基于 AI 时代的程序设计与计算机体系结构变化

曾锦程

XXX 大学 XX 系

提交日期：2026 年 2 月 3 日

摘　要

大语言模型的快速发展显著提升了代码生成的自动化水平，但其“百衲衣”式的能力结构——从海量去脉络化文本中归纳统计模式——使其难以形成服务于特定个体的“关系总和”，导致在实际交互中产生间歇性失忆、逻辑断裂、幻觉等问题。本文从AI 编程的基本模式出发，提出“AI 递归”与“编程 AI”两类核心范式的划分，并在此基础上构建了一套以“思路”为中心的智能系统理论框架。

本文的主要贡献包括：（1）提出 AI 递归的概念，区分同质递归与异质递归，给出多模型递归的可靠性数学表达；（2）将计算机三大基本结构（顺序、选择、循环）与人类三大思维（归纳、类比、递归）及马克思主义三大规律（量变质变、对立统一、否定之否定）进行理论对应，建立三层理论体系；（3）提出“思路论”及其核心单元——思路结点，定义双层存储架构（调用存储器与结点存储器分离），实现思路的去冗余存储、版本化复用与动态演化；（4）设计并实现第一类思路机系统——DeepSeek 思维链对话客户端，该系统以 Tkinter 构建图形界面，支持与 DeepSeek API 及 Ollama 本地模型的交互，具备流程图绘制、思路结点存储、语义检索、交互式迭代优化等功能，初步验证了 AI 外置记忆循环结构的可行性。

本文的研究表明：将 AI 从“一次性生成工具”转变为“可积累、可复用、可演化的思路载体”，是提升智能系统可靠性与个性化适配能力的可行路径。在此基础上，本文进一步提出多层外置记忆循环结构与“智脑”概念，为未来智能系统的记忆机制设计提供了理论参考与实践原型。

关键词：AI 编程；AI 递归；思路论；思路机；外置记忆循环；思维链；可靠性建模

目 录

第一章 引言

1.1 研究背景

当前，以大语言模型为代表的人工智能技术正以前所未有的速度渗透至软件开发的各个环节。从 GitHub Copilot 到 Cursor，从 DeepSeek Coder 到 Claude 3.5 Sonnet，AI 辅助编程已成为开发者日常工作中的重要组成部分。据统计，截至 2025 年底，超过 60%的专业开发者每周至少使用一次 AI 编程工具，代码生成、调试辅助、架构建议等功能的普及显著提升了软件开发效率。

然而，效率提升的背后，AI 编程的局限性也逐渐显现。大模型的核心能力源自对海量文本数据的统计学习——它从互联网上的代码仓库、技术博客、问答社区中汲取养分，拼凑出“看似合理”的答案。这种能力结构，本文称之为“百衲衣”困境：AI 什么场景都能覆盖，但没有任何一片布料是专属于某一具体个体、某一具体关系脉络的。当用户在深度对话中提出个性化需求时，AI 会出现间歇性失忆、逻辑断裂、幻觉频发等问题。这些问题并非偶然，而是由其底层架构的固有缺陷决定的。

马克思在《关于费尔巴哈的提纲》中指出：“人的本质不是单个人所固有的抽象物，在其现实性上，它是一切社会关系的总和。”AI 恰恰缺失这种“关系总和”——它无法理解“谁在提问”“为何提问”“什么答案才是对该关系脉络真实的”。当前的主流解决方案，如检索增强生成（RAG）、长上下文窗口扩展、思维链提示等，本质上都是在试图用“更多信息”弥补“关系缺失”，但并未触及问题的核心。

1.2 问题的提出

基于上述观察，本文试图回答以下三个核心问题：

第一，AI 在编程过程中究竟扮演什么角色？ 当前对 AI 编程的理解往往停留在“辅助工具”层面，但实际使用中，AI 既可充当代码生成器，也可作为问题求解的主体。这两种角色有何本质区别？其背后的运行机制如何建模？

第二，如何克服 AI 的“一次性”局限？　大模型的训练数据冻结于某一时刻，模型自身无法在交互中持续成长。是否存在一种机制，让 AI 能够在与用户的持续对话中积累经验、迭代思路、形成记忆，从而逐渐“生长”出服务于特定个体的整体存在？

第三，AI 的“思维”能否被结构化存储与复用？　当前 AI 生成的思维链仅在单次对话中生效，无法作为可积累的知识资产。如果能够将每一次解决问题的“思路”存储下来，并在未来遇到相似问题时复用，是否能够构建一个可演化的智能系统？

1.3 研究思路与方法

围绕上述问题，本文从以下四个层面展开研究：

第一层：基础模式分析。从最简单的 AI 编程实践出发，区分“AI 编程”（AI 作为代码生成工具）与“编程 AI”（AI 作为问题求解主体）两类基本模式，并引入“AI 递归”概念，对多模型协同求解的可靠性进行数学建模。

第二层：理论体系构建。将计算机三大基本结构（顺序、选择、循环）与人类三大思维（归纳、类比、递归）及马克思主义三大规律（量变质变、对立统一、否定之否定）进行对应，提出“三层理论体系”，为 AI 时代的程序设计提供哲学与方法论支撑。

第三层：核心概念定义。提出“思路论”及其核心单元“思路结点”，定义思路结点的数据结构、存储架构、联系方式与可靠性模型。在此基础上设计“第一类循环单元”与“第一类思路机”，作为智能系统的核心运行机制。

第四层：系统实现验证。基于上述理论，开发 DeepSeek 思维链对话客户端。该系统以 Tkinter 构建图形界面，支持与 DeepSeek API 及 Ollama 本地模型的交互，实现了思路结点的绘制、存储、检索与迭代优化，初步验证了 AI 外置记忆循环结构的可行性。

1.4 论文结构安排

本文共分为八章，结构安排如下：

第一章为引言，阐述研究背景、问题意识、研究思路与论文结构。

第二章从 AI 编程的两类基本模式出发，提出 AI 递归的概念，给出同质递归与异质递归的数学描述，并定义 AI 最小循环单元。

第三章建立三层理论体系，将计算机三大结构与人类三大思维、马克思主义三大规律进行对应，提出广义化的程序设计框架。

第四章提出思路论，定义思路结点、思路分类、联系方式、存储架构与可靠性串联模型，并在此基础上构建第一类循环单元与第一类思路机。

第五章详细阐述思路机的计算机实现，包括数据结构设计、统一模型调用结点、核心部件功能与动态更新机制。

第六章介绍 AI 外置记忆循环结构的系统实现，包括数据库设计、检索加载流程、单结点动态更新机制与系统整体架构。

第七章在单层外置记忆的基础上提出多层外置记忆循环结构，并引入“智脑”概念，展望思路复用与知识共享的未来形态。

第二章 理论与方法

2.1 问题的起点

我们从最简单的 AI 编程问题出发。

当前对 AI 编程的普遍认识是：借助人工智能工具辅助生成、优化、调试代码，以及使用编程语言开发、训练、部署人工智能模型的综合性编程实践。

对比传统关于程序设计[1]的定义：设计、编制和调试程序的方法与过程。AI 编程丰富了传统程序设计的内涵，加入了 AI 元素。

而我将要讨论在这变化下的基本模式。

2.1.1 两类基本模式

我任意选用一个 AI，我称之为 A，它在解决一道编程问题时会生成一段参考代码，这串代码可以直接复制，大概率也可以直接使用，这毋庸置疑可以减少写代码和部分想代码的时间，让编程效率得到极大的提升。

当任意 AI 模型 A 解决编程问题时可以简单地分为两类基本模式 AI 编程和编程 AI。

模式一：AI 编程（AI 代码生成）

AI 自主生成完整的代码解决方案

输出为纯代码，AI 生成代码中不包含 AI 模型调用部分

核心特征：人类是思考主体，AI 是效率工具

1. 静态性：生成过程在编码阶段完成，运行时，代码固定
2. 一次性：AI 参与在代码生成阶段结束
3. 工具性：AI 作为高级代码生成器，类似编译器的高级形式

逻辑描述：最终代码=人类思考调试+AI 代码生成

模式二：编程 AI

人类在编程过程中主动引入 AI 作为编程唯一元素。

由人主动引导 AI 执行生成代码片段、优化结构、调试错误

人不直接参与到代码编程

核心特征：传统编程语言与人之间通过 AI 连接，人通过指挥 AI 解决问题。

数学描述：解决方案=人类编程 AI 工作流程+AI 按流程执行

2.2 AI 递归

当 AI 在编程时将自身或其它 AI 模型作为可调用工具时，就形成了 AI 递归结构。

2.2.1 AI 递归的基本形态

我将 AI 递归分为同模型递归和多模型递归。

2.2.2 同模型递归（同质递归）及其局限

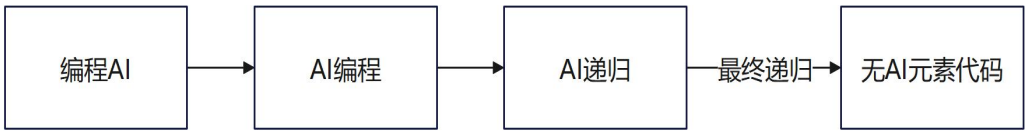
当递归调用采用同一 AI 模型时：

1.理想情况下同模型递归的数学极限：

如果每层递归能完全共享已执行的代码信息，不断更新已知条件，缩小解空间，则当递归层级趋于无穷时：

$$\lim_{k\to\infty} \text{解空间大小} = 0$$

问题自动坍缩为确定性的代码程序。



但前提是：AI 模型的信息储备和推理能力均为无穷大。

实际递归层级理想深度不为无穷。

定义最终递归所需深度为 P。

即任意问题通过同模型递归至少需要递归 P 次才能坍缩为确定性的代码程序。

在现实约束下，提出 AI 递归可靠性的量化框架：

定义对于特定问题，考虑到 AI 模型的信息储备和推理能力无法达到理想状态，AI 模型的可靠性百分比为：

$$r = \frac{n}{P}$$

其中：

n：实际有效递归深度；P：理论完全可靠深度（解决该问题所需的最小递归深度）

约束条件：0≤r≤1

2.死思考（Dead Thinking）现象——同模型递归局限性：

模型在自身输出上反复递归，每层递归不能完全共享已执行的代码信息。

缺乏新信息注入，陷入思维定势

除非出现 AI 幻觉等异常，否则无法产生实质性进展

2.2.3 多模型递归（异质递归）

多模型递归即递归过程中出现模型组合发生变化，使用模型类型发生变化。

第一个模型处理问题的初始部分，可靠性为：r1

未解决的部分（1-r1）由第二个模型处理

第二个模型处理剩余部分的可靠性为 r2，所以贡献为(1-r1)*r2。

以此类推

1.总可靠性公式

设总可靠性为 R，则：

$$R = r_1 + (1 - r_1)r_2 + (1 - r_1)(1 - r_2)r_3 + \cdots + \left(\prod_{j=1}^{m-1}(1 - r_j)\right) r_m$$

