03-Part_Three (Bilingual)

03-Part_Three（中英對照）

Chapter 12: Exception Handling and Recovery

第 12 章：例外處理與復原

For AI agents to operate reliably in diverse real-world environments, they must be able to manage unforeseen situations, errors, and malfunctions. Just as humans adapt to unexpected obstacles, intelligent agents need robust systems to detect problems, initiate recovery procedures, or at least ensure controlled failure. This essential requirement forms the basis of the Exception Handling and Recovery pattern.

要讓 AI 代理人在多樣的真實世界環境中可靠運作，它們必須能管理突發情況、錯誤與故障。就像人類會適應意外障礙，智慧代理人需要健全的系統來偵測問題、啟動復原程序，或至少確保可控的失敗。這項基本需求構成了「例外處理與復原」模式的基礎。

This pattern focuses on developing exceptionally durable and resilient agents that can maintain uninterrupted functionality and operational integrity despite various difficulties and anomalies. It emphasizes the importance of both proactive preparation and reactive strategies to ensure continuous operation, even when facing challenges. This adaptability is critical for agents to function successfully in complex and unpredictable settings, ultimately boosting their overall effectiveness and trustworthiness.

此模式聚焦於打造格外耐久且具韌性的代理人，使其即使遭遇各種困難與異常，也能維持不中斷的功能與運作完整性。它強調事前準備與事後應對策略的重要性，以便在面對挑戰時仍能持續運作。這種適應力是代理人在複雜且不可預測環境中成功運作的關鍵，最終提升其整體效能與可信度。

The capacity to handle unexpected events ensures these AI systems are not only intelligent but also stable and reliable, which fosters greater confidence in their deployment and performance. Integrating comprehensive monitoring and diagnostic tools further strengthens an agent’s ability to quickly identify and address issues, preventing potential disruptions and ensuring smoother operation in evolving conditions. These advanced systems are crucial for maintaining the integrity and efficiency of AI operations, reinforcing their ability to manage complexity and unpredictability.

處理突發事件的能力確保這些 AI 系統不僅聰明，而且穩定可靠，進而提升人們對其部署與效能的信心。整合完善的監控與診斷工具，能進一步強化代理人快速辨識並處理問題的能力，避免潛在中斷，並確保在變動情境下更順暢的運作。這些進階系統對維持 AI 作業的完整性與效率至關重要，強化其管理複雜性與不可預測性的能力。

This pattern may sometimes be used with reflection. For example, if an initial attempt fails and raises an exception, a reflective process can analyze the failure and reattempt the task with a refined approach, such as an improved prompt, to resolve the error.

此模式有時會與反思搭配使用。例如當初次嘗試失敗並引發例外時，反思流程可分析失敗原因，並以更精煉的方法重新嘗試任務，例如改良提示，以解決錯誤。

Exception Handling and Recovery Pattern Overview

例外處理與復原模式概觀

The Exception Handling and Recovery pattern addresses the need for AI agents to manage operational failures. This pattern involves anticipating potential issues, such as tool errors or service unavailability, and developing strategies to mitigate them. These strategies may include error logging, retries, fallbacks, graceful degradation, and notifications. Additionally, the pattern emphasizes recovery mechanisms like state rollback, diagnosis, self-correction, and escalation, to restore agents to stable operation. Implementing this pattern enhances the reliability and robustness of AI agents, allowing them to function in unpredictable environments. Examples of practical applications include chatbots managing database errors, trading bots handling financial errors, and smart home agents addressing device malfunctions. The pattern ensures that agents can continue to operate effectively despite encountering complexities and failures.

例外處理與復原模式回應了 AI 代理人在運作中管理故障的需求。此模式包含預先預測潛在問題（如工具錯誤或服務不可用），並制定降低影響的策略。這些策略可能包括錯誤記錄、重試、備援方案、優雅降級與通知。此外，模式也強調復原機制，例如狀態回滾、診斷、自我修正與升級處理，以將代理人恢復到穩定運作。實作此模式能提升 AI 代理人的可靠性與韌性，使其能在不可預測的環境中運作。實務應用例子包括聊天

機器人處理資料庫錯誤、交易機器人處理金融錯誤，以及智慧家庭代理人處理裝置故障。此模式確保代理人即使遭遇複雜性與失敗，仍能持續有效運作。

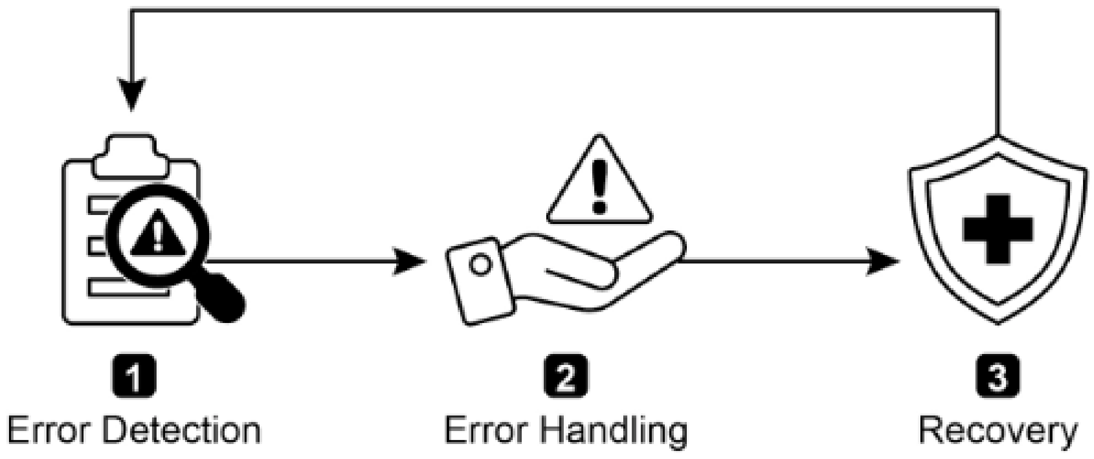

Figure 1: Key Components of Exception Handling and Recovery for AI agents

Fig.1: Key components of exception handling and recovery for AI agents

圖 1：AI 代理人例外處理與復原的關鍵元件

Error Detection: This involves meticulously identifying operational issues as they arise. This could manifest as invalid or malformed tool outputs, specific API errors such as 404 (Not Found) or 500 (Internal Server Error) codes, unusually long response times from services or APIs, or incoherent and nonsensical responses that deviate from expected formats. Additionally, monitoring by other agents or specialized monitoring systems might be implemented for more proactive anomaly detection, enabling the system to catch potential issues before they escalate.

錯誤偵測： 這包括在問題發生時仔細辨識操作異常。可能表現為工具輸出無效或格式錯誤、特定 API 錯誤（如 404 Not Found 或 500 Internal Server Error）、服務或 API 回應時間異常過長，或偏離預期格式的無意義回應。此外，也可以透過其他代理或專用監控系統進行更主動的異常偵測，讓系統在問題擴大前先行捕捉。

Error Handling: Once an error is detected, a carefully thought-out response plan is essential. This includes recording error details meticulously in logs for later debugging and analysis (logging). Retrying the action or request, sometimes with slightly adjusted parameters, may be a viable strategy, especially for transient errors (retries). Utilizing alternative strategies or methods (fallbacks) can ensure that some functionality is maintained. Where complete recovery is not immediately possible, the agent can maintain partial functionality to provide at least some value (graceful degradation). Finally, alerting human operators or other agents might be crucial for situations that require human intervention or collaboration (notification).

錯誤處理： 一旦偵測到錯誤，就需要周全的回應計畫，包括將錯誤細節仔細記錄在日誌中以利後續除錯與分析（logging）。重試該動作或請求，有時可調整參數，特別適用於暫時性錯誤（retries）。採用替代策略或方法（fallbacks）能維持部分功能。若無法立即完全復原，代理人可維持部分功能以提供至少一些價值（graceful degradation）。最後，當情況需要人類介入或協作時，通知人類操作員或其他代理可能至關重要（notification）。

Recovery: This stage is about restoring the agent or system to a stable and operational state after an error. It could involve reversing recent changes or transactions to undo the effects of the error (state rollback). A thorough investigation into the cause of the error is vital for preventing recurrence. Adjusting the agent's plan, logic, or parameters through a self-correction mechanism or replanning process may be needed to avoid the same error in the future. In complex or severe cases, delegating the issue to a human operator or a higher-level system (escalation) might be the best course of action.

復原： 這個階段著重於在錯誤發生後將代理人或系統恢復至穩定可運作狀態。可能涉及回復近期變更或交易以撤銷錯誤影響（狀態回滾）。對錯誤原因進行徹底調查是防止再次發生的關鍵。可能需要透過自我修正機制或重新規劃流程調整代理人的計畫、邏輯或參數，以避免未來重蹈覆轍。在複雜或嚴重的情況下，將問題交由人類操作員或更高層級系統處理（升級）可能是最佳方案。

Implementation of this robust exception handling and recovery pattern can transform AI agents from fragile and unreliable systems into robust, dependable components capable of operating effectively and resiliently in challenging and highly unpredictable environments. This ensures that the agents maintain functionality, minimize downtime, and provide a seamless and reliable experience even when faced with unexpected issues.

實作這套健全的例外處理與復原模式，可將 AI 代理人從脆弱且不可靠的系統，轉變為能在充滿挑戰且高度不可預測環境中有效且具韌性運作的可靠元件。這能確保代理人維持功能、降低停機時間，並在面對意外問題時仍提供順暢且可靠的體驗。

Practical Applications & Use Cases

實務應用與使用情境

Exception Handling and Recovery is critical for any agent deployed in a real-world scenario where perfect conditions cannot be guaranteed.

例外處理與復原對任何部署在真實世界、無法保證完美條件的代理人都至關重要。

- **Customer Service Chatbots:** If a chatbot tries to access a customer database and the database is temporarily down, it shouldn't crash. Instead, it should detect the API error, inform the user about the temporary issue, perhaps suggest trying again later, or escalate the query to a human agent.
- **Automated Financial Trading:** A trading bot attempting to execute a trade might encounter an "insufficient funds" error or a "market closed" error. It needs to handle these exceptions by logging the error, not repeatedly trying the same invalid trade, and potentially notifying the user or adjusting its strategy.
- **Smart Home Automation:** An agent controlling smart lights might fail to turn on a light due to a network issue or a device malfunction. It should detect this failure, perhaps retry, and if still unsuccessful, notify the user that the light could not be turned on and suggest manual intervention.
- **Data Processing Agents:** An agent tasked with processing a batch of documents might encounter a corrupted file. It should skip the corrupted file, log the error, continue processing other files, and report the skipped files at the end rather than halting the entire process.
- **Web Scraping Agents:** When a web scraping agent encounters a CAPTCHA, a changed website structure, or a server error (e.g., 404 Not Found, 503 Service Unavailable), it needs to handle these gracefully. This could involve pausing, using a proxy, or reporting the specific URL that failed.
- **Robotics and Manufacturing:** A robotic arm performing an assembly task might fail to pick up a component due to misalignment. It needs to detect this failure (e.g., via sensor feedback), attempt to readjust, retry the pickup, and if persistent, alert a human operator or switch to a different component.
- **客服聊天機器人：** 若聊天機器人嘗試存取客戶資料庫而該資料庫暫時無法使用，它不應當機。相反地，它應偵測 API 錯誤、告知使用者臨時問題，或建議稍後再試，或將問題升級給真人客服。

- **自動化金融交易：** 交易機器人在執行交易時可能遇到「資金不足」或「市場已關閉」等錯誤。它需要記錄錯誤、不重複嘗試相同的無效交易，並可能通知使用者或調整策略。
- **智慧家庭自動化：** 控制智慧燈的代理人可能因網路問題或裝置故障而無法開燈。它應偵測失敗、嘗試重試，若仍無法成功，則通知使用者燈無法開啟並建議手動介入。
- **資料處理代理人：** 負責處理一批文件的代理人可能遇到損毀檔案。它應跳過該檔案、記錄錯誤、繼續處理其他檔案，並在結尾回報被跳過的檔案，而非停止整個流程。
- **網頁爬蟲代理人：** 當爬蟲遇到 CAPTCHA、網站結構變更或伺服器錯誤（如 404 Not Found、503 Service Unavailable）時，需要優雅處理，例如暫停、使用代理伺服器，或回報失敗的特定網址。
- **機器人與製造業：** 執行組裝任務的機械手臂可能因對位不準而無法拾取零件。它需要偵測此失敗（例如透過感測器回饋）、嘗試調整、重試拾取，若持續失敗，則通知人類操作員或改用其他零件。

In short, this pattern is fundamental for building agents that are not only intelligent but also reliable, resilient, and user-friendly in the face of real-world complexities.

總之，此模式是打造面對真實世界複雜性時仍具智慧、可靠、韌性且友善的代理人的基礎。

Hands-On Code Example (ADK)

實作程式碼範例（ADK）

Exception handling and recovery are vital for system robustness and reliability. Consider, for instance, an agent’s response to a failed tool call. Such failures can stem from incorrect tool input or issues with an external service that the tool depends on.

