

https://youtu.be/o1mC-RCALSg?list=PLIAzIZzCjtLJQzoK-1n6R_KVTXm4tHZqP

https://youtu.be/We8BONFMSZU?list=PLIAzIZzCjtLJQzoK-1n6R_KVTXm4tHZqP

這份原始資料摘錄自台大網頁程式設計課程的講義，主要探討如何運用 **Agentic AI Flow** 與 **LINE Bot** 實作對話式使用者體驗。文中強調開發者不應僅將聊天機器人當作 AI 的**傳聲筒**，而應學會設計完整的**對話劇本與流程**，透過引導式互動來克服使用者意圖不明或平台技術限制。為了降低開發複雜度，講者建議採用 **Incremental 開發策略**，先從基礎的 Echo Bot 啟動，再逐步疊加資料庫查詢與 **Prompt Engineering** 等功能。最後，該課程旨在讓學生體會，即便在 AI 輔助下，建構一個能處理複雜邏輯並維持**系統乾淨度**的網頁應用程式，仍需深度的工程思考與創意設計。

這份教學源自台大網頁設計課程，旨在指導學生如何從基礎的 **AI 對話機器人** 進化為具備商業競爭力的 **對話式 UI/UX 系統**。講師強調，開發者不應僅依賴 AI 工具（如 Cursor）自動生成程式碼，而必須深入理解 **系統流程與狀態管理（Stateful vs. Stateless）**，並運用 **console.log** 與數據分析來排除 AI 無法解決的邏輯盲點。文中詳盡介紹了 **LINE Messaging API** 的多樣化元件，如 **Flex Message**、**Image Map** 與 **LIFF** 網頁框架，教導如何透過選單引導避免使用者迷失。最後，課程提升至 **Agentic AI Flow** 的層次，解釋如何利用 **Prompt Template** 結構化對話資訊，並結合 **Zod** 驗證資料格式，使 AI 回應能精確對接後端邏輯，從而打造出真正具備良好用戶體驗的數位服務。

Discuss 對話式情境設計.

對話式情境設計（Conversational UI/UX Design） **理解並回應使用者的「意圖」**，提供如同真人對話般、個人化且流暢的互動體驗。

以下根據來源內容，深入探討對話式情境設計的關鍵要素、挑戰與實作策略：

1. 對話式情境的核心挑戰

對話式介面的開發者面臨兩大核心挑戰：**理解（聽懂）** **回應（行動）**。

- **語言的模糊性**：人類語言通常不夠精確且高度隱晦，充滿了背景知識、風俗文化或約定俗成的含義。例如「不要跟我老婆亂講」在特定情境下意味著「什麼都不要講」，而非單純的語法解析能理解。
- **上下文（Context）的缺失**：直接呼叫大語言模型（LLM）的 API 時，預設是沒有記憶的，這會讓機器人顯得很笨。因此，設計者必須建立 **Session 或 Context 的觀念**，將過去的對話歷史包裝給模型，使其能掌握當前的溝通進度。

2. 從「傳聲筒」轉向「代理人流程」（Agentic Flow）

來源強調，一個好的對話式設計不應只是將使用者的話丟給 AI 後再回傳的「傳聲筒」，而應該具備**處理流程的控制能力**。

- **引導與訪談**：使用者通常無法一次完整表達需求。設計者應思考如何透過**多輪對話**，像專業代理人（如旅遊代辦）一樣，引導使用者提供資訊、提供選項，最後完成複雜任務。
- **劇本式規劃**：為了避免使用者天馬行空地亂問，建議預先規劃**對話流程與腳本**（類似狀態圖或劇本遊戲），預判使用者的意圖並將其引導至設定好的情境中。

3. 善用平台組件優化體驗

在對話式情境中，單純文字並非唯一手段。善用平台的 UI 組件（如 LINE 的按鈕、圖文選項）能大幅提升體驗：

- **降低表達負擔：** 當介面提供按鈕時，使用者傾向直接點擊而非打字，這不僅簡化了互動，也將使用者的行為限制在開發者可控的範圍內，達成雙贏。
- **結構化回應：** 為了讓後端伺服器（B Server）能理解 LLM 的產出，設計者應要求模型以**結構化格式（如 JSON）**回覆，以便觸發後續的程式動作（Call to action）。

4. 實作建議與開發精神

- **漸進式開發（Incremental Development）：** 建議從最簡單的 Echo Bot（傳聲筒）開始，確保基本連線與佈署成功後，再逐步疊加功能。
- **工程化處理：** 在將使用者需求丟給模型前，伺服器應先進行 **Prompt Engineering**，結合資料庫（如 RAG 技術）或歷史紀錄來補足資訊，讓 AI 的回覆更精準且符合業務規範。

比喻理解： 設計對話式情境就像是在製作一張**捲軸式的迷宮地圖**。你不能預期玩家會乖乖走直線，但你可以透過在地圖上標註明顯的「路標」（按鈕）或設置「導遊」（Agent），引導玩家在有限的路線中探索，最終抵達你為他們準備好的寶藏終點，而不至於讓他們在無邊際的語言荒野中迷路。

LINE 的 push notification 每月免費次數為何？

何謂 Agentic Flow 中的 Agent 定義？

傳統圖形介面與對話式介面的互動差異為何？

Discuss LINE 平台開發限制.