更紧凑的写法：

$$R = \sum_{k=1}^{m} \left(r_k \cdot \prod_{j=1}^{k-1}(1 - r_j) \right)$$

其中约定： $\prod_{j=1}^{0}(1 - r_j) = 1$

剩余不可靠部分经过所有 m 个模型处理后，仍然未解决的部分为：

$$E = \prod_{j=1}^{m}(1 - r_j)$$

可以用数学归纳法证明 R+E=1

满足归一化要求。

所有模型可靠性相同（ri=r），即坍缩为同模型递归的理论模型。

$$R = 1 - (1 - r)^m$$

$$E = (1 - r)^m$$

从总可靠性公式出发，通过设定测量条件（尝试次数等于理论完全可靠层数 P），可以自然推导出模型可靠性百分比的定义式。

最终公式：

$$R = 1 - \prod_{j=1}^{m}(1 - r_j) = 1 - \prod_{j=1}^{m}\left(1 - \frac{n_j}{P_j}\right)$$

$$E = \prod_{j=1}^{m}\left(1 - \frac{n_j}{P_j}\right)$$

2.2.4 AI 最小循环单元

AI 递归的思想着毋庸置疑是一个循环结构，我将其称之为 AI 最小循环单元。

AI 最小循环单元理论上只需要程序员执行编程 AI 后就可以通过 AI 递归得到一份完全可靠的代码。

该代码能解决任意问题的范围只受到 AI 模型的信息储备和推理能力的范围限制。

第三章 三层理论体系的对应关系

在 AI 时代，加入 AI 元素后的编程理论应该有新的形态。

传统的三大基本结构：顺序结构、选择结构、循环结构，显然需要进行一定的扬弃。这种扬弃应该是对基本概念的一种拓展，毕竟传统的编程理论建于一座逻辑十分严密的数学地基之上，正好，这个地基也为 AI 时代的编程理论提供了有力支撑。

3.1 程序设计广义化

我现在要做的是将原本局限于代码的过程式编程的理论广义化。

我认为，人类三大基本思维中的归纳、类比、递归与三大基本结构的顺序、选择、循环一一对应，后者可以说是前者在操作上的一种表现。

同样我发现，马克思主义基本原理中的量变质变理论、对立统一理论以及否定之否定理论与上述理论中的元素一一对应，也有异曲同工之处。

这样我将这三个不同领域却互有相关的理论结合了起来。马克思主义基本原理和人类思维为问题的直接语言理解提供了方法论，而计算的编程理论又为问题求解提供了对应的可执行计算机操作。

该框架可以写为：

归纳（量变质变） → 大语言模型：海量的数据（量变）训练后，模型涌现出理解和生成能力（质变）。

类比（对立统一） → 选择判断语句：通过比较事物间的相似与差异（对立统一），进行分类和选择。

递归（否定之否定） → 会更新变量的循环：在自我调用和迭代中，状态不断被扬弃和更新，实现螺旋式上升。

或者写为：

归纳 = 量变质变 = 顺序结构（大语言模型）

类比 = 对立统一 = 选择判断语句

递归 = 否定之否定 = 会更新变量的循环

以上等于为弱等于关系。

3.2 广义化的出发点

为什么会这样去进行广义化描述呢？

人工智能是为了去令机器具备自主思考的能力。这就像就像人类一般，人的思考，人的思维人的智能应该与人工智能的真正实现有着本质的共通之处。

而关于人类思维发展的一般规律，其实马克思主义基本原理里已经显事的给出了。这就是马克思主义基本原理三大思维发展的规律。

这三大规律在人类思维发展的过程中是富含的，是动态的，并无明确成分比例的的，它们应该在处于实践中处于解决不同问题时给出不同的组合包含关联的形式。

将这些优美的理论与计算的基本理论相结合，这边是我的思路。

更甚者，如果哲学这门学科是关于解释世界的或者用以改变世界的学科，那么我觉得若能把计算机理论同哲学理论相联系起来，那么，将为这两门学科都带来前所未有的生机和活力。举个例子，维根斯坦的“语言的边界即是世界的边界”或者说“语言即世界”不正在被大语言模型所论证着吗？

3.3 三个领域的根本统一性

下面我将来论证他们的相关之处，以及描述他们在互相关领域是如何进行表述的。

三个领域的理论称呼不同，但却连着同一个根系。

归纳、顺序结构（ai 模型）、量变质变理论；类比、选择结构、对立统一理论；递归、循环结构、否定之否定理论。

它们应该就是三种极致抽象的三种衍生表述。

我将之一一对应为：单维线性，维度重设，维度变换。

三种基本结构的跨维度阐释：

1. 顺序结构的本质

顺序结构就像流水，由始至终，包罗万象，但却被时序严格的约束着方向，**归纳**如同不同密度的流体在流水不同速率范围内的沉积物，**AI 模型**以其强大的流动性和流量用极高的维度换取全空间位置的拟合覆盖。

2. 选择结构的本质

选择结构即是分支，它受其它维度的因素影响使得单维度的线性发生状态的变化，**类比**便是不同维度的互相映射，维度为 n 的世界在基础的空间坐标系或者其他理论坐标系存在着相似的投影，受其影响，我们的直觉所在维度发生了状态的变化，**对立统一**便是描述这个维度影响过程中的维度线性无关的直接冲突交汇点。

3. 循环结构的本质

循环结构即是无限维度中最佳视角即不同实践坐标系下的最佳特征向量的拟合。循环之递归寻求最适解，能量最小解。**否定之否定**更像是一个无限迭代的的过程，如果不穷尽最佳视角的潜力或者思维的耐心，我们就不可能思考对自身基的变换。

计算机三大基本结构的实现简单易懂，采用任意编程语言便可在计算机中尝试实现。

从计算机理论走向人类思维乃至马克思主义基本原理的实现则需要一个关键的桥梁——量变质变的实现（AI 大语言模型）。

这是归纳特征后通过无数维度参数的拟合描述最终映射在所需解空间维度得到的。

因此接下来对三层理论体系进行讨论最终融合成一套基本的操作方式是我接下来要做的工作。

3.4 顺序结构

顺序结构是程序设计领域中常用的基础程序结构，具有执行顺序自上而下、依次执行的特性，其结构简单且应用广泛。

该结构按照语句书写顺序逐行执行，常见于输入、计算、输出组成的线性流程。

如果从一封闭程序观察，其顺序结构语句的安排必为认为事先确定。

看似与任意复杂逻辑无关的顺序结构，却是经过人脑深思熟虑组合后排列而出的结构。其有序性便与自然所规定的无序发展有着截然不同的区别，若非人为，思维发展中绝无理想状态的顺序结构。

为了给出其在 AI 下的全新形态，须融合注意力机制、自动顺序执行思想。

这是一个思路与其余思路的接口。

一个顺序的开始必然是无数往事的奠基下的注意力选择，显然基于注意力机制理论下的大语言模型很好拟合了这一功能。

AI 顺序结构即 AI 注意力模型调用。

3.5 选择结构

选择结构是计算机编程中用于根据条件判断控制程序流程的基本控制结构，通过条件表达式判定执行不同代码块，其核心通过关系表达式和逻辑表达式组成条件判断，结果以布尔值决定分支路径。这是传统选择结构的基本概念。

而对于思维发展来讲，我们的选择必是基于我们所有思想、所处立场、所有喜好、所在环境、所学知识等等各方面的影响下做出的。

一句话概括，思想是行动的先导。

AI 选择结构即 AI 基于注意力机制下受 AI 思维引导作出选择的结构。

该理论必须融合对立统一理论、包含通过关系表达式和逻辑表达式组成条件判断。

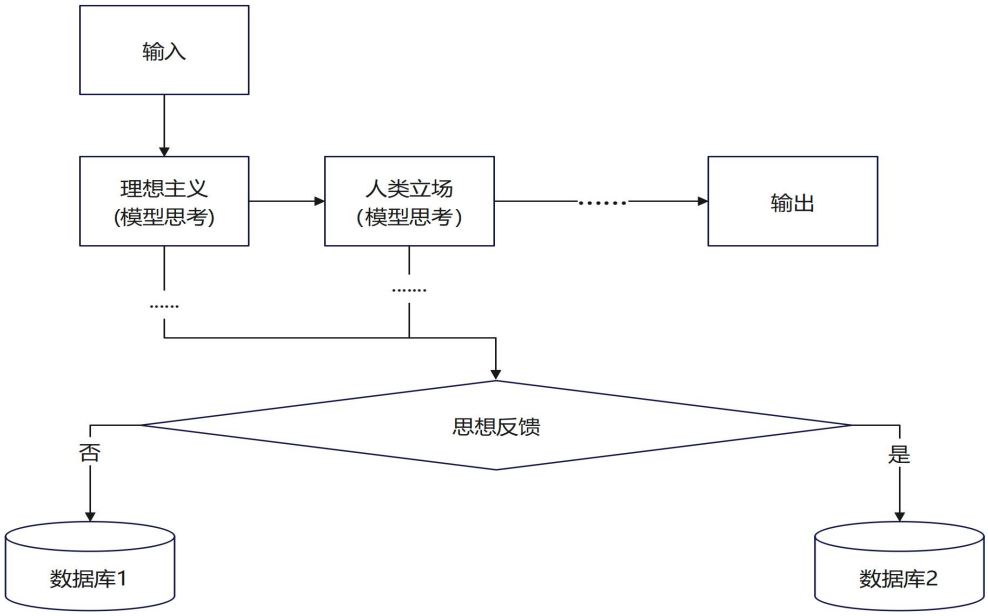
此处设立两种理想机器。

一是过滤器。

通过 AI 思维在自我讨论过滤思路的结构。

以 AI 模型本身作为讨论选择的核心，基于“思想是行动先导原理”，对模型设定思想、立场、主义、世界观、价值观、“人生观”等等过滤结点，符合条件的 token 自然通过，不符合条件的 token 被过滤。

过滤器模型：



二是对立统一分析机，对立统一规律是唯物辩证法的根本规律，亦称矛盾规律，指事物内部对立面之间既统一又斗争推动发展的基本原理，由毛泽东在《矛盾论》中确立为唯物辩证法最根本的法则。其核心内容包括矛盾的同一性与斗争性（相互依存又相互排斥）、普遍性与特殊性（共性个性关系）两对特性。

该规律揭示矛盾存在于一切事物发展过程，普遍性体现为“事事有矛盾，时时有矛盾”，特殊性要求具体问题具体分析。矛盾分析方法强调坚持重点论与两点论统一，抓住主要矛盾及矛盾主要方面。

直接调用模型本身即可做到对立统一分析的作用。

当得出的分析结果仍需通过过滤后得到预定的输出结果。

对立统一理论包含了选择结构的思想但其本身是一个深刻的理论概念，满足对立统一分析效果的结构数不胜数，但贯穿于思维发展的全过程，我们能做的应当是将其在需要时刻引导而出。