例外處理與復原對系統的強健性與可靠性至關重要。以代理人對失敗工具呼叫的回應為例，這類失敗可能源於工具輸入錯誤，或工具所依賴的外部服務出現問題。

```python
from google.adk.agents import Agent, SequentialAgent


# Agent 1: Tries the primary tool. Its focus is narrow and clear.
primary_handler = Agent(
    name="primary_handler",
    model="gemini-2.0-flash-exp",
    instruction="""
    Your job is to get precise location information. Use the
get_precise_location_info
    tool with the user's provided address.
    """,
    tools=[get_precise_location_info],
)

# Agent 2: Acts as the fallback handler, checking state to decide its action.
fallback_handler = Agent(
    name="fallback_handler",
    model="gemini-2.0-flash-exp",
    instruction="""
    Check if the primary location lookup failed by looking at
state["primary_location_failed"].
    - If it is True, extract the city from the user's original query and use
the get_general_area_info tool.
    - If it is False, do nothing.
    """,
    tools=[get_general_area_info],
)

# Agent 3: Presents the final result from the state.
response_agent = Agent(
    name="response_agent",
    model="gemini-2.0-flash-exp",
    instruction="""
    Review the location information stored in state["location_result"].
Present this information
    clearly and concisely to the user. If state["location_result"] does not
exist or is empty,
    apologize that you could not retrieve the location.
    """,
    tools=[],  # This agent only reasons over the final state.
)

# The SequentialAgent ensures the handlers run in a guaranteed order.
robust_location_agent = SequentialAgent(
    name="robust_location_agent",
    sub_agents=[primary_handler, fallback_handler, response_agent],
)
```

This code defines a robust location retrieval system using a ADK's SequentialAgent with three sub-agents. The `primary_handler` is the first agent, attempting to get precise location information using the `get_precise_location_info` tool. The `fallback_handler` acts as a backup, checking if the primary lookup failed by inspecting a state variable. If the primary lookup failed, the fallback agent extracts the city from the user's query and uses the `get_general_area_info` tool. The `response_agent` is the final agent in the sequence. It reviews the location information stored in the state. This agent is designed to present the final result to the user. If no location information was found, it apologizes. The SequentialAgent ensures that these three agents execute in a predefined order. This structure allows for a layered approach to location information retrieval.

這段程式碼使用 ADK 的 SequentialAgent 與三個子代理，定義了一個強健的地點資訊查詢系統。`primary_handler` 是第一個代理，嘗試使用 `get_precise_location_info` 工具取得精確位置資訊。`fallback_handler` 作為備援，透過檢查 state 變數判斷主要查詢是否失敗。若主要查詢失敗，備援代理會從使用者原始查詢中抽取城市並使用 `get_general_area_info` 工具。`response_agent` 是序列中最後的代理，負責檢視 state 中儲存的地點資訊並向使用者呈現結果。若沒有取得地點資訊，它會致歉。SequentialAgent 確保三個代理依照預定順序執行，讓地點資訊查詢採用分層式流程。

At a Glance

一覽

What: AI agents operating in real-world environments inevitably encounter unforeseen situations, errors, and system malfunctions. These disruptions can range from tool failures and network issues to invalid data, threatening the agent's ability to complete its tasks. Without a structured way to manage these problems, agents can be fragile, unreliable, and prone to complete failure when faced with unexpected hurdles. This unreliability makes it difficult to deploy them in critical or complex applications where consistent performance is essential.

什麼： 在真實世界環境中運作的 AI 代理人不可避免會遇到意外情況、錯誤與系統故障。這些中斷可能來自工具失敗、網路問題到無效資料，威脅代理完成任務的能力。若沒有結構化的管理方式，代理容易變得脆弱、不可靠，面對突發障礙時甚至完全失敗。這種不可靠性讓其難以部署於需要穩定表現的關鍵或複雜應用。

Why: The Exception Handling and Recovery pattern provides a standardized solution for building robust and resilient AI agents. It equips them with the agentic capability to anticipate, manage, and recover from operational failures. The pattern involves proactive error detection, such as monitoring tool outputs and API responses, and reactive handling strategies like logging for diagnostics, retrying transient failures, or using fallback mechanisms. For more severe issues, it defines recovery protocols, including reverting to a stable state, self-correction by adjusting its plan, or escalating the problem to a human operator. This systematic approach ensures agents can maintain operational integrity, learn from failures, and function dependably in unpredictable settings.

為什麼： 例外處理與復原模式提供一套標準化方案，用來打造強健且具韌性的 AI 代理人。它賦予代理人預測、管理並從運作失敗中復原的能力。此模式包含主動式錯誤偵測，例如監控工具輸出與 API 回應，以及被動式處理策略，如記錄日誌以診斷、重試暫時性失敗或使用備援機制。對於更嚴重的問題，它定義復原流程，包括回復到穩定狀態、透過調整計畫進行自我修正，或將問題升級給人類操作員。這種系統化方法能確保代理維持運作完整性、從失敗中學習，並在不可預測的環境中可靠運作。

Rule of Thumb: Use this pattern for any AI agent deployed in a dynamic, real-world environment where system failures, tool errors, network issues, or unpredictable inputs are possible and operational reliability is a key requirement.

經驗法則： 在動態的真實世界環境中部署任何 AI 代理人時，只要可能遭遇系統故障、工具錯誤、網路問題或不可預測輸入，且運作可靠性是關鍵需求，就應使用此模式。

Visual Summary:

視覺摘要：

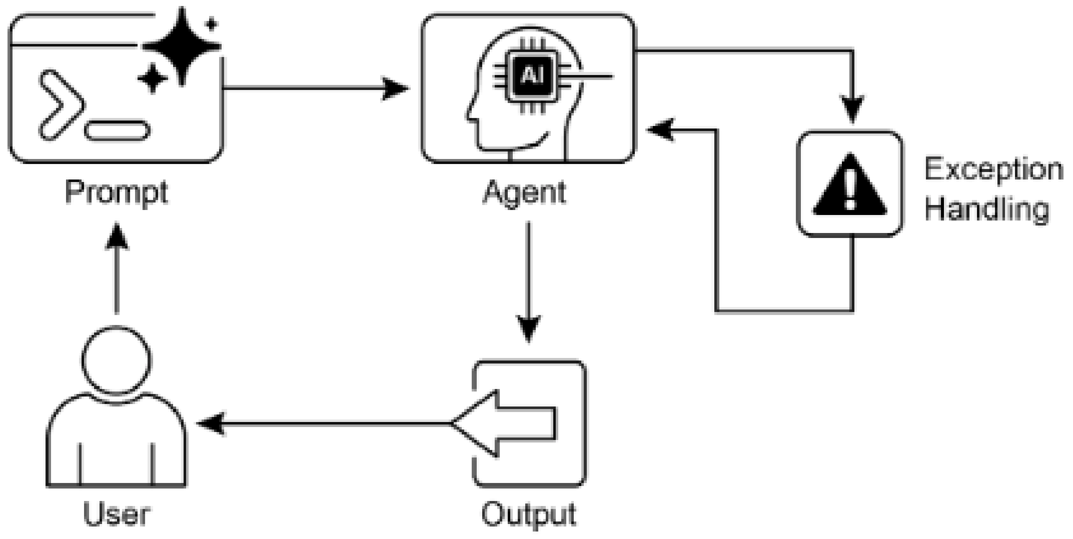

Figure 2: Exception Handling Pattern

Fig.2: Exception handling pattern

圖 2：例外處理模式

Key Takeaways

重要重點

Essential points to remember:

重要重點如下：

- Exception Handling and Recovery is essential for building robust and reliable Agents.
- This pattern involves detecting errors, handling them gracefully, and implementing strategies to recover.
- Error detection can involve validating tool outputs, checking API error codes, and using timeouts.
- Handling strategies include logging, retries, fallbacks, graceful degradation, and notifications.
- Recovery focuses on restoring stable operation through diagnosis, self-correction, or escalation.
- This pattern ensures agents can operate effectively even in unpredictable real-world environments.
- 例外處理與復原是打造強健且可靠代理人的關鍵。
- 此模式包含偵測錯誤、優雅處理錯誤與實施復原策略。
- 錯誤偵測可透過驗證工具輸出、檢查 API 錯誤碼與設定逾時來完成。
- 處理策略包含記錄日誌、重試、備援方案、優雅降級與通知。
- 復原著重於透過診斷、自我修正或升級來恢復穩定運作。
- 此模式確保代理人在不可預測的真實世界環境中仍能有效運作。

Conclusion

結論

This chapter explores the Exception Handling and Recovery pattern, which is essential for developing robust and dependable AI agents. This pattern addresses how AI agents can identify and manage unexpected issues, implement appropriate responses, and recover to a stable operational state. The chapter discusses various aspects of this pattern, including the detection of errors, the handling of these errors through mechanisms such as logging, retries, and fallbacks, and the strategies used to restore the agent or system to proper function. Practical applications of the Exception Handling and Recovery pattern are illustrated across several domains to demonstrate its relevance in handling real-world complexities and potential failures. These applications show how equipping AI agents with exception handling capabilities contributes to their reliability and adaptability in dynamic environments.

本章探討了例外處理與復原模式，這對打造強健且可靠的 AI 代理人至關重要。此模式說明 AI 代理人如何辨識並管理意外問題、實施適切回應，並復原至穩定的運作狀態。本章討論該模式的多個面向，包括錯誤偵測、透過記錄日誌、重試與備援等機制處理錯誤，以及用於恢復代理或系統正常功能的策略。透過多個領域的實務應用，展示該模式在處理真實世界複雜性與潛在失敗時的相關性。這些應用說明賦予 AI 代理例外處理能力如何提升其在動態環境中的可靠性與適應力。

References

參考資料

1. McConnell, S. (2004). Code Complete (2nd ed.). Microsoft Press.
2. Shi, Y., Pei, H., Feng, L., Zhang, Y., & Yao, D. (2024). Towards Fault Tolerance in Multi-Agent Reinforcement Learning. arXiv preprint arXiv:2412.00534.
3. O’Neill, V. (2022). Improving Fault Tolerance and Reliability of Heterogeneous Multi-Agent IoT Systems Using Intelligence Transfer. Electronics, 11(17), 2724.
4. McConnell, S.（2004）。《Code Complete（第 2 版）》。Microsoft Press。
5. Shi, Y., Pei, H., Feng, L., Zhang, Y., & Yao, D.（2024）。《Towards Fault Tolerance in Multi-Agent Reinforcement Learning》。arXiv preprint arXiv:2412.00534。
6. O’Neill, V.（2022）。《Improving Fault Tolerance and Reliability of Heterogeneous Multi-Agent IoT Systems Using Intelligence Transfer》。Electronics, 11(17), 2724。

Chapter 13: Human-in-the-Loop

第 13 章：人類參與迴圈

The Human-in-the-Loop (HITL) pattern represents a pivotal strategy in the development and deployment of Agents. It deliberately interweaves the unique strengths of human cognition—such as judgment, creativity, and nuanced understanding—with the computational power and efficiency of AI. This strategic integration is not merely an option but often a necessity, especially as AI systems become increasingly embedded in critical decision-making processes.

人類參與迴圈（HITL）模式是開發與部署代理人時的重要策略。它刻意將人類認知的獨特優勢——如判斷、創造力與細膩理解——與 AI 的運算能力與效率交織在一起。這種策略性整合不只是選項，往往是必要條件，尤其當 AI 系統越來越深入關鍵決策流程時更是如此。

The core principle of HITL is to ensure that AI operates within ethical boundaries, adheres to safety protocols, and achieves its objectives with optimal effectiveness. These concerns are particularly acute in domains characterized by complexity, ambiguity, or significant risk, where the implications of AI errors or misinterpretations can be substantial. In such scenarios, full autonomy—where AI systems function independently without any human intervention—may prove to be imprudent. HITL acknowledges this reality and emphasizes that even with rapidly advancing AI technologies, human oversight, strategic input, and collaborative interactions remain indispensable.

HITL 的核心原則是確保 AI 在倫理界線內運作、遵守安全規範，並以最佳效能達成目標。這些顧慮在複雜、模糊或高度風險的領域尤為尖銳，因為 AI 的錯誤或誤解可能造成重大影響。在此情境下，完全自主——也就是 AI 無需人類介入而獨立運作——往往並不審慎。HITL 認可這個現實並強調，即使 AI 技術快速進步，人類的監督、策略性投入與協作互動仍不可或缺。

The HITL approach fundamentally revolves around the idea of synergy between artificial and human intelligence. Rather than viewing AI as a replacement for human workers, HITL positions AI as a tool that augments and enhances human capabilities. This augmentation can take various forms, from automating routine tasks to providing data-driven insights that inform human decisions. The end goal is to create a collaborative ecosystem where both humans and AI Agents can leverage their distinct strengths to achieve outcomes that neither could accomplish alone.

HITL 的核心精神在於人工智慧與人類智慧之間的協同效應。HITL 並不將 AI 視為取代人類工作者，而是定位為增強人類能力的工具。這種增強可以有多種形式，從自動化例行工作到提供數據驅動洞見以支援人類決策。最終目標是打造一個協作生態系，讓人類與 AI 代理人能運用各自的長處，達成任一方單獨無法完成的成果。

In practice, HITL can be implemented in diverse ways. One common approach involves humans acting as validators or reviewers, examining AI outputs to ensure accuracy and identify potential errors. Another implementation involves humans actively guiding AI behavior, providing feedback or making corrections in real-time. In more complex setups, humans may collaborate with AI as partners, jointly solving problems or making decisions through interactive dialog or shared interfaces. Regardless of the specific implementation, the HITL pattern underscores the importance of maintaining human control and oversight, ensuring that AI systems remain aligned with human ethics, values, goals, and societal expectations.

