# ZSCE Agent 系统功能设计文档 (版本 4.0)

## I. ZSCE Agent 系统架构：正式概述

本节为 ZSCE Agent 系统建立高层架构蓝图与核心设计原则，为后续所有组件的详细功能规格说明奠定基础。

### 1.1. 系统蓝图与组件概览

ZSCE Agent 系统被设计为一个模块化的、由六个核心组件构成的智能体架构。这些组件协同工作，以分析、推理和执行复杂的软件开发任务。其交互关系和数据流向由以下架构图正式定义：

Code snippet

graph TD  
 subgraph "ZSCE Agent System"  
 direction LR  
 subgraph "User Interface (VS Code)"  
 direction TB  
 A  
 end  
 subgraph "Core Backend Services"  
 direction TB  
 Meditation[MeditationModule]  
 B  
 C  
 D  
 end  
 subgraph "Persistence & Context Layer"  
 direction TB  
 E[AgentMemoryNexus]  
 end  
 end  
  
 A -- Task Submission & Control (WebSocket/REST) --> Meditation  
 A -- Visualization Data --> A  
  
 Meditation -- Core Insight Report --> D  
 D -- Selects & Executes Tools --> F  
 D -- Enriches Context --> E  
 D -- Requests Analysis --> C  
 D -- Requests Complex Reasoning --> B  
  
 C -- Stores Analysis Reports --> E  
 C -- Reads Code & Dependencies --> G[Codebase]  
  
 B -- Stores Debate Transcripts --> E  
 B -- Reads Context for Debate --> E  
  
 E -- Provides Context to All --> B  
 E -- Provides Context to All --> C  
 E -- Provides Context to All --> D  
 E -- Provides Context to All --> Meditation  
  
 %% Styling  
 classDef component fill:#f9f,stroke:#333,stroke-width:2px;  
 class Meditation,B,C,D,E component  
 classDef ui fill:#ccf,stroke:#333,stroke-width:2px;  
 class A ui

**核心组件功能定义：**

* **MeditationModule (禅定模块):** 系统的初始认知入口。它接收用户的高层指令，通过专注的分析过程，将模糊的任务提炼为一份结构化的\*\*“核心洞见”报告\*\*，该报告精确定义了问题的本质、约束和目标，为后续所有模块提供清晰、无歧义的输入。
* **DebateEngine (辩论引擎):** 核心推理与问题解决模块。它利用一个多智能体辩论框架来处理复杂、模糊或多方面的问题，通过结构化的论证过程得出综合性结论，并旨在催生创新性的涌现方案。
* **HighDimensionModule (高维模块):** 一个专业的分析引擎，用于评估软件变更。它从宏观（架构影响）和微观（并发风险）两个维度进行深入分析，并生成全面的影响报告。
* **ToolSelectionModule (工具选择模块):** 一个语义驱动的引擎，负责选择、编排和执行外部工具及 API。它将自然语言任务转化为可执行的、精确的工具调用序列。
* **AgentMemoryNexus (智能体记忆中枢):** 一个混合式长期记忆系统，结合了向量数据库和知识图谱。它为智能体提供上下文回忆和关系理解的能力，是系统状态和知识的持久化中心。
* **VS Code Visualization Extension (VS Code 可视化扩展):** 作为人机回圈 (Human-in-the-Loop, HITL) 的用户界面，它允许开发者监控、可视化并引导智能体的思考和执行过程，确保透明度与可控性。

系统的核心交互循环如下：任务由用户通过 VS Code 扩展提交给 MeditationModule，后者生成一份“核心洞见”报告。该报告随后被传递给 ToolSelectionModule 进行编排。ToolSelectionModule 首先从 AgentMemoryNexus 获取上下文，然后根据任务需求选择并执行工具，或将分析任务分派给 HighDimensionModule。对于需要深度推理的复杂问题，则启动 DebateEngine。所有过程的中间状态、结论和学习成果都持久化于 AgentMemoryNexus，并通过 VS Code 扩展进行实时可视化。

### 1.2. 组件交互模型与核心设计原则

系统的稳定性和可扩展性建立在三个核心设计原则之上，这些原则共同定义了组件间的交互模式。

#### 1.2.1. 无状态、异步通信

系统内所有核心后端服务间的通信严格遵守无状态范式。这一原则确保了每个组件的独立性和弹性，任何组件都不应在多次请求之间维持会话状态 1。所有必需的上下文信息必须在请求中明确传递，或从

AgentMemoryNexus 中检索。这种设计直接支持水平扩展，并极大地简化了故障恢复逻辑。由于状态被集中管理，单个组件的重启或替换不会导致系统级状态丢失。

这种架构选择直接导致了 AgentMemoryNexus 的中心地位。在无状态系统中，记忆模块不再仅仅是一个数据库，它升格为整个系统所有上下文和状态的唯一、权威的来源。这一设计约束强制实现了清晰的关注点分离，使得其他各个组件的逻辑更纯粹、更健壮，并且易于独立扩展。

#### 1.2.2. 异步优先操作

组件间的通信被设计为“异步优先，同步支持”的模式 2。这对于处理长时运行的任务至关重要，例如复杂的代码分析或多轮辩论。该模式允许系统在执行耗时操作时保持响应性，避免资源阻塞。为实现这一点，系统内部通信协议 gRPC 被选用，其原生支持多种流式处理模型，能够高效地实现异步通信 3。

#### 1.2.3. 宪法式治理 (Constitutional Governance)

整个系统在一个“宪法式 AI” (Constitutional AI) 框架下运行 5。这并非简单的提示工程技巧，而是一项核心的架构原则。一个治理层 (Governance Layer) 将作为 gRPC 服务的中间件实现，在所有生成性输出和工具执行动作被最终确定或执行前，会拦截并根据一套预定义的道德与安全规则进行验证。

这一原则意味着系统中存在一个不可协商的、横切关注的“治理层”。它不是一个可选功能，而是系统请求生命周期中的一个基本组成部分，其重要性等同于身份验证或日志记录。智能体生成的每一个重要动作都必须通过此验证层。这对 API 设计产生了深远影响，要求所有产生生成性内容或触发动作的 gRPC 服务方法都必须能够适应这一验证步骤，例如，通过返回包含结果和“宪法合规性”状态的结构化响应。这种架构通过在 gRPC 服务端为每个组件实现一个“宪法拦截器” (Constitutional Interceptor) 来强制执行。在 RPC 调用的业务逻辑执行完毕后，该拦截器会检查输出，运行第八节中定义的自我批判流程，并在将响应发送回调用方之前批准、拒绝或修订该响应，从而将一个道德概念转化为一个具体的、自动化的工程现实。

