# 自動啟動並持續執行 AI 攻擊學習的框架設計

## 開機自動啟動與持續練習執行

為實現系統開機後即啟動的**持續攻擊學習**，可以設計一個常駐的訓練服務。在部署時將此服務設為開機自動啟動（例如在 Docker Compose EntryPoint 或系統服務中配置）。啟動後，該服務會初始化核心 AI 指揮官（AICommander）或主控控制器（BioNeuronMasterController），並立即進入**持續訓練迴圈**。訓練迴圈基於現有的強化學習架構，反覆執行「載入場景 → 生成攻擊計畫 → 執行計畫 → 收集經驗 → 模型訓練 → 評估改進」的閉環流程[[1]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L32-L40)。透過TrainingOrchestrator提供的批次訓練方法（如 run\_training\_batch），可以按場景反覆多回合訓練，並在每回合完成後自動進行下一回合，直到系統關機為止[[2]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L229-L237)。為避免單次訓練結束後退出，可在批次訓練完成後重新啟動下一批訓練（例如在迴圈中無限調用 run\_training\_batch）。整個過程在**AI 自主模式**下進行，無需人工介入，每個回合自動決策並執行[[3]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_ARCHITECTURE.md#L110-L119)。

為確保不中斷執行與錯誤復原，需在主訓練迴圈中加入健壯的錯誤處理機制。這包含對每次訓練回合的例外處理：一旦發生未預期錯誤，記錄錯誤日誌並安全地跳過或重啟該回合，而非中止整個服務。例如，TrainingOrchestrator.run\_training\_batch內部已對單個回合錯誤進行捕獲並繼續下一回合[[4]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L242-L251)；在最高層級的持續迴圈中也應增加 try/except，確保任何未捕獲異常都不會終止服務，而是記錄後重啟訓練流程。此外，可以利用AICommander.save\_state()在適當時機保存AI狀態，包括知識庫和經驗數據，方便在崩潰後復原[[5]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L473-L482)[[6]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L477-L485)。

為了讓進度**可觀察**，需充分利用日誌系統記錄訓練動態。當前架構中的Plan Executor已使用TraceLogger詳細追蹤攻擊步驟執行[[7]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L6-L14)。建議將日誌輸出定向至檔案或集中日誌服務，並調高詳細級別以記錄關鍵事件。例如，每當開始執行新的攻擊計畫或步驟時，都透過logger輸出提示，目前在執行第幾步及其動作[[8]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L101-L109)。每個訓練回合結束後，記錄成功與否、執行步驟數量和績效指標等摘要[[9]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L160-L168)[[10]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L184-L188)。透過持續的日誌輸出，管理員可以在控制台或日誌檔中**觀察訓練進度**與AI行為，符合需求1中透過log監控的目標。

## AI 練習過程中的 CLI 指令產生機制

在**自主練習**過程中，讓 AI 產生並執行 CLI 指令，以模擬實際系統操作。例如，AI在不同任務階段可能需要：「檢查系統狀態」、「執行安全工具」、「調整環境設定」等。為此，可擴充 AI 引擎的工具集，讓**AI代理**能夠調用系統級操作工具。一種做法是在services/core/aiva\_core/ai\_engine/tools.py中新增對應的工具類，例如：

* **StatusChecker**：封裝現有的系統狀態檢查腳本（如check\_status.ps1），提供方法檢視各服務健康狀態。
* **SystemCommandTool**（或 ShellExecutor）：允許 AI 輸入任意系統指令字串，由程式以子程序方式執行（可設定在受控沙箱環境中運行）。
* **ConfigAdjuster**：提供修改設定檔或調整應用配置的介面，例如更新掃描參數（執行內部API或編輯設定檔）。

透過這些工具類，AI 在練習時可以根據需要產生對應的CLI操作指令。例如：如果AI判斷需要更深入掃描，可調用SystemCommandTool執行nmap掃描；或在訓練過程中定期使用StatusChecker取得服務資源使用情況。這些操作都可以被封裝為標準的Tool接口，供AI決策模組使用。現有架構中的AI Commander會協調多模組並具備執行任務的功能[[11]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L4-L12)[[12]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L24-L31)；我們可在AI決策環中，讓BioNeuron AI模型在產生行動方案時附帶適當的**系統指令步驟**。舉例而言，當AI制定攻擊計畫（AttackPlan）時，除了漏洞掃描步驟，也插入「環境檢查」等步驟，其action可對應如“check\_system\_status”，tool\_type為“system”，parameters指定檢查內容。Plan Executor在執行此步驟時，會識別tool\_type為system，從而調用上述StatusChecker工具執行實際指令。整個過程實現了**AI自主產生並執行CLI命令**的能力，涵蓋從系統狀態查詢、外部工具調用到系統設定調整等操作。