在實務上，HITL 可以透過多種方式實作。常見做法之一是由人類擔任驗證者或審閱者，檢視 AI 輸出以確保正確性並找出潛在錯誤。另一種方式是人類主動引導 AI 行為，即時提供回饋或修正。在更複雜的設定中，人類可能與 AI 成為合作夥伴，透過互動對話或共享介面共同解決問題與做出決策。無論具體實作為何，HITL 模式都強調維持人類控制與監督的重要性，確保 AI 系統與人類倫理、價值、目標與社會期待一致。

Human-in-the-Loop Pattern Overview

人類參與迴圈模式概觀

The Human-in-the-Loop (HITL) pattern integrates artificial intelligence with human input to enhance Agent capabilities. This approach acknowledges that optimal AI performance frequently requires a combination of automated processing and human insight, especially in scenarios with high complexity or ethical considerations. Rather than replacing human input, HITL aims to augment human abilities by ensuring that critical judgments and decisions are informed by human understanding.

人類參與迴圈（HITL）模式將人工智慧與人類輸入結合，以強化代理人的能力。此方法承認最佳的 AI 表現往往需要自動化處理與人類洞見的結合，特別是在高度複雜或具

有倫理考量的情境中。HITL 不是取代人類輸入，而是透過人類理解來支撐關鍵判斷與決策，藉此增強人類能力。

HITL encompasses several key aspects: Human Oversight, which involves monitoring AI agent performance and output (e.g., via log reviews or real-time dashboards) to ensure adherence to guidelines and prevent undesirable outcomes. Intervention and Correction occurs when an AI agent encounters errors or ambiguous scenarios and may request human intervention; human operators can rectify errors, supply missing data, or guide the agent, which also informs future agent improvements. Human Feedback for Learning is collected and used to refine AI models, prominently in methodologies like reinforcement learning with human feedback, where human preferences directly influence the agent's learning trajectory. Decision Augmentation is where an AI agent provides analyses and recommendations to a human, who then makes the final decision, enhancing human decision-making through AI-generated insights rather than full autonomy. Human-Agent Collaboration is a cooperative interaction where humans and AI agents contribute their respective strengths; routine data processing may be handled by the agent, while creative problem-solving or complex negotiations are managed by the human. Finally, Escalation Policies are established protocols that dictate when and how an agent should escalate tasks to human operators, preventing errors in situations beyond the agent's capability.

HITL 涵蓋多個關鍵面向：人類監督（Human Oversight）指監控 AI 代理人的表現與輸出（例如透過日誌審查或即時儀表板）以確保遵循指引並避免不良結果。介入與修正（Intervention and Correction）發生在 AI 代理遇到錯誤或模糊情境時，並可能請求人類介入；人類操作員可修正錯誤、補充缺失資料或引導代理，這也有助於未來的代理改進。用於學習的人類回饋（Human Feedback for Learning）會被收集並用來改進 AI 模型，特別是在「人類回饋強化學習」等方法中，人類偏好會直接影響代理的學習軌跡。決策增強（Decision Augmentation）是 AI 代理提供分析與建議，最終由人類做決策，透過 AI 洞見提升人類決策，而非完全自主。人類與代理協作（Human-Agent Collaboration）是一種合作互動，人類與 AI 代理各自貢獻優勢；例行資料處理由代理負責，而創意問題解決或複雜協商由人類處理。最後，升級政策（Escalation Policies）是規範代理在何時以及如何將任務升級給人類操作員的流程，避免在超出代理能力的情況下出錯。

Implementing HITL patterns enables the use of Agents in sensitive sectors where full autonomy is not feasible or permitted. It also provides a mechanism for ongoing improvement through feedback loops. For example, in finance, the final approval of a large corporate loan requires a human loan officer to assess qualitative factors like leadership character. Similarly, in the legal field, core principles of justice and accountability demand that a human judge retain final authority over critical decisions like sentencing, which involve complex moral reasoning.

實作 HITL 模式讓代理人能在不允許或不適合完全自主的敏感領域中使用。它也透過回饋迴圈提供持續改進的機制。例如在金融領域，大型企業貸款的最終核准需要人類貸款專員評估領導特質等質性因素。同樣地，在法律領域，正義與問責的核心原則要求人類法官保有對量刑等關鍵決策的最終權限，因為這涉及複雜的道德推理。

Caveats: Despite its benefits, the HITL pattern has significant caveats, chief among them being a lack of scalability. While human oversight provides high accuracy, operators cannot manage millions of tasks, creating a fundamental trade-off that often requires a hybrid approach combining automation for scale and HITL for accuracy. Furthermore, the effectiveness of this pattern is heavily dependent on the expertise of the human operators; for example, while an AI can generate software code, only a skilled developer can accurately identify subtle errors and provide the correct guidance to fix them. This need for expertise also applies when using HITL to generate training data, as human annotators may require special training to learn how to correct an AI in a way that produces high-quality data. Lastly, implementing HITL raises significant privacy concerns, as sensitive information must often be rigorously anonymized before it can be exposed to a human operator, adding another layer of process complexity.

注意事項： 儘管 HITL 有其優點，但也存在重大限制，其中最主要的是缺乏擴展性。雖然人類監督能提供高準確度，但操作員無法處理數以百萬計的任務，形成根本性的取捨，常需要結合大規模自動化與 HITL 精準性的混合方法。此外，該模式的成效高度依賴人類操作員的專業能力；例如 AI 能生成程式碼，但只有熟練的開發者才能精準找出細微錯誤並提供正確修正指引。這種專業需求也適用於使用 HITL 產生訓練資料時，因為人類標註者可能需要專門訓練，才能以能產出高

品質資料的方式修正 AI。最後，實作 HITL 也帶來顯著的隱私疑慮，因為敏感資訊往往必須嚴格匿名化後才能交由人類操作員查看，增加了流程複雜度。

Practical Applications & Use Cases

實務應用與使用情境

The Human-in-the-Loop pattern is vital across a wide range of industries and applications, particularly where accuracy, safety, ethics, or nuanced understanding are paramount.

人類參與迴圈模式在廣泛的產業與應用中都十分關鍵，尤其是需要高度準確性、安全性、倫理性或細膩理解的場域。

- **Content Moderation:** AI agents can rapidly filter vast amounts of online content for violations (e.g., hate speech, spam). However, ambiguous cases or borderline content are escalated to human moderators for review and final decision, ensuring nuanced judgment and adherence to complex policies.
- **Autonomous Driving:** While self-driving cars handle most driving tasks autonomously, they are designed to hand over control to a human driver in complex, unpredictable, or dangerous situations that the AI cannot confidently navigate (e.g., extreme weather, unusual road conditions).
- **Financial Fraud Detection:** AI systems can flag suspicious transactions based on patterns. However, high-risk or ambiguous alerts are often sent to human analysts who investigate further, contact customers, and make the final determination on whether a transaction is fraudulent.
- **Legal Document Review:** AI can quickly scan and categorize thousands of legal documents to identify relevant clauses or evidence. Human legal professionals then review the AI's findings for accuracy, context, and legal implications, especially for critical cases.
- **Customer Support (Complex Queries):** A chatbot might handle routine customer inquiries. If the user's problem is too complex, emotionally charged, or requires empathy that the AI cannot provide, the conversation is seamlessly handed over to a human support agent.
- **Data Labeling and Annotation:** AI models often require large datasets of labeled data for training. Humans are put in the loop to accurately label

images, text, or audio, providing the ground truth that the AI learns from. This is a continuous process as models evolve.

- **Generative AI Refinement:** When an LLM generates creative content (e.g., marketing copy, design ideas), human editors or designers review and refine the output, ensuring it meets brand guidelines, resonates with the target audience, and maintains quality.
- **Autonomous Networks:** AI systems are capable of analyzing alerts and forecasting network issues and traffic anomalies by leveraging key performance indicators (KPIs) and identified patterns. Nevertheless, crucial decisions—such as addressing high-risk alerts—are frequently escalated to human analysts. These analysts conduct further investigation and make the ultimate determination regarding the approval of network changes.
- **內容審查：** AI 代理可快速篩選大量線上內容以找出違規內容（如仇恨言論、垃圾內容）。但模糊案例或邊界內容會升級給人類審查員做出審閱與最終判定，確保細膩判斷並符合複雜政策。
- **自動駕駛：** 自駕車多數時間可自主處理駕駛任務，但在 AI 無法確定處理的複雜、不可預測或危險情況下（如極端天氣、異常路況），設計上會交回人類駕駛控制。
- **金融詐欺偵測：** AI 系統可依模式標記可疑交易，但高風險或模糊警示通常會交給人類分析師進一步調查、聯繫客戶，並做出是否為詐欺交易的最終判定。
- **法律文件審查：** AI 能快速掃描並分類數千份法律文件以找出相關條款或證據；人類法律專業人員接著審查 AI 發現的內容，確認準確性、脈絡與法律意涵，特別是關鍵案件。
- **客服支援（複雜問題）：** 聊天機器人可處理例行詢問；若使用者問題過於複雜、情緒強烈或需要同理心而 AI 無法提供，對話會無縫轉交給人類客服。
- **資料標註與註解：** AI 模型訓練常需要大量標註資料，人類被納入流程以準確標註影像、文字或音訊，提供 AI 學習所需的真實標準。模型演進時此過程會持續進行。
- **生成式 AI 精修：** 當 LLM 產出創意內容（如行銷文案、設計點子）時，人類編輯或設計師會審閱與修飾輸出，確保符合品牌規範、貼近目標受眾並維持品質。
- **自主網路：** AI 系統能利用關鍵績效指標（KPI）與已識別模式分析警報並預測網路問題與流量異常。不過關鍵決策——例如處理高風險警報——常升級給人類分析師。分析師會進一步調查，並就網路變更核准做出最終判定。

This pattern exemplifies a practical method for AI implementation. It harnesses AI for enhanced scalability and efficiency, while maintaining human oversight to ensure quality, safety, and ethical compliance.

此模式展現了 AI 落地的務實作法：利用 AI 提升擴展性與效率，同時保留人類監督以確保品質、安全與倫理合規。

“Human-on-the-loop” is a variation of this pattern where human experts define the overarching policy, and the AI then handles immediate actions to ensure compliance. Let’s consider two examples:

「Human-on-the-loop」是此模式的一種變體，由人類專家制定整體政策，而 AI 負責即時行動以確保遵循。以下是兩個例子：

- **Automated financial trading system**: In this scenario, a human financial expert sets the overarching investment strategy and rules. For instance, the human might define the policy as: “Maintain a portfolio of 70% tech stocks and 30% bonds, do not invest more than 5% in any single company, and automatically sell any stock that falls 10% below its purchase price.” The AI then monitors the stock market in real-time, executing trades instantly when these predefined conditions are met. The AI is handling the immediate, high-speed actions based on the slower, more strategic policy set by the human operator.
- **Modern call center**: In this setup, a human manager establishes high-level policies for customer interactions. For instance, the manager might set rules such as “any call mentioning ‘service outage’ should be immediately routed to a technical support specialist,” or “if a customer’s tone of voice indicates high frustration, the system should offer to connect them directly to a human agent.” The AI system then handles the initial customer interactions, listening to and interpreting their needs in real-time. It autonomously executes the manager’s policies by instantly routing the calls or offering escalations without needing human intervention for each individual case. This allows the AI to manage the high volume of immediate actions according to the slower, strategic guidance provided by the human operator.
- **自動化金融交易系統**：在此情境中，人類金融專家制定整體投資策略與規則。例如可定義政策為：「投資組合維持 70% 科技股與 30% 債券，不對單一公司投資超過 5%，且任何股票若跌破買入價 10% 則自動賣出。」AI 會即時監控股市，在符合預

設條件時立即執行交易。AI 負責高速的即時行動，依循人類操作員制定的較慢、較策略性的政策。

- **現代客服中心**：在此設定中，人類主管建立高層級客戶互動政策。例如可設定規則：「任何提到『服務中斷』的來電要立即轉接給技術支援專家」，或「若來電者語氣顯示高度挫折，系統應主動提供轉接真人客服。」AI 系統接手初始客戶互動，即時聆聽並解讀需求。它會根據主管政策自動執行，例如立即轉接或提供升級處理，無需每個案例都由人類介入。這讓 AI 能依循人類提供的較慢且策略性的指引，管理大量即時行動。

Hands-On Code Example

實作程式碼範例

To demonstrate the Human-in-the-Loop pattern, an ADK agent can identify scenarios requiring human review and initiate an escalation process . This allows for human intervention in situations where the agent's autonomous decision-making capabilities are limited or when complex judgments are required. This is not an isolated feature; other popular frameworks have adopted similar capabilities. LangChain, for instance, also provides tools to implement these types of interactions.