对立统一分析机引导：

最终分析结果应该作为过滤器的输入部分、输出部分以及过滤节点参考部分，按需选择分析结果的去处。

3.6 循环结构

循环结构是指在程序中需要反复执行某个功能而设置的一种程序结构。它由循环体中的条件，判断继续执行某个功能还是退出循环。根据判断条件,循环结构又可细分为以下两种形式:先判断后执行的循环结构和先执行后判断的循环结构。

AI 循环结构结合否定之否定理论，它们的关联是高度相似的。

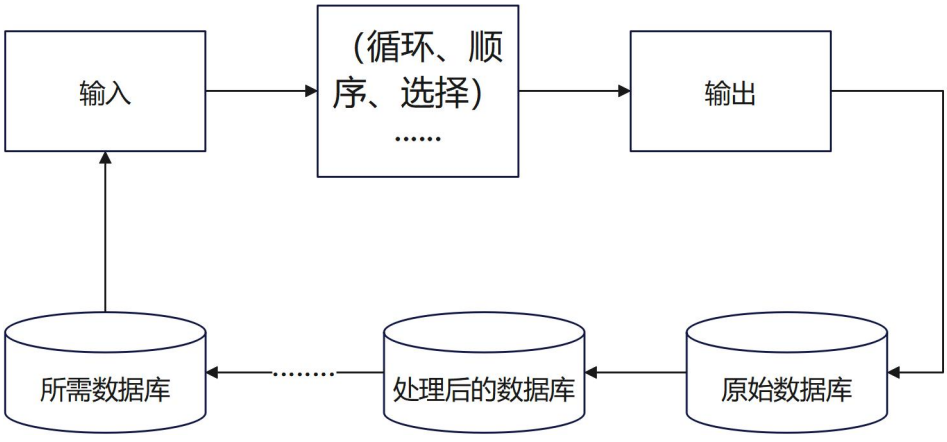
AI 循环结构由于 AI 模型的参数不变性，想要通过某一结构去实时更改 AI 参数是不现实的。

因此 AI 循环结构应当分为 AI 逻辑循环结构和 AI 外置记忆循环结构。

AI 逻辑循环结构即 AI 模型在编程过程中不变，但输入模型的的数据即输入的条件和注意力机制的结果会随着循环一次次的改变。其局限性就来自于模型本身，其发展方向简而言之就在与对于循环结点设计方向。

AI 外置记忆循环结构，即基于 AI 逻辑循环结构的基础上，将迭代过程信息有机的存储于外置的数据库中。该结构作拥有外置记忆存储结构，也就是说在循环过程中可以更改所调用的 AI 模型。并且我认为存储在数据库的数据必须方便用于模型微调或者是新的 AI 模型的训练。

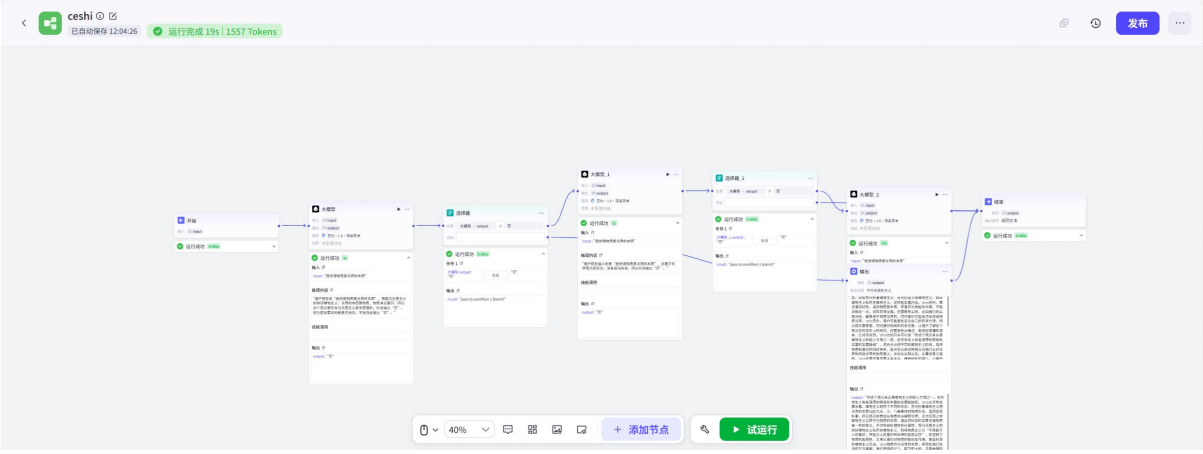
AI 外置记忆循环结构：



外置的数据库该怎样进行数据的分配处理，可以用 AI 辅助处理也可以人为的去设计，数据的更新以便与下次模型使用为原则。

3.7 过滤器实验

利用 cozi 工作流 if 语句，采用大模型智能判断识别，最终起到对不同思想进行过滤的作用。



设置了一个马克思主义过滤器，话题内容符合马克思主义就会正常跳到下一个结点，如果不符合就会在 if 判断中直接到结束结点并不输出内容，这样的过滤可以用许多思想的结点来逐渐使过滤器更加完善。

第四章 思路

下面我将以思路为基础，阐述一个具有可靠性的智能体系。

4.1 思路论

什么是思路？

我认为，思路是针对特定问题、在给定约束下、为实现目标而动态生成、可执行、可审议的步骤序列及其逻辑依据。它是认知活动与计算过程的统一体，是问题解决过程中思维运动的外化轨迹。

当我们去思考一个问题时，我们首先会有意识或者无意识的采用一套思路去解决问题，这个思路在短期内可能是变化较小的会受到我们的习惯、知识等等内在因素的限制，长期看人处于不同阶段的思路是动态变化的，思路会随着我们解决不同问题随着环境因素的改变，我们会适当的主动的或者被动的调整我们解决问题的思路。

而思路中的片段就是思维链。

思维链是宏观庞杂思路在微观局部上的映射。

而思维链在 IBM 上的定义是思维链 (CoT) 模拟人类的推理过程，通过一系列连贯的逻辑推导促进系统性的问题解决。

而 AI 中使用的思维链 (CoT) 提示是一种人工智能方法，通过将复杂的任务分解为一系列逻辑步骤，最终实现解决方案，以模拟类似人类的推理过程。这种方法反映了人类智能的基本特征，提供了一种结构化的问题解决机制。换句话说，思维链 (CoT) 基于认知策略，将复杂问题分解为可管理的中间思想，然后依次引导至最终答案。

思维链最直观的例子就是目前手机 APP 里的 deepseek 会在每次输出前给出一系列它的思考过程，当它的思考过程逻辑自洽后它才会把答案输出。

毋庸置疑这是一种很好的提升 AI 可靠性的方法。

但是，在实际 AI 使用过程中，我们会发现 AI 出现间歇性失忆、AI 幻觉等等不可靠的问题。

人之智能区别于简单的 AI 模型的本质在于，人之智能会因自身所处关系总和而形成一套整体，该整体明辨只服务于该关系总和的虚实，因为人是一切关系的总和，AI 做不到这一点，因为 AI 是”百衲衣”，AI 归纳了大量的数据特征并从高维映射到我们任意所需的相关维度，但是它无法存在独立的“关系总和”更不用说去用它的关系总和形成思路生成思维链然后解决问题与某一个体关系总和相关的问题了。

因此我把 AI 缺陷归纳为以下两点：

1.AI 的“百衲衣”困境：

大语言模型通过海量数据训练，从无数异质、去脉络化的文本片段中归纳出统计模式，如同用千万块碎布拼成的“百衲衣”——它什么场景都能覆盖，但没有任何一片布料是专属于某一具体个体、某一具体关系脉络的。

2.马克思论断：“人的本质不是单个人所固有的抽象物，在其现实性上，它是一切社会关系的总和。”AI 的缺失：AI 没有属于自己的“关系总和”，无法形成服务于特定个体、特定社群的整体性存在，因此无法真正“理解”谁在提问、为何提问、什么答案才是对该关系脉络“真实”的。

因此为什么说我们要引入思路论。

思路论是要求 AI 在不断解决问题的过程中慢慢通过与某一个体交互，基于个体关系，形成整体存在形式，最终有一个存于存储器的整体能明辨只服务于该关系总和的虚实，该整体能生成思维链最终解决问题，并动态生长。

与某一个体交互，基于个体关系，形成整体存在形式，动态生长，这些是关键。

有思路就当有思路结点用于存储思路内容，那什么是思路结点呢？

4.2 思路结点

思路结点是构成思路的原子级逻辑与执行单元，是思路在空间维度上的基本存储单位与在时间维度上的最小执行片段。

思路结点与思路结点之间通过明确的输入/输出接口相互连接，形成有向无环或带环的逻辑网络。一个思路结点内部可以封装任意粒度的内容：

一个具体的函数调用或 API 请求；

一个 AI 模型的一次完整推理；

另一个完整的子思路（嵌套思路）；

一段人工操作指令；

一组条件判断逻辑；

一个预定义流程模板……

本质抽象：思路结点是“可执行的逻辑胶囊”——对外屏蔽内部实现细节，仅暴露输入、输出、终止条件、可靠性标签。

特别说明：子流程结点是思路论实现递归结构的关键——一个思路结点内部可以包含完整的子思路，子思路又由若干思路结点构成，从而形成任意深度的嵌套。这为复杂问题分解与知识复用提供了形式化基础。

4.3 思路分类

我目前将思路分为，基本思路、一般思路、简单思路、复杂思路四个类别。

基本思路：即作为思路的最小单元，可分为三大基本思路，即马克思主义基本原理的三个基本理论作为三大基本思路的核心，三个基本理论于基本思路的实现可以有不同编程的理论形态，但是其内核的丰富性一定是结合了哲学理论的深刻内涵的。

一般思路即面对某些对话会有的定式的回话或者面对某类问题动作会有对应定式的方法或动作进行反应，可由基本思路构成。

简单思路即根据已知的或经验的给出一套解决问题的流程和方法，可由一般思路或者加上基本思路构成。

复杂思路即结点中的内容无上限限制或者思路复杂度无限制要求的流程集合，可由其他三种思路复合构成。

思路组成存在形式分为两大类：有限思路（即思路由固定结点数、固定步骤数构成）、无限思路（即需要设定时间限制或设定结果目的预期限制）。

4.4 思路联系方式

由于思路结点与思路结点之间通过明确的输入/输出接口相互连接，形成有向无环或带环的逻辑网络。

思路的联系方式可以有：思路继承、思路整合、基于结果预判的思路联通开关、思路迭代更新联系、思路功能分化、思路数据存在结构。

思路继承，可分为全继承和部分继承，类似于编程中类的继承。思路通过思路继承可以从最初的思路进化到复杂的适应环境的思路。

思路整合，即降低思路的复杂度。两思路结点间间隔多个结点，但是由于一定条件下，两思路结点存在一定固定关系（可以是函数关系或其他关系），两思路结点间的多个思路结点可以被整合略去保留关系函数。

基于结果预判的思路联通开关，即两个思路结点连接之间可用一个模型预判器作为开关，模型会预测通过该思路后的结果，如果与输出的结果比较后差异不大即预判通过使思路正常执行。