在 LINE 平台上開發對話式應用程式（如 Chatbot）時，開發者會面臨來自**技術規範、成本機制以及介面彈性**等多方面的限制。根據來源內容，以下詳細討論這些開發限制：

1. 推播通知的嚴格限制 (Push Notification Limits)

這是 LINE 開發中最顯著的技術與成本限制之一：

- **免費額度有限：** LINE 平台對於「主動推播」（Push Notification，即在使用者未主動發話的情況下，由伺服器主動發訊息給使用者）有非常嚴格的規定。**每個月僅提供 100 次免費推播額度**，超過此限額則需要額外付費。
- **影響功能設計：** 這項限制會直接影響需要主動提醒、通知的功能（如遊戲換手提醒或定時通知）。
- **回覆與推播的區別：** 開發者通常需優先利用 **reply token** 進行回覆。當使用者主動傳訊給機器人時，系統會獲得一個 reply token，此時的回傳不計入推播次數；但若無此 Token 卻想發訊，就必須動用收費的推播功能。

2. 使用者互動流轉的限制

- **難以跨使用者傳遞對話：**在 LINE 的架構下，若想將使用者 A 的對話內容即時轉發或串接給使用者 B，在實作上會遇到相當大的限制與挑戰。
- **被動互動模式：**系統多數時候處於「被動」狀態，必須依賴使用者主動開啟對話，這對設計主動式的代理人流程（Agentic Flow）增加了難度。

3. UI/UX 介面的設計彈性 (UI/UX Flexibility)

- **預設介面單純化：**LINE 作為聊天平台，其介面本質上是高度規範且單純的，這雖然降低了設計複雜度，但也限制了 UI/UX 的表現空間與彈性。
- **環境先天的限制：**開發者無法像開發網頁一樣隨心所欲地佈局，必須在 LINE 既有的對話框架構下進行設計。
- **引導與限制的雙刃劍：**為了優化體驗，開發者必須善用 LINE 內建的組件（如**按鈕**、**圖文選項**）。這雖然能簡化互動並引導使用者，但同時也將使用者的行為**侷限在預設的選項中**。

4. 系統開發與維護的複雜性

- **技術債與維護困難：**由於 LINE 的對話邏輯通常與後端伺服器（B Server）高度耦合，若在開發初期沒有將介面、資料結構設計乾淨，隨著功能增加，系統會變得極其複雜，甚至出現「牽一髮而動全身」而難以修正的情況。
- **理解意圖的負擔：**使用者在 LINE 上的表達往往是不精確、隱晦且充滿個人色彩的（Freestyle）。對於開發者而言，若要在有限的資訊中精準回應使用者的意圖，必須投入大量的工程成本處理上下文（Context）與對話歷史。

比喻理解：在 LINE 上開發就像是在一間**受限制的預售屋**內做裝潢。你不需要擔心建築結構（介面框架），但你不能隨意拆牆或改窗戶（UI 限制），且每當你想主動打電話給住戶（推播通知）時，電信公司（LINE 平台）都會收取高額費用。因此，你必須在有限的空間內，精準地放置「按鈕」與「圖文家具」，讓使用者能順著你設計好的動線移動。

什麼是「推播通知」(push notification) 的免費額度限制？

在開發對話式 UI 時，何謂「意圖」(intent)？

請簡述 Agentic Flow 中「代理人」(Agent) 的概念。

Discuss 代理人流程控管。

代理人流程控管（Agentic Flow Control）在對話式系統中，是指讓程式不再僅僅是簡單的指令回應器，而是像一個具備主動性、能處理複雜任務的「代理人（Agent）」。

根據來源內容，有效的代理人流程控管應包含以下核心策略與機制：

1. 從「指令式」轉向「引導式」流程

代理人流程的核心在於**多輪對話與訪談**。

- **訪談與溝通**：使用者往往無法一次性完整表達需求。代理人的任務是透過多輪溝通，引導使用者提供必要的資訊。

- **提供選項**：像旅遊代理人（Travel Agent）一樣，當使用者目標模糊時（例如：只想去日本但不知去哪），代理人應主動提供選項（Options）協助使用者縮小範圍，最終完成任務。