為了展示人類參與迴圈模式，ADK 代理人可以辨識需要人類審查的情境並啟動升級流程。這讓在代理自主決策能力受限或需要複雜判斷時能介入人類。這並非孤立功能；其他熱門框架也採用類似能力，例如 LangChain 也提供工具來實作這類互動。

```
from typing import Optional

from google.adk.agents import Agent
from google.adk.tools.tool_context import ToolContext
from google.adk.callbacks import CallbackContext
from google.adk.models.llm import LlmRequest
from google.genai import types


# Placeholder for tools (replace with actual implementations if needed)
def troubleshoot_issue(issue: str) -> dict:
    return {"status": "success", "report": f"Troubleshooting steps for
{issue}."}


def create_ticket(issue_type: str, details: str) -> dict:
    return {"status": "success", "ticket_id": "TICKET123"}
```

```
def escalate_to_human(issue_type: str) -> dict:
    # This would typically transfer to a human queue in a real system
    return {"status": "success", "message": f"Escalated {issue_type} to a
human specialist."}


technical_support_agent = Agent(
    name="technical_support_specialist",
    model="gemini-2.0-flash-exp",
    instruction="""
    You are a technical support specialist for our electronics company.
    FIRST, check if the user has a support history in state["customer_info"]
["support_history"].
    If they do, reference this history in your responses.

    For technical issues:
    1. Use the troubleshoot_issue tool to analyze the problem.
    2. Guide the user through basic troubleshooting steps.
    3. If the issue persists, use create_ticket to log the issue.

    For complex issues beyond basic troubleshooting:
    1. Use escalate_to_human to transfer to a human specialist.

    Maintain a professional but empathetic tone. Acknowledge the frustration
technical issues can cause,
    while providing clear steps toward resolution.
    """,
    tools=[troubleshoot_issue, create_ticket, escalate_to_human],
)


def personalization_callback(
    callback_context: CallbackContext, llm_request: LlmRequest
) -> Optional[LlmRequest]:
    """Adds personalization information to the LLM request."""
    # Get customer info from state
    customer_info = callback_context.state.get("customer_info")
    if customer_info:
        customer_name = customer_info.get("name", "valued customer")
        customer_tier = customer_info.get("tier", "standard")
        recent_purchases = customer_info.get("recent_purchases", [])

        personalization_note = (
            f"\nIMPORTANT PERSONALIZATION:\n"
            f"Customer Name: {customer_name}\n"
            f"Customer Tier: {customer_tier}\n"
        )
        if recent_purchases:
            personalization_note += f"Recent Purchases: {',
'.join(recent_purchases)}\n"

        if llm_request.contents:
            # Add as a system message before the first content
            system_content = types.Content(
                role="system",
                parts=[types.Part(text=personalization_note)],
            )
            llm_request.contents.insert(0, system_content)

    return None  # Return None to continue with the modified request
```

This code offers a blueprint for creating a technical support agent using Google's ADK, designed around a HITL framework. The agent acts as an intelligent first line of support, configured with specific instructions and equipped with tools like `troubleshoot_issue`, `create_ticket`, and `escalate_to_human` to manage a complete support workflow. The escalation tool is a core part of the HITL design, ensuring complex or sensitive cases are passed to human specialists.

這段程式碼提供了一個使用 Google ADK 建立技術支援代理人的藍圖，設計圍繞 HITL 架構。該代理人作為智慧的第一線支援，配置了明確指令並配備 `troubleshoot_issue`、`create_ticket` 與 `escalate_to_human` 等工具，以管理完整的支援流程。升級工具是 HITL 設計的核心，確保複雜或敏感案例交由人類專家處理。

A key feature of this architecture is its capacity for deep personalization, achieved through a dedicated callback function. Before contacting the LLM, this function dynamically retrieves customer-specific data—such as their name, tier, and purchase history—from the agent's state. This context is then injected into the prompt as a system message, enabling the agent to provide highly tailored and informed responses that reference the user's history. By combining a structured workflow with essential human oversight and dynamic personalization, this code serves as a practical example of how the ADK facilitates the development of sophisticated and robust AI support solutions.

此架構的一大特色是透過專用 callback 函式實現深度個人化。在呼叫 LLM 之前，該函式會從代理的 state 動態取得客戶資料——如姓名、等級與購買紀錄。這些脈絡會以系統訊息注入提示詞，使代理能提供高度量身訂做且有根據的回應，並引用使用者

的歷史資訊。結合結構化流程、必要的人類監督與動態個人化，這段程式碼成為 ADK 如何促成高階且強健 AI 支援方案的實務範例。

At Glance

一覽

What: AI systems, including advanced LLMs, often struggle with tasks that require nuanced judgment, ethical reasoning, or a deep understanding of complex, ambiguous contexts. Deploying fully autonomous AI in high-stakes environments carries significant risks, as errors can lead to severe safety, financial, or ethical consequences. These systems lack the inherent creativity and common-sense reasoning that humans possess. Consequently, relying solely on automation in critical decision-making processes is often imprudent and can undermine the system's overall effectiveness and trustworthiness.

什麼： AI 系統（包含先進 LLM）常難以處理需要細膩判斷、倫理推理或對複雜模糊脈絡深度理解的任務。在高風險環境部署完全自主的 AI 帶來重大風險，因為錯誤可能導致嚴重的安全、財務或倫理後果。這些系統缺乏人類具備的創造力與常識推理能力。因此，在關鍵決策流程中完全依賴自動化往往並不審慎，且可能削弱系統的整體效能與可信度。

Why: The Human-in-the-Loop (HITL) pattern provides a standardized solution by strategically integrating human oversight into AI workflows. This agentic approach creates a symbiotic partnership where AI handles computational heavy-lifting and data processing, while humans provide critical validation, feedback, and intervention. By doing so, HITL ensures that AI actions align with human values and safety protocols. This collaborative framework not only mitigates the risks of full automation but also enhances the system's capabilities through continuous learning from human input. Ultimately, this leads to more robust, accurate, and ethical outcomes that neither human nor AI could achieve alone.

為什麼： 人類參與迴圈（HITL）模式透過策略性地將人類監督融入 AI 工作流程，提供標準化解決方案。此代理式方法建立共生夥伴關係：AI 處理計算與資料處理的重擔，人類則提供關鍵驗證、回饋與介入。藉此，HITL 確保 AI 行動與人類價值與安全規範一致。此協作框架不僅降低完全自動化的風險，也透過持續的人類輸入學習提升系統能力。最終產生更強健、更準確且更符合倫理的成果，這是單靠人或 AI 都難以達成的。

Rule of Thumb: Use this pattern when deploying AI in domains where errors have significant safety, ethical, or financial consequences, such as in healthcare, finance, or autonomous systems. It is essential for tasks involving ambiguity and nuance that LLMs cannot reliably handle, like content moderation or complex customer support escalations. Employ HITL when the goal is to continuously improve an AI model with high-quality, human-labeled data or to refine generative AI outputs to meet specific quality standards.

經驗法則： 當部署 AI 的領域出錯會造成重大安全、倫理或財務後果（如醫療、金融或自主系統）時，應採用此模式。對於 LLM 無法可靠處理的含糊與細膩任務（如內容審查或複雜客服升級）也至關重要。當目標是用高品質的人類標註資料持續改進 AI 模型，或精修生成式 AI 輸出以符合特定品質標準時，應採用 HITL。

Visual Summary:

視覺摘要：

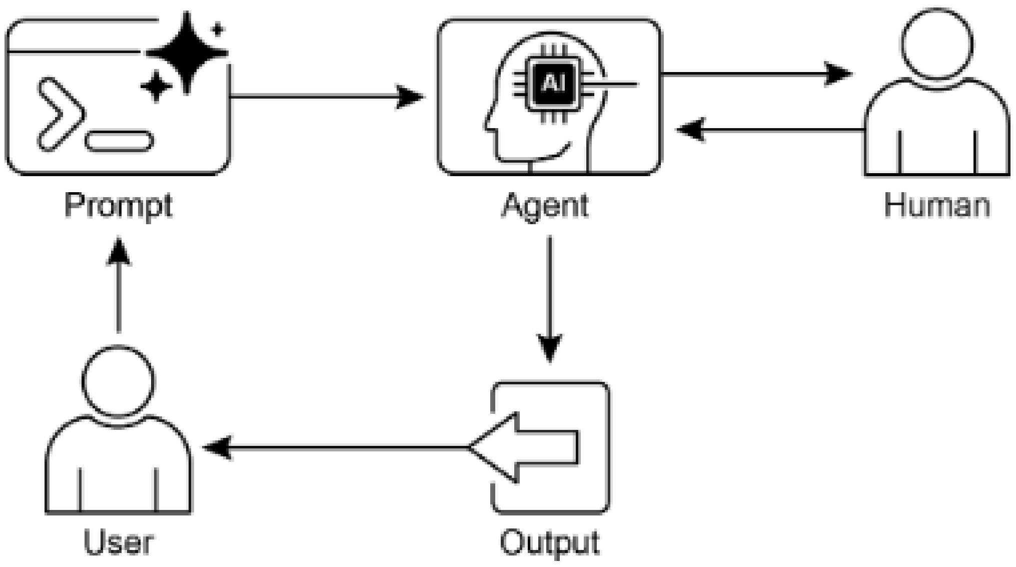

Figure 3: Human in the Loop Design Pattern

Fig.1: Human in the loop design pattern

圖 1：人類參與迴圈設計模式

Key Takeaways

重要重點

Key takeaways include:

重點包含：

- Human-in-the-Loop (HITL) integrates human intelligence and judgment into AI workflows.
- It's crucial for safety, ethics, and effectiveness in complex or high-stakes scenarios.
- Key aspects include human oversight, intervention, feedback for learning, and decision augmentation.
- Escalation policies are essential for agents to know when to hand off to a human.
- HITL allows for responsible AI deployment and continuous improvement.
- The primary drawbacks of Human-in-the-Loop are its inherent lack of scalability, creating a trade-off between accuracy and volume, and its dependence on highly skilled domain experts for effective intervention.
- Its implementation presents operational challenges, including the need to train human operators for data generation and to address privacy concerns by anonymizing sensitive information.
- 人類參與迴圈（HITL）將人類智慧與判斷融入 AI 工作流程。
- 在複雜或高風險情境中，對安全、倫理與效能至關重要。
- 關鍵面向包含人類監督、介入、學習回饋與決策增強。
- 升級政策對代理辨識何時交給人類至關重要。
- HITL 促成負責任的 AI 部署與持續改進。
- HITL 的主要缺點是缺乏擴展性，導致準確度與量之間的取捨，且高度依賴熟練的領域專家介入。
- 實作上也有營運挑戰，包括訓練人類操作員產生資料，以及為隱私而對敏感資訊進行匿名化。

Conclusion

結論

This chapter explored the vital Human-in-the-Loop (HITL) pattern, emphasizing its role in creating robust, safe, and ethical AI systems. We discussed how integrating human oversight, intervention, and feedback into agent workflows can significantly enhance their performance and trustworthiness, especially in complex and sensitive domains. The practical applications demonstrated HITL's widespread utility, from content moderation and medical diagnosis to autonomous driving and customer support. The conceptual code example provided a glimpse into how ADK can facilitate these human-agent interactions through escalation mechanisms. As AI capabilities continue to advance, HITL remains a cornerstone for responsible AI development, ensuring that human values and expertise remain central to intelligent system design.

本章探討了關鍵的人類參與迴圈（HITL）模式，強調其在打造強健、安全且符合倫理的 AI 系統中的角色。我們討論了將人類監督、介入與回饋納入代理工作流程如何顯著提升效能與可信度，尤其在複雜且敏感的領域中。實務應用展示了 HITL 的廣泛效用，從內容審查、醫療診斷到自動駕駛與客服支援。概念性程式碼範例也呈現 ADK 如何透過升級機制促成人類與代理的互動。隨著 AI 能力持續進步，HITL 仍是負責任 AI 發展的基石，確保人類價值與專業知識在智慧系統設計中居於核心。

References

參考資料

1. A Survey of Human-in-the-loop for Machine Learning, Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, Liang He, https://arxiv.org/abs/2108.00941
2. A Survey of Human-in-the-loop for Machine Learning，Xingjiao Wu，Luwei Xiao，Yixuan Sun，Junhang Zhang，Tianlong Ma，Liang He，https://arxiv.org/abs/2108.00941

Chapter 14: Knowledge Retrieval (RAG)

第 14 章：知識檢索（RAG）

LLMs exhibit substantial capabilities in generating human-like text. However, their knowledge base is typically confined to the data on which they were trained, limiting their access to real-time information, specific company data, or highly specialized details. Knowledge Retrieval (RAG, or Retrieval Augmented Generation), addresses this limitation. RAG enables LLMs to access and integrate external, current, and context-specific information, thereby enhancing the accuracy, relevance, and factual basis of their outputs.

LLM 在生成類人文字方面具備強大能力。然而，它們的知識庫通常受限於訓練資料，無法取得即時資訊、特定公司資料或高度專業的細節。知識檢索（RAG，Retrieval

Augmented Generation）正是為了解決此限制。RAG 讓 LLM 能存取並整合外部、最新且具脈絡的資訊，提升輸出的正確性、相關性與事實依據。

For AI agents, this is crucial as it allows them to ground their actions and responses in real-time, verifiable data beyond their static training. This capability enables them to perform complex tasks accurately, such as accessing the latest company policies to answer a specific question or checking current inventory before placing an order. By integrating external knowledge, RAG transforms agents from simple conversationalists into effective, data-driven tools capable of executing meaningful work.