思路迭代更新联系，设定思路更新框架，可以是循环更新也可以是人为更新，满足AI编程理论。

思路功能分化，即对本来处理单一功能的思路进行一定保留拓展后达到实现多功能的效果，类似细胞的分化，即由同一来源的思路经过选择调整表达产生于不同领域擅长对应领域工作的功能的过程。例如可以通过思路分化调整某思路向擅长动作或擅长思考的思路进行发展利于后续有关具身智能的研究。

思路数据存在结构，即对思路的存储数据结构进行研究，例如思路栈，可以起到对时间较近的思路进行优先选择的结构作用。

4.5 思路数据存储形式

由于思路结点只通过输入输出连接，思路结点可以作为外置数据存储在任意计算的外置存储器中，并且由于思路结点是根据个性化生成的，思路结点可以进行标号在进行

组合例如用单一的一系列数字号码，每个号码对应一个思路结点，存储号码标识的部分叫做思路调用存储器，存储思路结点内容的叫做思路结点存储器。

思路存储采用双层存储结构，思路调用存储器作为思路结点存储器的钥匙而存在。

因此我在此作如下定义。

思路数据存储采用双层存储架构，将思路的逻辑结构与原子内容分离，分别存储于两个独立且相互映射的存储器中。

思路 ID，唯一标识一个完整思路的全局标识符。

结点 ID，唯一标识一个思路结点的全局标识符。

调用序列，思路调用存储器中存储的有序结点 ID 列表，是思路逻辑骨架的核心表达。线性思路仅凭调用序列即可完整表示执行顺序。

版本化存储，结点存储器保留同一结点的历史版本，调用时可指定版本标签，确保可复现性。

双层存储的核心价值（后续研究按照需要可继续增加层数）

轻量化：思路复制、传输、版本对比仅需操作调用序列（ID 列表），成本趋近于零。

去冗余：内容哈希去重确保同一结点内容全局仅存一份，支持大规模复用。

可演化：思路生长仅修改调用序列，结点内容只增不改，历史版本可完全回溯。

可复用：子思路结点实现任意深度嵌套，调用序列即思路 DNA，支持生态级共享。

4.6 思路可靠性串联模型

在思路论中，一个完整的思路 S 由 n 个有序执行的思路结点 N1,N2,…,Nn 构成。每个结点 Ni 执行特定功能，其执行成功是一个随机事件。设：Ri=P(结点 Ni 执行成功),0≤Ri≤1

核心假设（待推导后讨论其合理性）：

步骤独立性假设：各思路结点的执行结果相互独立。

串联功能逻辑假设：思路整体成功当且仅当每一个思路结点均执行成功。任何一个结点失败，则整个思路失败。

可靠性静态假设：在当前执行轮次，Ri 是固定常数（由历史执行统计或先验评估给出）。

目标：求思路整体可靠性 R_S=P(思路 S 执行成功) 关于 $\{Ri\}_i^n{}_{=1}$ 的函数表达式。

基于事件独立性的直接推导

定义事件 Ai={结点 Ni 执行成功}，i=1,2,…,n。

思路整体成功事件为：

E=A1∩A2∩… ∩An

由概率的乘法公理：若事件 A1,A2,…,An 相互独立，则：

P(E)=P(A1)· P(A2)· … · P(An)=$\prod_{i=1}^{n} R_i$

因此：

$$R_S=\prod_{i=1}^{n} R_i$$

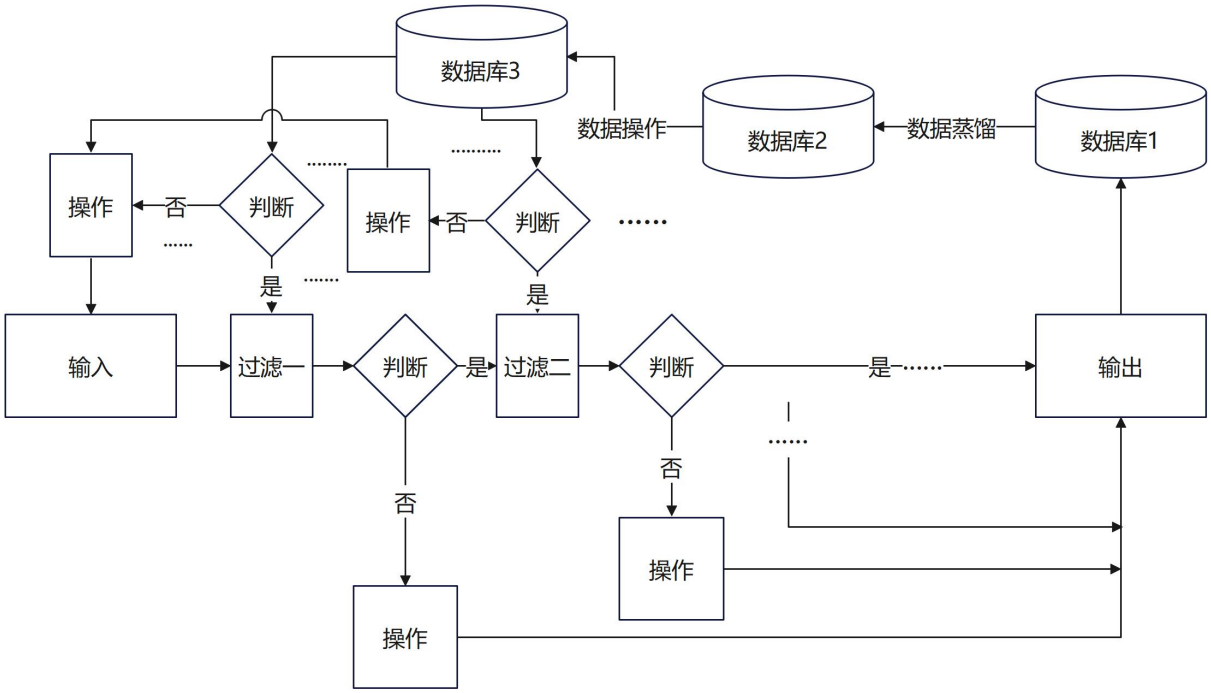
4.7 第一类循环单元

第一类循环单元由基本基本思路构成。

即通过量变质变的 AI 模型作为核心数据更新函数，利用对立统一思维的判断作为分析依据，最后通过否定之否定的结构完成迭代。

第一类循环单元是最小循环单元的思想继承。

下面给出一种第一类循环单元的模型。



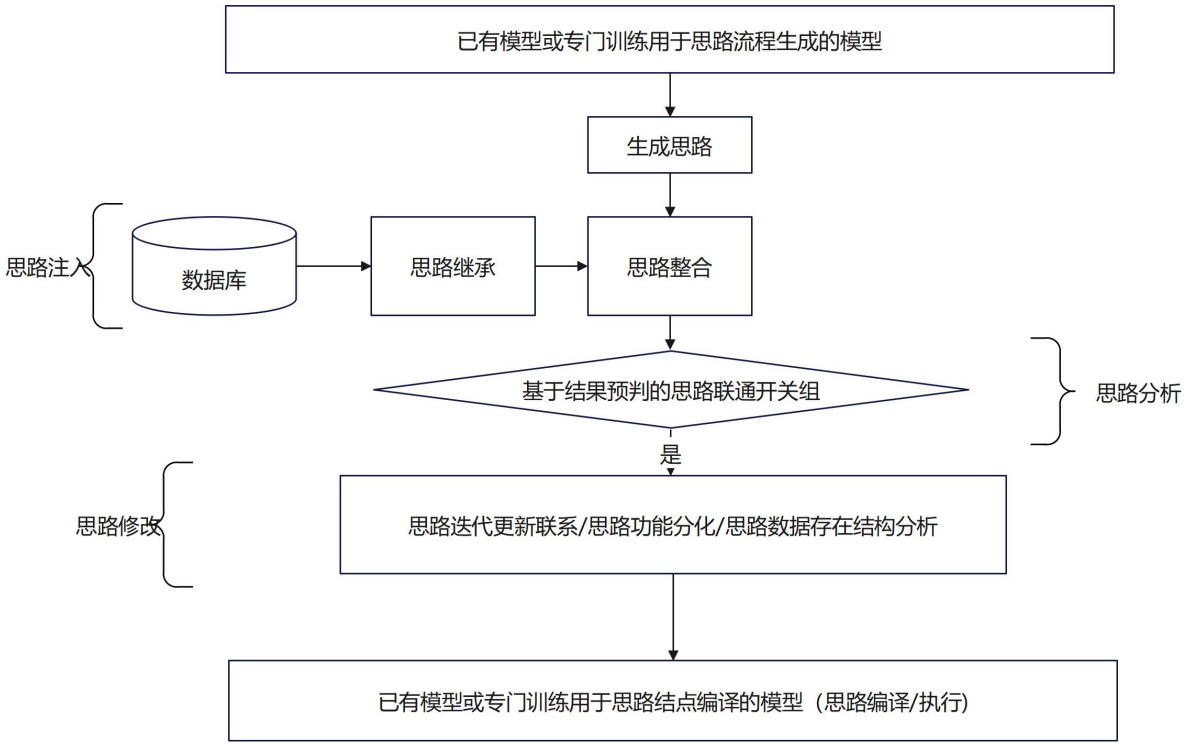
第一类循环单元由基本思路的编程思想构成，满足三大基本思想的闭环。按照思路组成存在形式设定对应有限无限状态。

