# 智能体白皮书总结

- 引言 (Introduction)
  - 背景
    人类处理复杂模式识别能力强，但常借助工具弥补知识不足。
    生成式 AI 模型同样可以训练使用工具访问实时信息或执行现实世界操作。<br>
  - 智能体概念的引出
    模型不仅需访问外部工具，还需自主规划和执行任务。
    这种将推理、逻辑及访问外部信息能力结合到生成式 AI 模型上的做法，引出了智能体 (agent) 的概念。...
- 什么是智能体? (What is an agent?)
  - 定义
    基本形式：一个应用程序，通过观察世界并使用其可支配的工具采取行动来达成目标。
    特性：自主 (autonomous)、主动 (proactive)。
    本文重点：基于「生成式 AI 模型」构建的特定类型智能体。<br>
  - 核心组件与认知架构 (Cognitive Architecture)
    驱动智能体行为 (behavior)、行动 (action) 和决策 (decision making) 的基础组件组合。
    关注核心功能，智能体认知架构包含三个基本组件：<br>
    - 模型 (The model)
      智能体流程的中心决策者，通常是语言模型 (LM)。
      能够遵循指令进行推理和逻辑，如 ReAct、思维链 (CoT)、思维树 (ToT)。
      可以是通用、多模态或微调过的模型。...
    - 工具 (The tools)
      弥合模型与外部世界交互的差距。
      使智能体能与外部数据和服务互动，解锁超越模型本身的能力。
      形式多样，常与 Web API (GET, POST 等) 类似。...
    - 编排层 (The orchestration layer)
      描述智能体处理信息、内部推理、指导行动/决策的周期性过程。
      循环持续直到目标达成或遇停止点。
      复杂性各异，可包含简单规则、链式逻辑、机器学习算法或概率推理。<br>
  - 智能体 vs. 模型 (Agents vs. models)
    - 模型 (Models)
      知识局限于训练数据。
      基于用户查询的单次推理/预测，无原生会话历史或上下文管理。
      无原生工具实现。
      无原生逻辑层，需通过提示工程（如 CoT, ...
    - 智能体 (Agents)
      知识通过工具连接外部系统扩展。
      管理会话历史，支持多轮推理/预测，每次交互 (turn) 包含输入和响应。
      原生实现工具。
      原生认知架构使用推理框架 (CoT, ReAct) 或预...
  - 认知架构: 智能体如何运作 (Cognitive architectures: How agents operate)
    - 运作方式
      类比厨师：收集信息 -内部推理 -> 采取行动 -> 根据反馈调整。
      核心是编排层，负责维护记忆、状态、推理和规划。
      利用提示工程及相关框架指导推理和规划。<br>
    - 常用框架和技术
      - ReAct
        结合推理 (Reason) 和行动 (Act)。<br>
      - 思维链 (Chain-of-Thought, CoT)
        通过中间步骤启用推理能力。有自洽性、主动提示、多模态等技术。<br>
      - 思维树 (Tree-of-thoughts, ToT)
        探索多种思维链作为问题解决步骤，适合探索或战略性任务。<br>
    - ReAct 示例流程
      1. 用户查询输入。
      2. 智能体启动 ReAct 序列。
      3. 智能体向模型提供提示，要求生成 ReAct 步骤（思考、行动、行动输入、观察）。
         - 行动 (Action): 决定采取何种行动（如选择工具或无工具）。
         - 行动输入 (Action input): 决定向工具提供何种输入。
         - 观察 (Observation): 行动/行动输入序列的结果。
         - (思考/行动/行动输入/观察可重复 N 次)
         - 最终答案 (Final answer): 模型提供给用户的最终答案。
      4. ReAct 循环结束，返回最终答案。
- 工具: 我们通往外部世界的钥匙 (Tools: Our keys to the outside world)
  - 工具的重要性
    语言模型缺乏直接感知和影响现实世界的能力。
    工具赋予模型与外部系统实时、上下文感知的交互能力。<br>
  - 工具类型
    主要有三种：扩展 (Extensions)、函数 (Functions)、数据存储 (Data Stores)。<br>
  - 扩展 (Extensions)
    - 定义
      以标准化方式弥合 API 和智能体之间空隙的桥梁。
      允许智能体无缝执行 API，无需关心底层实现。<br>
    - 工作原理
      通过示例教智能体如何使用 API 端点及所需参数。
      智能体根据用户查询和已知扩展示例，动态选择最适合任务的扩展。
      扩展独立于智能体制作，但作为配置一部分提...
