使用指南

Niurx

2025 年 8 月 15 日

导言

这是早期版本中演示沙盒的使用指南。本沙盒是一个用于演示流图式沙盒的构建和使用的示例。

在阅读本指南前，请先阅读同一目录下的 1.intro.md 文件和 2.graph.md 文件来获取对 Hevno 沙盒的基本理解。沙盒就是酒馆中的角色卡

这是一个具有一定门槛的新平台，请保持足够的耐心。

如果你发现有些部分看不懂，在同一目录下有一个 code_collection.md 文件，那里面是 Hevno 的几乎所有代码。你可以直接把里面的内容全部贴给 Gemini 然后说你有什么问题。经过测试，Gemini 可以准确回答几乎所有问题。

1 沙盒概述

图 1: 默认沙盒概览

如图**??**所示，Hevno 的沙盒分为三个区域：definition、lore 和 moment。每个区域下都会以类似文件夹的形式展示。

- **Definition**：定义区域，决定了一个沙盒在第一次启动时的初始状态。其中的 initial_lore 和 initial_moment 文件夹会在沙盒初次运行时拷贝到 lore 和 moment 区域。
- **Lore**：传说区域。这里包含了所有不会随着读档而改变的内容。它可以被视为角色卡的静态部分。
- **Moment**：刹那区域。这里包含了所有会随着读档而改变的内容。你每进行一次对话，都会产生一个新的 moment 区域。在你回到某个历史时刻时，moment 区域会恢复到当时的状态。

1.1 为什么要这么设计?

想象这样一个场景：你在探索中遇到了一只巨龙，LLM 在你和巨龙的战斗中描述巨龙释放了一个强大的远古魔法。

这个魔法是在你遇到巨龙时才诞生的吗? 并不是。这个魔法存在于整个游戏世界的历史中，只是在现实里这个魔法直到你遇到巨龙时才被 LLM 编写出来。

那么如果你不幸被巨龙杀死了，你回档到了某个历史时刻，重新开始挑战巨龙，此时我们需要确保巨龙依然会释放那个强大的远古魔法，LLM 需要能够重新获得那个曾经被他创造过的魔法的描述。

这就需要一个跨越读档的机制来确保巨龙的远古魔法不会因为你回档而消失。这就是 lore 区域的作用。你可以在 lore 区域存放一本关于巨龙的书，里面记载了巨龙所会的一切。

所有的存档都会共享同一个 lore 区域，这样就可以确保你在任何时候遇到的巨龙都能使用同样的远古魔法。

如果你对酒馆的提示词模版有了解，你可以认为 lore 就是提示词模版的 local variables 部分，而 moment 就是 message variables 部分。

2 图

下面请点那个名为 main 的图标最右边的铅笔。main 是我们的主图，管理着 LLM 的执行流程。在点击后你应该会看到类似图**??**的界面，如图**??**所示。

图 2: 主图概览

图中的每个条目是一个节点。节点代表了某些功能或操作。你可以通过点击节点来查看其详细信息。每个节点有三个区域，节点 ID、依赖于和指令列表。让我们来以生成思考链节点为例，如图**??**所示。

- **节点 ID**：每个节点都有一个唯一的 ID，用于标识和引用该节点。
- **依赖于**：这个区域列出了当前节点依赖的其他节点。只有当这些依赖节点完成后，当前节点才能执行。（大部分情况下你不需要手动指定这个区域，如果你不明白这个区域的作用，请不要填写）
- **指令列表**：这是一个指令列表，LLM 会根据这些指令来执行相应的操作。

2.1 生成思考链节点

下面我们来详细讲解指令的意思。当前生成思考链节点具有一个 llm.default 指令。让我们点击右侧红框框处的小钱包图标来查看指令的详细信息，如图**??**所示。