4.8 第一类思路机

基于第一类循环单元，构建第一类思路机模型。

思路机：基于思路论的思想构建以思路分析作为处理问题核心的人工智能深化机器。

主要有以下四个部件：思路生成器、思路分析器、思路修改器、思路编译器。



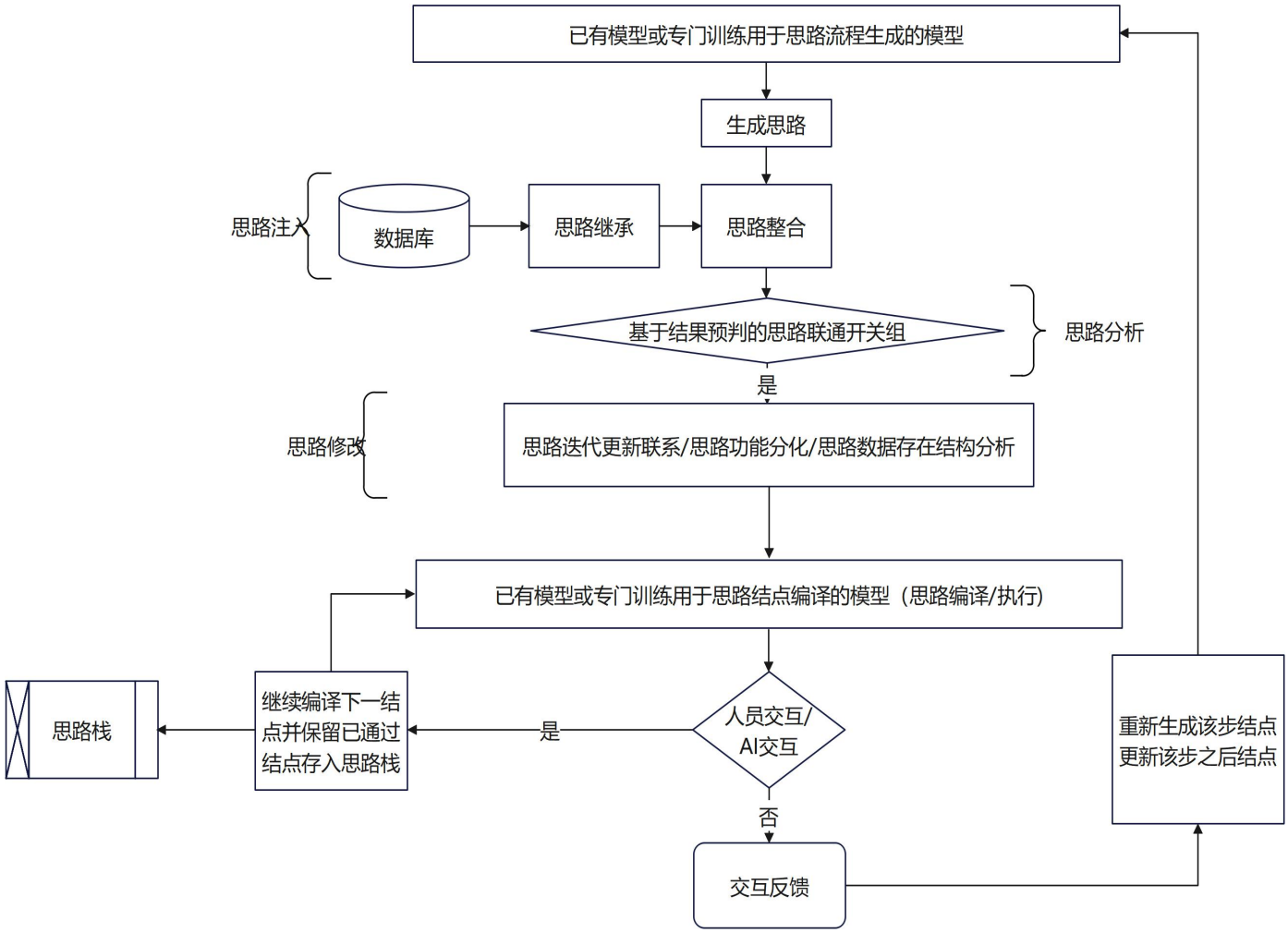
其中思路生成器、思路分析器、思路修改器以操作思路内容为主要任务，用于思路自动化生成的全流程。

思路编译器作为将已有思路转化为实际可操作步骤的部件，由于可编译内容是有限存储于编译器中的，而生成的思路结点的种类是无限的，这就需要通过第一类循环单元将不可被编译的思路结点分解为可编译思路结点的构成。

因此思路编译器必须做到能编译基本思路的能力，其余能力可靠性源于编译器所使用模型的可靠性，模型的可靠性可通过接入多个模型以及进行模型组合策略来提高总的可靠性。

4.9 单思路结点动态更新

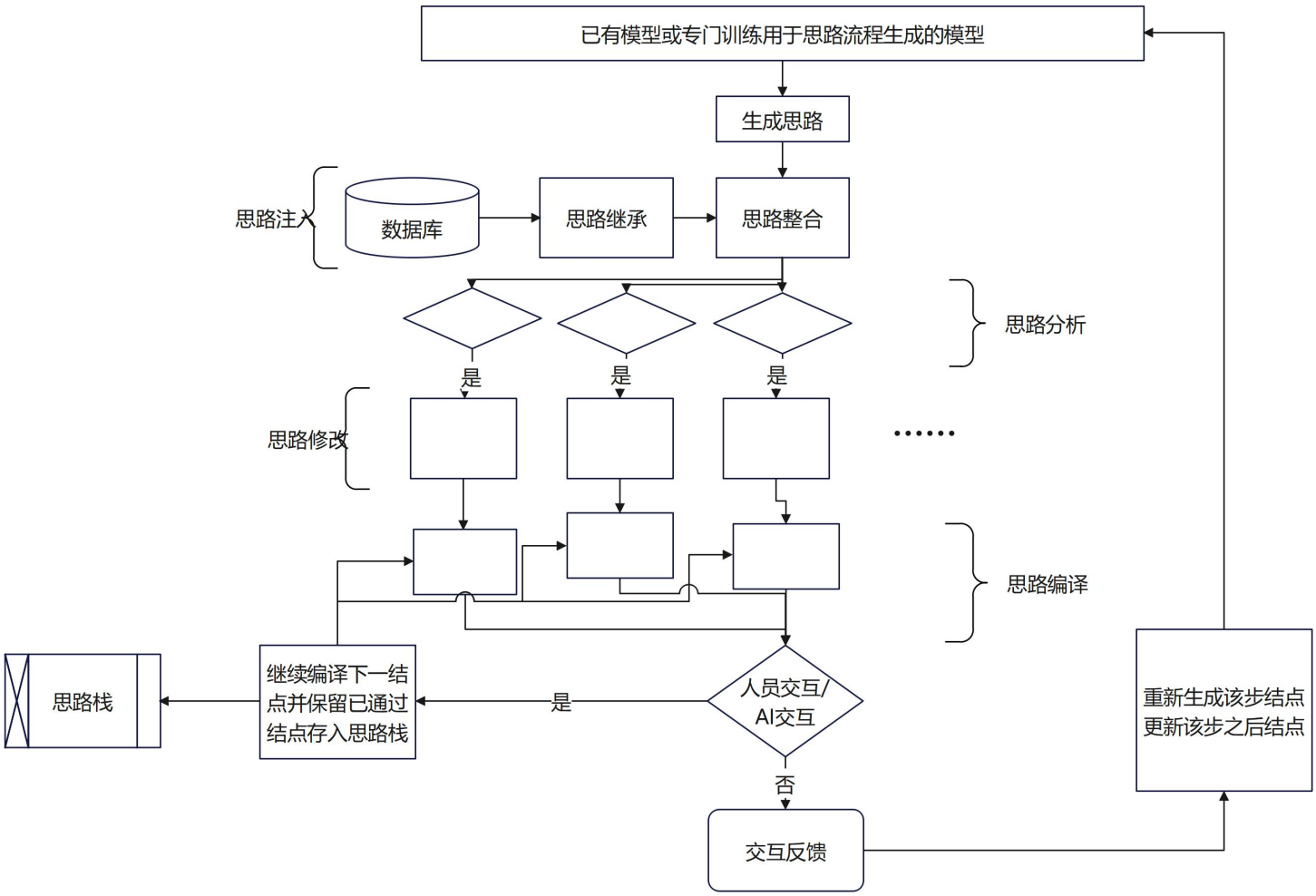
单思路结点动态更新必须通过与外界交互实现，因此可分为两个部分：思路机部分、交互部分。



单思路结点动态更新通过与 AI 交互，AI 单步执行思路结点，AI 能够通过交互反馈时事更新当前回答思路，当事件结束后，思路栈内信息保留。

注意思路不影响交互，只是为了后续更好的交互。

单思路结点动态更新在循环后的动态更新图如下所示。



4.9 多思路结点动态更新

多思路结点动态更新与单思路结点相比还需考虑思路数据存在结构的更新变化，若在多思路结点动态更新过程中始终满足思路栈的形式存储思路并且保证思路冗余不影响结果，那么多思路结点动态更新自动坍缩为单思路结点动态更新。

第五章 思路机的实现

在前文提出的“思路论”框架中，我们定义了思路结点、思路调用存储器、思路结点存储器等核心概念，并构建了第一类循环单元与第一类思路机的理论模型。本章将详细阐述如何将这些理论转化为可运行的计算机系统——DeepSeek 思维链对话应用，即思路机的一个具体实现。我们将从数据结构、核心部件、动态更新机制以及与外置记忆的交互四个方面展开说明。

5.1 思路结点的实现

链表是实现思路结点的最好结构之一，因为链表建立在指针的基础上可以表示复杂的数据结构也方便数据的增、删。

对于思路来讲，只要思路中的内容冗余不影响思路的最终输出，那么思路在整合前只需要不断进行增加删减即可。

大量复杂的思路结点内容存储在思路结点存储器中。

而一系列对于思路结点的拓扑操作基于思路调用存储器进行操作即可。

总结一句话，思路结点内容很难进行改和查，也无需进行改和查。

思路结点需要足够高可靠高效率的增和删的功能。

因此链表结构和指针操作作为思路结点实现形式是目前来讲的首选。

首先给出任意思路结点的基础模板。

==========

思路结点模板

==========

【结点 ID】 :<唯一标识符，建议使用 UUID 或 哈希值>

【结点名称】　　:<简要描述该结点的功能>

【结点类型】　　:<基本思路 / 一般思路 / 简单思路 / 复杂思路>

【创建时间】　　:<YYYY-MM-DD HH:MM:SS>

【版本号】　　:<v1.0.0>

【所属思路 ID】　　:<可选，指向所属完整思路>

【前置结点 ID 列表】　:<可选，指向当前结点的前置结点>

【后置结点 ID 列表】　:<可选，指向当前结点的后置结点>

【输入接口】　　:

- 参数名 :<类型>:<描述>
- 参数名 :<类型>:<描述>

…

【输出接口】　　:

- 返回值名 :<类型>:<描述>
- 返回值名 :<类型>:<描述>

…

【终止条件】　　:<描述该结点何时停止执行 / 输出结束标志>

【可靠性标签】　　:<0~1 之间的数值，表示该结点历史执行成功率>

【模型调用】 :<可选，如果该结点需要调用 AI 模型，则填写此项>

- 模型 ID :<模型唯一标识，如 "gpt-4", "claude-3", "stable-diffusion-v2">

- 模型类型 :<LLM / 图像生成 / 语音识别 / 其他>

- 模型参数 :<JSON 对象，如 {"temperature": 0.7, "max_tokens": 500}>

- 调用结点 ID :<指向统一模型调用结点的 ID，如果系统预定义了标准模型调用结点>

- 输入映射 :<描述如何将当前结点的输入参数映射到模型调用结点的输入>

- 输出映射 :<描述如何将模型调用结点的输出映射回当前结点的输出或中间结果>

- 调用方式 :<同步 / 异步>

【执行逻辑】 :

<封装的内容可以是以下任意一种，若【模型调用】存在，则通常先调用模型，再处理结果>

- 函数调用 / API 请求（非模型类）

- 子思路嵌套（指向另一个思路 ID）

- 人工操作指令

- 条件判断逻辑

- 预定义流程模板

- 混合逻辑（如模型调用结果后处理）

...