2. 腳本式規劃與狀態控管

為了避免使用者「天馬行空」導致系統無法回應，設計者必須主動**控管流向**：

- **預設腳本（Scenario/Script）**：設計者應規劃好對話流程，將可能的情境預先定義好（類似狀態圖），將使用者引導至設定好的路徑中。

- **意圖預判**：透過流程設計，在使用者尚未表達完整前，就根據當前所處的流程步驟預判其意圖，並給予相應的提示。

3. 工程端的品質控管機制

要確保代理人流程的品質，後端伺服器（B Server）必須介入並執行以下操作：

- **上下文與 Session 管理**：由於大語言模型（LLM）的 API 預設是沒有記憶的，B Server 必須負責**儲存對話歷史（History）**，並在發送新請求時將過去的脈絡包裝進去，否則機器人會顯得非常笨拙。

- **提示詞工程（Prompt Engineering）與「腦補」**：B Server 在將使用者的簡短對話丟給模型前，應進行 **Prompt Template** 的填充，加入背景知識或資料庫資訊（如 RAG 技術），讓模型的產出更具精準度與符合業務邏輯。

- **結構化回傳（Call to Action）**：代理人流程通常涉及後續動作。設計者應要求模型以**結構化格式（如 JSON）**回覆，以便程式能解析並執行具體的後端操作（如：查資料庫、發送推播）。

4. 善用介面組件作為流程約束

在 LINE 等平台開發時，**UI 組件**是流程控管的重要工具：

- **按鈕與圖文選單**：當介面出現按鈕時，使用者傾向點擊而非打字。這不僅降低了溝通負擔，更重要的是將**使用者的行為限制在開發者預設的流程內**，達成雙贏的控管效果。

5. 漸進式開發精神

由於複雜的代理人流程涉及多層次的互動，來源強烈建議採取**漸進式開發（Incremental Development）**：

- **從小到大**：先確保基本的傳聲筒功能運作，再逐步疊加多輪對話、歷史記憶與資料庫檢索功能。

- **快速疊代**：透過快速測試與修復「隱藏的 Bug」，確保每一階段的功能都穩定，避免系統因過於複雜而導致開發者無法掌控。

比喻理解：代理人流程控管就像是一位**資深飯店櫃檯經理**。他不會只站在櫃檯等客人下令，而是會主動詢問客人的旅遊偏好，提供地圖與推薦餐廳（選項），並在客人提問時，記住客人的房號與之前的特殊要求

(Context)。雖然他給予客人親切對話的自由，但實際上是將客人引導在飯店能提供的服務流程內，確保服務既個人化又不會出錯。

LINE 的 push notification 免費額度是多少？

LINE 機器人開發中 reply token 的作用為何？

什麼是對話式 UI/UX 設計中的「意圖」？

Discuss 提示詞工程模板.

在對話式系統的開發中，**提示詞工程模板 (Prompt Template)** 扮演著橋樑的角色，負責將使用者的簡短輸入轉化為大語言模型 (LLM) 能理解並高品質回應的詳細指令。以下根據來源內容探討其核心觀念與實作策略：

1. 提示詞模板的必要性

在 LINE 或其他聊天平台上，使用者的表達往往非常簡約或不精確 (Freestyle)。

- **消除模型的「中性」特質**：大語言模型本身是相對中性且無辜的，給予什麼樣的資訊，它就回傳什麼樣的品質 (Garbage in, garbage out)。
- **腦補背景資訊**：提示詞模板的作用在於由後端伺服器 (B Server) 進行「腦補」，在將訊息丟給模型前，先包裝一堆背景資訊 (Information)，使回應更貼心且符合需求。
- **克服 API 的限制**：直接呼叫 LLM 的 API 時，模型預設不具備對話歷史紀錄，會顯得很笨；開發者必須透過模板將 **Session 或 Context (上下文)** 重新餵給模型，它才能掌握目前的對話進度。

2. 模板的運作機制：填充與流轉

提示詞模板並非固定不變，而是根據**代理人流程 (Agentic Flow)** 動態生成的：

- **對應流程節點**：當對話進入不同的「劇本」或「流程節點」時，系統會對應不同的提示詞模板。
- **動態填充資料**：系統會根據使用者在流程中的輸入，將資料填入模板 (Template) 中，最後拼湊成一個完整的 Prompt 丟給模型。
- **結合外部資源**：為了提供精準的服務 (如保險諮詢或爬山建議)，模板中常需包含從**資料庫 (DB)** 或 **FAQ** 檢索出的結構化資訊，確保模型不會胡言亂語。

3. 控制輸出格式與後續行動

一個進階的提示詞模板不僅規範輸入，也規範**輸出**：

- **結構化回應**：為了讓後端伺服器能解讀模型的回答，模板應要求模型以**特定格式 (如 JSON)** 回覆。
- **觸發行動 (Call to Action)**：透過模板引導模型產出具備邏輯的指令，使系統能執行如「存入資料庫」或「顯示按鈕選項」等具體動作。

