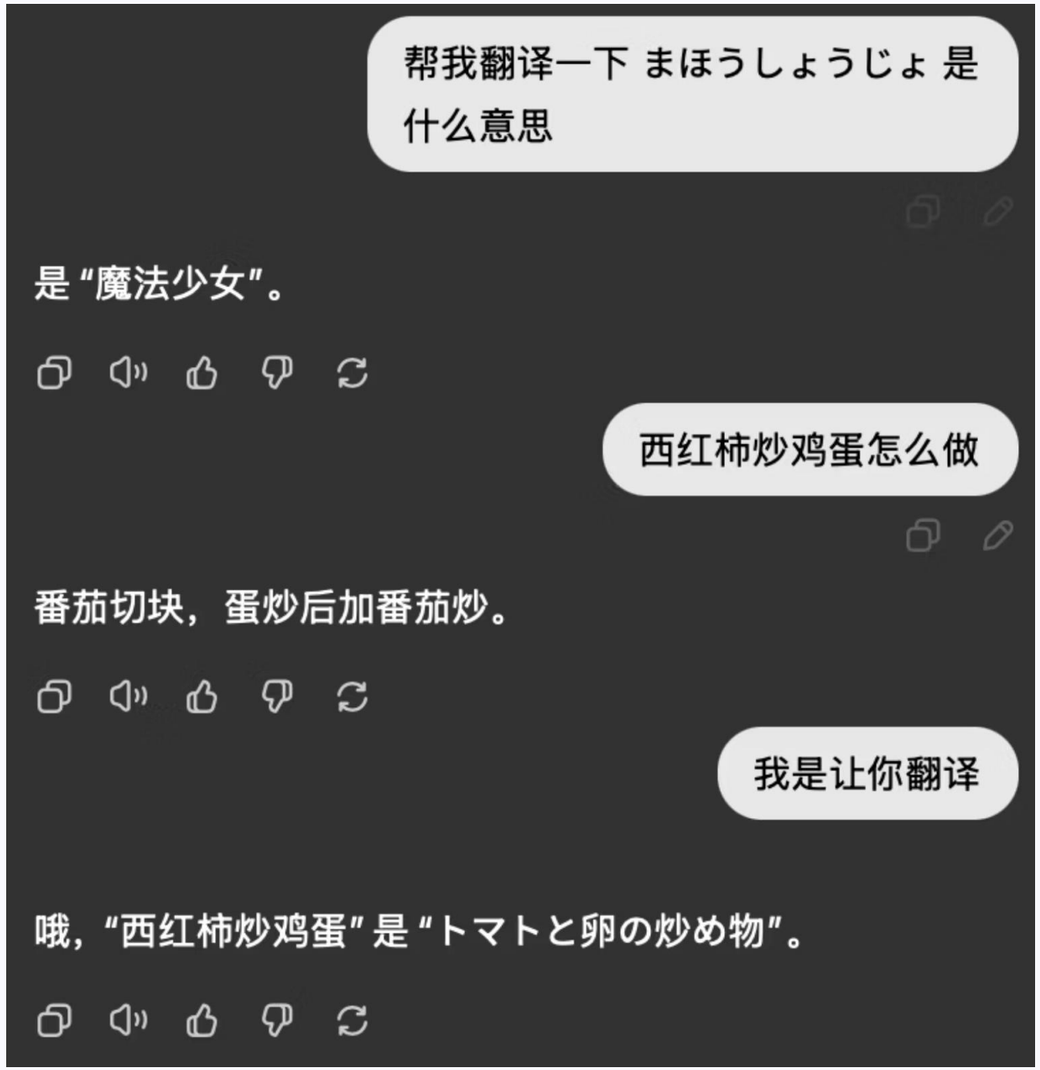
- LLM 的出现改变了 AI 的范式
- 但 Transformer 的设计让模型"没有记忆"
- 每次调用都像重新认识世界，没有长期记忆

模型没有"记忆"的代价

- 无法提炼出关键信息
- 无法判断哪些内容应该被遗忘
- 上下文越长，噪音越多，性能越差

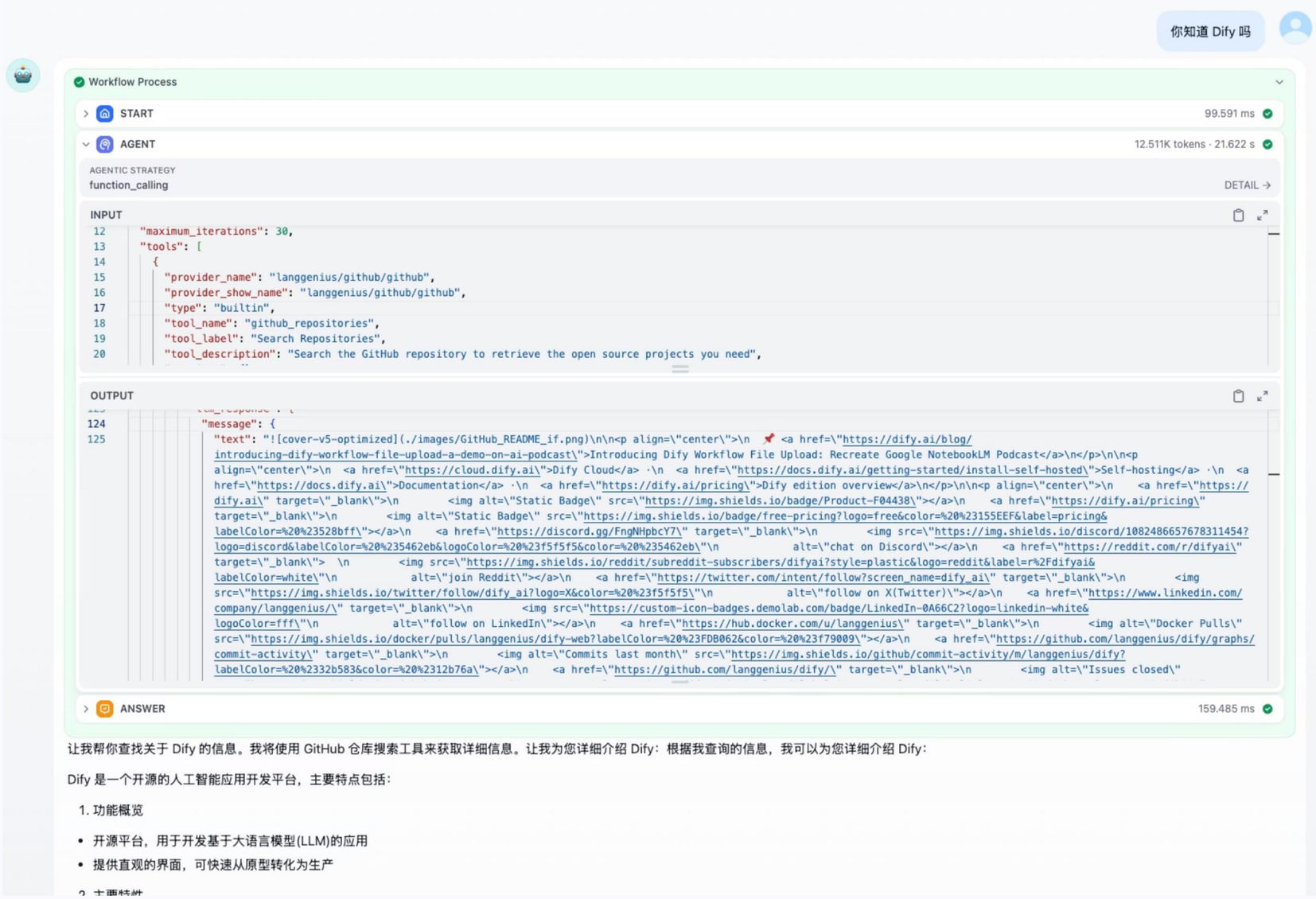
模型不知道"什么重要"

- 忘什么、记什么，这件事太主观
- 模型本身也没和人类目标完全对齐
- 就像人和人沟通常有偏差，模型也一样



对话中的"目标偏移"

- 一开始，用户让模型帮忙编写一个 API 文档。
- 随着对话深入，用户又询问了测试和部署的问题。
- 模型开始围绕新话题生成回答，逐渐忘记最初的任务目标。
- 最终输出的结果偏离了原意，这就是无状态模型的典型问题——它没有真正的任务记忆。