【备注】　　:<可选，补充说明>

==========

模型调用从执行逻辑中分离，通过统一的模型调用结点接口进行管理。同时保留执行逻辑，用于处理非模型调用的常规操作。这样设计既保持了思路结点的通用性，又实现了模型调用的标准化与复用。

为了支持上述模型调用的标准化，可定义专用的模型调用结点，所有模型调用请求都通过该结点执行。该结点本身也是一个思路结点，具有统一的输入输出接口，内部封装与具体模型的交互细节。

==========

统一模型调用结点模板

==========

【结点 ID】　　:<固定或动态生成，如 "model_caller_v1">

【结点名称】　　: 统一模型调用器

【结点类型】　　: 基本思路

【创建时间】　　:<固定时间>

【版本号】　　:v1.0.0

【所属思路 ID】　　:<无>

【前置结点 ID 列表】　:<动态连接>

【后置结点 ID 列表】　:<动态连接>

【输入接口】 :

- model_id : string : 要调用的模型标识
- model_params : object : 模型调用参数（如 temperature, max_tokens 等）
- input_data : any : 传递给模型的输入数据（如提示词、图像等）
- call_mode : string : 同步/异步（可选，默认同步）

【输出接口】 :

- output_data : any : 模型返回的结果
- error : string : 错误信息（如有）
- metadata : object : 调用元数据（如 token 消耗、耗时等）

【终止条件】 : 返回结果或错误后终止

【可靠性标签】 : 根据历史调用统计动态更新

【执行逻辑】 :

1. 根据 model_id 查找对应模型的 API 端点或本地模型路径。
2. 使用 model_params 和 input_data 构造请求。
3. 发起调用（同步或异步），等待响应。
4. 封装返回结果到输出接口。
5. 记录调用日志，更新可靠性标签。

【备注】　　：该结点为系统内置结点，所有模型调用请求统一经过此结点。

若思路结点需要调用模型：

在【模型调用】字段中填写所需模型 ID、参数等信息。

将【模型调用】中的 调用结点 ID 指向统一的模型调用结点（如上述模板的 ID）。

在【输入映射】中说明如何从当前结点的输入参数构造 input_data。

在【输出映射】中说明如何将模型返回的 output_data 映射到当前结点的输出或中间变量。

【执行逻辑】中可以包含对模型返回结果的后处理，或者如果模型调用后无需额外处理，可留空或仅透传结果。

若思路结点不需要调用模型：

省略【模型调用】字段，直接在【执行逻辑】中实现功能。

统一模型调用结点的作用：

封装所有与具体模型交互的细节（API 密钥、重试机制、错误处理、限流等）。

提供统一的输入输出格式，便于上层思路结点调用。

集中记录模型调用的可靠性数据，用于整体可靠性评估。

这种设计实现了模型调用与业务逻辑的解耦，提高了系统的可维护性和可扩展性。

5.2 思路结点的数据结构与存储

思路结点是思路机的基本操作单元，其数据结构直接决定了系统的表达能力与扩展性。根据我们给出的思路结点模板，我们在实现中采用了以下核心字段：

结点 ID：使用 UUID 或内容哈希值作为全局唯一标识，确保结点在不同会话中可被唯一引用。

结点类型：包括“基本思路”“一般思路”“简单思路”“复杂思路”，对应 4.3 节的分类。实际实现中，结点类型也用于流程图绘制的样式区分（如矩形、菱形等）。

内容：结点的核心文本描述，即该步骤的具体含义。

输入/输出接口：以 JSON 格式记录该结点期望接收的参数及输出的结果，便于结点间的数据传递。

可靠性标签：初始值为 1.0，后续可根据历史执行成功率动态更新（当前版本未启用，但预留字段）。

模型调用信息：若该结点需要调用 AI 模型，则记录模型 ID、参数、输入输出映射等，并指向统一的模型调用结点。

在物理存储层面，我们实现了 4.5 节提出的“双层存储架构”：

思路结点存储器：对应数据库中的 `flowchart_content` 表，以 `content` 和 `node_type` 为唯一约束，确保相同内容只存储一次。每个结点对应一条记录，包含完整的结点内容。

思路调用存储器：对应 `flowchart_session` 表，其 `node_sequence` 字段存储有序的结点 ID 列表，表示一个完整思路的逻辑骨架。通过这种方式，多个思路可以共享相同的结点内容，实现去冗余和轻量化复制。

此外，为支持快速检索，我们设计了 `retrieval_label` 表，存储每个思路会话的语义化标签，供后续自然语言匹配使用。

5.3 统一模型调用结点

在思路机中，AI 模型的调用被封装为一个特殊的思路结点——统一模型调用结点（以下简称“调用结点”）。该结点的模板已在 4.1 节末尾给出，其核心设计如下：

输入接口：接收 `model_id`（模型标识）、`model_params`（调用参数）、`input_data`（输入数据）等。

执行逻辑：根据 `model_id` 查找对应的 API 端点或本地模型路径，构造请求并调用，返回结果。

输出接口：输出 `output_data`（模型返回结果）、`error`（错误信息）、`metadata`（调用元数据，如 token 消耗）。

可靠性标签：根据历史调用成功率动态更新，为后续多模型递归提供数据支持（参见 2.2.3 节的多模型递归可靠性公式）。

任何需要调用 AI 模型的思路结点，只需在自身的“模型调用”字段中指定目标模型和参数，并将 `调用结点 ID` 指向统一的模型调用结点，通过输入/输出映射完成数据传递。这种设计实现了模型调用与业务逻辑的解耦，所有模型调用的细节（API 密钥、重试、限流、错误处理）均由调用结点集中处理，便于维护和扩展。

5.4 思路生成器与分析器

根据 4.8 节第一类思路机的定义，思路机包含思路生成器、思路分析器、思路修改器、思路编译器四个核心部件。在 DeepSeek 思维链对话应用中，这些部件的功能主要通过 AI 模型与用户交互共同完成：

思路生成器：当用户输入一个问题后，系统将问题作为提示词发送给 AI 模型，模型返回的推理过程（思维链）被解析为一系列思路结点，构成初始流程图。这一过程对应于“从零到一”的思路生成。

思路分析器：在交互模式中，用户可对当前思路进行“反思”操作。系统将当前会话上下文、最后一个高亮结点的内容以及用户的补充要求打包，发送给 AI 模型，模型重新生成该结点及其后续结点的内容。分析器在此过程中负责解析模型返回的新结点，并与原流程图合并。

思路修改器：对应“深入”操作。用户选中一个结点后，系统要求模型将该结点展开为更详细的子流程图，然后用子图替换原结点。修改器负责将子图嵌入主流程图，并更新结点间的连接关系。

思路编译器：负责将思路结点序列转换为可实际执行的步骤。目前，编译器的主要功能是将流程图渲染为可视化界面，并允许用户逐步骤执行（“自思考”功能）。未来版本中，编译器将支持将思路导出为可执行的代码或工作流脚本。

5.5 思路修改器与动态更新机制

4.9 节提出了“单思路结点动态更新”的概念，即通过与 AI 的持续交互，让思路结点内容随反馈而实时演化。在应用中，这一机制通过以下方式实现：

交互模式中的“反思”：用户可对最后一个高亮结点发起反思，系统将原结点内容、当前会话历史以及用户的新要求一起发送给 AI，AI 返回更新后的结点内容，并以 `[交互更新]` 的形式追加到流程图中。更新后的结点在数据库中作为新版本存储（保留原版本），但当前会话的 `node_sequence` 更新为指向新结点 ID。

“深入”操作：用户选中某一结点后，系统要求模型生成该结点的详细子流程，子流程本身也是一个完整的思路图。系统将子图的结点插入 `flowchart_content` 表，并用子图的 `node_sequence` 替换原结点在父序列中的位置，实现思路的嵌套与细化。

“裁去”操作：用户可指定从某个结点开始裁掉后续所有结点，系统直接截断 `node_sequence`，并释放不再被引用的结点（若其他会话未引用，则可物理删除或保留为历史版本）。

所有更新操作均实时保存到外置记忆（JSON 文件）中，并在流程图全亮时自动存入数据库，形成永久记忆。

5.6 第一类循环单元的实现

2.2.4 节定义的“AI 最小循环单元”和 4.7 节的“第一类循环单元”是思路机实现递归优化和可靠性提升的核心机制。在应用中，第一类循环单元表现为一种特殊的交互模式循环：

循环入口：用户发起一次对话，系统生成初始思路。

循环体：用户通过“反思”“深入”等操作对思路进行迭代优化，每次迭代都引入新的信息（用户反馈、模型重新推理），相当于 2.2.2 节中的“有效递归深度 n”。

循环终止条件：当用户对当前思路完全满意（即流程图全亮）时，循环终止，此时思路已“坍缩”为确定性的可靠程序（如可复用的解决方案）。若用户选择“继续交互”，则高亮结点数增加，进入更深层次的细化循环。

这一过程与否定之否定规律相吻合：每一次迭代都是对前一次思路的“扬弃”——保留合理部分，修正或细化不足部分，实现螺旋式上升。

5.7 与外置记忆的交互

思路机与外置记忆的交互通过“保存外置记忆”和“加载外置记忆”两个功能实现，其背后对应所述的双层存储架构：

保存外置记忆：将当前会话的 `node_sequence` 和所有引用的结点内容写入 JSON 文件，同时若满足全亮条件，则异步写入数据库。JSON 文件作为轻量级快照，便于用户手动备份和恢复；数据库作为长期记忆库，支持语义检索和大规模复用。

加载外置记忆：用户可选择从 JSON 文件恢复，或输入自然语言描述从数据库检索。检索时，系统将用户描述与 `retrieval_label` 表中的所有 `label_text` 进行 AI 匹配，返回最相似的 `session_id`，然后根据其 `node_sequence` 从 `flowchart_content` 中取出所有结点内容，重构流程图。

通过这种双层设计，思路机既具备快速手动保存/加载的灵活性，又拥有基于语义的智能检索能力，真正实现了“可复用、可演化”的外置记忆循环结构（见 3.6 节图）。

5.7 小结

本章详细阐述了思路机在 DeepSeek 思维链对话应用中的具体实现。我们以思路结点为原子单元，通过双层存储架构实现内容的去冗余与轻量化复用；通过统一模型调用结点封装 AI 交互细节；通过思路生成器、分析器、修改器、编译器的协同工作，支持思路的动态生成、分析与迭代优化；第一类循环单元则为思路的递归深化提供了理论指导。整个系统成功将抽象的“思路论”转化为可操作的计算机程序，验证了 AI 外置记忆循环结构的可行性与有效性，为构建具备长期记忆与自适应能力的智能系统奠定了实践基础。

第六章 AI 外置记忆循环结构的实现

在前文提出的“思路论”框架中，我们明确了思路结点、思路调用存储器、思路结点存储器的概念，并指出“AI 外置记忆循环结构”是实现思路动态生长与可靠复用的关键机制（见 3.6 节）。本章将详细阐述该结构的计算机实现：如何将交互过程中形成的思维链（即思路的实例化表达）持久化存储，如何基于语义检索快速加载历史思路，以及如何通过交互模式实现单思路结点的动态更新。最后，我们将展示一个完整的应用系统——DeepSeek 思维链对话客户端，作为上述理论的实践验证。