4. 實作建議：漸進式優化

來源建議開發者不應追求一步到位，而應透過**迭代 (Iteration)** 來優化模板：

- **從傳聲筒開始：** 第一步可先做簡單的 Echo Bot（傳聲筒），確保連線正常。
- **加入歷史紀錄：** 第二步在模板中加入過去幾句的對話紀錄，讓模型具備基本的連貫性。
- **細化劇本：** 根據實際測試結果，預判使用者意圖並細化各個流程節點的模板內容。

比喻理解： 提示詞模板就像是給演員（LLM）看的「劇本提詞機」。使用者傳來的簡訊只是一句台詞，但提詞機會立刻在後台補上「現在的場景、之前的劇情摘要、你應有的口氣」，讓演員能在幾秒鐘內接戲，演出一段既專業又符合邏輯的對話，而不至於在舞台上發愣或胡言亂語。

LINE 的 push notification 免費次數限制為何？

何謂 Agentic Flow 中的 Agent 定義？

開發對話式 UI 時，聽懂與回應何者較困難？

Discuss 漸進式功能開發。

漸進式功能開發（Incremental Development） 是來源中針對對話式系統開發所提倡的核心精神，特別是在處理結合 LINE 平台與大語言模型（LLM）等複雜技術時，這種開發策略能有效降低失敗率並確保系統品質,,。

以下根據來源內容，探討漸進式功能開發的實作步驟、優勢與關鍵精神：

1. 核心開發策略：從「傳聲筒」開始

來源建議開發者不要試圖一步到位完成複雜功能，而應將開發過程切分為多個 **Milestones（里程碑）**, :

- **第一步：實作 Echo Bot（傳聲筒）：** 先確保系統能成功串接 LINE 平台與伺服器，達成「使用者講什麼，機器人就回什麼」的最簡功能。
- **第二步：串接 LLM API：** 在連線穩定的基礎上，再將使用者的話傳送給大語言模型並回傳其答案。
- **第三步：加入上下文與歷史紀錄：** 透過資料庫（DB）儲存過去的對話，並在 Prompt 中包裝歷史訊息（如過去 10 句對話），讓模型具備記憶能力。
- **後續進階：** 逐步疊加代理人流程（Agentic Flow）、資料庫檢索（RAG）或結構化回應處理等功能,。

2. 為什麼對話式系統適合漸進式開發？

來源指出，對話式介面（CUI）在開發上有其獨特優勢：

- **結構穩定性：** 傳統的網頁介面（GUI）若隨意增加功能，常會發生後來的功能弄壞前面介面的情況。
- **對話情境的包容性：** 由於對話式介面始終維持在「對話」的情境中，功能可以像堆積木一樣慢慢疊加（Incremental），且每一步都可以直接透過對話進行實測，較不容易發生功能衝突,。

3. 開發中的品質與風險控管

在快速迭代（Iteration）的過程中，開發者應維持以下工程紀律：

- **每一步都需驗證：** 每完成一個步驟就要檢查是否運作正常 (Work)，並養成打 Tag (版本標籤) 與加測試的習慣，確保出錯時能隨時回到上一個穩定的版本,。
- **提早佈署與測試：** 建議盡早將應用程式佈署到雲端並進行實測，因為許多隱藏的 Bug 往往是在非預期的使用者互動中才被發現,。
- **理解 AI 寫的程式碼：** 即便使用 AI 輔助開發 (如 Cursor)，也應避免過度依賴而產生「偷懶」心理。開發者必須理解 AI 產出的程式碼邏輯，否則當系統變得龐大複雜時，將無法進行有效的維護或修正,。

4. 克服先天限制與折衷

漸進式開發也包含對環境限制的理解。開發者在開發過程中會逐漸意識到平台 (如 LINE 的推播限制) 或技術 (如 LLM 的不確定性) 的瓶頸，進而透過**工程化的手段 (Engineering Solution)** 進行功能上的取捨與平衡 (Compromise)，。

比喻理解： 漸進式功能開發就像是在**建造一座樂高城堡**。你不是一覺醒來就擁有一整座城堡，而是先鋪好穩固的地基 (Echo Bot)，接著一塊一塊地砌上城牆 (LLM 串接) 與塔樓 (歷史記憶)。因為是逐塊疊加，你可以隨時停下來檢查每一塊積木是否穩固，就算發現某一層蓋歪了，也能輕易拆除重來，而不會導致整座建築瞬間崩塌。

LINE Push Notification 的免費額度與限制為何？

何謂 Agentic Flow 中的「代理人」概念？

在開發對話式 UI 時，處理使用者意圖的兩大核心任務為何？

Discuss 輔助開發與調試.