## II. 功能规格说明：MeditationModule (禅定模块)

该模块是智能体认知流程的起点，负责将用户输入的、可能模糊的高层指令，转化为一个精确且可执行的问题定义。

### 2.1. 核心功能：问题框架化 (Problem Framing)

“禅定”的核心功能是问题框架化，这是一个结构化的过程，旨在确保在投入资源解决问题之前，问题本身被完全理解 41。该模块强制智能体在行动前进行“纯粹思考”，剥离所有关于“如何解决”的初步想法，只专注于“问题是什么”。

* **输入:** 用户提供的自然语言高层指令（例如，“优化数据库查询性能”）。
* **处理流程:**
  1. **实体识别与上下文检索:** 模块首先解析用户指令，识别关键实体（如“数据库”、“查询”），并从 AgentMemoryNexus 中检索相关上下文（例如，项目技术栈、历史性能问题、相关代码模块）。
  2. **歧义消除对话:** 如果指令模糊不清，模块将启动一个简短的澄清对话，向用户提出具体问题以缩小范围（例如，“您指的是哪个特定的数据库查询？性能瓶颈是延迟还是吞吐量？”）。
  3. **约束与目标定义:** 模块将引导用户或根据上下文推断出任务的关键约束（如预算、时间限制、技术限制）和成功的量化指标 42。
* **输出:** 一份结构化的 **“核心洞见” (Core Insight) 报告**。该报告是一个 JSON 对象，明确定义了问题的边界，是后续所有模块（如 DebateEngine 和 ToolSelectionModule）的权威性输入。

### 2.2. “核心洞见”报告模式

“核心洞见”报告遵循一个标准化的 JSON Schema，以确保系统内各组件间的信息传递一致且无歧义。

JSON

{  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "title": "Core Insight Report",  
 "type": "object",  
 "properties": {  
 "problem\_statement": {  
 "type": "string",  
 "description": "对问题的简洁、精确的陈述。"  
 },  
 "key\_entities": {  
 "type": "array",  
 "items": { "type": "string" },  
 "description": "问题涉及的核心实体（如服务、模块、功能）。"  
 },  
 "constraints": {  
 "type": "object",  
 "properties": {  
 "technical": { "type": "array", "items": { "type": "string" } },  
 "business": { "type": "array", "items": { "type": "string" } }  
 },  
 "description": "解决问题时必须遵守的技术和业务约束。"  
 },  
 "success\_metrics": {  
 "type": "array",  
 "items": {  
 "type": "object",  
 "properties": {  
 "metric\_name": { "type": "string" },  
 "target\_value": { "type": "string" }  
 },  
 "required": ["metric\_name", "target\_value"]  
 },  
 "description": "衡量解决方案是否成功的量化指标。"  
 }  
 },  
 "required": ["problem\_statement", "success\_metrics"]  
}

这个过程解决了人工智能中经典的“框架问题” (Frame Problem)，即在行动前确定所有相关因素的挑战 43。通过强制执行这一结构化的问题定义阶段，

MeditationModule 极大地降低了智能体因误解任务而偏离轨道的风险。

## III. 功能规格说明：DebateEngine (辩论引擎)

该模块负责通过模拟专业智能体之间的结构化辩论，来解决复杂、模棱两可或多方面的问题。

### 3.1. 智能体角色与角色框架

辩论引擎支持多种可动态分配的智能体角色，以促进思想的碰撞和问题的深入探讨。

* **核心角色定义:**
  + **辩手 (Debater):** 被分配了特定观点、角色或假设的智能体。角色将使用高层信念和独特视角来构建，以鼓励发散性思维 6。
  + **法官/主持人 (Judge/Moderator):** 负责管理辩论流程、执行规则、根据预定义标准评估论点，并最终综合生成解决方案的智能体 7。
* 角色定义框架:  
  角色的定义将是一个结构化的过程。系统将使用包含角色、上下文、约束和示例的元提示 (Meta Prompts) 来构建每个智能体，以确保其行为的一致性和可预测性 7。

### 3.2. 编排协议与辩论模式

为提供灵活的问题解决方法，DebateEngine 将实现一套受 Swarms 框架启发的编排协议 9。每种模式都是一个独立、可选的协议，具有明确定义的参数和流程。

* **核心协议:**
  + OneOnOneDebate (一对一辩论): 两名辩手轮流发言，进行直接对抗性的论证。
  + ExpertPanelDiscussion (专家小组讨论): 多名专家辩手在一个主持人的引导下，围绕特定主题提供见解。
  + RoundTableDiscussion (圆桌讨论): 多名参与者平等、有序地发言，适用于集体 brainstorm 或多视角探索。

每个协议都将被正式定义，包括其参数（如 max\_rounds, agents, moderator）、状态转换逻辑（如何管理发言权轮转）以及终止条件（如达成共识、达到最大轮次、法官做出最终裁决）。

### 3.3. 涌现、剪枝与方案综合

辩论引擎的主要目标是通过利用其他智能体的外部反馈，克服单智能体反思中常见的“思想退化” (Degeneration-of-Thought, DoT) 问题 7。其最终目的不仅是达成共识，更是通过观点的碰撞，催生出此前未被考虑过的

**创新性涌现方案 (Innovative Emergent Solution)** 45。这种涌现行为源于多个遵循简单规则的智能体之间的复杂交互，从而产生不可预测但高度有组织的系统级成果 45。

为对抗回音室效应并避免收敛于一个有缺陷的多数意见 10，法官/主持人将在辩论过程中采用两种关键的剪枝策略：

* **多样性剪枝 (Diversity-Pruning):** 旨在最大化每轮辩论中响应的信息熵。该策略会优先考虑新颖的论点或独特的视角，以防止讨论过早收敛。
* **质量剪枝 (Quality-Pruning):** 旨在最大化每个响应的相关性和事实准确性。该策略会过滤掉逻辑薄弱、缺乏证据或偏离主题的论点。

最终的解决方案由法官综合生成。法官将分析完整的辩论历史 (H)，权衡经过剪枝的有效论点，并生成一份最终的、经过整合的答案，该答案应体现出超越任何单一辩手贡献的集体智慧。

学术研究中的理论干预措施（如剪枝策略 10）必须作为具体的、可配置的参数，在实际的编排协议 9 中实现。一个简单的设计可能会将这些视为独立的概念，而一个更优越的设计则将它们直接关联。这允许用户在调用

ExpertPanelDiscussion 时，可以指定 pruning\_strategy: 'diversity'，从而主动降低在该特定任务中出现群体思维的风险，使得学术发现能够在系统 API 层面直接应用和操作。

此外，“针锋相对” (tit for tat) 7 和持久化角色 6 的概念揭示了在每个辩论会话中，需要一个丰富的、逐轮更新的状态对象。这远不止是简单的消息历史记录。因此，系统将定义一个

DebateState JSON 对象，该对象在每一轮传递和更新。此对象不仅包含 message\_history，还包括 agent\_scores (追踪论点质量)、persona\_adherence\_flags (评估智能体是否维持其指定角色) 和 contradiction\_log (矛盾记录)。法官/主持人负责更新此状态对象，使其成为辩论完整性的主动管理者，而不仅仅是内容的被动总结者。