图 3: 生成思考链节点

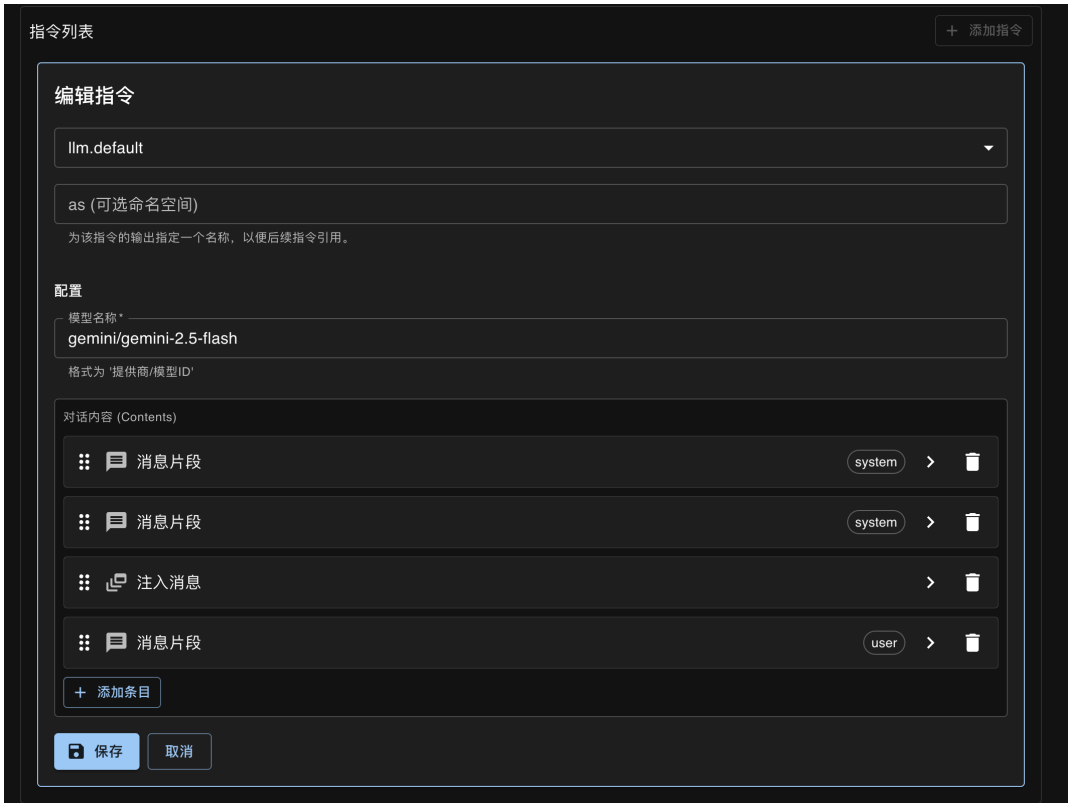

图 4: llm.default 指令

展开后的面板的第一栏用于切换不同的指令类型。当前我们选择的是 llm.default 指令。

第二栏 `as` 可以直接忽略。我们以后再讲解。

第三栏，用于配置这个 LLM 调用使用什么模型。当前我们使用的是 gemini-2.5-flash。

第四栏，对话内容。这是 llm.default 指令的核心内容。你可以视为酒馆的预设。

让我们逐一检查每个部分。首先让我们来看前两条，如图**??**所示。

如果你熟悉酒馆的预设，你会很容易地发现第一条是一条普通的 system 指令。

第二条就比较复杂，让我们来拆分一下。首先，这条 prompt 前后被双大括号包裹，这意味着这是一个宏，就像酒馆的宏一样用来执行一些高级功能。

但是不同于酒馆，Hevno 的宏不是一些固定的命令，而是任意的 pythonn 代码。这意味着你可以在这里编写任何 Python 代码来生成你想要的内容。以这条为例，其首先构建一个字符串，其中的关键内容是尝试获取一个叫做“已知信息”的节点的输出，将其注入到本条指令中，如果没有获取到，则输出“目前对世界一无所知”。

图 5: llm.default 指令的前两条内容

那么已知信息是什么节点？让我们先略过这一点，继续看下两条指令，如图**??**所示。

图 6: llm.default 指令的后两条内容

你会发现第三条指令叫做注入消息。这个指令是用来注入历史聊天记录到 prompt 的。其实际实现也是用宏，其从一个叫做获取聊天记录的节点获取到了 output，将其作为这条的实际内容。

获取聊天记录是什么节点？也不用管，我们以后再讲。

最后一个条目同样使用了宏来嵌入内容，被双大括号包裹的 moment._user_input 就是获取当前用户输入的内容。moment 就是刹那区域的意思，其中包含了本次会话中的所有变量。