在對話式系統與 Web 應用程式的開發過程中，利用 AI 工具輔助開發與進行有效的調試 (Debugging) 已成為現代開發者的核心技能。根據來源內容，以下詳細討論輔助開發的策略、調試的方法以及確保系統品質的關鍵流程：

1. AI 輔助開發的核心策略

現代開發者常利用 **Cursor** 等 AI 工具來加速編碼，但這並不代表開發變得輕鬆，而是開發重心的轉移。

- **節省開發成本：** AI 能大幅縮減撰寫程式碼、思考邏輯與閱讀文件的時間，讓開發者能更專注於 UI/UX 的設計與系統深入思考。
- **引導與重構 (Refactoring)：** 開發者應具備程式概念，主動引導 AI 進行**代碼重構**，確保程式碼結構乾淨且易於後續維護。
- **避免盲目依賴：** 來源強烈提醒，即便程式碼是 AI 寫的，開發者仍必須**理解並能解釋其邏輯**。若系統變複雜後發生故障，開發者若因偷懶而不了解代碼，將無法進行修正。

2. 系統調試 (Debugging) 的方法與挑戰

調試是確保服務穩定運行的關鍵。當系統出現 bug 時，來源建議採取以下行動：

- **克服 AI 的盲點：**工程師常因「盲點」而寫出 bug，而 AI 同樣會有盲點。若 AI 產出的代碼有問題，單純要求它修正可能會陷入無效循環，此時開發者必須**親自跳進程式碼中分析流程**。
- **觀察日誌 (Logging)：**利用 **console.log** 是最基本且有效的觀察手段。開發者需要清楚日誌會出現在前端 (Frontend)、後端伺服器 (B Server) 還是雲端佈署環境中，才能精準定位問題。
- **錯誤訊息處理：**當出現錯誤 (Error message) 時，可將其複製給 AI 輔助修正，這能解決大部分明顯的問題。
- **資料驗證與防錯：**為了避免資料型別錯誤導致系統崩潰，建議利用如 **Zod** 等工具定義 **Data Schema** 並進行驗證，確保回傳資料符合預期格式。

3. 效能調試與技術方案

除了功能性錯誤，**效能 (Performance) 問題**也是調試的重點：

- **效能瓶頸分析：**例如當發現 B Server 與資料庫 (DB) 溝通太慢時，需判斷是程式寫法不佳，還是查詢本身就耗時。
- **技術升級方案：**針對資料庫讀取過慢的問題，開發者可主動引導 AI 導入 **Redis** 等記憶體快取 (Memory Cache) 機制，將頻繁讀取的對話歷史暫存，以大幅提升回應速度與流暢度。

4. 流程控管與品質保證

良好的開發流程能從源頭減少調試的壓力：

- **漸進式開發與測試 (Incremental Testing)：**建議將開發切分為多個里程碑 (Milestones)，每完成一個功能就進行完整測試，確保其運作正常後再進行下一步。
- **版本管理：**養成打 **Tag (版本標籤)** 的習慣，這能讓開發者在系統改壞時，隨時有路可退，回到上一個穩定的版本。
- **提早佈署與實測：**應盡早將應用程式佈署到雲端。許多隱藏的 Bug 只有在使用者實際測試、面對非預期的操作案例 (Case) 時才會顯露出來。

比喻理解：輔助開發與調試就像是駕馭一架具備「自動駕駛」功能的飛機。AI 是強大的自動駕駛系統，能幫你處理繁瑣的航行細節 (寫扣)，但身為機長的開發者必須清楚飛行原理與航線 (邏輯)，隨時監控儀表板上的數據 (Logging)。當遇到亂流或系統判斷錯誤時 (Bug)，機長必須能立刻切回手動模式修正航向，否則僅靠自動駕駛可能無法避開預料外的障礙。

為什麼對話式介面建議設計「主選單」？

試比較無狀態與具狀態對話設計的差異。

開發者如何利用 JSON Schema 控管模型輸出？

Discuss 對話腳本設計。

在對話式系統開發中，**對話腳本設計 (Conversational Script Design)** 是決定使用者體驗 (UX) 好壞的核心，其目的是將使用者從「漫無目的的自由對話」引導至「有目標的服務流程」中。

以下根據來源內容，整理對話腳本設計的關鍵原則與實作步驟：

1. 定義服務範圍與目標受眾 (TA)

設計對話腳本的第一步是**明確界定服務範圍**。開發者不應試圖製作一個無所不知的通用型機器人（如 ChatGPT），而應專注於特定場域（如保險諮詢、登山引導或餐廳訂位），並定義該範圍內的具體服務項目，超出範圍則不予處理。