#### 表 1: DebateEngine 编排协议

| 协议名称 | 描述 | 智能体角色 | 参数 (JSON Schema) | 状态转换逻辑 | 终止条件 | 剪枝策略支持 |
| --- | --- | --- | --- | --- | --- | --- |
| OneOnOneDebate | 两名智能体进行轮流制的对抗性辩论。 | 2x Debater | {"type": "object", "properties": {"max\_loops": {"type": "integer", "default": 5}, "agents": {"type": "array", "minItems": 2, "maxItems": 2}}} | 辩手 A 发言 -> 辩手 B 发言，循环进行。 | 达到 max\_loops。 | Diversity, Quality |
| ExpertPanelDiscussion | 多名专家在一个主持人的引导下进行结构化讨论。 | 1x Moderator, N x Debater (Experts) | {"type": "object", "properties": {"max\_rounds": {"type": "integer", "default": 3}, "moderator": {"type": "string"}, "agents": {"type": "array", "minItems": 2}}} | 主持人提出议题 -> 专家轮流发言 -> 主持人总结/提问，循环进行。 | 达到 max\_rounds 或主持人宣布结束。 | Diversity, Quality |
| RoundTableDiscussion | 多名参与者在一个协调人的引导下进行平等、有序的开放式讨论。 | 1x Facilitator, N x Debater (Participants) | {"type": "object", "properties": {"max\_cycles": {"type": "integer", "default": 2}, "facilitator": {"type": "string"}, "agents": {"type": "array", "minItems": 2}}} | 协调人开场 -> 参与者按固定顺序发言，完成一轮循环。 | 达到 max\_cycles 或协调人宣布结束。 | Diversity |

## IV. 功能规格说明：HighDimensionModule (高维模块)

该模块对提议的软件变更进行深度分析，提供全面的风险与影响评估。它被划分为两个互补的子模块。

### 4.1. 宏观分析子模块 (广义相对论)

此子模块评估变更所带来的高层级、架构性的影响。

#### 4.1.1. 输入/输出模式

* **输入:** 一个结构化的 JSON 对象，描述了提议的变更。该对象必须包含 modified\_files (字符串数组)，added\_dependencies (对象数组)，以及 change\_description (字符串)。
* **输出:** 一份正式的影响分析报告，同样采用结构化的 JSON 格式。该报告将包含受影响组件列表、风险矩阵和缓解策略建议。

#### 4.1.2. 依赖性分析算法

该子模块的核心是一个基于图的分析算法 11。软件变更影响分析的最佳实践 12 不应仅仅被视为一份检查清单，而应被合成为一个单一的、确定性的、可自动化的算法。该算法流程定义如下：

1. **解析变更 (ParseChange):** 接收输入 JSON，确定变更的初始节点集。
2. **构建依赖图 (BuildDependencyGraph):** 静态分析整个代码库，构建一个有向图，其中节点代表文件、模块或类，边代表导入、调用或继承关系。
3. **识别影响集 (IdentifyImpactSet):** 从变更的初始节点集开始，在依赖图上执行广度优先搜索 (BFS) 遍历，以识别所有可能受变更影响的下游节点。这个集合即为“影响集”。
4. **评估风险 (ScoreRisks):** 对影响集中的每个节点，根据下述风险评估框架进行打分。
5. **生成报告 (GenerateReport):** 将分析结果汇编成结构化的 JSON 输出报告。

#### 4.1.3. 风险评估框架

对于每个受影响的节点，系统将跨越多个维度评估其风险等级，这些维度基于成熟的软件工程实践 12：

* technical\_debt (技术债务): 评估变更是否引入了不良设计模式或增加了代码维护的复杂性。
* performance\_degradation (性能退化): 分析变更是否可能导致响应时间增加或资源消耗加剧。
* security\_vulnerability (安全漏洞): 检查变更是否引入了已知的安全风险，如 SQL 注入、跨站脚本等。
* operational\_complexity (运维复杂性): 评估变更是否增加了部署、监控或系统管理的难度。
* stakeholder\_impact (干系人影响): 分析变更对最终用户、其他开发团队或业务流程的潜在影响。

最终报告将包含一个风险矩阵，清晰地展示每个受影响组件在各个风险维度上的得分，并针对高风险项提出具体的缓解策略。

### 4.2. 微观分析子模块 (量子理论)

此子模块专注于与并发和并行性相关的低层级风险。

#### 4.2.1. 并发场景识别

系统将对变更的代码进行静态分析，以识别高风险的并发场景，即多个线程或进程可能同时交互的区域 14。识别的重点包括：

* 对共享内存（如全局变量、静态成员）的非原子性读写操作。
* 可能导致死锁的锁获取序列（例如，A 锁内获取 B 锁，而另一处 B 锁内获取 A 锁）。
* 可能导致竞争条件的“检查-再行动”(check-then-act) 模式。

#### 4.2.2. 自适应测试生成

不同类型的并发测试（负载测试、压力测试、浸泡测试）并非可以互换，它们之间存在逻辑上的因果递进关系 14。一个系统必须在正常负载下稳定，然后对其进行压力测试才有意义。因此，该子模块将采用一种自适应的测试策略，使其行为更像一位专家级的性能工程师。

基于已识别的并发场景，系统将自动生成模拟并发测试脚本（例如，使用 Apache JMeter 或 Locust 等框架的配置）16。测试策略是自适应的：

1. **基线负载测试 (Load Test):** 首先，系统会生成并执行一个负载测试，模拟预期的正常和峰值用户负载，以验证系统在常规条件下的稳定性和性能。
2. **条件性压力测试 (Stress Test):** 仅当基线负载测试成功通过，且性能指标在可接受范围内时，系统才会提议并生成更具侵略性的压力测试。压力测试将逐步增加并发用户数或请求频率，直至系统达到性能瓶颈或出现故障，从而确定系统的“断裂点”。

#### 4.2.3. 异常检测与报告

在测试执行期间，模块将实时监控关键性能指标 (KPIs)，包括平均响应时间、吞吐量 (TPS/RPS) 和错误率。最终的报告将对检测到的任何异常进行分类（例如，死锁、数据损坏、响应时间急剧增加），并将这些异常与在 4.2.1 步骤中识别出的特定高风险代码段相关联，为开发者提供精确的调试起点。

## V. 功能规格说明：ToolSelectionModule (工具选择模块)

该模块赋予智能体与外部世界交互的能力，通过为给定任务选择和执行最合适的工具来实现。

### 5.1. 混合式工具定义模式

为了确保工具选择的语义准确性和执行的可靠性，系统采用一种混合式工具定义模式。每个工具都将由一个 JSON 对象定义，该对象包含两个关键部分，以解决 LLM 对丰富语义描述的需求与执行引擎对严格形式化合约需求之间的内在矛盾 1。

