## BioNeuron 模型 (AI核心大腦)

**目前狀況：** AIVA 的 BioNeuron 模型由 BioNeuronRAGAgent 實現，結合一個約500萬參數的生物啟發神經網路核心，以及防幻覺機制與知識檢索增強[[1]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L140-L148)。代碼中透過 ScalableBioNet 等類別構建了該模型的結構，並使用 PyTorch 等庫進行訓練（Training SOP 文件中示範了建立 ScalableBioNet 模型並統計參數）[[2]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md#L120-L128)。目前此模型已能基本運行並與 RAG 知識庫和強化學習迴路集成，不過**實際可能仍處於雛形**：例如雖有宣稱500萬參數，但實際訓練的效果尚未達到目標（近期基線通過率80%，目標95%+）[[3]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md#L146-L154)。**防幻覺機制**（AntiHallucinationModule）雖已有類別框架，但推測仍較粗略，可能僅作簡單的結果校驗或尚未充分實作。

**問題分析：** BioNeuron 模型作為決策大腦，目前主要挑戰在於**模型複雜度與可靠性**：大規模參數帶來的訓練難度和資源需求較高，而訓練資料與經驗樣本是否足夠支撐如此複雜的模型仍存疑。此外，**幻覺風險**尚未完全消除，模型可能在知識不足時產生不切實際的攻擊步驟。現有實作可能僅部分達成設計，例如抗幻覺可能只是對生成的計畫進行簡單檢驗而未有深度算法支撐。最後，**性能與整合**也是問題：如此大的模型在實時決策時的延遲和資源佔用需要觀察，需確保與其他模組（如 Planner、Tracer 等）協調順暢[[4]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L142-L146)。

**增強建議：**  
- **強化模型訓練與精度：** 建議增加高品質的訓練數據和經驗樣本，以提高BioNeuron網路決策精度。可引入更多模擬攻擊場景或歷史滲透測試數據來豐富模型學習。此外，利用強化學習時應調整獎勵機制，確保模型逐步逼近目標性能（例如對比 PlanComparator 計算的獎勵得分進行梯度更新）[[5]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L48-L56)[[6]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L144-L151)。  
- **完善抗幻覺機制：** 為防止模型輸出不合理步驟，可在 AntiHallucinationModule 中實作**知識校驗**：例如將模型生成的攻擊步驟與知識庫比對，若發現與已知漏洞原理不符或超出目標系統範圍，則予以修正或濾除。這可透過向量檢索相關知識 (KnowledgeBase.search) 檢驗模型建議的可信度來完成[[7]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L92-L101)[[8]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L102-L110)。保持這一模組的實作簡潔（純 Python 判斷邏輯），以符合現有架構風格。  
- **性能優化：** 在不引入新框架前提下，可考慮利用 NumPy/向量化運算優化部分計算，以及透過現有的 OptimizedBioSpikingLayer 等類別進行**微調**[[9]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_engine/__init__.py#L14-L21)。例如在推理階段使用預先批次計算、簡化激活函數等手段，加速模型決策。同時確保模型僅在需要時調用，減少不必要的佔用。

**（選用）補充實作：** 如需對抗幻覺機制的具體實作，可考慮增補如下模擬代碼，以現有風格對接：

# AntiHallucinationModule 補充：基於知識庫驗證模型輸出  
class AntiHallucinationModule:  
 def \_\_init\_\_(self, knowledge\_base):  
 self.knowledge\_base = knowledge\_base  
 def validate\_plan(self, attack\_plan):  
 """驗證整個攻擊計畫，移除明顯不合理的步驟"""  
 refined\_steps = []  
 for step in attack\_plan.steps:  
 # 利用知識庫搜尋與步驟相關的知識條目  
 results = self.knowledge\_base.search(step.description)  
 if results: # 知識庫中有相關內容  
 refined\_steps.append(step)  
 else:  
 # 若知識庫無相關條目，標記該步驟可能是幻覺  
 print(f"[AntiHallucination] 移除可疑步驟: {step.description}")  
 attack\_plan.steps = refined\_steps  
 return attack\_plan

上述補充代碼展示了一種可能的抗幻覺處理：對每個攻擊步驟在知識庫中尋找相關支持，無相關知識的步驟則被視為幻覺而移除。這種實作遵循了現有架構的風格（使用簡單的類和方法調用），並充分利用了 KnowledgeBase 的檢索能力。實際部署時，可將此模組融入 BioNeuronRAGAgent 在生成計畫後的流程中，以提高決策合理性。

## DecisionAgent (決策代理)

**目前狀況：** 決策代理負責基於AI模型與環境資訊，制定攻擊策略和行動順序。在現有架構中，DecisionAgent 的職能主要由 AICommander 類承擔。AICommander 提供了一系列決策相關的方法，例如 execute\_command() 統一入口，內部調用 \_plan\_attack() 生成攻擊計畫、\_make\_strategy\_decision() 決定策略、\_detect\_vulnerabilities() 執行漏洞檢測、以及 \_learn\_from\_experience() 觸發經驗學習等[[10]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L148-L154)。這表明系統已具備基本的決策流程框架。目前AICommander與BioNeuron核心、Plan執行器等模組皆有串聯，**應能在模擬環境下跑通流程**（如 demo\_bio\_neuron\_master.py 演示完整流程）。然而，決策代理的具體決策邏輯可能較為簡單：例如 \_make\_strategy\_decision 可能僅以固定規則或條件分支實現，尚未充分利用模型預測或風險評估結果。決策代理目前應能運行，但**智能程度有限**，許多策略判斷可能仍標記為 TODO 或以簡單佔位符實現。