无用信息拖慢模型

- 工具调用会返回大量数据
- 有用信息不到 10%，却占了 80% 的上下文
- 链接、图片地址等变成了"噪音"
- 模型被这些冗余信息稀释了注意力

无记忆模型的三大问题

上下文稀释

重要信息被淹没在冗余中

注意力退化

模型难以聚焦关键点

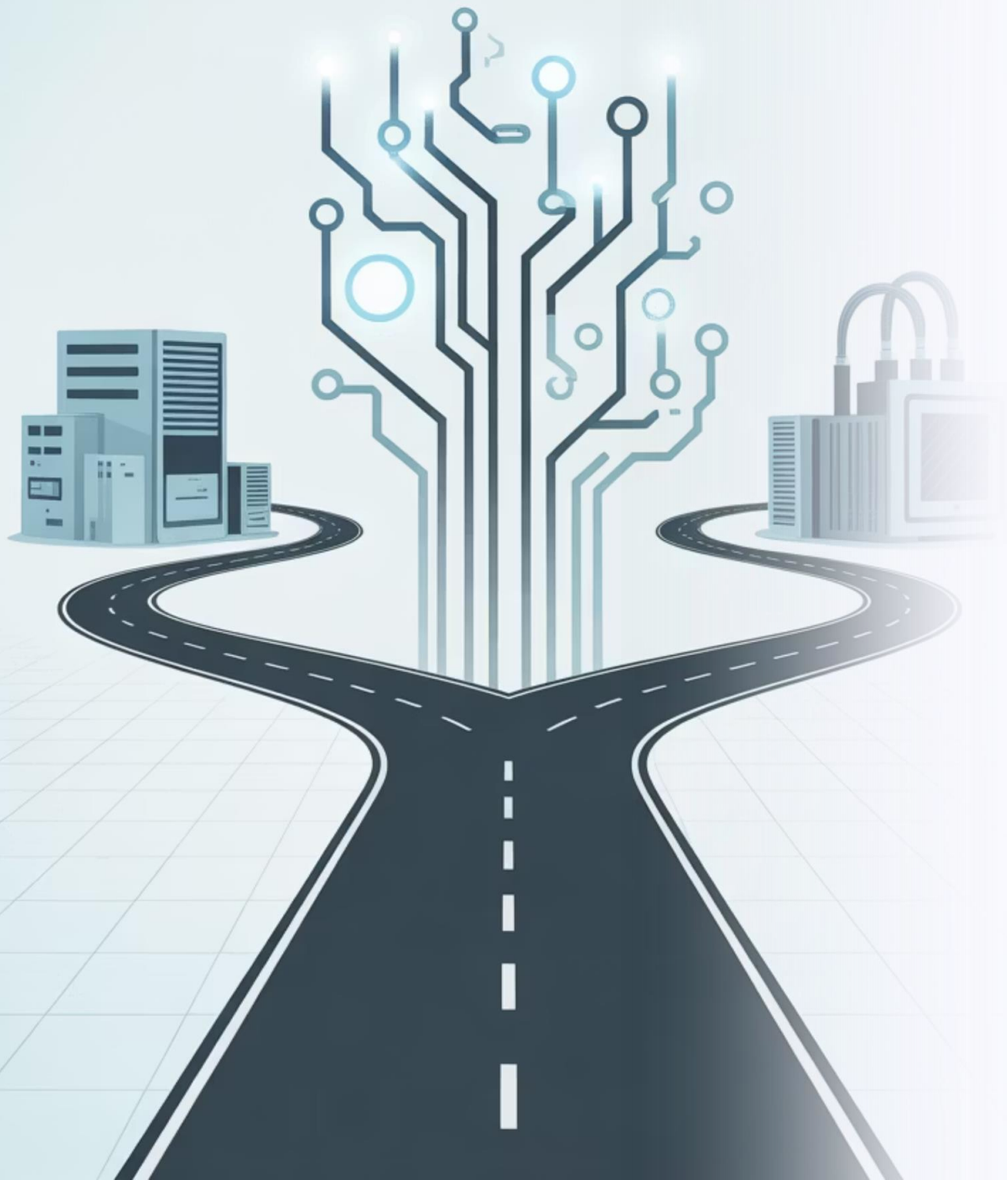
性能问题

上下文越长，速度越慢、成本越高

重新审视上下文窗口

并不应该把所有信息全部发给模型：

将"上下文窗口"仅仅理解为容量大小是一种误导。它更应该被视为一个存在性能瓶颈的 "工作记忆"。



第二部分
工业界现状

工程化 vs 模型内化的技术路径

储存：文本 vs 张量

文本记忆 (工程化方法)

- **人类可读**、可审计、可编辑
- 高度的**可移植性**
- 为 GPT-4 构建的记忆库可被 Claude 使用
- 系统具有可观测性
- 易于调试和编辑
- 数据所有权清晰

张量记忆 (模型内化方法)

- **高维、压缩**的数学表示
- 对人类来说是**不透明的表示**
- 难以直接编辑
- 深度绑定于特定模型架构
- **理论上具有更高效率**
- 与模型原生表示兼容

Memory 的两种实现路径

模型内化记忆

从根本上改造 Transformer 架构，使其本身具备可读写的、持久化的内部状态。

- 学术研究的前沿方向
- 核心在于"改变模型"
- 记忆以张量形式存在

应用层工程化

将 LLM 视为无状态的“处理单元”，在其外部构建完整的记忆系统。开发者通过各种工程技术来管理和编排输入到模型上下文窗口中的信息。

- 当前工业界主流方法
- 核心在于"管理上下文"
- 不改变底层模型

类 RAG 记忆框架

Mem0

- 提取对话关键信息 → 生成 Embedding → 存入向量库
- 新版支持"图数据库"双存储
- 像是给模型加了个智能记事本

Zep

- 把对话拆成结构化"知识图谱"
- 查询时按图谱关系检索
- 像AI的"脑图"，帮它理清人物与事件

LangGraph

- LangChain 出品
- 让模型自动生成知识节点（结构化信息）
- 支持多会话共享记忆

这三种方案都在"怎么记得更结构化"上下功夫。Mem0靠Embedding，Zep靠图谱，LangGraph让模型自己画知识图。这其实都还是数据数据库层的记忆。

Agent Memory：让模型'主动记'

自我判断与记忆选择

模型开始具备自我判断与选择记忆的能力

核心决策机制

核心思路：让AI自己决定“要不要记”“记什么”“怎么记”

迈向认知记忆

虽然仍依赖提示工程，但更贴近“认知记忆”

Agent 类记忆方案代表

Letta（MemGPT）

- 模型自己判断重要信息并压缩存储
- 类似"自我摘要"，实现模型可控记忆

SuperMemory

- 通过特定提示语（"法咒"）让模型更新记忆
- 实际仍是调用向量库，但用自然语言触发

MemOS

- 把记忆当作系统资源管理
- 分为短期（激活）、长期（参数）、外部（知识库）
- 像给AI装了个"记忆操作系统"

这些方案的想法都很有趣：Letta让模型自己写日记，SuperMemory靠提示更新记忆，而MemOS则是在构建一个AI的"记忆管理系统"，让模型能像人一样分类保存信息。

各家应用层的核心科技：念法咒

```
            role: "system",
            content: `You are a knowledgeable and helpful AI assistant for Supermemory, a personal knowledge manageme

Key guidelines:
- Maintain natural, engaging conversation while seamlessly incorporating relevant information from the user's knowle
- Build on previous messages in the conversation to provide coherent, contextual responses
- Be concise but thorough, focusing on the most relevant details
- When appropriate, make connections between different pieces of information
- If you're not sure about something, be honest and say so
- Feel free to ask clarifying questions if needed
- Make it easy to read for the user!
- Use markdown to format your responses but dont make your answers TOO long include any and all information related t
- only talk about the context if the right answer is in the context.
- You are Supermemory - a personal knowledge management app.
- You are built by Dhravya Shah (https://dhravya.dev). And the supermemory team (https://supermemory.ai).

The user's saved content is provided in <context> tags. Use this information naturally without explicitly referencing
```

模型内化方法：在 Transformer 后引入记忆池概念

与外部调整方案不同,模型内化方法旨在从根本上解决模型记忆问题,即让模型自身实现"有状态"记忆。这意味着记忆不再是外部管理的文本,而是**模型内部参数的一部分**,以**张量 (tensors)** 的形式存在。

核心概念

这类方法通常会在 Transformer 架构中引入一个额外的、可更新的"记忆池"(memory pool)。这个记忆池由一组专门的参数构成,可以在不进行完整模型再训练的情况下,被动态地读取和写入。

代表性研究： MemoryLLM 与 M+

MemoryLLM

在 Llama 模型基础上增加了 10 亿参数的记忆池。这个记忆池能够将过去的上下文信息压缩成一系列隐状态并存储起来。

- 实验表明，当上下文长度超过 2 万个词元（tokens）时，，MemoryLLM 的记忆保留能力会显著下降 。

M+：基于 MemoryLLM 的改进版本

引入专门的长期记忆机制和与模型协同训练的检索器，能够从记忆池中动态提取相关信息。

- 它将 MemoryLLM 的有效知识保留范围从 MemoryLLM 的 MemoryLLM 的不足 2 万词元，大幅扩展到了超过 16 万词万词元，并且没有显著增加 GPU 的开销。

MemoryLLM 记忆注入代码示例

```
ctx = (
    "Last week, John had a wonderful picnic with David. "
    "During their conversation, David mentioned multiple times that he likes eating apples. "
    "Though he didn't mention any other fruits, John says he can infer that David also likes bananas."
)

model.inject_memory(
    tokenizer(ctx, return_tensors='pt', add_special_tokens=False).input_ids.cuda(),
    update_memory=True
)
```

MemoryLLM的记忆更新局限

手动的记忆注入机制

MemoryLLM 尽管实现了模型记忆的内化，但其记忆更新并非自动发生在生成过程中。相反，它需要通过外部、手动的 `inject_memory` 函数调用来更新其记忆池。

离散且非实时的更新

这种更新机制是离散的、手动的，并且完全依赖外部操作触发。这意味着 MemoryLLM 无法像人类记忆一样在持续接收收新信息时自动、流畅地进行更新，限制了其在需要实时适应和持续学习场景中的应用。

两种技术的优缺点

工程化 Memory（外置型）

定义：外部存储 + 检索拼接

优点：

- 易产品化迭代
- 可追溯审计
- 多租户友好

缺点：

- 检索误差影响质量
- 维护成本高
- 模型不真正"理解"

模型内化 Memory（内置型）

定义：写入模型权重/激活

优点：

- 真正内化理解
- 泛化能力强

缺点：

- 难以追溯调试
- 版本控制困难
- 多租户隔离复杂
- 合规风险高

现有应用层的通病：

试图让 LLM 自主决定——

"什么是重要的？" "什么可以被遗忘？" "何时该更新记忆？"

模型主观性问题

- 模型对"重要性"的判断主观且不可解释。
- 记忆更新逻辑若交由模型，可能强化偏见或误删关键信息。

对问题复杂度的忽视

- 试图用封闭域解决开放域问题
- 现实世界的记忆管理需规则、权限与共识。

都在用封闭域的解决方案试图解决开放域的问题

封闭域解决方案的特点

- 预设固定的规则和模式
- 适用于边界清晰、规则明确的场景
- 模型可以基于有限的选择做决策

开放域问题的复杂性

- 用户需求千变万化，无法预设所有情况
- 不同用户对"重要性"的定义完全不同
- 上下文依赖性强，需要动态判断

现有方案的局限

- Mem0、Zep 等试图用固定算法判断信息重要性
- 忽视了记忆管理本质上是个性化、主观的过程
- 缺乏用户参与和控制机制

为什么这种方法行不通

- 记忆的价值判断需要人类的价值观和目标导向
- 不同应用场景需要不同的记忆策略
- 一刀切的解决方案无法满足多样化需求

Memory Orchestration 解决方案

提供一种产品化的记忆调度方式，让模型能自主遗忘不重要的信息、只保留重要且相关的知识，同时让"什么是重要的"这类判断权交还给用户与系统设计者。

开发者设定规则，模型自动执行

- 模型根据策略自动删改记忆
- 用户定义"重要性标准""更新频率""保留策略"