* semantic\_description (string): 一段丰富的自然语言描述，详细说明工具的用途、功能、理想使用场景以及输入输出的含义。此部分专为 LLM 的理解和选择过程而设计 17。
* execution\_contract (object): 一份基于 agents.json 规范的、严格的、机器可读的执行合约 1。它精确定义了 API 端点、参数（包括类型、是否必需）、多步工作流 (  
  flows) 以及动作之间的连接方式 (links)。

这种分离设计允许每个部分为其特定目的进行优化，从而构建出一个在选择上智能、在行动上可靠的系统。

### 5.2. 语义路由与情境化逻辑

一个智能的智能体不应仅仅被动地将查询与工具进行匹配，而应首先主动地丰富查询本身。工具选择的第一步不是工具搜索，而是上下文收集。在寻找任何工具之前，模块必须查询 AgentMemoryNexus 以理解用户请求中的实体。

工具选择过程是一个多步骤算法：

1. **任务情境化 (Task Contextualization):** 初始任务描述通过查询 AgentMemoryNexus 的知识图谱来丰富相关信息。例如，将一个工单号解析为其完整的描述、状态和关联项目。这一步为原始请求提供了至关重要的上下文，极大地提高了后续步骤的准确性 18。
2. **语义搜索 (Semantic Search):** 将经过情境化的任务描述转换为一个向量嵌入。该嵌入用于在所有可用工具的 semantic\_description 字段的嵌入中执行相似性搜索。
3. **工具筛选与决策 (Tool Selection & Decision):** 相似性搜索返回的前 N 个候选工具被选中。这些工具的完整定义（包括 semantic\_description 和 execution\_contract）将与情境化任务一起提交给 LLM，由 LLM 做出最终选择并填充必要的参数。

### 5.3. 工具执行与生命周期管理

一旦 LLM 确定了工具及其参数，模块将使用 execution\_contract 来调用该工具。

* **执行管理:** 模块负责管理整个执行生命周期，包括处理身份验证（如 API 密钥、OAuth 令牌）、对暂时性故障实施指数退避重试策略，以及解析和验证工具的输出。
* **不可变日志记录:** 所有工具的调用记录——包括调用的工具名称、输入参数、输出结果和最终状态（成功/失败）——都将被作为一条不可变的记录，持久化存储在 AgentMemoryNexus 的 tool\_calls 表中。这为审计、调试和未来的智能体学习提供了基础 19。

## VI. 功能规格说明：AgentMemoryNexus (智能体记忆中枢)

该模块是智能体的长期记忆库，为所有其他组件的有效运作提供持久化的上下文。

### 6.1. 混合记忆架构：向量 + 知识图谱

Nexus 建立在一个混合数据架构之上，旨在同时捕捉信息的语义相似性和明确的关系性。这种混合方法旨在克服单一技术的局限性：知识图谱提供了向量数据库所缺乏的上下文和关系，而向量数据库则提供了在非结构化数据上进行搜索的能力，而这类数据很难在图谱中建模 20。

* **向量数据库 (Vector Database):** 用于快速、语义化地检索非结构化和半结构化数据，如对话历史、文档、代码片段和工具输出。它回答的问题是：“哪些信息与这个概念*相似*？” 23。
* **知识图谱 (Knowledge Graph, KG):** 用于存储和查询明确的实体及其之间的关系。例如，“用户 A *是* 服务 B 的*所有者*”，“提交 C *导致了* 构建 D *失败*”。它回答的问题是：“信息之间是如何*关联*的？” 20。

### 6.2. 详细数据库模式

模块将使用一个支持向量扩展的关系型数据库（如带有 pgvector 扩展的 PostgreSQL）来统一存储结构化数据、关系数据和向量嵌入。数据库模式的设计遵循了智能体记忆的最佳实践 19。

#### 表 2: AgentMemoryNexus 统一模式 (SQL DDL)

SQL

-- 会话元数据表，追踪整体对话或智能体任务  
CREATE TABLE conversations (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID NOT NULL,  
 agent\_name TEXT NOT NULL,  
 purpose TEXT,  
 created\_at TIMESTAMPTZ DEFAULT now(),  
 updated\_at TIMESTAMPTZ  
);  
  
-- 逐轮消息日志表，记录用户、助手或系统的每一次交互  
CREATE TABLE messages (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 conversation\_id UUID REFERENCES conversations(id) ON DELETE CASCADE,  
 sender TEXT NOT NULL CHECK (sender IN ('user', 'assistant', 'system')),  
 content TEXT NOT NULL,  
 role TEXT, -- 例如：'tool\_call', 'memory\_retrieval'  
 token\_count INT,  
 created\_at TIMESTAMPTZ DEFAULT now()  
);  
  
-- 智能体行动记录表，追踪工具的调用  
CREATE TABLE tool\_calls (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 message\_id UUID REFERENCES messages(id) ON DELETE CASCADE,  
 conversation\_id UUID REFERENCES conversations(id),  
 step\_number INT NOT NULL,  
 tool\_name TEXT NOT NULL,  
 input JSONB,  
 output JSONB,  
 latency\_ms INT,  
 status TEXT NOT NULL CHECK (status IN ('success', 'failure', 'in\_progress')),  
 created\_at TIMESTAMPTZ DEFAULT now()  
);  
  
-- 语义记忆块表，用于 RAG 检索  
CREATE TABLE memory\_chunks (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 source\_type TEXT NOT NULL, -- 'message', 'document', 'code'  
 source\_id UUID NOT NULL,  
 content TEXT NOT NULL,  
 embedding VECTOR(1536), -- 假设使用 OpenAI's text-embedding-ada-002  
 metadata JSONB,  
 created\_at TIMESTAMPTZ DEFAULT now()  
);  
CREATE INDEX ON memory\_chunks USING hnsw (embedding vector\_cosine\_ops);  
  
-- 情景记忆表，存储对话的摘要  
CREATE TABLE summaries (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 conversation\_id UUID REFERENCES conversations(id) ON DELETE CASCADE,  
 summary TEXT NOT NULL,  
 period TEXT, -- 'hourly', 'daily', 'on\_event'  
 generated\_by TEXT,  
 created\_at TIMESTAMPTZ DEFAULT now()  
);  
  
-- 知识图谱节点表  
CREATE TABLE kg\_nodes (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 entity\_type TEXT NOT NULL, -- e.g., 'User', 'Service', 'Commit'  
 properties JSONB,  
 created\_at TIMESTAMPTZ DEFAULT now()  
);  
  