那么这个节点做了什么事情？这个节点进行了一次简单的 LLM 调用，生成了一个思考链。这就是 Hevno 和旧的酒馆的最大区别，我们不再用一次 LLM 调度来完成所有消息的生成，而是将每个消息的生成分解为多个节点，每个节点负责一个小的任务。比如目前我们所在的生成思考链节点，它只负责生成思考链，而不是整个消息。

那么这个思考链生成了以后要去哪里？这个思考链会被存储在变量系统中。

后续的节点可以用 {{nodes. 生成思考链节点.output}}来获取到这个思考链。

比如说下面的生成回复节点。

2.2 生成回复节点

生成回复节点同样具有一个 llm.default 指令。其中的前两个条目如图**??**所示。

图 7: 生成回复节点

在这里我们就能发现第二个条目使用{{nodes. 生成思考链节点.output}}来获取到这个思考链。并将其作为本次 LLM 调用的 prompt 的一部分。

这就意味着我们将思维链的生成完全地分离出来了。不会再有思维链太重导致带不动的问题。无论我们的思维链有多复杂，我们最后都能用一次独立的 LLM 调用来生成回复。完全不会有任何注意力不足的问题。

2.3 还有更酷炫的

让我们查看“更新知识库节点”，如图??所示。

这个功能比较复杂且需要一定的代码基础，如果看不懂，请不要担心。你只需要大概了解这个功能能达到什么效果即可

图 8: 更新知识库节点

这个节点有两个指令，一个是 llm.default 指令，另一个是 system.execute

让我们先来看 llm.default 指令，如图??所示。这是一个普通的 LLM 调用指令。其内容是分析刚刚的思考链中，AI 是否获得了新的知识，并用一个 JSON 来描述所有知识。