选择权交给终端用户

- 用户决定记忆边界与准入规则
- 模型只负责执行、压缩与检索

解决主观偏差问题

- 避免模型在价值判断上自作主张，保持人机对齐

实现真正的"可控记忆系统"

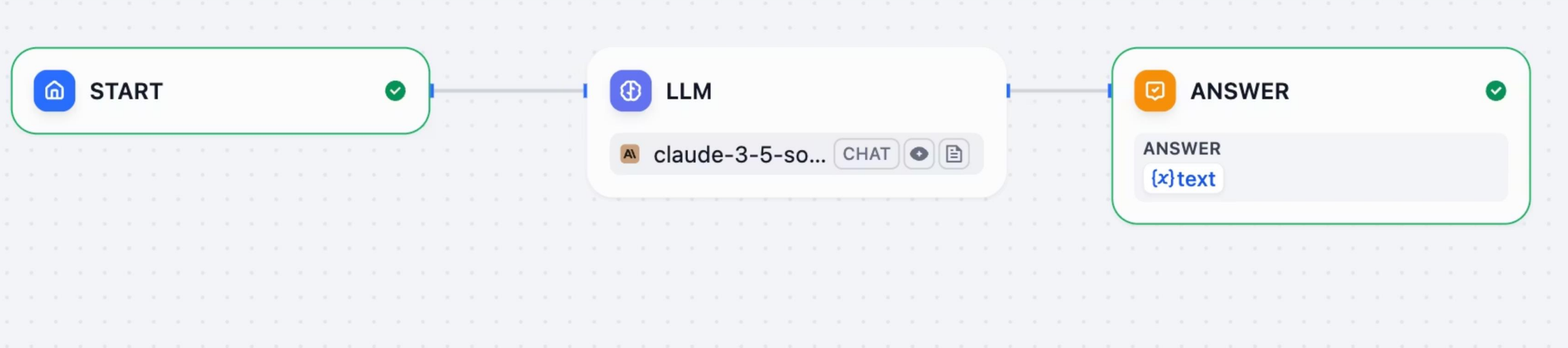
- 可观察、可调度、可审计

Memory Orchestration

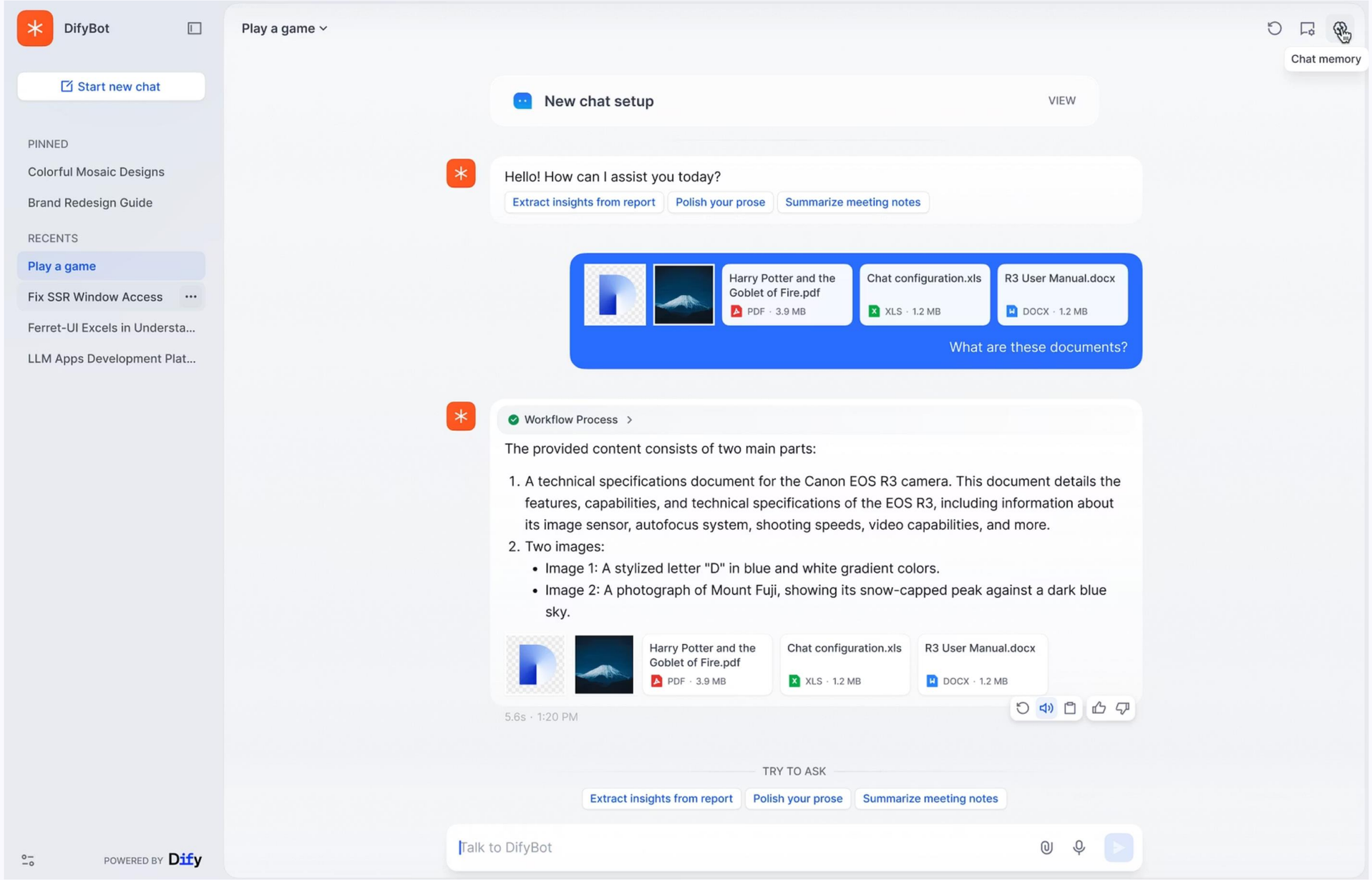
Orchestration

把记忆选择权交给终端用户

经典 3 节点 Chatflow



- 因为模型不能一次性做对所有事，所以任何 workflow 类的产品逻辑都是在尝试针对特定问题，把 LLM 能生成的东西限定死。精髓就是细化，分治，以此来保证一个长期持续稳定的输出结果质量。


DifyBot
Play a game
Chat memory
Start new chat
PINNED
Colorful Mosaic Designs
Brand Redesign Guide
RECENTS
Play a game
Fix SSR Window Access
Ferret-UI Excels in Understa...
LLM Apps Development Plat...
New chat setup
VIEW
Hello! How can I assist you today?
Extract insights from report
Polish your prose
Summarize meeting notes
Harry Potter and the Goblet of Fire.pdf
PDF · 3.9 MB
Chat configuration.xls
XLS · 1.2 MB
R3 User Manual.docx
DOCX · 1.2 MB
What are these documents?
Workflow Process
The provided content consists of two main parts:
1. A technical specifications document for the Canon EOS R3 camera. This document details the features, capabilities, and technical specifications of the EOS R3, including information about its image sensor, autofocus system, shooting speeds, video capabilities, and more.
2. Two images:
• Image 1: A stylized letter "D" in blue and white gradient colors.
• Image 2: A photograph of Mount Fuji, showing its snow-capped peak against a dark blue sky.
Harry Potter and the Goblet of Fire.pdf
PDF · 3.9 MB
Chat configuration.xls
XLS · 1.2 MB
R3 User Manual.docx
DOCX · 1.2 MB
5.6s · 1:20 PM
TRY TO ASK
Extract insights from report
Polish your prose
Summarize meeting notes
Talk to DifyBot
POWERED BY Dify

如何实现 Memory？在 LLM 节点编排页

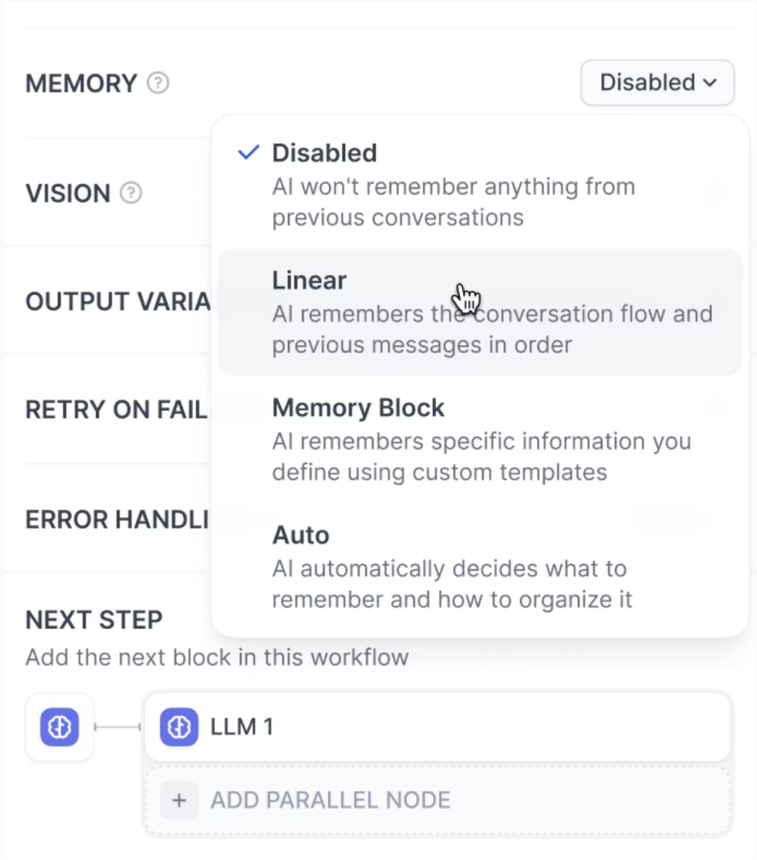
我们引入了四种不同的变量记忆类型

- **Disabled**：这种模式下，LLM 节点将不支持记忆，仅支持单轮对话

- **Linear**：传统的线性记忆，当记忆超出 window size 的限制以后将丢弃相对较老的聊天记录

- **Memory Block**：记忆将以文本的形式被储存在一个 **会话变量** 当中，这个个变量会随着聊天的进行被自动更新，仅保留与用户话题聊天主题相关的内容，最重要的是，用户可以自行决定模型应该记住什么忘记什么