2. 核心架構：將腳本圖譜化 (Graph Structure)

一個好的對話腳本應被視為一種圖（**Graph**）結構或狀態機（**State Machine**）：

- **節點 (Node)**：代表對話的當前狀態。
- **子節點與連線**：代表使用者在該狀態下可選擇的路徑。
- **終點**：每個服務流程都應有明確的終點，即完成任務並給予回應的地方，且通常應在終點提供「回到主選單」的選項。

3. 主選單設計與引導

由於使用者面對機器人時常不知如何開口，**主選單**是不可或缺的導航工具：

- **歡迎訊息**：當使用者加入好友時，應立即彈出包含主選單的歡迎訊息。
- **限制行為**：透過選單選項（按鈕或圖文選單），開發者能將使用者的行為限制在可控的流程內，避免其「天馬行空」地發問導致系統無法處理。
- **容錯機制**：當系統遇到「聽不懂」或無法處理的情況時，最保險的做法是引導使用者回到主選單重新選擇。

4. 靜態與動態對話設計 (Stateless vs. Stateful)

- **靜態對話 (Stateless)**：每個請求都是獨立的（如簡單的一問一答 FAQ），系統不需記憶先前的對話。
- **狀態對話 (Stateful)**：系統會記錄對話歷史與上下文（Context），能理解如「取消它」之類的代名詞，適合複雜的交易流程。實作上需透過資料庫（DB）儲存對話狀態，以便模型判斷當前流程進度。

5. 善用平台元件優化流程

對話腳本不應只依賴純文字，應結合 LINE 等平台提供的 UI 元件：

- **按鈕 (Buttons) 與 快速回覆 (Quick Reply)**：讓使用者點擊而非打字，不僅降低互動負擔，也能確保輸入資訊的精確度。
- **圖文選單 (Image Map) 與 輪播介面 (Carousel)**：適合展示商品或多樣化的功能入口。
- **彈性訊息 (Flex Message)**：類似微型網頁，可自訂結構化的佈局來呈現資訊。

6. 漸進式迭代與優化

腳本設計不需要一步到位。建議從最直覺的 core 功能開始設計：

- **從小到大**：先完成一兩項最重要的腳本，上線後收集回饋。
 - **行為分析**：記錄使用者的對話路徑。如果發現許多使用者在多層點擊後才找到某功能，可考慮將該功能移至主選單（快捷路徑），以提升效率。
 - **靈活調整**：對話式介面的優勢在於，調整選項層級（如合併 A、B 選項至新選單 X）對使用者體驗的衝擊較小，自然語言會協助緩衝這些變動。
-

比喻理解：設計對話腳本就像是在為遊客**規劃一場主題公園的導覽**。你不需要讓遊客在整座荒山隨意探索（自由對話），而是要修築好路徑（流程節點）、設置清楚的指示牌（主選單與按鈕），並在適當的景點安排導覽員（LLM 的專業回應）。即便遊客走錯路，只要路標清楚，隨時都能引導他們回到規劃好的主幹道上。

LINE 的 push notification 免費額度與限制為何？

何謂 stateful 設計及其在對話流程中的作用？

如何確保大語言模型的格式化回應以便程式解析？

Discuss LINE UI 元件.

在 LINE 平台的對話式開發中，善用內建的 **UI 元件** 不僅能提升使用者體驗，還能有效地透過視覺化引導來控管對話流程，降低使用者輸入的負擔。

以下根據來源內容，詳細討論 LINE 平台提供的各類 UI 元件及其應用場景：

1. 基礎訊息元件

- **文字訊息 (Text Message)**：最基本的溝通方式。進階的文字訊息 (V2) 支援使用變數（如 `{{name}}`）來直接稱呼使用者名子，增加互動的人性化感受。
- **多媒體訊息**：包括 **Sticker**（貼圖）、**Image**（圖片）、**Video**（影片）及 **Audio**（音訊）。其中圖片訊息包含「預覽圖」與點擊後的「完整圖」兩階段呈現。
- **特殊訊息**：如 **Location**（位置訊息）及 **Coupon**（優惠券）。

2. 互動引導元件

為了避免讓使用者「天馬行空」地輸入文字而導致機器人難以理解，設計者常使用以下元件來限制與引導行為：

- **主選單 (Main Menu)**：這是使用者與機器人接觸的第一步（如歡迎訊息中）。良好的主選單設計能協助使用者快速開始，並將其引導至設定好的服務路徑中。
- **按鈕 (Buttons)**：通常以「Bubble」形式呈現，包含頁首圖 (Hero)、內文 (Body) 與頁尾按鈕 (Footer)。按鈕在對話中**不會消失**，可用於需要長期留存的操作選單。
- **快速回覆 (Quick Reply)**：出現在對話框底部的選項按鈕，**點擊後即會消失**。適合用於一次性的選擇，例如詢問「是/否」或提供暫時性的選項。