    - 示例
      Google Flights 扩展、Google Maps 扩展、代码解释器 (Code Interpreter) 扩展。<br>
  - 函数 (Functions)
    - 定义
      类似软件工程中的函数：完成特定任务的自包含代码模块。
      模型决定使用哪个函数及所需参数，但不直接调用 API。<br>
    - 与扩展的区别
      1. 模型输出函数调用及其参数，不进行实时 API 调用。
      2. 函数在客户端 (client-side) 执行，扩展在智能体端 (agent-side) 执行。
    - 使用场景
      需要更精细控制数据流和执行：
      API 调用需在应用堆栈其他层进行（中间件、前端）。
      安全或身份验证限制阻止智能体直接调用 API。
      时间或操作顺序限制（批处理、人工审核）。...
    - 示例流程 (Function Calling)
      用户查询 -模型生成包含函数名和参数的 JSON -> 客户端应用接收 JSON -> 客户端执行相应逻辑（如调用 API） -> 返回结果给用户。<br>
  - 数据存储 (Data Stores)
    - 定义
      解决模型知识静态的问题，提供对动态和最新信息的访问。
      确保模型响应基于事实和相关性。<br>
    - 工作原理
      允许以原始格式（如 PDF, CSV, HTML, 数据库）提供数据给智能体。
      通常实现为向量数据库 (vector database)。
      将传入文档转换为向量嵌入 (vector embeddings)。...
    - 实现与应用 (RAG)
      常见应用：检索增强生成 (Retrieval Augmented Generation, RAG)。
      RAG 流程：<br>
      1. 用户查询 -> 嵌入模型 -> 查询嵌入。
      2. 查询嵌入与向量数据库内容匹配 (e.g, 使用 ScaNN)。
      3. 检索匹配内容 (文本格式)。
      4. 智能体接收用户查询和检索内容，制定响应或行动。
      5. 最终响应发送给用户。
  - 工具回顾 (Tools recap)
    - 总结表
- 通过定向学习增强模型性能 (Enhancing model performance with targeted learning)
  - 方法
    - 上下文学习 (In-context learning)
      推理时提供提示、工具、少量示例 (few-shot examples)。
      模型“即时 (on the fly)”学习任务。ReAct 是一个例子。<br>
    - 基于检索的上下文学习 (Retrieval-based in-context learning)
      从外部存储（如示例存储库、RAG 数据存储）检索相关信息、工具、示例，动态填充模型提示。<br>
    - 基于微调的学习 (Fine-tuning based learning)
      推理前用更大的特定示例数据集训练模型。
      帮助模型预先理解何时及如何应用工具。<br>
  - 权衡与结合
    每种方法在速度、成本、延迟方面各有优劣。
    结合使用可实现更健壮、适应性更强的解决方案。<br>
- 使用 Vertex AI 智能体的生产应用 (Production applications with Vertex AI agents)
  - 背景
    生产级应用需要将核心智能体组件与 UI、评估框架、持续改进机制等集成。<br>
  - Vertex AI 平台
    提供全托管环境简化生产应用构建。
    特点：
    自然语言界面定义智能体要素（目标、指令、工具、子智能体、示例）。
    开发工具支持测试、评估、衡量性能、调试、...
- 使用 LangChain 快速开始智能体 (Agent quick start with LangChain)
  - 概述
    使用 LangChain 和 LangGraph 库构建智能体原型。
    允许将逻辑、推理、工具调用序列“链接”起来。<br>
  - 示例
    使用 `gemini-1.5-flash-001` 模型、SerpAPI (搜索) 和 Google Places API 工具。
    演示了模型、编排 (ReAct) 和工具如何协同工作实现目标。<br>
- 通过定向学习增强模型性能 (Enhancing model performance with targeted learning)
  - 背景
    模型选择正确工具的能力至关重要，尤其在生产环境。
    现实场景常需超出通用训练数据的知识。<br>
  - 方法
    - 上下文学习 (In-context learning)
      推理时提供提示、工具、少量示例 (few-shot examples)。
      模型“即时 (on the fly)”学习任务。ReAct 是一个例子。<br>
    - 基于检索的上下文学习 (Retrieval-based in-context learning)
      从外部存储（如示例存储库、RAG 数据存储）检索相关信息、工具、示例，动态填充模型提示。<br>
    - 基于微调的学习 (Fine-tuning based learning)
      推理前用更大的特定示例数据集训练模型。
      帮助模型预先理解何时及如何应用工具。<br>
  - 权衡与结合
    每种方法在速度、成本、延迟方面各有优劣。
    结合使用可实现更健壮、适应性更强的解决方案。<br>
- 总结 (Summary)
  - 关键要点
    1. 智能体通过利用工具访问实时信息、建议行动、自主规划执行，扩展了语言模型能力。
    2. 核心是编排层（认知架构），构建推理、规划、决策（如 ReAct, CoT, ToT）。
    3. 工具（扩展、函数、数据存储）是智能体通往外部世界的钥匙。
  - 未来展望
    工具将更复杂，推理能力将增强。
    “智能体链 (agent chaining)”/“混合智能体专家 (mixture of agent experts)”策略将发展。
    构建复杂智能体需要迭代方法，实验和改进是关键。...