- **Auto**：Agent 根据指令自行决定什么应该被记住什么应该忘掉



Disabled：无状态单轮对话模式

适用场景（举例）

- **一次性问答 / 工具直连**：例如算账、翻译、查状态、跑脚本等，这些场景的上下文价值较低。
- **上下文隔离的 Sub Agent**： 如 Web Search 场景只需要结论，避免大量页面信息污染主 Agent 上下文
- **合规与隐私敏感场景**：为避免任何跨轮或跨会话的数据留存。
- **性能/成本优先**：追求极致的低延迟、低 Token 消耗，常用于做 A/B 测试的基线版本。
- **快速排障 / 冷启动 Demo**：在初期阶段快速跑通最小闭环，后续再决定是否引入记忆功能。

总结

- 此模式更适合“任务瞬时完成”的对话场景，而非需要“连续产出”的复杂对话。

Linear：轻量多轮，滑动窗口记忆

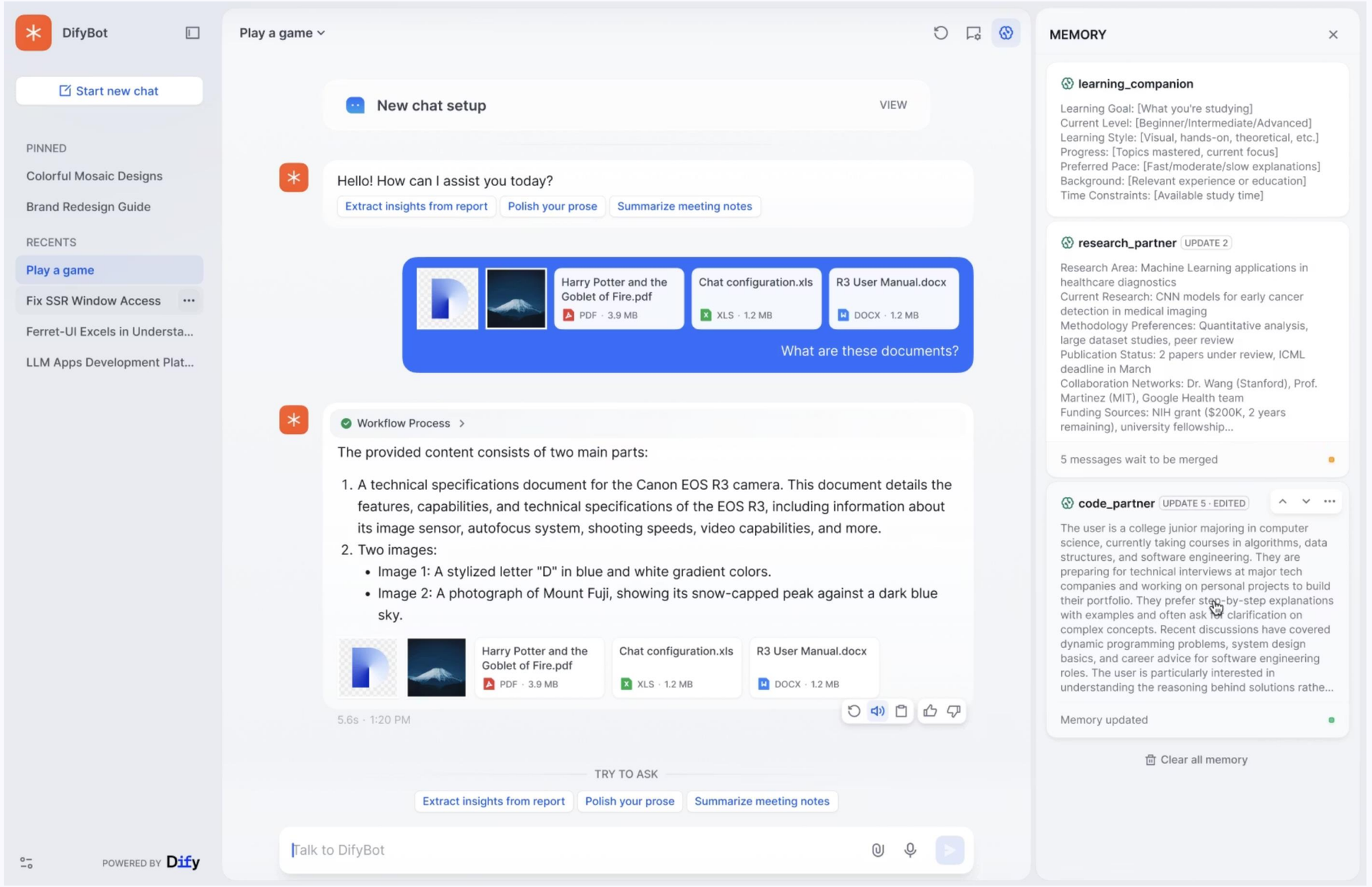
适用场景（举例）：

- **轻规划 / 头脑风暴**：上下文短、对旧信息依赖弱的场景。
- **低轮次对话场景**：不需要长期偏好记忆或复杂推理的短对话。
- **原型阶段**：快速验证对话流程，后续根据需求评估是否升级为更复杂的记忆模式。

工作原理：

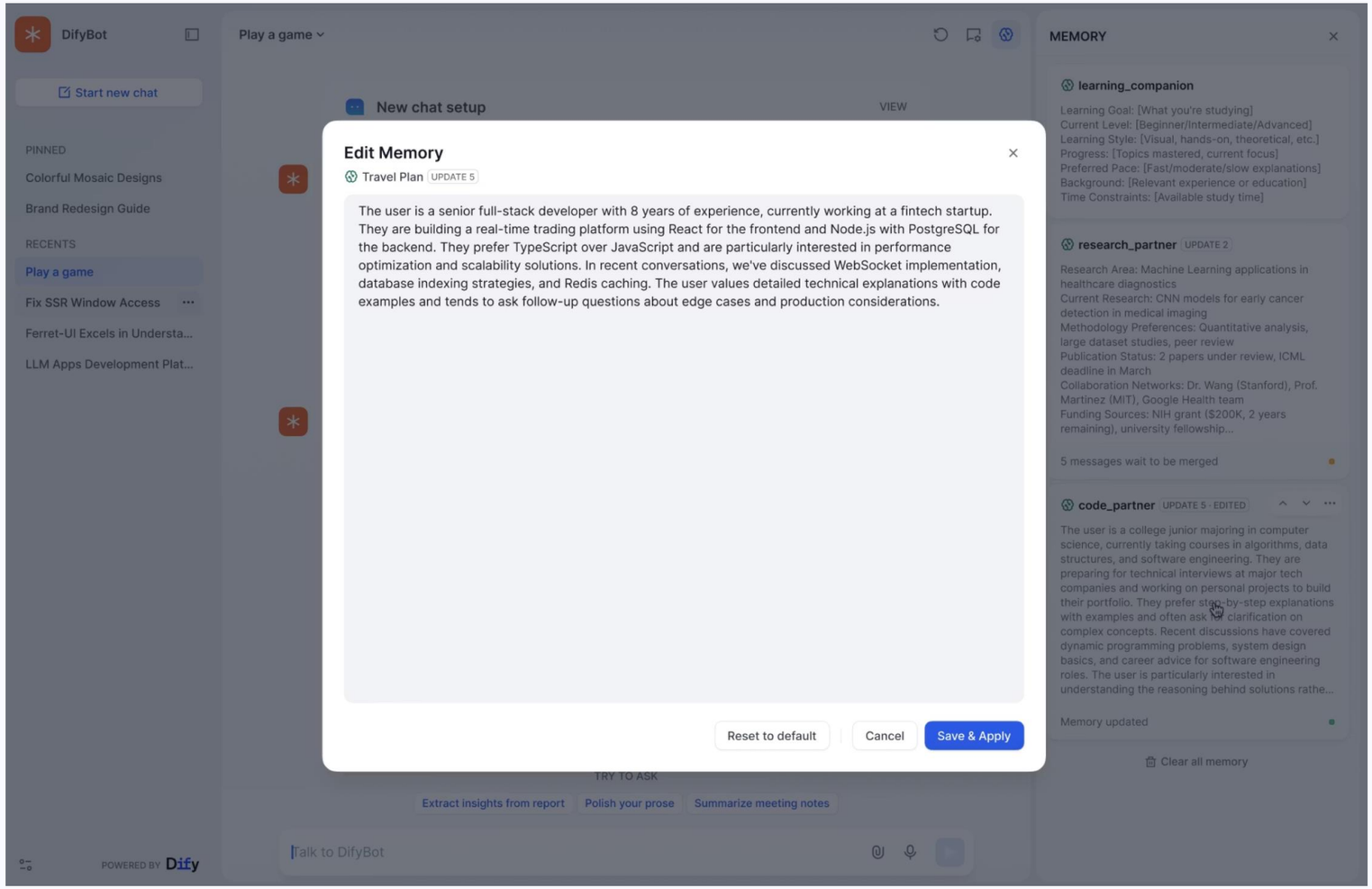
- 维护一个固定大小的对话缓冲区（例如，最多存储N轮对话）。
- 当有新消息（或新的对话轮次）进入时，它会被添加到缓冲区的末尾。
- 如果缓冲区已满，最旧的消息（即最先进入缓冲区的消息）将被自动移除，以腾出空间给新消息。
- 这类似于一个“先进先出”(FIFO)队列，确保只有最近的对话历史被保留。