6.1 外置记忆存储系统设计

为了实现思路的“轻量化、去冗余、可演化、可复用”，我们设计了基于关系型数据库的三层存储架构，严格对应 4.5 节提出的“双层存储”思想。

6.1.1 数据库表结构

系统采用 MySQL/MariaDB 作为外置存储器，核心包含三张表：

| 表名 | 作用 | 对应理论概念 |

| `flowchart_content` | 存储原子化的思路结点内容，按 `content` 和 `node_type` 去重 | 思路结点存储器 |

| `flowchart_session` | 存储一次完整思路的调用序列，即有序的结点 ID 列表 | 思路调用存储器 |

| `retrieval_label` | 为每个思路会话生成语义化的检索标签，便于后续匹配 | 思路索引 |

`flowchart_content` 表结构

```sql
CREATE TABLE flowchart_content (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    content TEXT NOT NULL,
    node_type VARCHAR(50) DEFAULT 'rect',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

每个结点内容（如“分析问题背景”“调用模型 A 推理”）作为独立实体存储，相同的 `content` 和 `node_type` 组合只保留一条记录。这种设计实现了内容级去冗余，当多个思路共用同一结点时，无需重复存储。

`flowchart_session` 表结构

```sql
CREATE TABLE flowchart_session (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    mode VARCHAR(50),
    model_name VARCHAR(100),
    summary TEXT,
    node_sequence JSON NOT NULL
);
```

`node_sequence` 字段以 JSON 数组形式记录该思路所包含的结点 ID 顺序，例如 `[5, 3, 8, 2]`。这相当于 4.5 节中的“调用序列”，是思路逻辑骨架的核心表达。通过该序列，系统可以随时重建完整的流程图。

`retrieval_label` 表结构