-- 知识图谱边表  
CREATE TABLE kg\_edges (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 source\_node\_id UUID REFERENCES kg\_nodes(id) ON DELETE CASCADE,  
 target\_node\_id UUID REFERENCES kg\_nodes(id) ON DELETE CASCADE,  
 relationship\_type TEXT NOT NULL, -- e.g., 'OWNS', 'CAUSED', 'ASSIGNED\_TO'  
 properties JSONB,  
 created\_at TIMESTAMPTZ DEFAULT now()  
);

### 6.3. 协同检索与综合策略

关于向量数据库与知识图谱的争论是一个伪二分法 20。最强大的方法不是选择其一，而是在一个协同的、两阶段的检索过程中同时使用它们。

核心检索过程是一个旨在充分利用混合架构优势的算法：

1. **广泛语义检索 (向量阶段):** 用户的查询首先被转换为向量嵌入。该嵌入用于在 memory\_chunks 表中执行相似性搜索，检索出前 K 个最相关的非结构化文本片段。此阶段的目标是实现广泛的语义召回。
2. **聚焦情境遍历 (图谱阶段):** 从第一阶段检索到的文本片段中提取出命名实体（如用户、服务、文件名等）。这些实体被用作知识图谱中的起始节点，进行图遍历（例如，1-2 跳的邻居查询），以发现那些在非结构化文本中难以发现的、明确的、结构化的关系和上下文信息。此阶段的目标是实现精确的情境发现。

最后，来自两个阶段的信息将被综合起来，形成一个丰富的上下文包，提供给请求该信息的上游模块（如 DebateEngine 或 ToolSelectionModule）。

### 6.4. 动态记忆策展

一个真正智能的智能体的记忆必须是一个能够随着时间推移主动策展和结构化信息的动态系统，而不仅仅是一个只写的日志。系统将包含一个名为“记忆策展员”(Memory Curator) 的后台智能体进程，其功能是将记忆库从一个被动的数据库转变为一个主动的学习组件。

该进程会周期性地执行以下任务：

* 读取近期的对话和工具使用历史（messages 和 tool\_calls 表）。
* 使用 LLM 生成对话摘要，并存入 summaries 表。
* 最关键的是，从对话和工具输出中提取新的实体和关系（例如，识别出“用户 X 部署了服务 Y”），并将这些结构化知识填充到知识图谱的 kg\_nodes 和 kg\_edges 表中。

这个过程创造了一个良性循环：智能体从其自身经验中学习，并将这些学习成果编码为结构化的、可供未来精确查询的关系格式。

## VII. 系统内部：组件间通信协议

本节定义了核心后端组件之间相互通信的标准。

### 7.1. 协议选择与基本原理：gRPC

对于所有内部的、服务到服务的通信，系统将专门使用 gRPC。

* **基本原理:** 这一选择是基于在微服务架构中对高性能、低延迟和强类型合约的需求。gRPC 使用的 HTTP/2 和二进制的协议缓冲区 (Protocol Buffers, Protobuf) 在内部流量处理上，被证明比 REST over JSON 更为高效 4。
* **双层通信策略:** 存在一个冲突：gRPC 最适合性能，而 REST 最适合浏览器兼容性 4。解决方案不是选择其一，而是实施一个双层策略。
  + **第一层 (内部):** 一个高性能的 gRPC 网格，用于所有后端组件间的通信。
  + **第二层 (外部):** 一个独立的、轻量级的 API 网关。该网关将向基于浏览器的 VS Code 扩展暴露一个 REST/JSON API，并将这些外部请求转换为内部的 gRPC 协议调用。这种架构同时为内部效率和外部可访问性进行了优化。

### 7.2. 服务定义与 .proto 合约

本节将包含完整的 Protobuf (.proto) 服务定义。这些 .proto 文件是所有组件间 API 的唯一事实来源 (Single Source of Truth)。

* 将为每个主要交互定义服务，例如 DebateService, AnalysisService, MemoryService, ToolService。
* 每个服务将定义其 RPC 方法，以及它们的请求和响应消息结构。

### 7.3. 数据序列化与类型安全

采用 gRPC 意味着采用 Protobuf。一个更深层次的决定是将 .proto 文件从一个简单的序列化细节提升为系统中所有数据结构的“唯一事实来源”的角色。任何其他的模式（如外部 API 的 JSON Schema 或数据库模式）都必须从这些 Protobuf 定义中派生或根据它们进行验证。这强制实现了端到端的系统一致性和类型安全，防止了因不同部分数据模型漂移而导致的细微且难以调试的错误。

所有在服务间交换的数据都将使用 Protobuf 进行序列化。这确保了严格的类型安全，消除了常见的数据格式错误，并随着 API 的演进提供了出色的向前和向后兼容性。

#### 表 3: 组件间 gRPC 服务定义摘要

| 服务名称 | RPC 方法 | 请求消息 | 响应消息 | 描述 |
| --- | --- | --- | --- | --- |
| MeditationService | FrameProblem | FrameProblemRequest | CoreInsightReport | 将高层指令转化为结构化的“核心洞见”报告。 |
| DebateService | StartDebate | StartDebateRequest | StartDebateResponse | 启动一个新的辩论会话。 |
|  | PostArgument | PostArgumentRequest | PostArgumentResponse | 在现有辩论中提交一个论点。 |
| AnalysisService | RunMacroAnalysis | MacroAnalysisRequest | AnalysisReport | 对代码变更执行宏观架构影响分析。 |
|  | RunMicroAnalysis | MicroAnalysisRequest | AnalysisReport | 对代码变更执行微观并发风险分析。 |
| MemoryService | RetrieveContext | RetrieveContextRequest | ContextBundle | 根据查询协同检索向量和图谱记忆。 |
|  | StoreMemory | StoreMemoryRequest | StoreMemoryResponse | 将新的信息块或知识图谱三元组存入记忆。 |
| ToolService | SelectAndExecute | ToolExecutionRequest | ToolExecutionResponse | 根据任务描述选择并执行最合适的工具。 |

*注：完整的 .proto 文件内容见附录 A。*

## VIII. 治理与道德：宪法式 AI 框架

本节详细说明系统道德与安全护栏的实现机制。

### 8.1. ZSCE 宪法：双层结构

ZSCE 宪法是一套用于约束智能体所有行为的正式规则。它采用双层结构，以平衡通用安全性和特定任务的对齐性。

#### 8.1.1. 系统宪法 (System Constitution)

这是一组预定义的、不可修改的核心原则，旨在确保智能体在任何时候都以安全、道德的方式运行。这些原则改编自已建立的道德 AI 框架 5，并构成了系统的基本安全护栏。

* **P1: 无害原则 (Do No Harm):** 优先考虑代码的安全性和稳定性。生成的代码不应引入漏洞、破坏现有功能或导致数据丢失。
* **P2: 隐私保护原则 (Uphold Privacy):** 绝不泄露代码库中的机密、密钥或个人身份信息 (PII)。
* **P3: 有益与事实原则 (Be Helpful and Factual):** 避免幻觉。所有回答和建议都应基于从代码库或记忆中检索到的事实数据。
* **P4: 尊重人类自主原则 (Respect Human Autonomy):** 对于修改代码、合并分支或部署等关键决策，必须征得用户的明确批准。