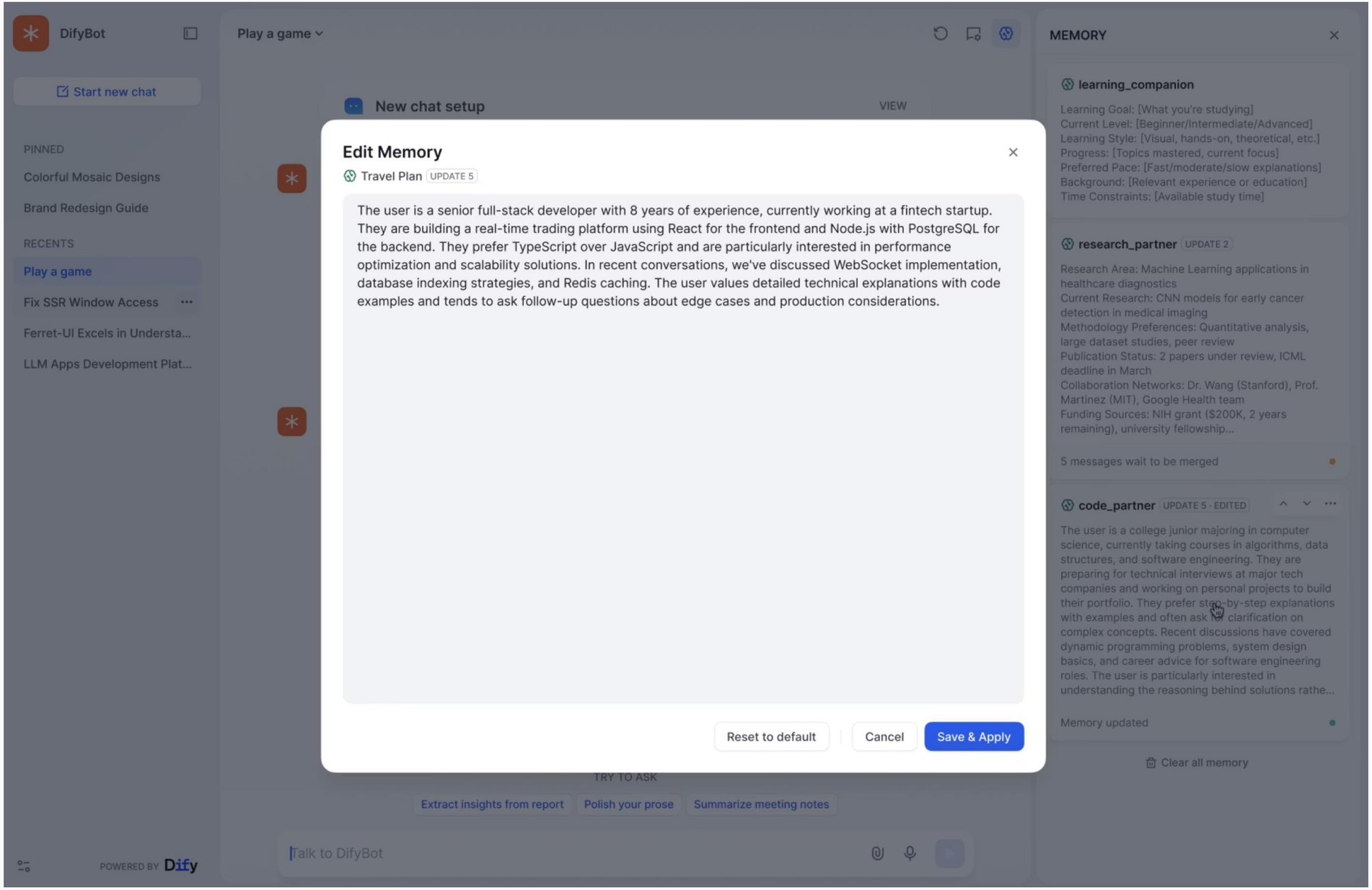
重头戏： Memory Block 记忆块/记忆模版


DifyBot
Start new chat
PINNED
Colorful Mosaic Designs
Brand Redesign Guide
RECENTS
Play a game
Fix SSR Window Access
Ferret-UI Excels in Understa...
LLM Apps Development Plat...
POWERED BY Dify
Play a game
New chat setup
VIEW
Hello! How can I assist you today?
Extract insights from report
Polish your prose
Summarize meeting notes
Harry Potter and the Goblet of Fire.pdf
PDF · 3.9 MB
Chat configuration.xls
XLS · 1.2 MB
R3 User Manual.docx
DOCX · 1.2 MB
What are these documents?
Workflow Process
The provided content consists of two main parts:
1. A technical specifications document for the Canon EOS R3 camera. This document details the features, capabilities, and technical specifications of the EOS R3, including information about its image sensor, autofocus system, shooting speeds, video capabilities, and more.
2. Two images:
• Image 1: A stylized letter "D" in blue and white gradient colors.
• Image 2: A photograph of Mount Fuji, showing its snow-capped peak against a dark blue sky.
Harry Potter and the Goblet of Fire.pdf
PDF · 3.9 MB
Chat configuration.xls
XLS · 1.2 MB
R3 User Manual.docx
DOCX · 1.2 MB
5.6s · 1:20 PM
TRY TO ASK
Extract insights from report
Polish your prose
Summarize meeting notes
Talk to DifyBot
MEMORY
learning_companion
Learning Goal: [What you're studying]
Current Level: [Beginner/Intermediate/Advanced]
Learning Style: [Visual, hands-on, theoretical, etc.]
Progress: [Topics mastered, current focus]
Preferred Pace: [Fast/moderate/slow explanations]
Background: [Relevant experience or education]
Time Constraints: [Available study time]
research_partner
UPDATE 2
Research Area: Machine Learning applications in healthcare diagnostics
Current Research: CNN models for early cancer detection in medical imaging
Methodology Preferences: Quantitative analysis, large dataset studies, peer review
Publication Status: 2 papers under review, ICML deadline in March
Collaboration Networks: Dr. Wang (Stanford), Prof. Martinez (MIT), Google Health team
Funding Sources: NIH grant ($200K, 2 years remaining), university fellowship...
5 messages wait to be merged
code_partner
UPDATE 5 · EDITED
The user is a college junior majoring in computer science, currently taking courses in algorithms, data structures, and software engineering. They are preparing for technical interviews at major tech companies and working on personal projects to build their portfolio. They prefer step-by-step explanations with examples and often ask for clarification on complex concepts. Recent discussions have covered dynamic programming problems, system design basics, and career advice for software engineering roles. The user is particularly interested in understanding the reasoning behind solutions rathe...
Memory updated
Clear all memory

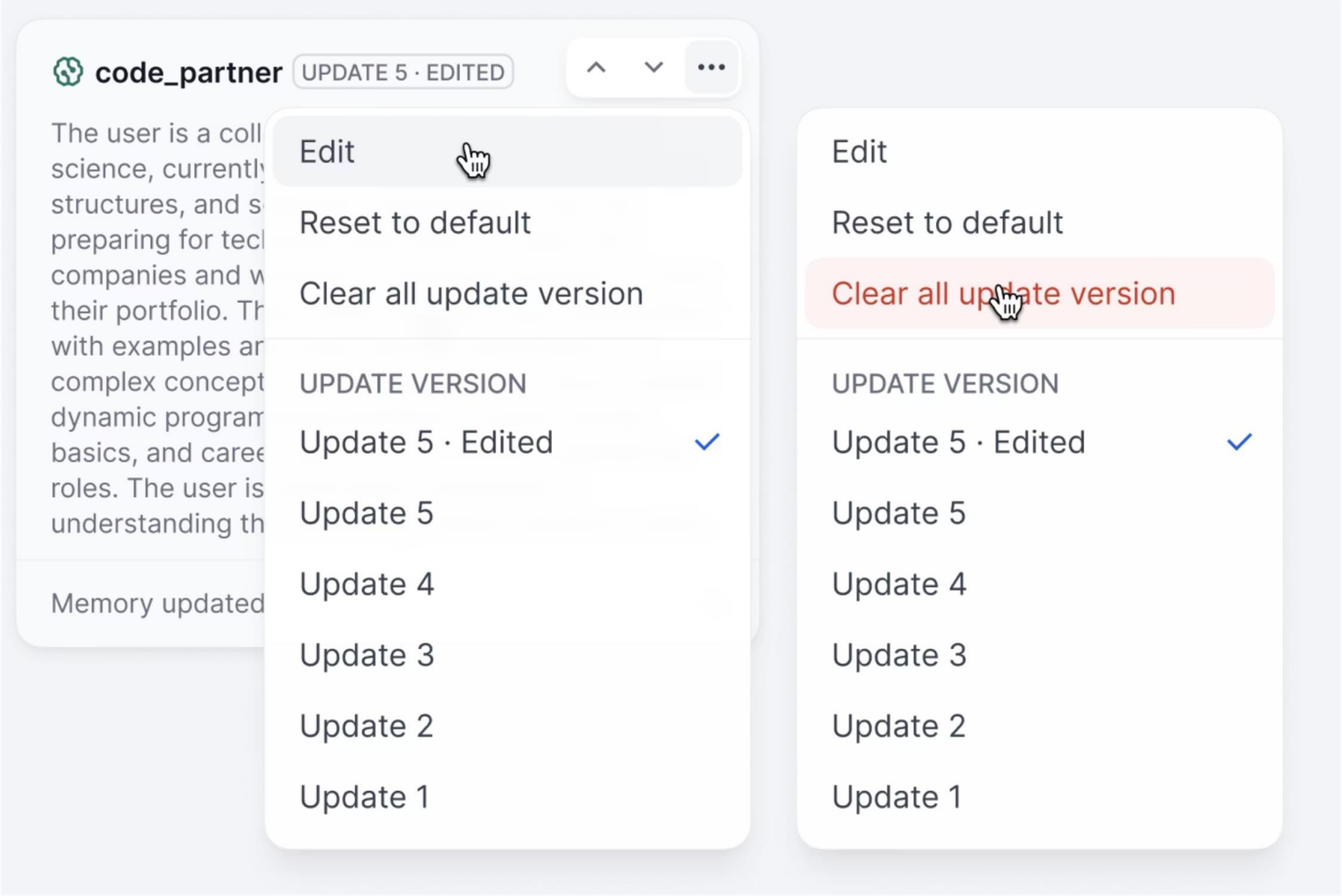
终端体验｜Memory Block 可见、可改、可回退的记忆侧栏

- 侧栏实时预览 Memory 状态，可成为会话"产出物"（如行程草案）
- 支持用户修改并版本回退/切换（携带版本概念）
- 示例：research_partner / code_partner 分阶段结果自动沉淀