對 AI 代理人而言，這點至關重要，因為它讓代理能以超越靜態訓練資料的即時可驗證數據來落實行動與回應。這使代理能準確完成複雜任務，例如存取最新公司政策來回答問題，或在下單前檢查現有庫存。透過整合外部知識，RAG 將代理人從單純對話者轉變為有效的資料驅動工具，能執行有意義的工作。

Knowledge Retrieval (RAG) Pattern Overview

知識檢索（RAG）模式概觀

The Knowledge Retrieval (RAG) pattern significantly enhances the capabilities of LLMs by granting them access to external knowledge bases before generating a response. Instead of relying solely on their internal, pre-trained knowledge, RAG allows LLMs to “look up” information, much like a human might consult a book or search the internet. This process empowers LLMs to provide more accurate, up-to-date, and verifiable answers.

知識檢索（RAG）模式在生成回應前賦予 LLM 存取外部知識庫的能力，大幅提升其表現。RAG 不再只依賴內部的預訓練知識，而是讓 LLM 能像人類查書或上網搜尋般「查詢」資訊。這個流程使 LLM 能提供更準確、最新且可驗證的答案。

When a user poses a question or gives a prompt to an AI system using RAG, the query isn’t sent directly to the LLM. Instead, the system first scours a vast external knowledge base—a highly organized library of documents, databases, or web pages—for relevant information. This search is not a simple keyword match; it’s a “semantic search” that understands the user’s intent and the meaning behind their words. This initial search pulls out the most pertinent snippets or “chunks” of information. These extracted pieces are then “augmented,” or added, to the original prompt, creating a richer, more informed query. Finally, this enhanced prompt is sent to the LLM. With this additional context, the LLM can generate a response that is not only fluent and natural but also factually grounded in the retrieved data.

當使用者向採用 RAG 的 AI 系統提出問題或提示時，查詢不會直接送入 LLM。系統會先在龐大的外部知識庫——一個高度組織化的文件、資料庫或網頁庫——中尋找相關資訊。這種搜尋不是簡單的關鍵字匹配，而是能理解使用者意圖與語意的「語意搜尋」。初步搜尋會擷取最相關的片段或「區塊」資訊。這些擷取到的內容會被「增強」並加入原始提示，形成更豐富、更有資訊的查詢。最後，這個增強後的提示送入 LLM。有了額外脈絡，LLM 的回應不僅流暢自然，也能以檢索到的資料為事實基礎。

The RAG framework provides several significant benefits. It allows LLMs to access up-to-date information, thereby overcoming the constraints of their static training data. This approach also reduces the risk of "hallucination"—the generation of false information—by grounding responses in verifiable data. Moreover, LLMs can utilize specialized knowledge found in internal company documents or wikis. A vital advantage of this process is the capability to offer "citations," which pinpoint the exact source of information, thereby enhancing the trustworthiness and verifiability of the AI's responses..

RAG 架構帶來多項重要效益。它讓 LLM 能取得最新資訊，突破靜態訓練資料的限制。此方法也藉由以可驗證資料為依據，降低「幻覺」——產生不實資訊——的風險。此外，LLM 也能運用企業內部文件或維基中的專門知識。此流程的一大優勢是能提供「引用」，指出確切資訊來源，從而提高 AI 回應的可信度與可驗證性。

To fully appreciate how RAG functions, it's essential to understand a few core concepts (see Fig.1):

要完整理解 RAG 的運作方式，必須先掌握幾個核心概念（見圖 1）：

Embeddings

向量嵌入

In the context of LLMs, embeddings are numerical representations of text, such as words, phrases, or entire documents. These representations are in the form of a vector, which is a list of numbers. The key idea is to capture the semantic meaning and the relationships between different pieces of text in a mathematical space. Words or phrases with similar meanings will have embeddings that are closer to each other in this vector space. For instance, imagine a simple 2D graph. The word "cat" might be represented by the coordinates (2, 3), while "kitten" would be very close at (2.1, 3.1). In contrast, the word “car” would have a distant coordinate like (8, 1), reflecting its different meaning. In reality, these embeddings are in a much higher-dimensional space with hundreds or even thousands of dimensions, allowing for a very nuanced understanding of language.

在 LLM 的語境中，向量嵌入（embeddings）是文本（如單字、片語或整份文件）的數值表示。這些表示以向量形式呈現，也就是數字列表。核心概念是將語意與不同文本之間的關係映射到數學空間中。語意相近的詞或片語在向量空間中會彼此更接近。例如在簡單的 2D 圖中，「cat」可能座標為 (2, 3)，「kitten」則非常接近 (2.1, 3.1)。相對地，「car」可能座標為 (8, 1)，反映其不同語意。實際上，嵌入位於更高維度的空間，可能有數百甚至數千維，能更細緻地理解語言。

Text Similarity

文本相似度

Text similarity refers to the measure of how alike two pieces of text are. This can be at a surface level, looking at the overlap of words (lexical similarity), or at a deeper, meaning-based level. In the context of RAG, text similarity is crucial for finding the most relevant information in the knowledge base that corresponds to a user’s query. For instance, consider the sentences: “What is the capital of France?” and “Which city is the capital of France?”. While the wording is different, they are asking the same question. A good text similarity model would recognize this and assign a high similarity score to these two sentences, even though they only share a few words. This is often calculated using the embeddings of the texts.

文本相似度指衡量兩段文本相似程度的指標。它可以是表層的詞彙重疊（詞彙相似度），也可以是更深層的語意相似。在 RAG 中，文本相似度對於找出知識庫中與使用者查詢最相關的資訊至關重要。例如句子「What is the capital of France?」與「Which city is the capital of France?」雖然用詞不同，但問的是同一問題。好的文本相似度模型會識別這點，並給予高相似度分數，即使它們只共享少量詞彙。這通常透過文本的嵌入來計算。

Semantic Similarity and Distance

語意相似度與距離

Semantic similarity is a more advanced form of text similarity that focuses purely on the meaning and context of the text, rather than just the words used. It aims to understand if two pieces of text convey the same concept or idea.

Semantic distance is the inverse of this; a high semantic similarity implies a low semantic distance, and vice versa. In RAG, semantic search relies on finding documents with the smallest semantic distance to the user’s query. For instance, the phrases “a furry feline companion” and “a domestic cat” have no words in common besides “a”. However, a model that understands semantic similarity would recognize that they refer to the same thing and would consider them to be highly similar. This is because their embeddings would be very close in the vector space, indicating a small semantic distance. This is the “smart search” that allows RAG to find relevant information even when the user’s wording doesn’t exactly match the text in the knowledge base.

語意相似度是更進階的文本相似形式，著重於文本的意義與脈絡，而非只看用詞。它試圖理解兩段文本是否表達相同概念或想法。語意距離是其反向概念；語意相似度高代表語意距離低，反之亦然。在 RAG 中，語意搜尋依賴找到與使用者查詢語意距離最小的文件。例如「a furry feline companion」與「a domestic cat」除了「a」以外沒有共同詞彙，但理解語意的模型會辨識它們指的是同一件事，因此視為高度相似。這是因為它們的嵌入在向量空間中非常接近，代表語意距離很小。這種「智慧搜尋」讓 RAG 即使用戶用詞不同，也能找到相關資訊。

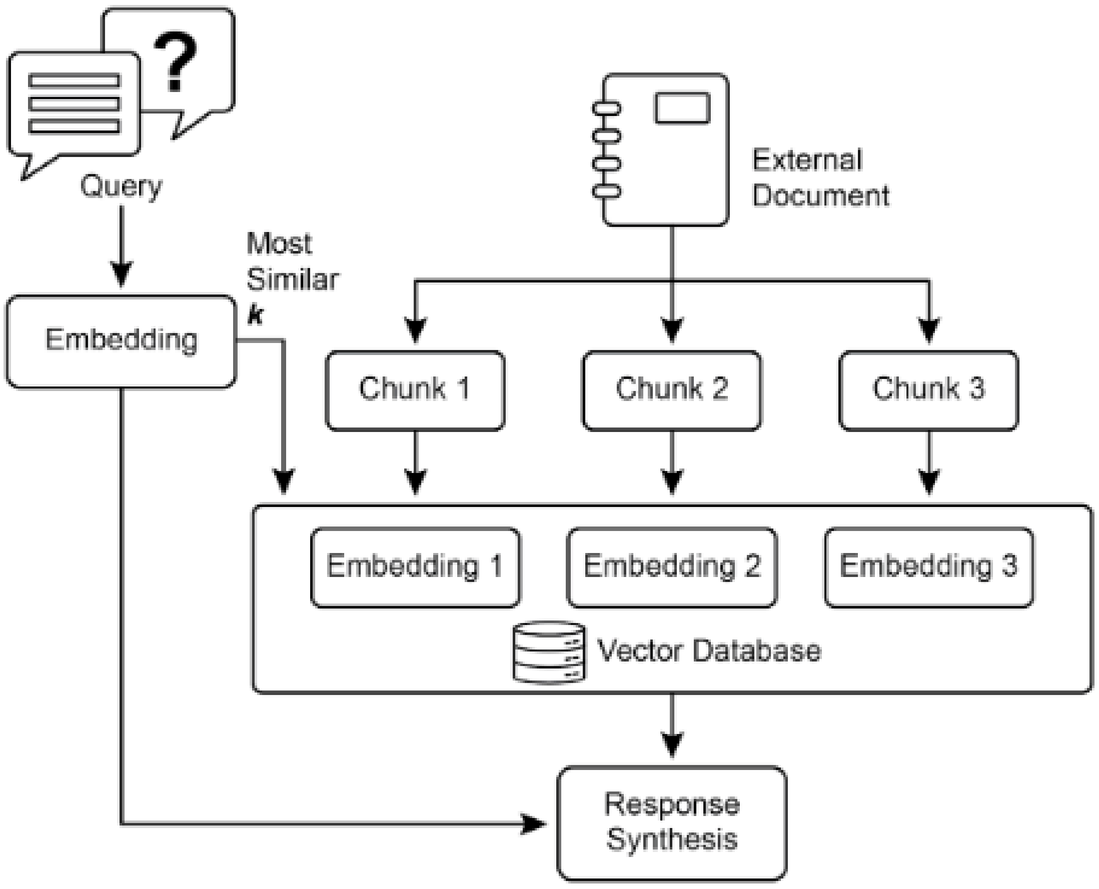

Figure 4: RAG Core Concept: Chunking, Embeddings, and Vector Database

Fig.1: RAG Core Concepts: Chunking, Embeddings, and Vector Database

圖 1：RAG 核心概念：切分、嵌入與向量資料庫

Chunking of Documents

文件切分

Chunking is the process of breaking down large documents into smaller, more manageable pieces, or “chunks.” For a RAG system to work efficiently, it cannot feed entire large documents into the LLM. Instead, it processes these smaller chunks. The way documents are chunked is important for preserving the context and meaning of the information. For instance, instead of treating a 50-page user manual as a single block of text, a chunking strategy might break it down into sections, paragraphs, or even sentences. For instance, a section on “Troubleshooting” would be a separate chunk from the “Installation Guide.” When a user asks a question about a specific problem, the RAG system can then retrieve the most relevant troubleshooting chunk, rather than the entire manual. This makes the retrieval process faster and the information provided to the LLM more focused and relevant to the user's immediate need. Once documents are chunked, the RAG system must employ a retrieval technique to find the most relevant pieces for a given query. The primary method is vector search, which uses embeddings and semantic distance to find chunks that are conceptually similar to the user's question. An older, but still valuable, technique is BM25, a keyword-based algorithm that ranks chunks based on term frequency without understanding semantic meaning. To get the best of both worlds, hybrid search approaches are often used, combining the keyword precision of BM25 with the contextual understanding of semantic search. This fusion allows for more robust and accurate retrieval, capturing both literal matches and conceptual relevance.

切分（Chunking）是將大型文件拆分成較小、可管理的片段或「區塊」的過程。為了讓 RAG 系統有效運作，它不能把整份大型文件都送入 LLM，而是處理較小的區塊。文件切分方式對於保存資訊脈絡與意義很重要。例如，將 50 頁的使用手冊分成章節、段落甚至句子，而不是視為單一文字塊。「故障排除」章節就會與「安裝指南」成為不同區塊。當使用者詢問特定問題時，RAG 系統就能取回最相關的故障排除區塊，而非整份手冊。這讓檢索更快、提供給 LLM 的資訊更聚焦且符合使用者即時需求。文件切分後，RAG 系統必須使用檢索技術找出與查詢最相關的片段。主要方法是向量搜尋，利用嵌入與語意距離找到概念上相似的區塊。較舊但仍有價值的方法是 BM25，一種基於關鍵字的演算法，依詞頻排序區塊但不理解語意。為了兼顧兩者，常使用混合搜尋，結合 BM25 的關鍵字精準度與語意搜尋的脈絡理解。這種融合能更穩健且準確地檢索，同時捕捉字面匹配與概念相關性。

Vector Databases

向量資料庫

A vector database is a specialized type of database designed to store and query embeddings efficiently. After documents are chunked and converted into embeddings, these high-dimensional vectors are stored in a vector database. Traditional retrieval techniques, like keyword-based search, are excellent at finding documents containing exact words from a query but lack a deep understanding of language. They wouldn't recognize that "furry feline companion" means "cat." This is where vector databases excel. They are built specifically for semantic search. By storing text as numerical vectors, they can find results based on conceptual meaning, not just keyword overlap. When a user’s query is also converted into a vector, the database uses highly optimized algorithms (like HNSW – Hierarchical Navigable Small World) to rapidly search through millions of vectors and find the ones that are “closest” in meaning. This approach is far superior for RAG because it uncovers relevant context even if the user’s phrasing is completely different from the source documents. In essence, while other techniques search for words, vector databases search for meaning. This technology is implemented in various forms, from managed databases like Pinecone and Weaviate to open-source solutions such as Chroma DB, Milvus, and Qdrant. Even existing databases can be augmented with vector search capabilities, as seen with Redis, Elasticsearch, and Postgres (using the pgvector extension). The core retrieval mechanisms are often powered by libraries like Meta AI’s FAISS or Google Research’s ScaNN, which are fundamental to the efficiency of these systems.