图 9: 更新知识库节点的 llm.default 指令

在获取这个 JSON 后，下面的 system.execute 指令会执行一个 Python 代码来将这个 JSON 中的内容更新到知识库中，如图??所示。

看不懂没关系，这段代码是 Gemini 写的我也看不懂（乐）

这个 Python 代码会将 JSON 中的内容添加到一个名为 knowing 的 codex 中。

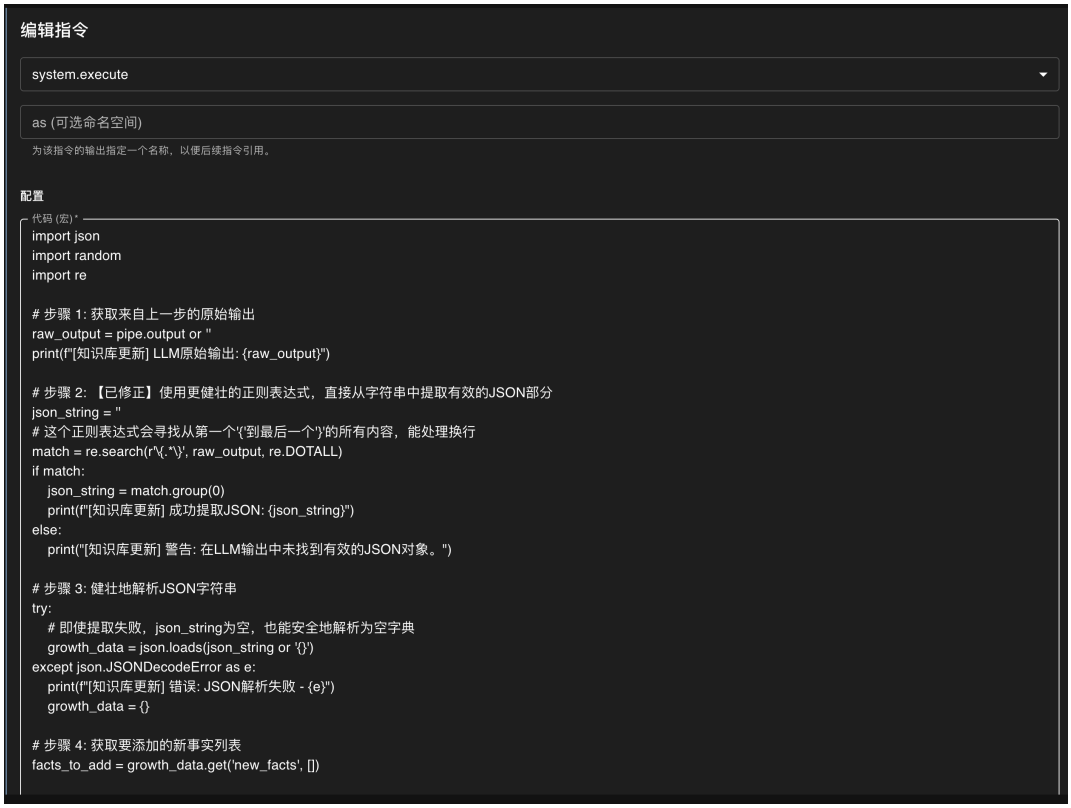

图 10: 更新知识库节点的 system.execute 指令

这是什么东西？codex 其实就是世界书，但是不同于酒馆的世界书，codex 可以主动选择用什么样的内容来触发关键字，并动态地在对话中添加条目。现在再让我们去看那个叫做“已知信息”的节点，如图**??**所示。

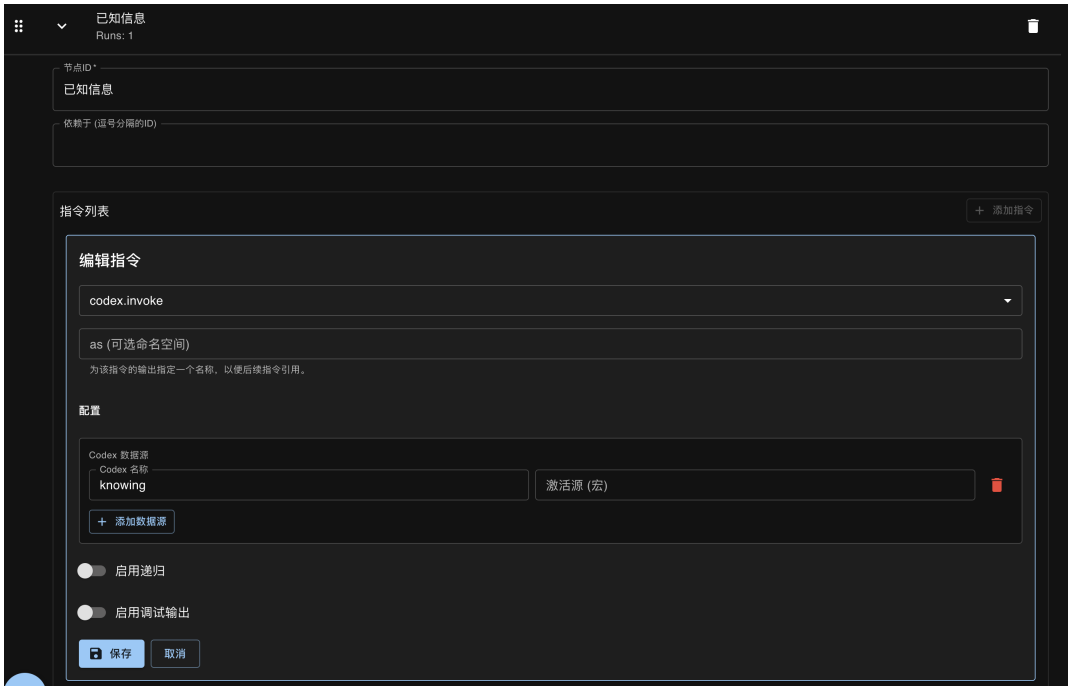

图 11: 已知信息节点

这个节点的功能就只是激活一个叫做 knowing 的 codex，并将其内容作为输出。并如同你在图**??**和图**??**中看到的那样，被多个 LLM 节点所调用。

2.4 小总结

这个沙盒展示了什么?

这是一个很简陋的沙盒，其主要功能就是用于演示如何将 LLM 的执行流程分解为多个步骤进行。比如生成思考链和根据生成的思考链生成回复被完全拆分。如果你愿意，你也可以在生成回复节点后加上一些格式修正、文风施加等节点。

另一方面，这个沙盒展示了动态添加世界书条目的功能，并演示了如何深度嵌入 Python 代码来实现复杂的功能。事实上添加世界书条目的功能并不需要这么复杂的 python 代码。未来会被更加简单的接口代替。这里只是用于展示 python 嵌入的可能性。