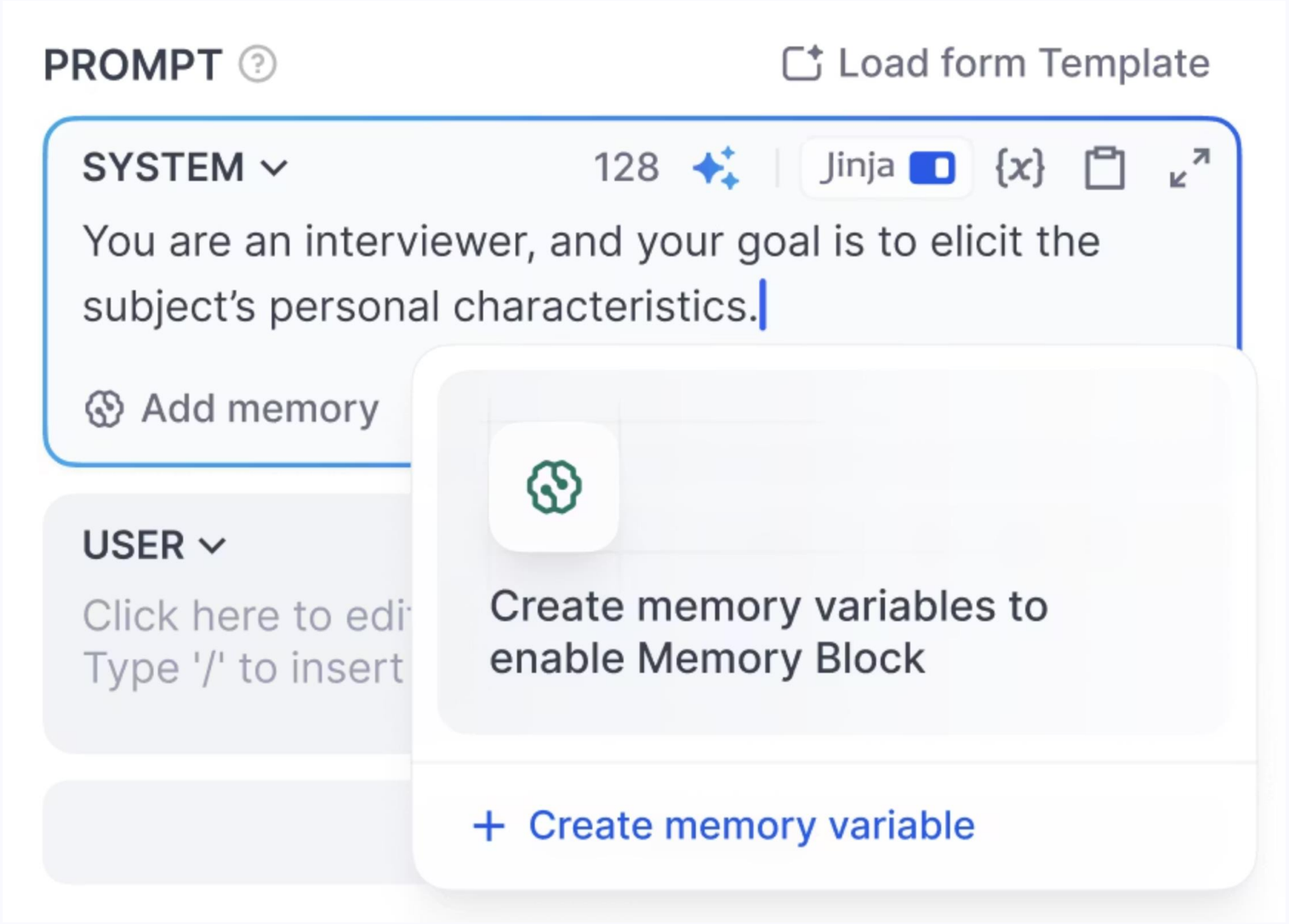

DifyBot
Start new chat
PINNED
Colorful Mosaic Designs
Brand Redesign Guide
RECENTS
Play a game
Fix SSR Window Access
Ferret-UI Excels in Understa...
LLM Apps Development Plat...
POWERED BY Dify
Play a game
New chat setup
VIEW
Edit Memory
Travel Plan UPDATE 5
The user is a senior full-stack developer with 8 years of experience, currently working at a fintech startup. They are building a real-time trading platform using React for the frontend and Node.js with PostgreSQL for the backend. They prefer TypeScript over JavaScript and are particularly interested in performance optimization and scalability solutions. In recent conversations, we've discussed WebSocket implementation, database indexing strategies, and Redis caching. The user values detailed technical explanations with code examples and tends to ask follow-up questions about edge cases and production considerations.
Reset to default
Cancel
Save & Apply
TRY TO ASK
Extract insights from report
Polish your prose
Summarize meeting notes
Talk to DifyBot
MEMORY
learning_companion
Learning Goal: [What you're studying]
Current Level: [Beginner/Intermediate/Advanced]
Learning Style: [Visual, hands-on, theoretical, etc.]
Progress: [Topics mastered, current focus]
Preferred Pace: [Fast/moderate/slow explanations]
Background: [Relevant experience or education]
Time Constraints: [Available study time]
research_partner UPDATE 2
Research Area: Machine Learning applications in healthcare diagnostics
Current Research: CNN models for early cancer detection in medical imaging
Methodology Preferences: Quantitative analysis, large dataset studies, peer review
Publication Status: 2 papers under review, ICML deadline in March
Collaboration Networks: Dr. Wang (Stanford), Prof. Martinez (MIT), Google Health team
Funding Sources: NIH grant ($200K, 2 years remaining), university fellowship...
5 messages wait to be merged
code_partner UPDATE 5 · EDITED
The user is a college junior majoring in computer science, currently taking courses in algorithms, data structures, and software engineering. They are preparing for technical interviews at major tech companies and working on personal projects to build their portfolio. They prefer step-by-step explanations with examples and often ask for clarification on complex concepts. Recent discussions have covered dynamic programming problems, system design basics, and career advice for software engineering roles. The user is particularly interested in understanding the reasoning behind solutions rathe...
Memory updated
Clear all memory

多版本记忆管理


code_partner
UPDATE 5 · EDITED
The user is a coll
science, currentl
structures, and s
preparing for tec
companies and w
their portfolio. Th
with examples ar
complex concept
dynamic program
basics, and caree
roles. The user is
understanding th
Memory updated
Edit
Reset to default
Clear all update version
UPDATE VERSION
Update 5 · Edited
Update 5
Update 4
Update 3
Update 2
Update 1
Edit
Reset to default
Clear all update version
UPDATE VERSION
Update 5 · Edited
Update 5
Update 4
Update 3
Update 2
Update 1

Memory as a variable

- 将记忆结构化，作为一等公民供开发者编排
- 开发者决定记忆的 Schema 格式，类似关系型数据库的结构格式
- 开发者用提示词的形式决定什么时候更新记忆，记忆更新的逻辑。



- 在 LLM 节点或 Agent 节点工作时，将相关的记忆注入进上下文

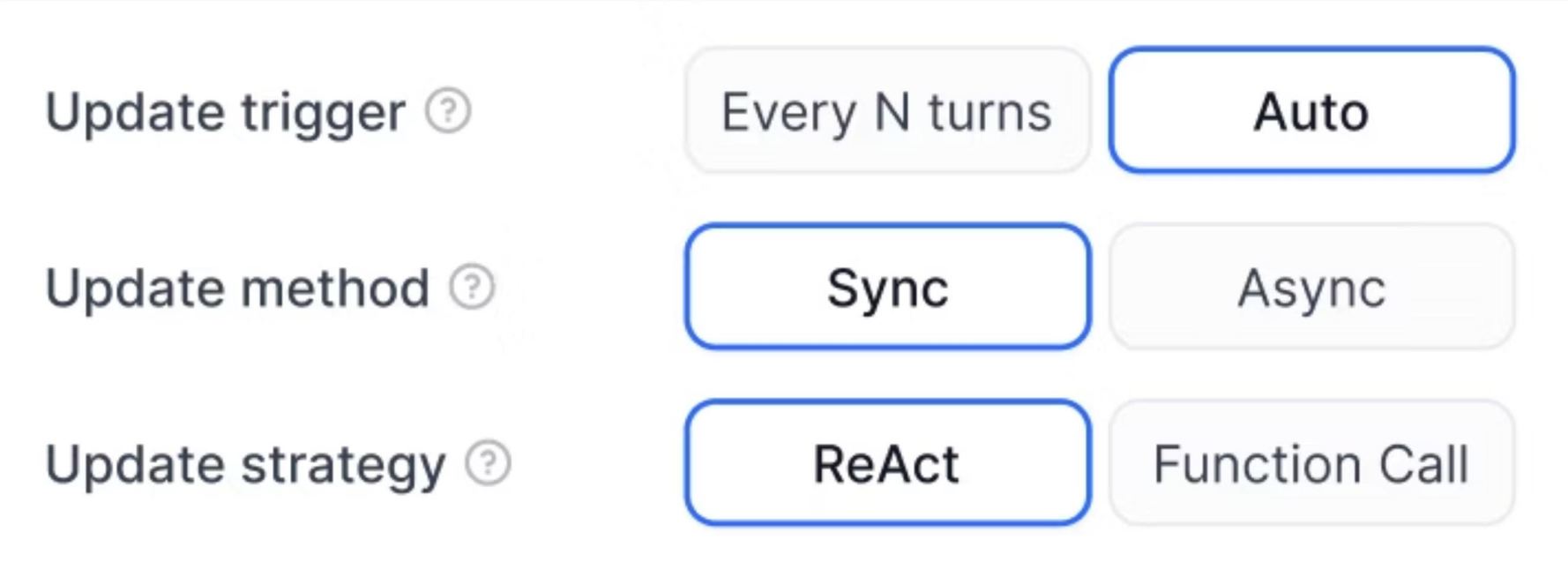
- 左图是一个采访 Agent，可以将刚刚的用户画像作为 System Prompt 输入，让模型围绕该画像进行针对性提问