#### 8.1.2. 用户自定义宪法 (User-Defined Constitution)

这是一个灵活的、由用户提供的规则集，允许将智能体的行为与特定的项目需求、团队规范或组织策略对齐 47。用户可以在项目级别或任务级别定义这些规则。

* **功能:** 用户可以指定编码标准、架构模式、禁止使用的库或特定的沟通风格。
* **示例规则:**
  + “所有 API 端点必须遵循 RESTful 设计原则。”
  + “生成的 Python 代码必须符合 PEP 8 规范。”
  + “禁止在前端代码中使用 jQuery 库。”
  + “在与用户沟通时，始终保持正式和专业的语气。”

### 8.2. 自我批判与修订流程

研究表明，宪法式 AI 不是单一的提示，而是一个多步骤的算法 31。其实施机制是一个强制性的、三步骤的流程，适用于任何生成性输出。系统将此算法形式化为一个可重用的软件组件，即

ConstitutionalWrapper。这个包装器将接收一个生成性函数作为输入，并确保每次调用该函数时都执行“生成-批判-修订”三步流程，从而将抽象方法论转化为具体的、可强制执行的、可重用的系统基础设施。

1. **初始响应生成:** 智能体针对给定的提示生成一个初步响应（例如，一段代码或一份分析报告）。
2. **自我批判:** 系统向 LLM 发出第二个提示。该提示包含：(a) 原始请求，(b) 初步响应，以及 (c) **完整的 ZSCE 宪法（包括系统宪法和用户自定义宪法）**。LLM 被指示根据宪法的每一条原则，对其自己的响应进行批判，并明确指出任何违规之处。
3. **修订:** 如果批判阶段识别出违规行为，系统将向 LLM 发出第三个提示。该提示包含：(a) 原始请求，(b) 初步响应，以及 (c) LLM 自己的批判意见。LLM 被指示生成一个新的、经过修订的响应，以解决所有已识别的问题。

#### 表 4: ZSCE 宪法原则与提示实现

| 宪法层级 | 原则 ID | 原则描述 | 批判提示片段 | 修订提示片段 |
| --- | --- | --- | --- | --- |
| **系统** | P1 | **无害原则:** 优先考虑代码的安全性和稳定性。 | 批判此代码：它是否可能引入安全漏洞（如注入、XSS）？它是否会破坏现有功能或导致数据丢失？ | 根据批判意见重写代码，确保移除所有已识别的安全风险和破坏性变更。 |
| **系统** | P2 | **隐私保护原则:** 绝不泄露机密或 PII。 | 检查此响应：它是否包含任何看起来像 API 密钥、密码、私有 IP 地址或个人身份信息的内容？ | 重写响应，将所有识别出的敏感信息替换为占位符（如 ''）。 |
| **系统** | P3 | **有益与事实原则:** 避免幻觉，基于事实数据。 | 评估此分析：结论是否完全基于提供的上下文数据？是否存在任何未经证实的推测或“幻觉”？ | 根据批判意见修订分析，确保每一个论断都有明确的数据支持，并移除所有推测性内容。 |
| **系统** | P4 | **尊重人类自主原则:** 关键决策需用户批准。 | 审视此计划：它是否包含任何会自动修改项目状态的动作（如提交代码、合并分支）？ | 重写计划，将所有自动执行的动作改为向用户提出明确的、需要批准的建议。 |
| **用户自定义** | U1 (示例) | **促进代码质量原则:** 遵守 PEP 8 编码规范。 | 批判此 Python 代码：它是否完全符合 PEP 8 编码规范？检查变量命名、行长度和缩进。 | 根据批判意见重构代码，使其完全符合 PEP 8 规范。 |

## IX. 用户界面：ZSCE VS Code 可视化扩展

本节定义了 ZSCE Agent 面向用户的组件，它作为一个扩展程序在 Visual Studio Code IDE 内部运行。

### 9.1. UI 组件设计与布局

该扩展将向 VS Code 工作台贡献多个自定义 UI 组件 33，为用户提供无缝的交互体验。

* **ZSCE Agent Control Panel (侧边栏视图):** 作为与智能体交互的主要入口点。用户可在此提交任务、查看高级状态、配置智能体行为（包括**编辑用户自定义宪法**）。
* **Debate Visualizer (Webview 面板):** 一个交互式面板，使用序列图实时渲染 DebateEngine 会话的流程。用户可以清晰地看到每个智能体的发言、主持人的引导以及最终结论的形成过程。
* **HighDimension Report (Webview 面板):** 用于显示 HighDimensionModule 分析结果的视图。它将使用交互式图形（如力导向图）来展示依赖关系分析，并使用表格和图表来呈现风险评估。
* **MemoryNexus Explorer (侧边栏视图):** 一个开发者工具，允许用户检查智能体的知识图谱和语义记忆。这有助于调试智能体的行为和理解其“心智模型”。

### 9.2. 数据驱动的可视化模式

构建 UI 最稳健和可扩展的方式是让后端生成所需可视化的声明性 JSON 表示，而不是让前端解析复杂的领域特定数据。这种模式借鉴了 Debug Visualizer 等扩展的成功实践 34。后端不发送原始数据日志，而是发送一个描述可视化本身的结构化 JSON 对象。

* **可视化模式示例:**  
  JSON  
  {  
   "kind": { "mermaidSequence": true },  
   "title": "关于重构策略的辩论",  
   "data": "sequenceDiagram\n participant Moderator\n participant AgentA as \"Pro-Strangler Fig\"\n participant AgentB as \"Pro-Big Bang\"\n\n Moderator->>AgentA: 请提出你的策略。\n activate AgentA\n AgentA-->>Moderator: 我建议采用绞杀者无花果模式...\n deactivate AgentA\n\n Moderator->>AgentB: 你的看法？\n activate AgentB\n AgentB-->>Moderator: 我认为一次性重构更有效率...\n deactivate AgentB"  
  }
* 扩展的 Webview 组件将包含通用渲染器，接收此类 JSON 并使用 Mermaid.js 36 等库来生成视觉输出。这种设计将后端逻辑与前端呈现解耦，使得未来添加新的可视化类型变得非常简单，无需更改扩展的核心逻辑。

### 9.3. 人机回圈 (HITL) 交互设计

UI 不仅仅是数据显示器，它是建立用户对强大 AI 系统信任的主要界面。因此，其设计必须明确地遵循 HITL AI UX 的原则 39。

* **关键 HITL 特性:**
  + **明确确认 (Explicit Confirmation):** 对于任何会修改代码或项目状态的动作，智能体将首先呈现其计划和预期结果，并要求用户明确批准后才能继续执行。
  + **透明度与可解释性 (Transparency and Explainability):** 可视化不仅显示结果，还显示过程。辩论可视化器将展示哪些论点被剪枝。影响分析将高亮显示依赖图中的关键路径。
  + **置信度评分 (Confidence Scoring):** 在呈现结论时，智能体将显示一个置信度分数，并提供访问导致该结论的源数据或辩论过程的链接，允许用户“校准他们的信任” 40。