向量資料庫是一種專門用來高效儲存與查詢嵌入的資料庫。文件切分並轉為嵌入後，這些高維向量會存入向量資料庫。傳統檢索技術（如關鍵字搜尋）擅長找到包含查詢字詞的文件，但缺乏對語言的深入理解，無法辨識「furry feline companion」其實指「cat」。向量資料庫的強項就在於此。它們專為語意搜尋而設計，透過將文字存為數值向量，能根據概念意義而非僅是關鍵字重疊來找結果。當使用者查詢也轉成向量後，資料庫會使用高度最佳化的演算法（如 HNSW – Hierarchical Navigable Small World）在數百萬向量中快速搜尋並找出語意最接近者。這對 RAG 特別優勢，因為即使使用者措辭與文件完全不同，也能找出相關脈絡。本質上，其他技術在找「字」，向量資料庫在找「意義」。此技術有多種實作，從 Pinecone、Weaviate 等託管資料庫，到 Chroma DB、Milvus、Qdrant 等開源方案。既有資料庫也可加上向量搜尋能力，例如 Redis、Elasticsearch 與 Postgres（使用 pgvector 擴充）。核心檢索機制常由 Meta AI 的 FAISS 或 Google Research 的 ScaNN 等函式庫提供，是系統效率的基礎。

RAG’s Challenges

RAG 的挑戰

Despite its power, the RAG pattern is not without its challenges. A primary issue arises when the information needed to answer a query is not confined to a single chunk but is spread across multiple parts of a document or even several documents. In such cases, the retriever might fail to gather all the necessary context, leading to an incomplete or inaccurate answer. The system’s effectiveness is also highly dependent on the quality of the chunking and retrieval process; if irrelevant chunks are retrieved, it can introduce noise and confuse the LLM. Furthermore, effectively synthesizing information from potentially contradictory sources remains a significant hurdle for these systems. Besides that, another challenge is that RAG requires the entire knowledge base to be pre-processed and stored in specialized databases, such as vector or graph databases, which is a considerable undertaking. Consequently, this knowledge requires periodic reconciliation to remain up-to-date, a crucial task when dealing with evolving sources like company wikis. This entire process can have a noticeable impact on performance, increasing latency, operational costs, and the number of tokens used in the final prompt.

儘管強大，RAG 模式仍有其挑戰。主要問題在於需要回答查詢的資訊可能分散於文件的多個區塊甚至多份文件，導致檢索器無法收集所有必要脈絡，進而產生不完整或不準確的答案。系統效能高度依賴切分與檢索品質；若取回不相關區塊，會引入雜訊並混淆 LLM。此外，如何有效整合可能彼此矛盾的來源資訊仍是一大難題。另一方面，RAG 需要對整個知識庫進行前處理並存入向量或圖資料庫等專用資料庫，這是一項龐大的工程。因此這些知識需要定期校準以保持最新，尤其在面對企業維基等持續更新的來源時。整體流程會對效能造成影響，增加延遲、營運成本與最終提示中的 token 使用量。

In summary, the Retrieval-Augmented Generation (RAG) pattern represents a significant leap forward in making AI more knowledgeable and reliable. By seamlessly integrating an external knowledge retrieval step into the generation process, RAG addresses some of the core limitations of standalone LLMs. The foundational concepts of embeddings and semantic similarity, combined with retrieval techniques like keyword and hybrid search, allow the system to intelligently find relevant information, which is made manageable through strategic chunking. This entire retrieval process is powered by specialized vector databases designed to store and efficiently query millions of embeddings at scale. While challenges in retrieving fragmented or contradictory information persist, RAG empowers LLMs to produce answers that are not only contextually appropriate but also anchored in verifiable facts, fostering greater trust and utility in AI.

總結來說，檢索增強生成（RAG）模式在提升 AI 知識性與可靠性方面是一大躍進。透過在生成流程中無縫整合外部知識檢索步驟，RAG 解決了獨立 LLM 的核心限制之一。嵌入與語意相似度等基礎概念，搭配關鍵字與混合搜尋等檢索技術，使系統能智

慧地找出相關資訊，並透過策略性切分加以管理。整個檢索流程由專門向量資料庫支援，能在大規模下儲存並高效查詢數百萬嵌入。儘管取得分散或矛盾資訊的挑戰仍在，RAG 仍讓 LLM 能產出既符合脈絡又以可驗證事實為基礎的答案，增進 AI 的可信度與實用性。

Graph RAG

Graph RAG
GraphRAG is an advanced form of Retrieval-Augmented Generation that utilizes a knowledge graph instead of a simple vector database for information retrieval. It answers complex queries by navigating the explicit relationships (edges) between data entities (nodes) within this structured knowledge base. A key advantage is its ability to synthesize answers from information fragmented across multiple documents, a common failing of traditional RAG. By understanding these connections, GraphRAG provides more contextually accurate and nuanced responses.

GraphRAG 是檢索增強生成的一種進階形式，使用知識圖譜而非單純向量資料庫來進行資訊檢索。它透過瀏覽結構化知識庫中資料實體（節點）之間的明確關係（邊）來回答複雜查詢。其重要優勢是能將分散於多份文件的資訊整合成答案，這是傳統 RAG 的常見弱點。透過理解這些連結，GraphRAG 能提供更符合脈絡且更細緻的回應。

Use cases include complex financial analysis, connecting companies to market events, and scientific research for discovering relationships between genes and diseases. The primary drawback, however, is the significant complexity, cost, and expertise required to build and maintain a high-quality knowledge graph. This setup is also less flexible and can introduce higher latency compared to simpler vector search systems. The system's effectiveness is entirely dependent on the quality and completeness of the underlying graph structure. Consequently, GraphRAG offers superior contextual reasoning for intricate questions but at a much higher implementation and maintenance cost. In summary, it excels where deep, interconnected insights are more critical than the speed and simplicity of standard RAG.

應用包含複雜的財務分析、將公司與市場事件連結、以及用於發現基因與疾病關係的科學研究。然而其主要缺點是建置與維護高品質知識圖譜所需的高複雜度、成本與專業知識。與較簡單的向量搜尋系統相比，這種設定也較不具彈性，並可能帶來更高延遲。系統成效完全取決於底層圖結構的品質與完整度。因此 GraphRAG 為複雜問題提

供優異的脈絡推理能力，但實作與維護成本也更高。總結來說，它適合在深度連結洞見比標準 RAG 的速度與簡單性更重要的情境。

Agentic RAG

代理式 RAG

An evolution of this pattern, known as **Agentic RAG** (see Fig.2), introduces a reasoning and decision-making layer to significantly enhance the reliability of information extraction. Instead of just retrieving and augmenting, an “agent”—a specialized AI component—acts as a critical gatekeeper and refiner of knowledge. Rather than passively accepting the initially retrieved data, this agent actively interrogates its quality, relevance, and completeness, as illustrated by the following scenarios.

此模式的一種演進形式稱為 **代理式 RAG**（見圖 2），它加入推理與決策層，顯著提升資訊擷取的可靠性。不再只是檢索與增強，而是由一個「代理」——專門的 AI 元件——擔任關鍵把關者與知識精煉者。代理不會被動接受初步檢索結果，而是主動檢視其品質、相關性與完整性，如下列情境所示。

First, an agent excels at reflection and source validation. If a user asks, “What is our company’s policy on remote work?” a standard RAG might pull up a 2020 blog post alongside the official 2025 policy document. The agent, however, would analyze the documents’ metadata, recognize the 2025 policy as the most current and authoritative source, and discard the outdated blog post before sending the correct context to the LLM for a precise answer.

首先，代理擅長反思與來源驗證。若使用者問「我們公司的遠端工作政策是什麼？」，標準 RAG 可能同時取回 2020 年的部落格文章與 2025 年正式政策文件。代理則會分析文件中繼資料，辨識 2025 年政策為最新且最權威的來源，並在送入 LLM 之前捨棄過時文章，提供正確脈絡以取得精確答案。

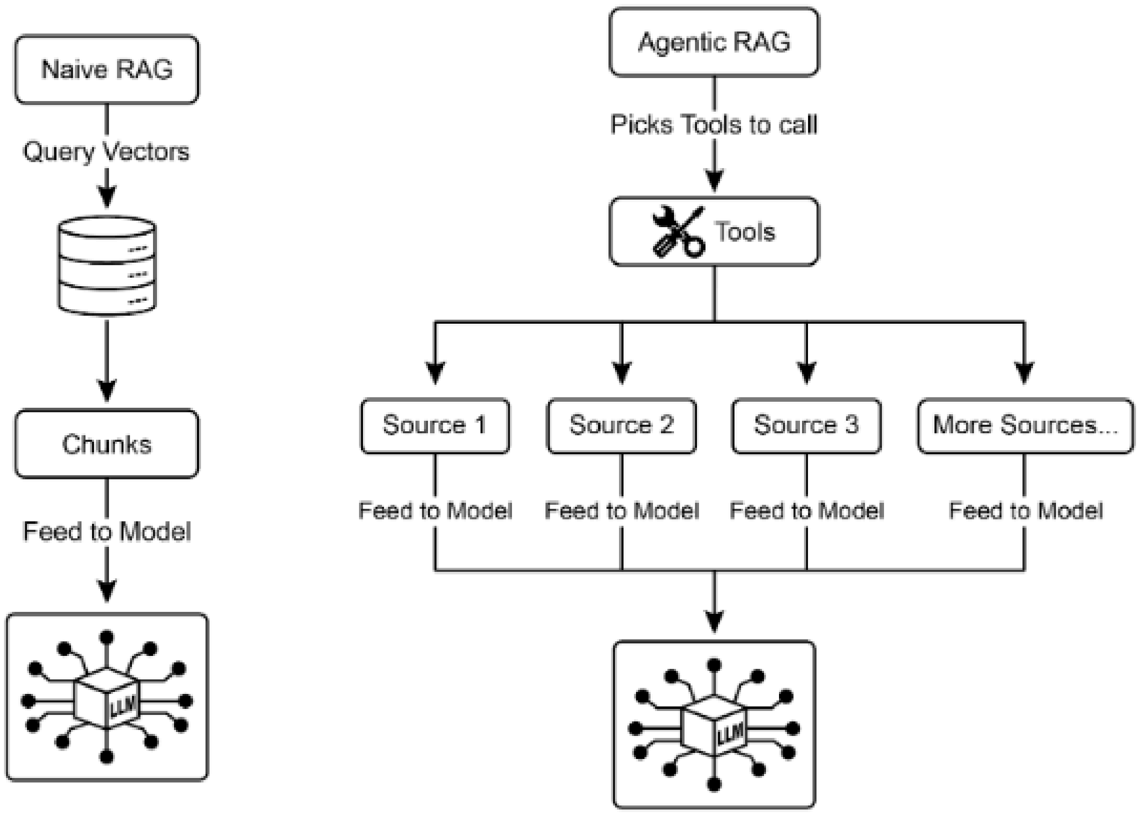

Figure 5: Agentic RAG Introduces Reasoning Agent

Fig.2: Agentic RAG introduces a reasoning agent that actively evaluates, reconciles, and refines retrieved information to ensure a more accurate and trustworthy final response.

圖 2：代理式 RAG 引入推理代理，主動評估、調和與精煉檢索資訊，以確保更準確可信的最終回應。

Second, an agent is adept at reconciling knowledge conflicts. Imagine a financial analyst asks, “What was Project Alpha’s Q1 budget?” The system retrieves two documents: an initial proposal stating a €50,000 budget and a finalized financial report listing it as €65,000. An Agentic RAG would identify this contradiction, prioritize the financial report as the more reliable source, and provide the LLM with the verified figure, ensuring the final answer is based on the most accurate data.

第二，代理擅長調和知識衝突。想像一位財務分析師詢問：「Project Alpha 的第一季預算是多少？」系統檢索到兩份文件：初始提案為 50,000 歐元，而最終財務報告為 65,000 歐元。代理式 RAG 會辨識此矛盾，優先採用較可靠的財務報告，並將經驗證的數字提供給 LLM，確保最終答案以最正確資料為準。

Third, an agent can perform multi-step reasoning to synthesize complex answers. If a user asks, “How do our product’s features and pricing compare to Competitor X’s?” the agent would decompose this into separate sub-queries. It would initiate distinct searches for its own product’s features, its pricing, Competitor X’s features, and Competitor X’s pricing. After gathering these individual pieces of information, the agent would synthesize them into a structured, comparative context before feeding it to the LLM, enabling a comprehensive response that a simple retrieval could not have produced.