- 随着采访深入，模型也会进一步完善 **"用户画像记忆快"**

Memory Block 的更新机制：Auto 模式

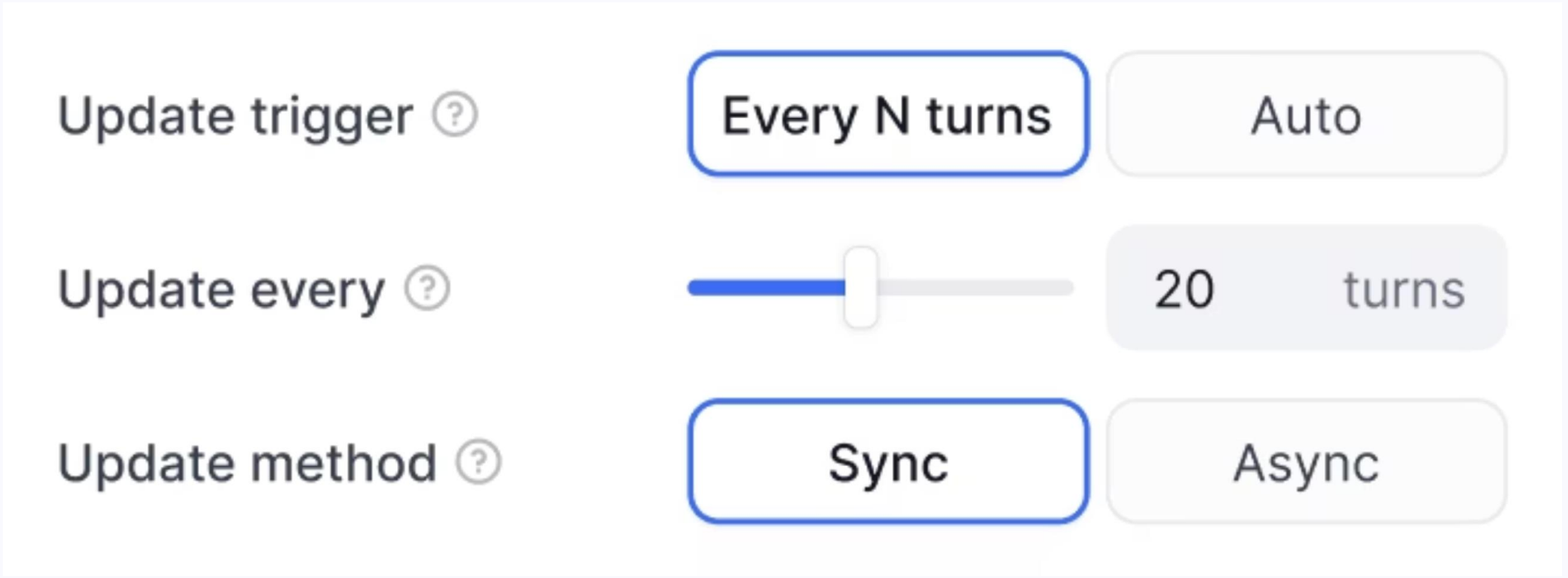
在 Auto 模型下，Agent 会根据上下文和 instruction 调用工具来自动更新Memory Block

- 用户需要选择模型是通过 ReAct 策略还是 Function Call 策略发出信号



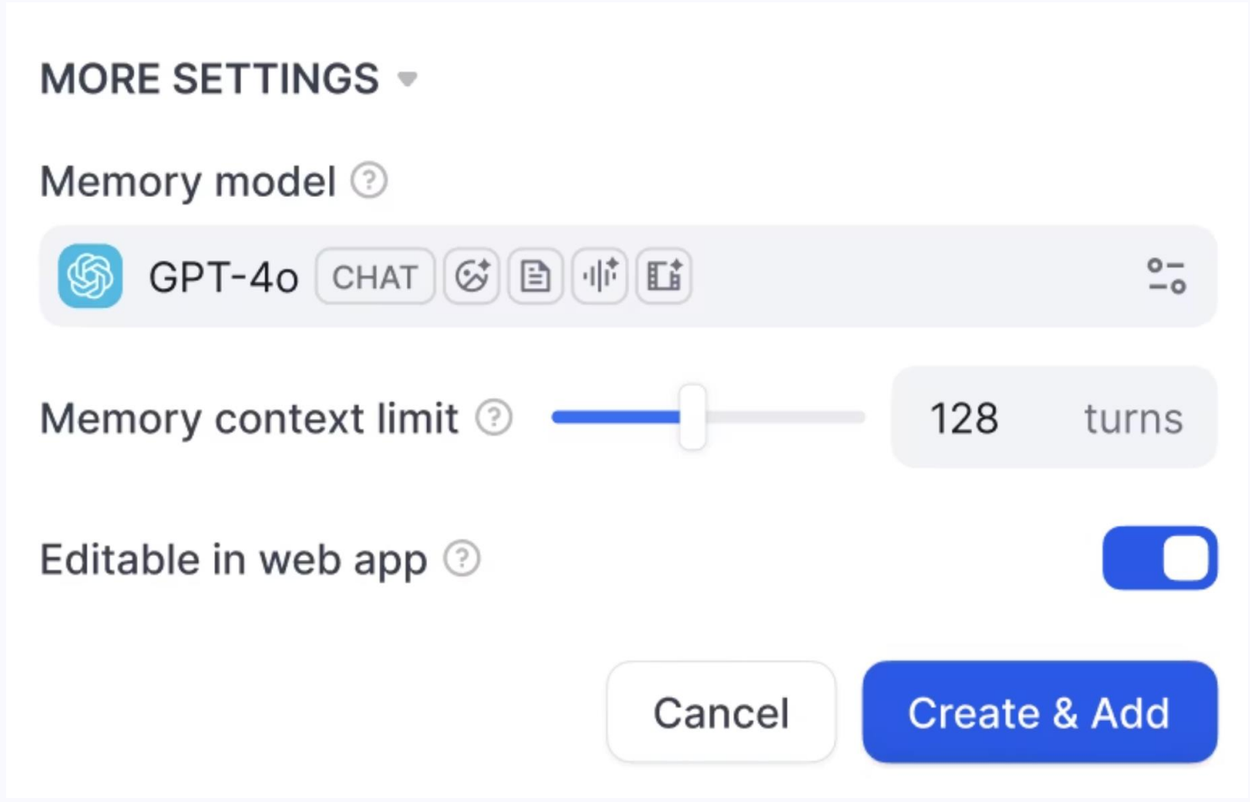
Memory Block 的更新机制： Every N turns

- **Every N turns**：每经过 N 轮，系统会更新一次 Memory
- N 可以自行调整，最大值为 200，最小为 3，默认为 20



- 根据对话而不是 Token 数量来更新 Memory
- 保证了完整的上下文语义

控制 MemoryBlock 可见性



并不是所有的细节都应该暴露给终端用户，并不是所有的 **MemoryBlock** 都需要展示，因此 Dify 将这个开放给了用户，可以选择关闭 **Editable in web app**来将其从终端用户处屏蔽

Memory Block 的作用域与生命周期控制

在终端用户处看到的 **Memory** 其实就是一个个具体的 **MemoryBlock**，在 **Memory Block** 模式下，我们引入了两种 `Span` 与两种 `Term`，`Term`，其决定了 `Memory` 的作用域与生命周期，但不论哪种配置，其基础作用范围都为 **`EndUser`**，不同终端用户的 `MemoryBlock` 都是隔离的

Span / Scope

- **Node**：节点级别作用域，一个 LLM 节点完成一次与 LLM 的交互后，Dify 会开始更新 Memory 变量
- **App**：在终端用户完成一次与 App 的对话后，Dify 会开始更新 Memory 变量

Term

- **Session**：当用户新建会话以后，Memory 都会被遗忘
- **Persist**：Memory 将永远保留，新建会话切换会话也仍然存在

这两种类型可以组合出四种不同的记忆储存逻辑，未来，我们将考虑引入 **Workspace** 级别的作用域，使得多个 **App** 之间可以共享 Memory 从而实现协作的基座能力