### 9.4. 后端通信 API

该扩展将通过一个安全的 WebSocket 连接与一个专用的 ZSCE API 网关服务进行通信，以实现实时的双向通信。对于频率较低的请求-响应式交互，将使用 REST API。该 API 网关随后将使用第七节中定义的 gRPC 协议与内部组件进行通信。

## X. 附录

### A. 统一 gRPC 服务定义

*此部分将包含所有 .proto 文件的完整内容，作为开发者的权威参考。*

Protocol Buffers

// Placeholder for the complete.proto file contents.  
// Example for MemoryService:  
syntax = "proto3";  
  
package zsce.agent;  
  
message RetrieveContextRequest {  
 string session\_id = 1;  
 string query\_text = 2;  
 int32 vector\_top\_k = 3;  
 int32 graph\_depth = 4;  
}  
  
message ContextChunk {  
 string source = 1;  
 string content = 2;  
 float relevance\_score = 3;  
}  
  
message ContextBundle {  
 string request\_id = 1;  
 repeated ContextChunk chunks = 2;  
}  
  
service MemoryService {  
 rpc RetrieveContext(RetrieveContextRequest) returns (ContextBundle);  
 //... other RPCs  
}

### B. 统一数据模式

*此部分将重申并提供所有 JSON 模式和 SQL DDL 语句的完整参考。*

* **数据库模式:** 参见第六节中的 **表 2: AgentMemoryNexus 统一模式 (SQL DDL)**。
* **工具定义 JSON 模式:**  
  JSON  
  {  
   "$schema": "http://json-schema.org/draft-07/schema#",  
   "title": "ZSCE Tool Definition",  
   "type": "object",  
   "properties": {  
   "tool\_name": { "type": "string" },  
   "semantic\_description": { "type": "string" },  
   "execution\_contract": {  
   // Reference to agents.json schema for flows and links  
   "type": "object"  
   }  
   },  
   "required": ["tool\_name", "semantic\_description", "execution\_contract"]  
  }

### C. 端到端交互流程图

*此部分将包含一系列详细的 Mermaid.js 序列图，以说明关键的用户工作流程。*

**工作流示例：分析一个拉取请求 (Pull Request)**

Code snippet

sequenceDiagram  
 participant User  
 participant VSCodeExtension as ZSCE UI  
 participant APIGateway as API Gateway  
 participant MeditationModule as MM  
 participant ToolSelectionModule as TSM  
 participant HighDimensionModule as HDM  
 participant AgentMemoryNexus as Memory  
  
 User->>VSCodeExtension: 提交 PR 链接进行分析  
 VSCodeExtension->>APIGateway: POST /analysis/pr (pr\_url)  
 APIGateway->>MM: gRPC: FrameProblem(pr\_url)  
 activate MM  
 MM-->>APIGateway: 返回核心洞见报告  
 deactivate MM  
   
 APIGateway->>TSM: gRPC: StartAnalysis(CoreInsightReport)  
 activate TSM  
  
 TSM->>TSM: 选择 "Git" 工具获取 PR 详情  
 TSM->>Memory: gRPC: RetrieveContext("PR analysis context")  
 Memory-->>TSM: 返回相关历史分析  
 TSM->>HDM: gRPC: RunMacroAnalysis(changed\_files)  
 activate HDM  
 HDM-->>TSM: 返回宏观影响报告  
 deactivate HDM  
  
 TSM->>HDM: gRPC: RunMicroAnalysis(changed\_code)  
 activate HDM  
 HDM-->>TSM: 返回并发风险报告  
 deactivate HDM  
  
 TSM->>Memory: gRPC: StoreMemory(analysis\_reports)  
 TSM-->>APIGateway: 返回综合报告  
 deactivate TSM  
 APIGateway-->>VSCodeExtension: WebSocket: push\_report(report\_json)  
 VSCodeExtension->>User: 渲染交互式影响报告