如此設計可滿足需求2：AI會根據任務動態產生相應的CLI指令執行。在強化學習閉環中，每當AI需要與系統互動時，就以這些命令形式表達操作意圖，達到**系統層級操作的自動化**。值得注意的是，在全自動（AI模式）下這些指令可直接執行；若未來結合UI混合模式，仍可將這些AI建議的指令呈現給使用者確認，以平衡安全。

## CLI 指令的 JSON 輸出格式設計

為方便日後在前端面板整合AI操作，可將每條AI執行的指令以結構化JSON格式輸出，包含**命令內容**、**描述**和**參數**欄位。這種格式便於前端解析，將AI行為可視化地展示給使用者。具體設計上，可以在每次AI決策執行操作時，同步生成一份JSON記錄，例如：

{  
 "command": "check\_status",  
 "description": "檢查核心服務運行狀態",  
 "parameters": {  
 "services": ["Core", "Scan", "DB"]  
 }  
}

上述範例表示一條“檢查狀態”的CLI指令，其說明和參數清晰標示。再如AI決定執行一次完整掃描，可能輸出：

{  
 "command": "run\_scan",  
 "description": "對目標 http://example.com 執行完整安全掃描",  
 "parameters": {  
 "target": "http://example.com",  
 "scan\_type": "full"  
 }  
}

這裡command為AI抽象行動（或對應具體工具名稱），parameters列出目標與模式，前端據此也能提供友好的展示或記錄。

實現方面，**AttackPlan/AttackStep**結構本身已定義了類似內容的欄位，如每步的action（描述操作）、tool\_type（工具類型）以及parameters（參數字典）[[13]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/aiva_common/schemas/ai.py#L114-L122)。因此可在Plan Executor執行步驟時，將AttackStep轉換為JSON物件：例如直接使用step.action作為command欄位，step.parameters作為parameters欄位，同時根據action或tool\_type提供一段說明文字作description。為了將此JSON方便地提供給面板，可採取以下措施：

* **日誌輸出**：在執行每個步驟時，logger記錄該JSON字串。例如在PlanExecutor.execute\_plan的迴圈中，於日誌中輸出json.dumps(command\_record)，其中command\_record包含上述結構。這樣前端日誌介面可實時抓取並解析。
* **消息佇列通知**：利用RabbitMQ或WebSocket，在每次AI下達指令時將JSON通過消息傳遞給UI模組，即時更新前端。現有架構支援RabbitMQ發布任務[[7]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L6-L14)，可擴充一個專門的CommandOutput隊列來廣播指令訊息。
* **歷史記錄**：將這些JSON物件追加至AI Commander的command\_history列表（目前該列表儲存了指令執行記錄[[14]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L182-L191)）。這樣不僅前端可查詢，事後分析也能從歷史記錄重建AI行為序列。

透過上述方式，每條AI產生的CLI命令都以結構化JSON對外呈現，包含**動作名稱、用途描述、相關參數**，滿足需求3對面板整合的格式要求。

## 現有架構分析與模組強化建議

**（1）CLI 指令產生與閉環訓練現狀：** 根據最新版本代碼，AIVA架構已具備強化學習的**訓練閉環**設計，但**CLI指令產生**機制仍處於構想階段。從架構文檔可知，AIVA的AI引擎設計了**經驗學習閉環**，流程涵蓋執行、追蹤、對比、學習和改進[[15]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_ARCHITECTURE.md#L369-L373)。實作上，TrainingOrchestrator已經協調了完整的訓練流程（1.場景載入、2.RAG增強計畫、3.執行攻擊、4.收集經驗、5.模型訓練、6.性能評估迭代）[[1]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L32-L40)。每次訓練會執行AttackPlan內多個步驟並迴圈更新AI模型，屬於**閉環強化學習**。不過，目前的代碼主要聚焦於漏洞掃描/攻擊步驟，並未明確實作AI輸出具體系統CLI命令的功能。從開發路線看，「CLI命令列模式」被列為未來多樣化操作模式之一[[16]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/DEVELOPMENT/FUNCTION_MODULE_EXPANSION_ROADMAP.md#L28-L33)，說明CLI整合在計畫中但細節尚未落地。