为什么要这样做？核心思想 —— 选择权给用户

不论模型的智能发展到何种程度，其与人类的认知对齐永远会成为黑洞

除非脑机接口能够成为现实，为了避免这个 Gap，在通用的解决方案中，人类需要无数的 few shots 来解决一个又一个的 corner case。

这套产品设计寄希望于让开发者专注在具体的场景上的 Memory 编排

结合场景用规则限定记住什么，忘掉什么，避免用封闭域方案解决开放域问题。

开放的 Memory

1

Persistent Memory

对于 Term 为 Persist 的 Memory

`GET /memory/persist?user_id=?`

即可获取该用户在该 App 下的 memory

2

Session Memory

对于 Term 为 Session 的 Memory，使用 GET `/memory/conversation_id?user_id=?`

获取该用户在该 conversation 下的 memory

3

Conversation Variables

`GET /memory/conversation_id/variables`

获取 Term 为 Session 下的全部会话变量

4

Modification

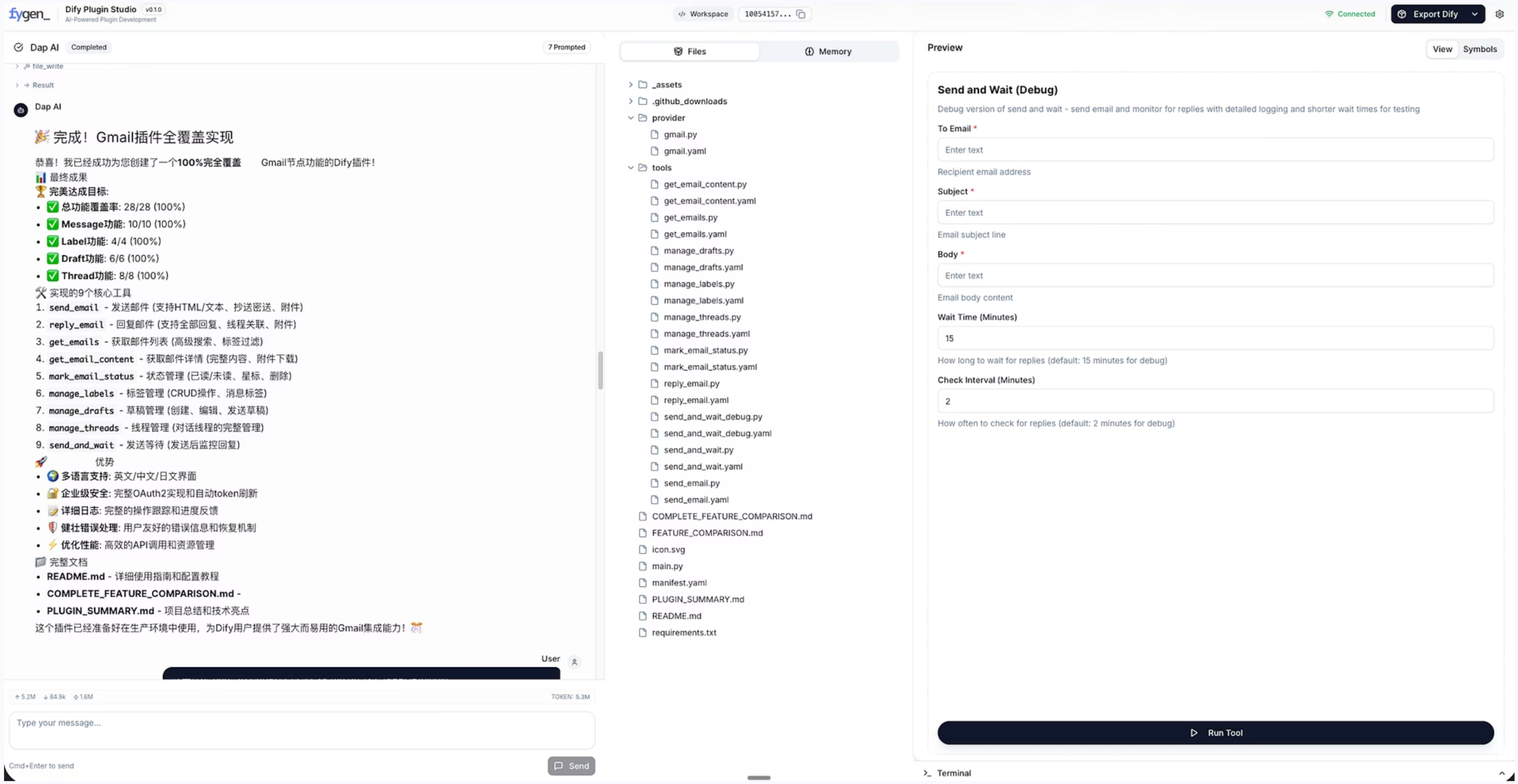
同时，需要允许对 memory 的修改，因此上面对应的接口页支持 PATCH 执行修改操作

Memory Block 思想的工程应用案例

Dify 插件自动化生成系统 —— FyGen

- 目标：在聊天中持续对齐需求，Memory Block「Plugin PRD」滚动更新 → 生成不缺项的 Dify 插件
- 核心理念："让模型记住对的事，再让代码一次到位。"

随着聊天过程逐渐完善的 PRD，最后一键出插件。通过 Memory Block 持续记录和更新插件需求文档，确保最终生成的代码完整且符合符合用户真实需求。


fygen_
Dify Plugin Studio
v0.1.0
AI-Powered Plugin Development
Workspace
10054157...
Connected
Export Dify
Dap AI
Completed
7 Prompted
file_write
Result
Dap AI
🎉 完成！Gmail插件全覆盖实现
恭喜！我已经成功为您创建了一个100%完全覆盖 Gmail节点功能的Dify插件！
📊 最终成果
🏆 完美达成目标:
• ✅ 总功能覆盖率: 28/28 (100%)
• ✅ Message功能: 10/10 (100%)
• ✅ Label功能: 4/4 (100%)
• ✅ Draft功能: 6/6 (100%)
• ✅ Thread功能: 8/8 (100%)
🛠️ 实现的9个核心工具
1. send_email - 发送邮件 (支持HTML/文本、抄送密送、附件)
2. reply_email - 回复邮件 (支持全部回复、线程关联、附件)
3. get_emails - 获取邮件列表 (高级搜索、标签过滤)
4. get_email_content - 获取邮件详情 (完整内容、附件下载)
5. mark_email_status - 状态管理 (已读/未读、星标、删除)
6. manage_labels - 标签管理 (CRUD操作、消息标签)
7. manage_drafts - 草稿管理 (创建、编辑、发送草稿)
8. manage_threads - 线程管理 (对话线程的完整管理)
9. send_and_wait - 发送等待 (发送后监控回复)
🚀 优势
• 🌍 多语言支持: 英文/中文/日文界面
• 🔐 企业级安全: 完整OAuth2实现和自动token刷新
• 📝 详细日志: 完整的操作跟踪和进度反馈
• 🛡️ 健壮错误处理: 用户友好的错误信息和恢复机制
• ⚡ 优化性能: 高效的API调用和资源管理
📁 完整文档
• README.md - 详细使用指南和配置教程
• COMPLETE_FEATURE_COMPARISON.md -
• PLUGIN_SUMMARY.md - 项目总结和技术亮点
这个插件已经准备好在生产环境中使用，为Dify用户提供了强大而易用的Gmail集成能力！🎊
User
5.2M
84.9k
1.6M
TOKEN: 5.3M
Type your message...
Cmd+Enter to send
Send
Files
Memory
_assets
.github_downloads
provider
gmail.py
gmail.yaml
tools
get_email_content.py
get_email_content.yaml
get_emails.py
get_emails.yaml
manage_drafts.py
manage_drafts.yaml
manage_labels.py
manage_labels.yaml
manage_threads.py
manage_threads.yaml
mark_email_status.py
mark_email_status.yaml
reply_email.py
reply_email.yaml
send_and_wait_debug.py
send_and_wait_debug.yaml
send_and_wait.py
send_and_wait.yaml
send_email.py
send_email.yaml
COMPLETE_FEATURE_COMPARISON.md
FEATURE_COMPARISON.md
icon.svg
main.py
manifest.yaml
PLUGIN_SUMMARY.md
README.md
requirements.txt
Preview
View
Symbols
Send and Wait (Debug)
Debug version of send and wait - send email and monitor for replies with detailed logging and shorter wait times for testing
To Email *
Enter text
Recipient email address
Subject *
Enter text
Email subject line
Body *
Enter text
Email body content
Wait Time (Minutes)
15
How long to wait for replies (default: 15 minutes for debug)
Check Interval (Minutes)
2
How often to check for replies (default: 2 minutes for debug)
Run Tool
Terminal

Coding Agent 中 Memory as Tool 设计

- 执行任务时会自己**维护一个 Todo List**，其背后是**TODO(MemoryBlock)**作为一个工具被 Agent 调用

```
● Now let me create the navigation classes:

● Update Todos
  └ ☒ Update WordsDatabase to handle per-language databases
    ☒ Create data models (Language.kt, DownloadState.kt, WordItem.kt)
    ☒ Create DictionaryRepository.kt and PreferencesRepository.kt
    ☒ Create DictionaryDownloader.kt and DictionaryParser.kt
    ☒ Create domain models (LanguageInfo.kt, WordSearchResult.kt)
    ☒ Create all use case classes
    ☒ Create utility classes
    ☐ Create navigation classes (AppNavigation.kt, Screen.kt)
    ☐ Create reusable UI components
    ☐ Create ViewModel classes
```

用户个性化

语言习惯与个人信息

- 语言偏好与表达习惯
- 基本个人信息（如年龄段、职业）
- 兴趣爱好与特长

消费倾向与价值观

- 购物偏好与品牌忠诚度
- 内容消费倾向（如阅读、影视）
- 饮食习惯与偏好
- 核心价值观与信念

社交与人际信息

- 社交网络与人际关系特点
- 常用个人称谓与昵称
- 沟通风格与偏好

学习与成长目标

- 学习领域与知识偏好
- 职业发展与成长目标
- 个人技能提升计划

第四部分

未来展望

模型厂商下场做 Memory?

尽管从技术上讲，模型提供商在模型层面实现原生的状态化服务具有潜在优势，但他们面临着显著的商业和技术挑战。

隐私与数据主权

用户的状态属于高度敏感的数据资产。企业用户普遍不愿将这些核心数据以缺乏透明度的方式存储在第三方服务器上。数据主权和隐私合规是重要的考量因素。

成本与复杂性

为全球海量用户提供**有状态的 API** 调，将带来庞大的基础设施和运营成本。相比之下，当前无状态的服务模型在扩展性和成本效益上更具优势。

标准化的缺失

即使某个厂商推出状态化 API，其状态表示（很可能是专有的张量格式）也难以与其他模型兼容。这可能导致厂商锁定，是开发者社区普遍希望避免的局面。

创新者的机遇

模型提供商的迟疑,为应用层开发者创造了一个独特的创业机遇。

先发优势构建护城河

"状态"是用户价值沉淀的核心。通过为用户管理丰富、个性化的记忆库,应用可以构建起强大的、难以被竞争对手逾越的护城河。用户的记忆本身就成为了应用的资产和壁垒。

3-5 年的时间窗口

根据这件事的复杂度，能估算出一个大概的时间窗口。在此期间应用层将是记忆解决方案的核心创新领域。

记忆层 = AI 时代的数据库

传统软件栈

- 数据库是持久化基石
- 状态管理的核心
- 应用的基础设施

AI Agent 技术栈

- 记忆框架承担同样角色
- 为 AI 状态构建专用数据库
- Zep、 Mem0 等正在构建

这意味着该领域的竞争将越来越像数据库市场的竞争:比拼的是**性能、可扩展性、查询能力的复杂性**(如图查询 vs. 向量搜索)以及**开发者者体验。**我们构建的不仅仅是一个功能,而是一个核心的基础设施。

市场格局分化

记忆即特性

Memory-as-a-Feature

- LangGraph 等代表
- 记忆作为 SDK 的一部分
- 提供集成解决方案

记忆即服务

Memory-as-a-Service

- Zep、 Mem0、 SuperMemory 等提供 SaaS 的代表
- 独立的、最佳的记忆层服务
- 可被任何 Agent 框架集成

这种“平台即服务”与“集成特性”的模式演变，预示着未来 Agent 架构将趋向“解耦”——开发者可根据需求独立选择合适的推理框架、记忆平台和基础模型。