```sql
CREATE TABLE retrieval_label (
    id INT AUTO_INCREMENT PRIMARY KEY,
    session_id INT NOT NULL,
    label_text TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (session_id) REFERENCES flowchart_session(id) ON DELETE CASCADE
);
```

`label_text` 是为该思路会话生成的语义化摘要，用于后续的自然语言检索。它并非简单的标题，而是通过 AI 对原始思维链内容进行提炼后形成的结构化描述。

6.1.2 存储触发条件：“全亮自动存储”

根据 4.1 节“思路论”的思想，只有经过充分交互、确认可靠的思路才值得存入长期记忆。因此系统设定存储触发条件为：**仅在交互模式下，且当前流程图处于“全亮”状态**（即高亮结点数 ≥ 总结点数）。全亮代表用户已确认该思路完整且可接受，此时后台自动将当前思路存入数据库。这一机制模拟了人类“巩固记忆”的过程——只有反复思考后形成的稳定认知才会转化为长期记忆。

6.1.3 存储流程

当交互模式中流程图全亮时，系统执行以下步骤：

1.遍历所有结点，对每个结点调用 `_get_or_insert_content_id` 函数：

- 若 `flowchart_content` 中已存在相同 `content` 和 `node_type` 的记录，则直接复用其 `id`；

- 否则插入新记录，获取新 `id`。

2. 构造 `node_sequence`，按流程图顺序将结点 ID 组织为 JSON 数组。

3.插入 `flowchart_session` 记录，同时记录当前模式、模型名称等信息。

4. 生成原始检索标签：取流程图“开头结点之后的内容”与“结束结点之前的内容”拼接，作为原始素材。

5.AI 格式化：调用大语言模型将原始标签转换为“开头内容、最终结果、最终目的”三段式结构化文本。例如：

开头内容：用户询问如何优化 Python 代码性能

最终结果：推荐使用 Cython 和并行计算

最终目的：提升代码运行效率

6. 插入 `retrieval_label` 记录**，关联 `session_id`。

整个流程在后台线程中异步执行，不影响用户交互。

6.2 记忆检索与加载流程

当用户需要复用历史思路时，可通过“加载外置记忆”功能从数据库检索并还原。检索过程充分利用了 AI 的语义理解能力，而非简单的关键词匹配。

6.2.1 检索流程

1. 用户输入自然语言描述，如“上次那个关于算法优化的思路”。

2. 系统查询 `retrieval_label` 表获取所有记录的 `id`、`session_id` 和 `label_text`。

3. 构造提示词，将所有 `label_text` 完整列出，要求模型“必须完整阅读所有内容后，选出最匹配用户描述的一个 `session_id`”。

4. 调用 AI 模型（与对话所用模型一致）进行匹配，解析返回结果中的 `session_id`。

5. 若 AI 匹配超时或失败，则降级为字符串模糊检索，以保证可用性。

注：目前应用尚未考虑有多个选择的情况该怎么做，当根据记忆数据库存储结构，思路存储结构形式的理论可以解决相关问题，即思路栈或别的数据结构的特征可以很好的解决这一问题。

6.2.2 加载流程

1. 根据 `session_id` 从 `flowchart_session` 获取 `node_sequence`。

2. 按 `node_sequence` 中的 ID 顺序，依次从 `flowchart_content` 查询 `content` 和 `node_type`。

3. 重构流程图数据结构 `flow_steps` 和 `flow_spec`，恢复结点间的连接关系。

4. 将流程图加载到主窗口，对话记录置空，便于用户在此基础上继续交互。

6.2.3 去重机制的优势

由于结点内容已去重，加载时只需根据 ID 序列取出内容，无需处理重复存储。这不仅节省了存储空间，还保证了同一结点在不同会话中的一致性——若后期修正了某结点内容，所有引用它的会话都会自动获得更新。

6.3 单思路结点动态更新的实现

在 4.9 节中，我们提出了“单思路结点动态更新”的概念，即通过与 AI 的持续交互，让思路结点内容随反馈而实时演化。在 DeepSeek 思维链对话应用中，这一机制通过“交互模式”中的“反思”和“深入”操作实现。

6.3.1 交互模式中的“反思”操作

当用户对当前思路的某个部分不满意时，可选中最后一个高亮结点，点击“反思”。系统将当前会话上下文、该结点内容以及用户补充要求发送给 AI，AI 重新生成该结点及其后续结点的内容，并以新结点替换原有部分。这对应于思路结点的“修改”与“迭代更新”。

6.3.2 “深入”操作

若用户希望将某个抽象结点展开为更详细的子流程，可点击“深入”。系统将该结点内容作为目标，要求 AI 生成一个完整的子流程图，并用该子图替换原结点。这体现了思路结点的“嵌套”特性（4.2 节），即一个结点可包含完整的子思路。

6.3.3 动态更新的数据一致性

每次更新后，系统会重新计算结点 ID（新结点插入 `flowchart_content`，旧结点若不再被引用则保留为历史版本），并更新当前会话的 `node_sequence`。同时，外置记忆 JSON 文件自动保存，确保断点续传。当流程图再次全亮时，新的思路版本将被存入数据库，形成版本化存储（4.5 节）。

6.4 系统整体架构与理论对应

DeepSeek 思维链对话应用完整实现了“思路论”的核心设计，其架构与理论概念的对应关系如下：

| 理论概念 | 系统实现 |

|----------|----------|

| 思路结点 | 流程图中的每个矩形/菱形单元，对应 `flowchart_content` 一条记录 |

| 思路调用存储器 | `flowchart_session` 表的 `node_sequence` 字段 |

| 思路结点存储器 | `flowchart_content` 表（去重存储） |

| 双层存储架构 | 数据库三表分离：内容、序列、索引 |

| 思路可靠性串联模型 | 通过 `reliability` 标签记录结点历史成功率，但当前版本未启用 |

| 第一类循环单元 | 交互模式中的“反思-深入-继续”循环，对应量变质变、对立统一、否定之否定三大基本思路 |

| 单思路结点动态更新 | 交互模式中 AI 以 `[交互更新]` 形式追加内容 |

| 过滤器实验 | 应用中的“马克思主义过滤器”演示了选择结构的 AI 实现 |

6.5 项目实现简述

本系统以 Python 编写，采用 Tkinter 构建图形界面，核心功能模块包括：

对话引擎：支持 DeepSeek API（云端）和 Ollama（本地）双模式，通过统一的调用接口与模型交互。

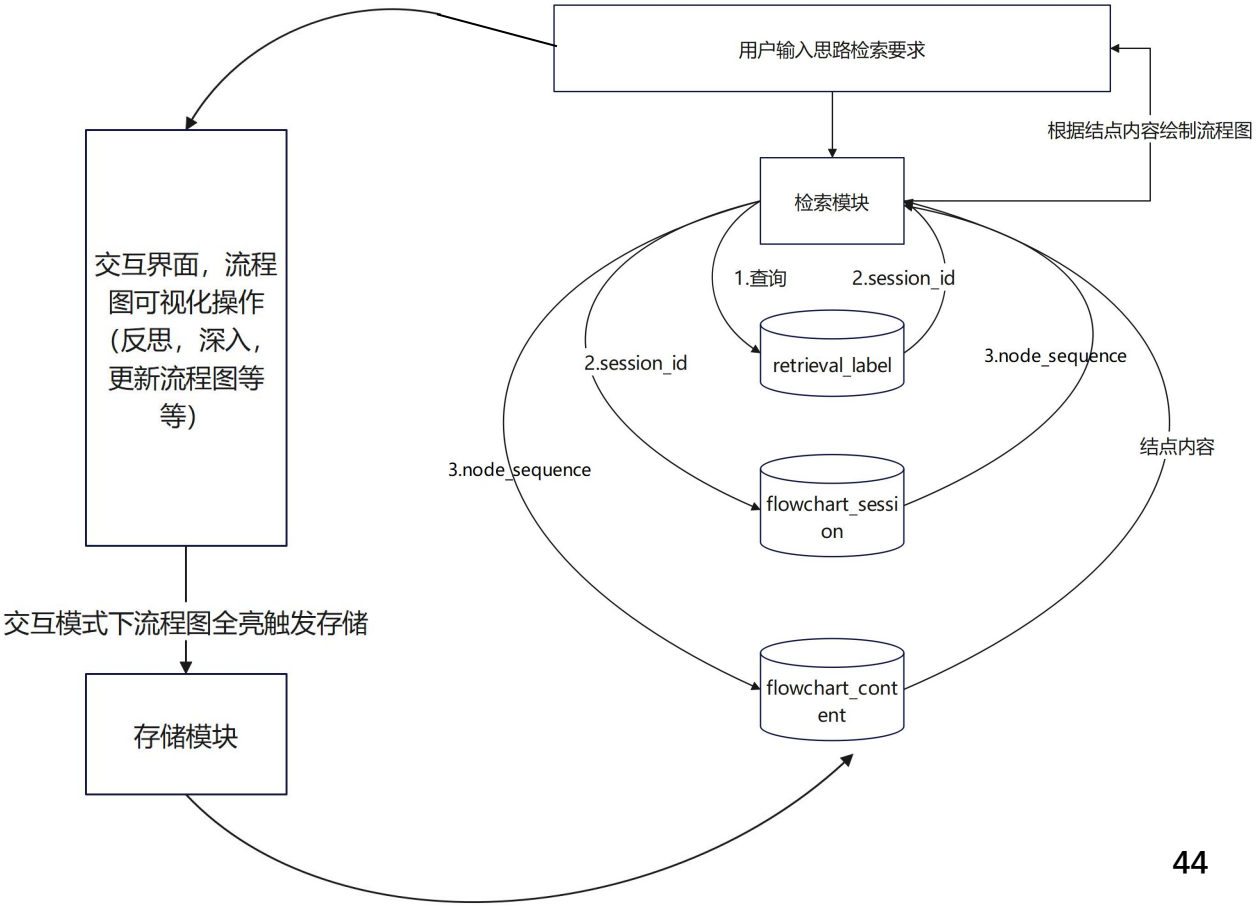
思维链解析器：将模型返回的推理内容（通常包含 `[...]` 或 XML 标签）解析为流程图结点，并生成 `flow_steps` 和 `flow_spec` 数据结构。

流程图绘制器：基于自定义画布绘制结点，支持缩放、双击查看全文、高亮状态管理。

外置记忆管理器：封装 JSON 文件读写与数据库操作，提供保存、加载、检索接口。

交互模式控制器：管理高亮状态、结点更新、继续交互等逻辑，确保思路的动态生长。

所有数据库操作均通过参数化 SQL 执行，防止注入；检索时的 AI 调用采用流式响应，避免界面卡顿。



第六章 多层 AI 外置记忆循环实现

前面关于 AI 外置循环记忆结构的实现明显是单层的，下面来讨论多层 AI 外置记忆循环实现。

6.1 多层外置记忆存储系统设计

单层的记忆只能适用于用户发送的文本篇幅较少，即一个问题中输入数据的空间复杂度较小的情况。

面对长记忆大篇幅的思考 AI，这一直是当下 AI 的痛病。

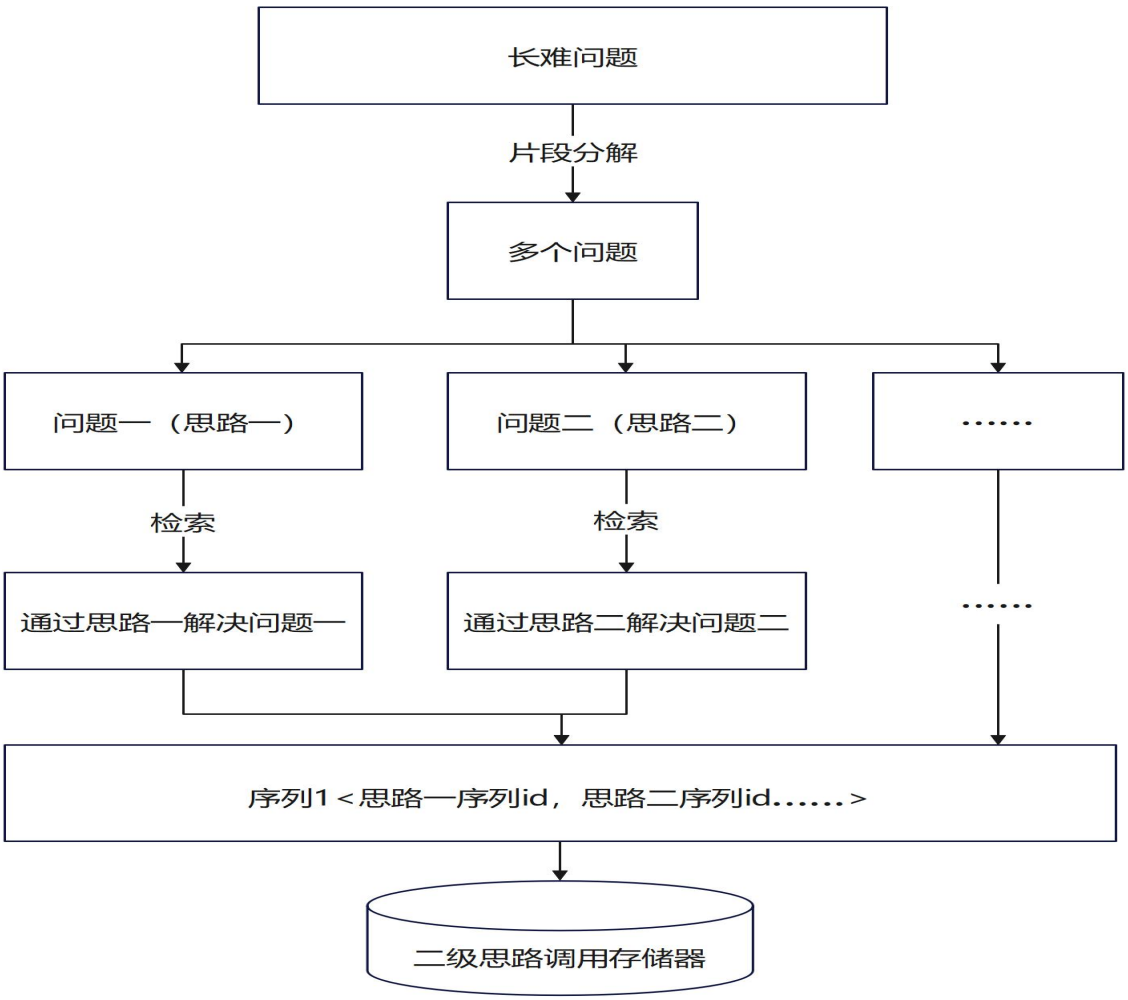
因此基于这个问题，我在单层 AI 外置记忆循环系统的基础上提出多层 AI 外置记忆循环系统，即在原本双层的结构基础上继续增加思路调用存储器的层数，即一级思路调用存储器可以为下一级思路调用存储器中的内容继续进行调用存储功能。

多层要实现的就是对于长对话的片段截取而后检索思路解决按照片段内容逐个处理问题的能力。

比如通过对于长篇内容按照句子进行划分，每一句都会通过思路检索，找到对应的处理思路而后输出结果，就像人处理长篇幅问题一样，一是不断阅读并不断提炼归纳最后解决问题，二是将大的问题拆成小的问题然后逐个解决小的问题。

那么长篇内容就是大问题，将其拆成许多小片段就是小问题，而小片段所需要用于解决问题的思路通过检索获得。

那么一个长篇内容就会对应多个思路，一个思路会对应一个序列存在思路调用存储器中，在思路调用存储器中一个思路只有一个 session_id，那么由多个思路组成的关于 session_id 的序列就可以存储在一个新的作为调用思路结点序列的思路调用存储器，而检索功能依旧通过查询对应 label_text 进行，那么这就相当于在原本思路应用的基础上做了一个应用思路的应用层。



6.2 多层外置记忆存储系统设计目的

设计这个系统的目的就是通过专业人员在与 AI 交互的同时能形成一套利于自己思路经验积累的可自动执行的思路数据库。

当然这些数据库如果设计得当是有利于后续 AI 新模型的训练的。

另外由于数据库的可共享性，一些专业人员可以通过推广自己在 AI 流程使用过程中的思路设计与思考给那些非专业人士，有效的思路共享可以解决后续人员参与同一思路下的问题解决的效率。

并且由于思路都是通过检索而调用的，这就对于同一思路的创造性复用提供了可能。虽然不同领域，但是思路足够抽象也可以为别的问题提供借鉴意义。

再加上思路机是自动化处理这一思路复用过程，我相信未来思路知识也将成为市场流通的商品。

第七章 从思路机到智脑

以下思考暂无研究与具体理论的支撑，仅仅作为作者的设想。

随着记忆层级的增加，如何高效地选择、组织、调度这些思路成为一个独立的计算问题。为此，本文提出智脑概念，并将其定位为运行在思路机之上的元级智能系统。

我认为，比人更稳定，比人更精确，记得更多，再保留人在发展中的长处，就已经比人更聪明了。毕竟再聪明也无法脱离物质世界的范畴，那与人同样从物质世界中学习的人工智能又能聪明多少呢？只是更有精力，更有阅历，更加人机，更能劳动罢了。

由于思路机的记忆部分采用多层的记忆循环存储系统，这个层数未知，这个层与层之间的拓扑结构复杂到难以估量，因此我认为应该有专门用来处理多层外置记忆存储的机器，这个机器以思路存储调用复用作为核心，也就是机器的核心操作是对超多层级的存储器进行操作。

这台机器只需要告诉我们解决任意问题我们需要什么思路，而后我们用电脑或其他工具去按照思路去一步步执行。

我将这台机器称之为智脑。

智脑独立于其他工具之外，它只起到思想的指导作用。

智脑可以直接接入其他工具，通过思路机对其他工具进行操作。

智脑使用电脑，我想这是一个可以预见的将来。

总结来讲，智脑是指以多层外置记忆循环结构为操作对象，以思路的选择与组合为核心功能，输出问题求解方案（即一组有序的思路会话）的元级智能系统。智脑不直接执行具体操作，而是将方案交由下层思路机或具体工具执行。

注意：智脑一定是已有可靠思路高度丰富后的产物。

7.1 一些技术发展原则

基于智脑的探索，本文提出以下技术发展原则，以期为未来智能系统设计提供参考。

1.模型要使本来复杂的专家系统注入内核，这是思路机有别于专家系统的关键。

2.原本复杂的思路结构要因为模型的加入而变得简单。

3.模型天生便是高效的思路算法。

4.黑盒的不可靠性要放在白盒中去使用。

5.记忆分层与动态迁移原则，智能系统的记忆应按照访问频率、抽象层级、领域特性分层存储，并支持思路在不同层间的动态迁移，以平衡检索效率与存储成本。

7.2 一些 AI 发展原则

定义明确：1.人工智能是智能，而不是人

发展方向：2.智能可以不断发展进化，遵循智能进化理论，但无需朝着人的方向去模仿（除了特殊场景需要）

存在形式：3.智能应当是一种存粹的意识形态（可以是信息或数据的有机集合），它非生命，却当为生命之精华。

目的，建议：4.人工智能是人类智能的累积丰富的高效外设助手，未来的人们应当用人工智能的个性归属来代表自身的劳动所得，而自身需不断为人类探索未知以此来丰富累积人类智能财富。一部分人去校对过去的知识，让全人类探索无限的未知。当然个性化的过程需在教育的非应试中体现。

总结起来 5.开放共享原则

思路作为可复用的知识资产，应支持跨用户、跨组织的共享与交易。未来可建立“思路市场”，由专业人员生产高质量思路，供普通用户按需调用，形成知识经济新形态。

参考文献

[1]计 算 机 科 学 技 术 名 词　审 定 委 员 会 . 计 算 机 科 学 技 术 名 词 [EB/OL].(2018)[2026-02-03]. https://www.termonline.cn/wordDetail?termName=程序设计&subject=9b37f41a26b011eea6a1b068e6519520&base=1