**（2）模組強化與補全建議：** 為滿足上述需求，應在現有架構基礎上增強以下幾方面： - **持續執行模組初始化與調度**：新增一個專門的自動訓練啟動模組（如AutoTrainService）。它在系統啟動時由主程式呼叫，負責創建 AICommander 實例並調用 run\_training\_session/run\_training\_batch開始循環[[17]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L431-L439)[[18]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L445-L453)。該模組可以啟動一條後台非同步任務，在事件迴圈中執行訓練迴圈，從而不阻塞主線程其它服務（例如API介面）。調度上，可利用 asyncio.create\_task 配合 while True 形成長循環，或定義一個定時計時器定期觸發訓練。由於TrainingOrchestrator.run\_training\_batch本身會遍歷所有場景多次迴圈[[2]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L229-L237)，直接在其外圍再套一層無限循環即可達成“持續到關機”的要求。 - **CLI指令產生閉環**：在AI計畫執行過程中引入CLI命令產生與反饋機制。建議擴充**BioNeuronRAGAgent**的決策輸出，使其不僅產生下一步攻擊動作，也產生對應的CLI指令建議。例如實作一個AttackPlanner子模組，將AI對下一步行動的決策轉換為AttackStep或直接的命令表示。針對沒有現成CLI工具對應的內部行動，可自定義虛擬命令名稱（如“LEARN\_MODEL”表示模型訓練）。每次執行完步驟後，將結果（成功/失敗、輸出資訊）封裝回饋給AI的經驗管理器與知識庫，強化下次決策。這樣閉環中增加了「AI產生命令→執行命令→結果學習」的小循環。 - **必要的 Python 類別補強**：實作上述功能需要增添若干類別和方法。例如，新增ShellCommandTool類（繼承自Tool介面）實現通用CLI執行；新增SystemMonitorTool類提供系統資源/服務狀態查詢。擴充AttackStep資料模型，增加一個可選欄位如cli\_cmd用于存放對應的CLI命令字串模板，或者在現有action基礎上約定其值同時可用作CLI命令名。還可以在PlanExecutor.\_execute\_step中加入對tool\_type的判斷分支，當tool\_type為system或cli時，調用上述Shell工具執行外部命令。最後，在AICommander或BioNeuronMasterController的初始化流程中，確保上述工具類和新模組正確加載與註冊（例如將工具註冊到AI Agent可用工具列表）。這樣，在系統啟動時整個自主訓練框架被搭建完畢，即刻開始運行。

經過上述增強，AIVA將具備開機自動進行持續攻擊訓練的能力：AI會全程自主規劃並執行攻擊步驟，過程中產生所需的系統CLI指令並執行，同時以JSON格式記錄每步命令和結果供前端監控。整個閉環架構將更加完善，AI可在不斷試錯中累積經驗並優化策略，真正達成**不中斷、自我恢復、可監控**的持續攻擊學習目標[[15]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_ARCHITECTURE.md#L369-L373)。各組件協同工作下，系統將在AI自主模式下長時間運行，不斷提高漏洞檢測與攻擊試探的智能化水準。确保在實踐中評估並調整這套框架的參數（如訓練回合數、命令執行沙箱限制等），以兼顧實戰效益與系統穩定性。こう

[[1]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L32-L40) [[2]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L229-L237) [[4]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L242-L251) [[9]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py#L160-L168) training\_orchestrator.py

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/training/training_orchestrator.py>

[[3]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_ARCHITECTURE.md#L110-L119) [[15]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_ARCHITECTURE.md#L369-L373) AI\_ARCHITECTURE.md

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_ARCHITECTURE.md>

[[5]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L473-L482) [[6]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L477-L485) [[11]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L4-L12) [[12]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L24-L31) [[14]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L182-L191) [[17]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L431-L439) [[18]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py#L445-L453) ai\_commander.py

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_commander.py>

[[7]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L6-L14) [[8]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L101-L109) [[10]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py#L184-L188) plan\_executor.py

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/execution/plan_executor.py>

[[13]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/aiva_common/schemas/ai.py#L114-L122) ai.py

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/aiva_common/schemas/ai.py>

[[16]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/DEVELOPMENT/FUNCTION_MODULE_EXPANSION_ROADMAP.md#L28-L33) FUNCTION\_MODULE\_EXPANSION\_ROADMAP.md

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/DEVELOPMENT/FUNCTION_MODULE_EXPANSION_ROADMAP.md>