- **圖文選單 (Image Map)**：使用者看到的圖檔 (PNG) 可根據座標定義不同的觸發區域，提供視覺化且具備導覽功能的互動界面。

3. 進階與複雜佈局元件

- **輪播模板 (Carousel Template)**：允許使用者左右滑動查看多個選項 (如商品清單)，每個選項可獨立包含圖片、文字與按鈕。

- **彈性訊息 (Flex Message)**：最強大的 UI 工具。它基於 JSON 格式定義，採用類似 **Flexbox** 的佈局觀念，讓開發者能像設計網頁一樣自定義複雜的界面結構 (如格線、分割線、按鈕佈局)。

- **LIF (LINE Front-end Framework)**：雖然外觀像瀏覽器分頁，但它是一個可與平台互動的 Web App。LIF 能直接取得使用者的 User ID、傳送訊息或關閉視窗，適合處理涉及安全性 (如輸入密碼) 或需高度客製化的頁面。

4. 特殊功能與動作 (Actions)

LINE UI 元件可綁定特定的 Action 來觸發後端邏輯：

- **Postback Action**：點擊按鈕後，會將隱藏的 Payload (包裝好的資訊) 傳回伺服器，讓後端執行特定動作 (如查詢特定基金編號) 而不顯示在對話框中。

- **內建工具觸發**：元件可連結至手機的功能，例如**打開相機、掃描 QR code 或開啟時間選擇器**。

5. 組合與回覆限制

開發者應注意，一次對話回覆最多可以**包裝五個元件** (例如：一個文字 + 一個圖片 + 一個 Flex Message)，利用這種打包方式可以創造出資訊極其豐富的回應效果。

比喻理解：LINE UI 元件就像是**自助餐廳裡的「餐盤與標示牌」**。如果你讓客人自己去廚房拿菜 (純文字 Freestyle)，場面會變得很混亂；但如果你準備好不同形狀的格子餐盤 (Flex Message)、提供清晰的導覽選單 (Image Map)，並在關鍵位置放上點餐按鈕，客人就能順著你安排好的動線，快速且精準地選到他們想要的餐點，讓整個服務過程既流暢又高效。

什麼是 LINE 平台的「回覆權杖」(reply token)？

開發對話式服務時，如何維持「有狀態」(stateful) 的設計？

何謂 Liff 元件及其在使用者識別上的優勢？

Discuss 代理式人工智慧流。

代理式人工智慧流 (Agentic AI Flow) 是指在對話式系統設計中，讓程式不再只是簡單的指令回應器 (傳聲筒)，而是具備主動性、能如同「代理人 (Agent)」般處理複雜任務的服務流程。這種設計的核心在於透過**多輪對話與流程控管**，引導使用者從模糊的需求轉向具體的結果。

以下根據來源內容，詳細討論代理式人工智慧流的關鍵特質與實作機制：

1. 代理人 (Agent) 的核心概念

代理人的概念類似於「旅遊代辦 (Travel Agent)」。當使用者只有模糊目標 (如「想去日本玩」) 時，代理人不會只等指令，而是會透過對話**提供選項 (Options)**、詢問細節，並根據專業知識協助使用者一步步規劃出完整的行程。在系統設計上，這意味著後端伺服器 (B Server) 必須在使用者與大語言模型 (LLM) 之間扮演**操縱與控制**的角色，而非僅僅轉發訊息。

2. 克服 LLM 的技術限制

實作代理式流程時，開發者必須處理 LLM API 的先天限制：

- **管理上下文與 Session**：LLM 的 API 預設是「無狀態的 (Stateless)」，即不具備記憶能力。代理式流程必須建立 **Session 或 Context 的觀念**，由伺服器端存儲對話歷史 (History)，並在每次請求時重新包裝給模型，使其能展現出如同真人的連貫性與貼心感。
- **提示詞工程模板 (Prompt Template)**：使用者的表達往往不夠精確 (Freestyle)，代理人流程會利用預設的**模板 (Template)** 進行「腦補」，將背景知識、資料庫資訊或先前收集的欄位資料填入，生成一份專業且精確的 Prompt 丟給模型，以提升回覆品質。

3. 對話腳本與流程控管

为了避免使用者「天馬行空」導致系統崩潰，代理式流程強調**劇本式規劃**：

- **圖形結構 (Graph)**：將服務流程抽象化為**狀態機 (State Machine)**，定義不同的節點與路徑。當對話進行到特定階段時，系統會預判使用者意圖，並引導其走向預設的終點。
- **UI 元件作為約束**：善用按鈕、快速回覆 (Quick Reply) 或圖文選單來取代自由輸入。這不僅降低使用者的表達負擔，更重要的是能將其行為**限制在開發者可控的範疇內**。