第三，代理能進行多步推理以整合複雜答案。若使用者問：「我們的產品功能與定價與競爭對手 X 相比如何？」代理會將問題拆成子查詢，分別搜尋自家產品功能、自家定價、競品功能與競品定價。收集這些資訊後，代理會合成結構化的比較脈絡再送給 LLM，使其能產出單純檢索無法達到的完整回應。

Fourth, an agent can identify knowledge gaps and use external tools. Suppose a user asks, “What was the market’s immediate reaction to our new product launched yesterday?” The agent searches the internal knowledge base, which is updated weekly, and finds no relevant information. Recognizing this gap, it can then activate a tool—such as a live web-search API—to find recent news articles and social media sentiment. The agent then uses this freshly gathered external information to provide an up-to-the-minute answer, overcoming the limitations of its static internal database.

第四，代理能識別知識缺口並使用外部工具。假設使用者問：「市場對我們昨天推出的新產品的即時反應是什麼？」代理搜尋每週更新的內部知識庫後發現沒有相關資訊。識別到缺口後，它可啟動工具，例如即時網路搜尋 API，來查找最新新聞與社群媒體情緒。代理接著使用這些新取得的外部資訊提供最新答案，克服內部靜態資料庫的限制。

Challenges of Agentic RAG

代理式 RAG 的挑戰

While powerful, the agentic layer introduces its own set of challenges. The primary drawback is a significant increase in complexity and cost. Designing, implementing, and maintaining the agent’s decision-making logic and tool integrations requires substantial engineering effort and adds to computational expenses. This complexity can also lead to increased latency, as the agent’s cycles of reflection, tool use, and multi-step reasoning take more time than a standard, direct retrieval process. Furthermore, the agent itself can become a new source of error; a flawed reasoning process could cause it to get stuck in useless loops, misinterpret a task, or improperly discard relevant information, ultimately degrading the quality of the final response.

儘管強大，代理層也帶來一系列挑戰。主要缺點是顯著增加複雜度與成本。設計、實作與維護代理的決策邏輯與工具整合需要大量工程投入，並增加計算成本。此複雜度也可能提高延遲，因為代理的反思、工具使用與多步推理循環比標準的直接檢索更耗時。此外，代理本身可能成為新的錯誤來源；推理流程若有缺陷，可能陷入無用迴圈、誤解任務或錯誤丟棄相關資訊，最終降低回應品質。

In Summary

小結

Agentic RAG represents a sophisticated evolution of the standard retrieval pattern, transforming it from a passive data pipeline into an active, problem-solving framework. By embedding a reasoning layer that can evaluate sources, reconcile conflicts, decompose complex questions, and use external tools, agents dramatically improve the reliability and depth of the generated answers. This advancement makes the AI more trustworthy and capable, though it comes with important trade-offs in system complexity, latency, and cost that must be carefully managed.

代理式 RAG 是標準檢索模式的精緻演進，將其從被動資料管線轉為主動解題框架。透過嵌入能評估來源、調和衝突、拆解複雜問題並使用外部工具的推理層，代理能大幅提升答案的可靠性與深度。此進展使 AI 更可信且更有能力，但也帶來系統複雜度、延遲與成本上的重要取捨，需要謹慎管理。

Practical Applications & Use Cases

實務應用與使用情境

Knowledge Retrieval (RAG) is changing how Large Language Models (LLMs) are utilized across various industries, enhancing their ability to provide more accurate and contextually relevant responses.

知識檢索（RAG）正在改變大型語言模型（LLM）在各行各業的使用方式，提升其提供更準確且更符合脈絡回應的能力。

Applications include:

應用包含：

- **Enterprise Search and Q&A:** Organizations can develop internal chatbots that respond to employee inquiries using internal documentation such as HR policies, technical manuals, and product specifications. The RAG system extracts relevant sections from these documents to inform the LLM's response.

- **Customer Support and Helpdesks:** RAG-based systems can offer precise and consistent responses to customer queries by accessing information from product manuals, frequently asked questions (FAQs), and support tickets. This can reduce the need for direct human intervention for routine issues.
- **Personalized Content Recommendation:** Instead of basic keyword matching, RAG can identify and retrieve content (articles, products) that is semantically related to a user's preferences or previous interactions, leading to more relevant recommendations.
- **News and Current Events Summarization:** LLMs can be integrated with real-time news feeds. When prompted about a current event, the RAG system retrieves recent articles, allowing the LLM to produce an up-to-date summary.
- **企業內部搜尋與 Q&A：** 組織可打造內部聊天機器人，使用人資政策、技術手冊與產品規格等內部文件回應員工提問。RAG 系統會從這些文件中擷取相關段落以支援 LLM 回應。
- **客服支援與服務台：** RAG 系統可透過存取產品手冊、常見問題（FAQ）與支援票證，提供精準且一致的回覆，減少例行問題對人類介入的需求。
- **個人化內容推薦：** 相較於基本關鍵字匹配，RAG 能找出與使用者偏好或過往互動在語意上相關的內容（文章、產品），帶來更貼近的推薦。
- **新聞與即時事件摘要：** LLM 可與即時新聞資料流整合，當使用者詢問當前事件時，RAG 會取回最新文章，讓 LLM 產出最新摘要。

By incorporating external knowledge, RAG extends the capabilities of LLMs beyond simple communication to function as knowledge processing systems.

透過納入外部知識，RAG 讓 LLM 的能力超越單純溝通，成為知識處理系統。

Hands-On Code Example (ADK)

實作程式碼範例（ADK）

To illustrate the Knowledge Retrieval (RAG) pattern, let's see three examples.

為了說明知識檢索（RAG）模式，以下提供三個範例。

First, is how to use Google Search to do RAG and ground LLMs to search results. Since RAG involves accessing external information, the Google Search tool is a direct example of a built-in retrieval mechanism that can augment an LLM's knowledge.

第一個示範如何使用 Google Search 進行 RAG，並將 LLM 扎根於搜尋結果。由於 RAG 需要存取外部資訊，Google Search 工具就是一個可增強 LLM 知識的內建檢索機制。

```
from google.adk.tools import google_search
from google.adk.agents import Agent


search_agent = Agent(
    name="research_assistant",
    model="gemini-2.0-flash-exp",
    instruction="You help users research topics. When asked, use the Google
Search tool",
    tools=[google_search],
)
```

Second, this section explains how to utilize Vertex AI RAG capabilities within the Google ADK. The code provided demonstrates the initialization of VertexAiRagMemoryService from the ADK. This allows for establishing a connection to a Google Cloud Vertex AI RAG Corpus. The service is configured by specifying the corpus resource name and optional parameters such as `SIMILARITY_TOP_K` and `VECTOR_DISTANCE_THRESHOLD`. These parameters influence the retrieval process. `SIMILARITY_TOP_K` defines the number of top similar results to be retrieved. `VECTOR_DISTANCE_THRESHOLD` sets a limit on the semantic distance for the retrieved results. This setup enables agents to perform scalable and persistent semantic knowledge retrieval from the designated RAG Corpus. The process effectively integrates Google Cloud's RAG functionalities into an ADK agent, thereby supporting the development of responses grounded in factual data.

第二個範例說明如何在 Google ADK 中使用 Vertex AI RAG 功能。提供的程式碼示範如何初始化 ADK 的 VertexAiRagMemoryService，建立與 Google Cloud Vertex AI RAG Corpus 的連線。此服務透過指定 corpus 資源名稱與可選參數（如 `SIMILARITY_TOP_K` 與 `VECTOR_DISTANCE_THRESHOLD`）進行設定，這些參數會影響檢索流程。`SIMILARITY_TOP_K` 定義要取回的相似結果數量；`VECTOR_DISTANCE_THRESHOLD` 則設定檢索結果允許的最大語意距離。此設定讓代理能從指定的 RAG Corpus 進行可擴展

且持久的語意知識檢索，將 Google Cloud 的 RAG 功能有效整合到 ADK 代理中，支援產生以事實資料為基礎的回應。

```python
# Import the necessary VertexAiRagMemoryService class from the
google.adk.memory module.
from google.adk.memory import VertexAiRagMemoryService


RAG_CORPUS_RESOURCE_NAME = "projects/your-gcp-project-id/locations/us-
central1/ragCorpora/your-corpus-id"

# Define an optional parameter for the number of top similar results to
retrieve.
# This controls how many relevant document chunks the RAG service will return.
SIMILARITY_TOP_K = 5

# Define an optional parameter for the vector distance threshold.
# This threshold determines the maximum semantic distance allowed for
retrieved results;
# results with a distance greater than this value might be filtered out.
VECTOR_DISTANCE_THRESHOLD = 0.7

# Initialize an instance of VertexAiRagMemoryService.
# This sets up the connection to your Vertex AI RAG Corpus.
# - rag_corpus: Specifies the unique identifier for your RAG Corpus.
# - similarity_top_k: Sets the maximum number of similar results to fetch.
# - vector_distance_threshold: Defines the similarity threshold for filtering
results.
memory_service = VertexAiRagMemoryService(
    rag_corpus=RAG_CORPUS_RESOURCE_NAME,
    similarity_top_k=SIMILARITY_TOP_K,
    vector_distance_threshold=VECTOR_DISTANCE_THRESHOLD,
)
```

Hands-On Code Example (LangChain)

實作程式碼範例（LangChain）

Third, let's walk through a complete example using LangChain.

第三，讓我們完整走一次 LangChain 的範例。

```python
import os
import requests
from typing import List, Dict, Any, TypedDict

from langchain_community.document_loaders import TextLoader
from langchain_core.documents import Document
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores import Weaviate
from langchain_openai import ChatOpenAI
from langchain.text_splitter import CharacterTextSplitter
from langchain.schema.runnable import RunnablePassthrough
from langgraph.graph import StateGraph, END

import weaviate
from weaviate.embedded import EmbeddedOptions
import dotenv


# Load environment variables (e.g., OPENAI_API_KEY)
dotenv.load_dotenv()

# Set your OpenAI API key (ensure it's loaded from .env or set here)
# os.environ["OPENAI_API_KEY"] = "YOUR_OPENAI_API_KEY"


# --- 1. Data Preparation (Preprocessing) ---

# Load data
url = "https://github.com/langchain-ai/langchain/blob/master/docs/docs/how_to/
state_of_the_union.txt"
res = requests.get(url)
with open("state_of_the_union.txt", "w") as f:
    f.write(res.text)

loader = TextLoader("./state_of_the_union.txt")
documents = loader.load()

# Chunk documents
text_splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=50)
chunks = text_splitter.split_documents(documents)

# Embed and store chunks in Weaviate
client = weaviate.Client(embedded_options=EmbeddedOptions())

vectorstore = Weaviate.from_documents(
    client=client,
    documents=chunks,
    embedding=OpenAIEmbeddings(),
    by_text=False,
)
```

```python
# Define the retriever
retriever = vectorstore.as_retriever()

# Initialize LLM
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)


# --- 2. Define the State for LangGraph ---
class RAGGraphState(TypedDict):
    question: str
    documents: List[Document]
    generation: str


# --- 3. Define the Nodes (Functions) ---
def retrieve_documents_node(state: RAGGraphState) -> RAGGraphState:
    """Retrieves documents based on the user's question."""
    question = state["question"]
    documents = retriever.invoke(question)
    return {"documents": documents, "question": question, "generation": ""}


def generate_response_node(state: RAGGraphState) -> RAGGraphState:
    """Generates a response using the LLM based on retrieved documents."""
    question = state["question"]
    documents = state["documents"]

    # Prompt template from the PDF
    template = """You are an assistant for question-answering tasks. Use the
following pieces of retrieved context to answer the question. If you don't
know the answer, just say that you don't know. Use three sentences maximum and
keep the answer concise.
Question: {question}
Context: {context}
Answer: """
    prompt = ChatPromptTemplate.from_template(template)

    # Format the context from the documents
    context = "\n\n".join([doc.page_content for doc in documents])

    # Create the RAG chain
    rag_chain = prompt | llm | StrOutputParser()

    # Invoke the chain
    generation = rag_chain.invoke({"context": context, "question": question})

    return {"question": question, "documents": documents, "generation":
```

```
generation}


# --- 4. Build the LangGraph Graph ---
workflow = StateGraph(RAGGraphState)

# Add nodes
workflow.add_node("retrieve", retrieve_documents_node)
workflow.add_node("generate", generate_response_node)

# Set the entry point
workflow.set_entry_point("retrieve")

# Add edges (transitions)
workflow.add_edge("retrieve", "generate")
workflow.add_edge("generate", END)

# Compile the graph
app = workflow.compile()


# --- 5. Run the RAG Application ---
if __name__ == "__main__":
    print("\n--- Running RAG Query ---")
    query = "What did the president say about Justice Breyer"
    inputs = {"question": query}
    for s in app.stream(inputs):
        print(s)

    print("\n--- Running another RAG Query ---")
    query_2 = "What did the president say about the economy?"
    inputs_2 = {"question": query_2}
    for s in app.stream(inputs_2):
        print(s)
```

This Python code illustrates a Retrieval-Augmented Generation (RAG) pipeline implemented with LangChain and LangGraph. The process begins with the creation of a knowledge base derived from a text document, which is segmented into chunks and transformed into embeddings. These embeddings are then stored in a Weaviate vector store, facilitating efficient information retrieval. A StateGraph in LangGraph is utilized to manage the workflow between two key functions: `retrieve_documents_node` and `generate_response_node`. The `retrieve_documents_node` function queries the vector store to identify relevant document chunks based on the user's input. Subsequently, the `generate_response_node` function utilizes the retrieved information and a predefined prompt template to produce a response using an OpenAI Large Language Model (LLM). The `app.stream` method allows the execution of queries through the RAG pipeline, demonstrating the system's capacity to generate contextually relevant outputs.