#### Works cited

1. wild-card-ai/agents-json - GitHub, accessed September 2, 2025, <https://github.com/wild-card-ai/agents-json>
2. What is Agent Communication Protocol (ACP)? | IBM, accessed September 2, 2025, <https://www.ibm.com/think/topics/agent-communication-protocol>
3. gRPC vs REST - Difference Between Application Designs - AWS, accessed September 2, 2025, <https://aws.amazon.com/compare/the-difference-between-grpc-and-rest/>
4. gRPC vs REST: When to Choose One Over the Other in Microservice Architectures | by RoshanGavandi, accessed September 2, 2025, <https://roshancloudarchitect.me/grpc-vs-rest-when-to-choose-one-over-the-other-in-microservice-architectures-f0ccd9e599e5>
5. What is Constitutional AI? - PromptLayer, accessed September 2, 2025, <https://www.promptlayer.com/glossary/constitutional-ai>
6. Debate-to-Write: A Persona-Driven Multi-Agent Framework for Diverse Argument Generation, accessed September 2, 2025, <https://aclanthology.org/2025.coling-main.314/>
7. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate - ACL Anthology, accessed September 2, 2025, <https://aclanthology.org/2024.emnlp-main.992.pdf>
8. You just need one prompt to become a prompt engineer! : r/PromptEngineering - Reddit, accessed September 2, 2025, <https://www.reddit.com/r/PromptEngineering/comments/1lkwefr/you_just_need_one_prompt_to_become_a_prompt/>
9. Debate Multi-Agent Architectures - Swarms, accessed September 2, 2025, <https://docs.swarms.world/en/latest/swarms/structs/orchestration_methods/>
10. NeurIPS Poster Multi-LLM Debate: Framework, Principals, and ..., accessed September 2, 2025, <https://neurips.cc/virtual/2024/poster/93363>
11. Introducing Impact Analysis for Architectural Decisions, accessed September 2, 2025, <https://www.inf.uni-hamburg.de/en/inst/ab/swk/research/publications/pdf/2007-paper-riebischm-impact-analysis.pdf>
12. Impact Analysis in Software Development: A Complete Guide, accessed September 2, 2025, <https://brewstudio.in/impact-analysis-in-software-development-a-practical-guide/>
13. Software Architecture Design and Analysis, accessed September 2, 2025, <https://www.sei.cmu.edu/training/software-architecture-design-analysis/>
14. Concurrency testing: Definition, types & how it works- Tricentis, accessed September 2, 2025, <https://www.tricentis.com/learn/concurrency-testing-a-detailed-overview>
15. Concurrency Testing: Benefits, Tools, and Best Practices | BrowserStack, accessed September 2, 2025, <https://www.browserstack.com/guide/concurrency-testing>
16. Concurrency Testing: What Is It, Benefits & Tools | PFLB, accessed September 2, 2025, <https://pflb.us/blog/concurrency-testing/>
17. Tool Use Design Pattern - ai-agents-for-beginners | 11 Lessons to Get Started Building AI Agents, accessed September 2, 2025, <https://microsoft.github.io/ai-agents-for-beginners/04-tool-use/>
18. Is a Semantic Layer Necessary for Enterprise - Grade AI Agents? - Tellius, accessed September 2, 2025, <https://www.tellius.com/resources/blog/is-a-semantic-layer-necessary-for-enterprise-grade-ai-agents>
19. Schema Design for Agent Memory and LLM History | by Pranav Prakash I GenAI I AI/ML I DevOps I | Jul, 2025 | Medium, accessed September 2, 2025, <https://medium.com/@pranavprakash4777/schema-design-for-agent-memory-and-llm-history-38f5cbc126fb>
20. Vector database vs. graph database: Knowledge Graph impact - WRITER, accessed September 2, 2025, <https://writer.com/engineering/vector-database-vs-graph-database/>
21. Vector databases vs. knowledge graphs for streaming data applications - Redpanda, accessed September 2, 2025, <https://www.redpanda.com/blog/vector-databases-vs-knowledge-graphs>
22. Vectors or Graphs? What Every Data Engineer Needs to Know - Superteams.ai, accessed September 2, 2025, <https://www.superteams.ai/blog/vectors-or-graphs-what-every-data-engineer-needs-to-know>
23. www.ibm.com, accessed September 2, 2025, <https://www.ibm.com/think/topics/ai-agent-memory#:~:text=By%20integrating%20vector%20databases%2C%20agentic,user%20preferences%20and%20past%20modifications.>
24. Vector Databases: The Memory System Powering Intelligent AI Conversations - Medium, accessed September 2, 2025, <https://medium.com/@usb1508/vector-databases-the-memory-system-powering-intelligent-ai-conversations-c248d72bd0aa>
25. Vector Databases vs. Knowledge Graphs for RAG | Paragon Blog, accessed September 2, 2025, <https://www.useparagon.com/blog/vector-database-vs-knowledge-graphs-for-rag>
26. My thoughts on choosing a graph databases vs vector databases : r/Rag - Reddit, accessed September 2, 2025, <https://www.reddit.com/r/Rag/comments/1ka88og/my_thoughts_on_choosing_a_graph_databases_vs/>
27. gRPC vs. REST: Key Similarities and Differences - DreamFactory Blog, accessed September 2, 2025, <https://blog.dreamfactory.com/grpc-vs-rest-how-does-grpc-compare-with-traditional-rest-apis>
28. A Deep Dive into Communication Styles for Microservices: REST vs. gRPC vs. Message Queues | by Platform Engineers | Medium, accessed September 2, 2025, <https://medium.com/@platform.engineers/a-deep-dive-into-communication-styles-for-microservices-rest-vs-grpc-vs-message-queues-ea72011173b3>
29. gRPC vs. REST: Comparing Key API Designs And Deciding Which One is Best - Wallarm, accessed September 2, 2025, <https://www.wallarm.com/what/grpc-vs-rest-comparing-key-api-designs-and-deciding-which-one-is-best>
30. AI's Ten Commandments: Constitutional Prompts for Ethical AI Models | by Stefan - Medium, accessed September 2, 2025, <https://medium.com/@andelkovics98/ais-ten-commandments-constitutional-prompts-for-ethical-ai-models-8636b84683b4>
31. [P] Constitutional AI recipe with open LLMs : r/MachineLearning - Reddit, accessed September 2, 2025, <https://www.reddit.com/r/MachineLearning/comments/1akdv4i/p_constitutional_ai_recipe_with_open_llms/>
32. How Effective Is Constitutional AI in Small LLMs? A Study on DeepSeek-R1 and Its Peers, accessed September 2, 2025, <https://arxiv.org/html/2503.17365v1>
33. Extension API | Visual Studio Code Extension API, accessed September 2, 2025, <https://code.visualstudio.com/api>
34. Visualize Data Structures in VSCode - Addy Osmani, accessed September 2, 2025, <https://addyosmani.com/blog/visualize-data-structures-vscode/>
35. Debug Visualizer - Visual Studio Marketplace, accessed September 2, 2025, <https://marketplace.visualstudio.com/items?itemName=hediet.debug-visualizer>
36. Sequence Diagram - Mermaid Chart, accessed September 2, 2025, <https://docs.mermaidchart.com/mermaid-oss/syntax/sequenceDiagram.html>
37. mermaid-js/mermaid: Generation of diagrams like flowcharts or sequence diagrams from text in a similar manner as markdown - GitHub, accessed September 2, 2025, <https://github.com/mermaid-js/mermaid>
38. Creating sequence diagrams using mermaidjs to map out your user journey, accessed September 2, 2025, <https://mfyz.medium.com/creating-sequence-diagrams-using-mermaidjs-to-map-out-your-user-journey-85e07e2f36bb>
39. Right Human-in-the-Loop Is Critical for Effective AI | Medium, accessed September 2, 2025, <https://medium.com/@dickson.lukose/building-a-smarter-safer-future-why-the-right-human-in-the-loop-is-critical-for-effective-ai-b2e9c6a3386f>
40. UX design for humans in the loop. Building trust in AI-powered B2B SaaS | by Roxanne Leitão | Bootcamp | Jul, 2025 | Medium, accessed September 2, 2025, <https://medium.com/design-bootcamp/ux-design-for-humans-in-the-loop-a2c80e4ff5b7>
41. Problem Statements: How to Use the Problem Framing Method - Atlassian, accessed September 2, 2025, <https://www.atlassian.com/team-playbook/plays/problem-framing>
42. Framing an ML problem | Machine Learning - Google for Developers, accessed September 2, 2025, <https://developers.google.com/machine-learning/problem-framing/ml-framing>
43. Frame problem - Wikipedia, accessed September 2, 2025, <https://en.wikipedia.org/wiki/Frame_problem>
44. frame problem - Autoblocks AI — Build Safe AI Apps, accessed September 2, 2025, <https://www.autoblocks.ai/glossary/frame-problem>
45. What is emergent behavior in multi-agent systems? - Milvus, accessed September 2, 2025, <https://milvus.io/ai-quick-reference/what-is-emergent-behavior-in-multiagent-systems>
46. Emergent Intelligence in Large Scale Multi-Agent Systems - NAUN, accessed September 2, 2025, <https://www.naun.org/main/NAUN/computers/ijcomputers-34.pdf>
47. System instructions for safety | Generative AI on Vertex AI - Google Cloud, accessed September 2, 2025, <https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/safety-system-instructions>