4. 結構化回應與行動觸發 (Call to Action)

代理式流程的終點通常不只是文字，而是具體的動作：

- **輸出格式化 (Output Formatting)**：設計者會要求 LLM 以 **JSON 格式** 回傳結果，以便後端伺服器進行邏輯判斷與型別檢查。
- **觸發後端任務**：透過結構化的回應，系統可以自動執行如「查詢資料庫」、「發送推播」或「更新訂單狀態」等 **Call to Action** 任務。

5. 結合外部資源 (如 RAG 技術)

代理人流程常結合 **RAG (檢索增強生成)** 機制。當模型需要回答專業領域問題 (如保險規範或步道資訊) 時，代理人會先從資料庫中檢索出正確的 FAQ 或文件內容，將其作為背景資訊提供給 LLM，確保回應精準且符合業務規範，而非讓模型憑空捏造。

比喻理解：代理式人工智慧就像是一位資深的**博物館導覽員**。他不會只在門口等遊客問路，而是會主動遞上地圖 (主選單)，詢問你的興趣 (多輪對話)，根據你的回覆帶你走進特定的展廳 (流程節點)，並隨時記住你剛才對哪個作品感興趣 (Context)。即便你問了無關的問題，他也會禮貌地引導你回到導覽路線上，確保你最後能帶著豐富的知識離開 (完成任務)。

LINE 的 reply token 與 push notification 有何區別？

在設計對話式 UI 時，如何利用「主選單」引導使用者？

為何建議在呼叫 LLM API 時使用 JSON 格式回應？

Discuss 輸出格式化校驗.

在對話式系統開發中，**輸出格式化與校驗 (Output Formatting and Validation)** 是確保後端伺服器 (B Server) 能有效理解並處理大語言模型 (LLM) 產出內容的關鍵技術。

以下根據來源內容，詳細討論輸出格式化校驗的必要性、實作工具與流程：

1. 為什麼需要輸出格式化？

大語言模型預設的回應通常是**非結構化**的自由文字 (Freestyle)，甚至包含 Markdown 表格或冗餘的解釋。

- **系統無法自動化**：如果模型只回傳一段文字，後端伺服器將難以解析其中的具體資訊，導致無法觸發後續的程式動作（如存入資料庫或顯示特定的 UI 按鈕）。
- **轉化為「Call to Action」**：格式化的目的在於將模型的分析結果轉換成**結構化回應**，使系統能根據這些數據執行特定的任務 (Call to Action)。

2. 實作機制：強制 JSON 輸出

為了確保輸出的穩定性，開發者需在呼叫模型時進行規格限制：

- **指定回應格式**：在呼叫 API 時，可以使用如 `response_format` 並設定型別為 `json_object`，這能強迫模型輸出有效的 JSON 格式，而非隨機的敘述文字。
- **提示詞約束**：在 Prompt 中明確規定模型必須遵守的格式規範，讓模型知道它必須扮演一個數據提供者，而非單純的聊天夥伴。

3. 校驗工具：利用 Zod 定義 Schema

僅有 JSON 格式尚不足夠，還必須確保資料內容的「正確性」。來源建議使用 **Zod** 等工具來進行校驗：

- **定義資料型別 (Data Schema)**：開發者應預先使用 Zod 定義好資料結構 (Schema)。例如，規定「溫度」欄位必須是數字，「日期」必須符合特定格式。
- **型別檢查與防錯**：透過 Zod 進行 **Validation (驗證)**，可以確保模型回傳的資料完全符合預期。這能避免系統因為收到錯誤的資料型別（例如該給數字卻給了字串）而崩潰，大幅減少錯誤處理的負擔。

4. 完整的處理流程

一個良好的對話式校驗流程應如下運作：

1. **定義**：先利用 Zod 定義好回應的 **Data Schema**。
 2. **請求**：包裝好 Prompt Template，並要求模型以 JSON 格式回覆。
 3. **校驗**：接收到 JSON 後，透過 Zod 檢查是否符合定義好的 Schema。
 4. **行動**：確認資料無誤後，再根據格式化內容執行後續的互動或業務邏輯。
-

比喻理解：輸出格式化校驗就像是銀行的「自動存款機」。如果你隨便投進一張寫著金額的紙條（非結構化文字），機器無法處理；但如果你按照規定的面額放入鈔票，且機器透過感應器核對防偽設計與幣值（格式化與校驗），它就能精準地將金額存入你的帳戶，並觸發後續的簡訊通知或帳單更新。

作業六延長的截止時間為何？

何謂對話式介面中的狀態設計（Stateful Design）？

如何確保大語言模型輸出的資料格式符合預期？