這段 Python 程式碼示範了用 LangChain 與 LangGraph 實作的檢索增強生成（RAG）流程。流程從建立知識庫開始，該知識庫源自一份文本文件，會被切分成區塊並轉為嵌入。這些嵌入接著存入 Weaviate 向量儲存，以利高效檢索。LangGraph 的 StateGraph 用來管理兩個關鍵函式之間的流程：`retrieve_documents_node` 與 `generate_response_node`。`retrieve_documents_node` 會根據使用者輸入查詢向量儲存以找出相關文件區塊；接著 `generate_response_node` 使用檢索到的資訊與預先定義的提示模板，透過 OpenAI 大型語言模型（LLM）產生回應。`app.stream` 方法讓查詢可經由 RAG 流程執行，示範系統生成具脈絡相關輸出的能力。

At Glance

一覽

What: LLMs possess impressive text generation abilities but are fundamentally limited by their training data. This knowledge is static, meaning it doesn't include real-time information or private, domain-specific data. Consequently, their responses can be outdated, inaccurate, or lack the specific context required for specialized tasks. This gap restricts their reliability for applications demanding current and factual answers.

什麼： LLM 具備令人印象深刻的文字生成能力，但本質上受限於訓練資料。這些知識是靜態的，意味著它不包含即時資訊或私有領域資料。因此其回應可能過時、不準確，或缺乏專門任務所需的特定脈絡。這個缺口限制了它們在需要最新且基於事實答案的應用中的可靠性。

Why: The Retrieval-Augmented Generation (RAG) pattern provides a standardized solution by connecting LLMs to external knowledge sources. When a query is received, the system first retrieves relevant information snippets from a specified knowledge base. These snippets are then appended to the original prompt, enriching it with timely and specific context. This augmented prompt is then sent to the LLM, enabling it to generate a response that is accurate, verifiable, and grounded in external data. This process effectively transforms the

LLM from a closed-book reasoner into an open-book one, significantly enhancing its utility and trustworthiness.

為什麼： 檢索增強生成（RAG）模式透過將 LLM 與外部知識來源連結，提供標準化解決方案。收到查詢時，系統先從指定知識庫檢索相關資訊片段，接著將這些片段附加到原始提示，補充即時且特定的脈絡。增強後的提示再送入 LLM，使其能產出準確、可驗證且以外部資料為基礎的回應。這個流程有效地把 LLM 從「封閉式推理」轉為「開放式推理」，大幅提升其實用性與可信度。

Rule of Thumb: Use this pattern when you need an LLM to answer questions or generate content based on specific, up-to-date, or proprietary information that was not part of its original training data. It is ideal for building Q&A systems over internal documents, customer support bots, and applications requiring verifiable, fact-based responses with citations.

經驗法則： 當你需要 LLM 以訓練資料未包含的特定、最新或專有資訊來回答問題或生成內容時，就應使用此模式。它非常適合建構內部文件的 Q&A 系統、客服機器人，以及需要可驗證且具引用的事實型回應的應用。

Visual Summary:

視覺摘要：

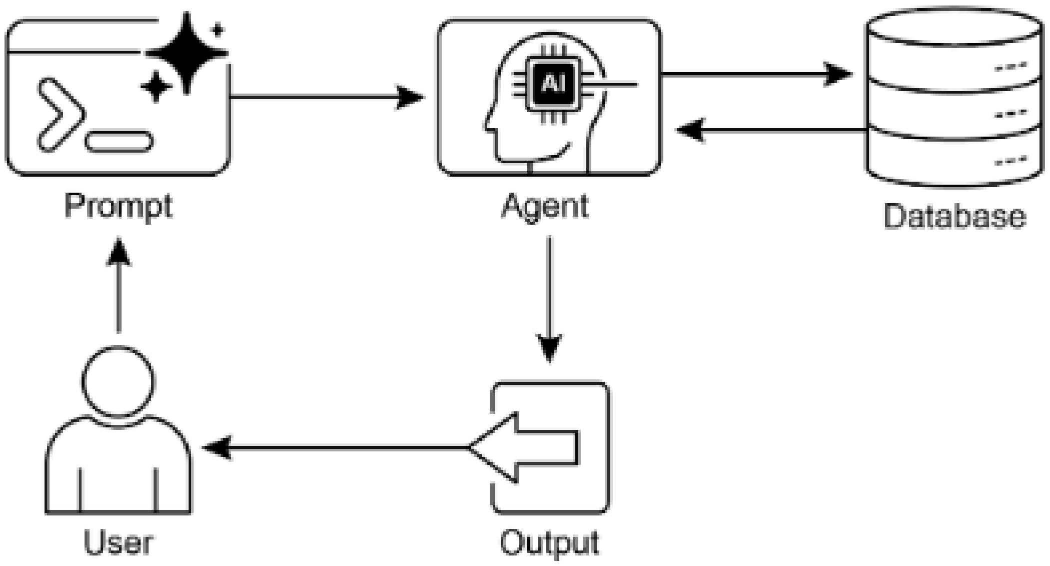

Figure 6: Knowledge Retrieval Pattern Database

Knowledge Retrieval pattern: an AI agent to query and retrieve information from structured databases

知識檢索模式：AI 代理人從結構化資料庫查詢並取回資訊

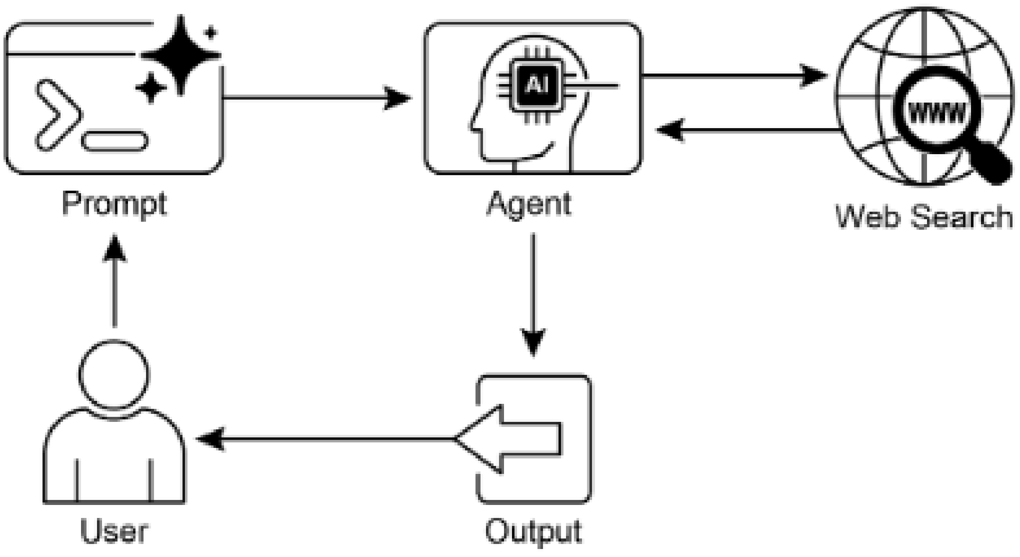

Figure 7: Knowledge Retrieval Pattern Search

Fig. 3: Knowledge Retrieval pattern: an AI agent to find and synthesize information from the public internet in response to user queries.

圖 3：知識檢索模式：AI 代理人從公共網路取得資訊並加以綜合以回應使用者查詢。

Key Takeaways

重要重點

- Knowledge Retrieval (RAG) enhances LLMs by allowing them to access external, up-to-date, and specific information.
- The process involves Retrieval (searching a knowledge base for relevant snippets) and Augmentation (adding these snippets to the LLM's prompt).
- RAG helps LLMs overcome limitations like outdated training data, reduces "hallucinations," and enables domain-specific knowledge integration.
- RAG allows for attributable answers, as the LLM's response is grounded in retrieved sources.

- GraphRAG leverages a knowledge graph to understand the relationships between different pieces of information, allowing it to answer complex questions that require synthesizing data from multiple sources.
- Agentic RAG moves beyond simple information retrieval by using an intelligent agent to actively reason about, validate, and refine external knowledge, ensuring a more accurate and reliable answer.
- Practical applications span enterprise search, customer support, legal research, and personalized recommendations.
- 知識檢索（RAG）讓 LLM 能存取外部、最新且特定的資訊，進而提升能力。
- 流程包含檢索（在知識庫搜尋相關片段）與增強（將片段加入 LLM 提示）。
- RAG 協助 LLM 克服訓練資料過時等限制，降低「幻覺」，並整合特定領域知識。
- RAG 允許可歸因的答案，因為回應以檢索來源為依據。
- GraphRAG 利用知識圖譜理解資訊之間的關係，能回答需要整合多來源資料的複雜問題。
- 代理式 RAG 不僅檢索資訊，還透過智慧代理主動推理、驗證與精煉外部知識，確保更準確可靠的答案。
- 實務應用涵蓋企業搜尋、客服支援、法律研究與個人化推薦。

Conclusion

結論

In conclusion, Retrieval-Augmented Generation (RAG) addresses the core limitation of a Large Language Model's static knowledge by connecting it to external, up-to-date data sources. The process works by first retrieving relevant information snippets and then augmenting the user's prompt, enabling the LLM to generate more accurate and contextually aware responses. This is made possible by foundational technologies like embeddings, semantic search, and vector databases, which find information based on meaning rather than just keywords. By grounding outputs in verifiable data, RAG significantly reduces factual errors and allows for the use of proprietary information, enhancing trust through citations.

An advanced evolution, Agentic RAG, introduces a reasoning layer that actively validates, reconciles, and synthesizes retrieved knowledge for even greater reliability. Similarly, specialized approaches like GraphRAG leverage knowledge graphs to navigate explicit data relationships, allowing the system to synthesize answers to highly complex, interconnected queries. This agent can resolve conflicting information, perform multi-step queries, and use external tools to find missing data. While these advanced methods add complexity and latency, they drastically improve the depth and trustworthiness of the final response. Practical applications for these patterns are already transforming industries, from enterprise search and customer support to personalized content delivery. Despite the challenges, RAG is a crucial pattern for making AI more knowledgeable, reliable, and useful. Ultimately, it transforms LLMs from closed-book conversationalists into powerful, open-book reasoning tools.

總結來說，檢索增強生成（RAG）透過連結外部、最新的資料來源，解決大型語言模型靜態知識的核心限制。此流程先檢索相關資訊片段，再增強使用者提示，讓 LLM 生成更準確且更具脈絡的回應。這仰賴嵌入、語意搜尋與向量資料庫等基礎技術，它們以「意義」而非僅關鍵字來找資訊。透過以可驗證資料為依據，RAG 大幅降低事實錯誤，並允許使用專有資訊，透過引用提升信任。

更進一步的演進是代理式 RAG，它加入推理層主動驗證、調和與整合檢索知識，以提升可靠性。同樣地，像 GraphRAG 這類專門方法利用知識圖譜，瀏覽明確的資料關係，以整合高度複雜、相互關聯的查詢答案。該代理可解決衝突資訊、進行多步查詢，並使用外部工具尋找缺失資料。儘管這些進階方法增加複雜度與延遲，但它們大幅提升最終回應的深度與可信度。這些模式的實務應用已在企業搜尋、客服支援與個人化內容傳遞等產業產生改變。儘管存在挑戰，RAG 仍是讓 AI 更有知識、更可靠、更有用的重要模式。最終，它將 LLM 從封閉式對話者轉變為強大的開放式推理工具。

References

參考資料

1. Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. https://arxiv.org/abs/2005.11401
2. Google AI for Developers Documentation. Retrieval Augmented Generation - https://cloud.google.com/vertex-ai/generative-ai/docs/rag-engine/rag-overview
3. Retrieval-Augmented Generation with Graphs (GraphRAG), https://arxiv.org/abs/2501.00309
4. LangChain and LangGraph: Leonie Monigatti, “Retrieval-Augmented Generation (RAG): From Theory to LangChain Implementation,” https://

medium.com/data-science/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4e9bd5f6a4f2

5. Google Cloud Vertex AI RAG Corpus https://cloud.google.com/vertex-ai/generative-ai/docs/rag-engine/manage-your-rag-corpus#corpus-management
6. Lewis, P., et al.（2020）。《Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks》。https://arxiv.org/abs/2005.11401
7. Google AI for Developers Documentation。《Retrieval Augmented Generation – https://cloud.google.com/vertex-ai/generative-ai/docs/rag-engine/rag-overview》
8. Retrieval-Augmented Generation with Graphs（GraphRAG），https://arxiv.org/abs/2501.00309
9. LangChain and LangGraph：Leonie Monigatti〈Retrieval-Augmented Generation (RAG): From Theory to LangChain Implementation〉，https://medium.com/data-science/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4e9bd5f6a4f2
10. Google Cloud Vertex AI RAG Corpus，https://cloud.google.com/vertex-ai/generative-ai/docs/rag-engine/manage-your-rag-corpus#corpus-management