**問題分析：** 現階段DecisionAgent面臨的主要問題有：  
- **策略動態性不足：** 可能缺乏根據掃描結果及風險水平動態調整策略的能力。例如，若初步掃描未發現高嚴重漏洞，代理應調整策略深度或範圍，但目前邏輯未必涵蓋此類動態變化。  
- **缺乏經驗驅動決策：** 雖然架構上接入了經驗學習迴路，但代理在每次決策時是否真正利用過往經驗仍未明確。若 \_learn\_from\_experience() 只是執行完後才學習，則在決策當下未充分引用過去案例。  
- **工具/步驟選擇有限：** 決策代理需要在多種攻擊技術與工具之間做選擇。目前可能僅根據預定策略（例如先SAST後DAST）執行，沒有依據目標環境的反饋自適應選擇最佳手段。舉例而言，代理或固定嘗試一系列漏洞掃描，而非根據系統反應調整掃描順序。

**增強建議：**  
- **引入風險評估決策：** 結合 BioNeuronMasterController.\_assess\_risk() 的結果，在 \_make\_strategy\_decision 中納入風險級別判斷。例如，高風險操作須更加保守（請求使用者確認或改用被動檢測），低風險操作則可自動執行[[11]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L162-L170)。透過在決策代理中閱讀 MasterController 的風險評估輸出，實現策略模式切換（UI/AI/Chat/Hybrid 模式）的自動化**智能平衡**[[12]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L156-L165)[[13]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L272-L278)。  
- **強化經驗利用：** 在決策階段即引用 ExperienceManager 中的高品質樣本。例如，可建立一個**策略推薦系統**：當遇到某類型目標或漏洞時，搜尋過去類似條件下成功率高的攻擊步驟，作為當前決策的參考。這可通過在 \_plan\_attack() 中調用 KnowledgeBase.get\_top\_entries(Experience) 或 ExperienceManager.get\_high\_quality\_samples() 實現[[14]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L62-L70)[[15]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L100-L105)。保持實作為簡單的列表過濾/排序，符合項目既有風格。  
- **細化工具選擇邏輯：** 利用現有模組的能力（如 Planner 的 ToolSelector），依據掃描目標類型和先前步驟結果動態選擇下一步的工具模組。例如，可基於MITRE ATT&CK技術類型決定是進行網應用掃描還是社工程測試[[16]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L12-L20)。建議在 \_make\_strategy\_decision 或 \_plan\_attack 中加入**簡單的規則引擎**：例如：「若發現SQL注入跡象則優先深入測試資料庫相關漏洞」，「若多次嘗試未果則切換攻擊途徑」等。這些規則可由枚舉或字典配置驅動，避免硬編碼，同時易於日後擴充。

*(註：AICommander 已有基本框架，增強時務必遵循其物件導向結構。以上建議透過調整/增加方法內邏輯即可實現，無需引入新的外部AI框架，以維持原專案風格。)*

**（選用）補充實作：** 以下是一段可能的 \_make\_strategy\_decision 內部邏輯範例，以展示如何根據先前結果調整策略：

# AICommander 類中的策略決策範例實作  
def \_make\_strategy\_decision(self, context: AttackContext) -> Decision:  
 """  
 根據目前上下文（包含先前步驟結果、風險評估等）選擇下一步決策。  
 """  
 high\_risk = context.risk\_level == "HIGH"  
 # 1. 若高風險且非UI模式，切換至UI模式要求確認  
 if high\_risk and self.mode != OperationMode.UI:  
 return Decision(action="SWITCH\_MODE", params={"mode": OperationMode.UI})  
 # 2. 根據已發現的漏洞線索調整策略  
 if "sql\_injection" in context.discovered\_vulns:  
 # 已發現SQLi跡象，深入測試SQLi相關漏洞  
 return Decision(action="RUN\_TOOL", params={"tool": "AdvancedSQLiScanner"})  
 if context.attempts\_without\_success > 3:  
 # 多次嘗試無成果，切換攻擊途徑，例如嘗試XSS  
 return Decision(action="RUN\_TOOL", params={"tool": "XSSDetector"})  
 # 3. 預設: 繼續下一步預定計畫  
 return Decision(action="PROCEED\_DEFAULT")

上述代碼模擬了DecisionAgent在不同情況下的策略調整：高風險時強制人工確認，多次失敗後改變攻擊手段等。這種實作使用簡單的條件判斷和內置類型，符合現有純Python的架構風格。同時透過 context 封裝決策所需資訊，保持模組介面清晰。實際代碼中，context 和 Decision 類需根據專案的 Schema/類別定義進行替換或擴充。總體而言，上述實作展示了**無須引入大型模型**即可加強代理決策靈活性的方法。

## 執行計畫 / Orchestrator (執行計畫生成與編排)

**目前狀況：** AIVA 平台已建立起完整的攻擊計畫生成與執行機制。**計畫生成**方面，系統使用 AttackPlan / AttackStep 資料結構來描述多步驟攻擊流程[[17]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L10-L18)。在AI決策引擎生成攻擊意圖後，Orchestrator（如 planner/orchestrator.py）會將其轉化為可執行的 AttackPlan，內含具體步驟（每步對應一個工具或攻擊技術）。**計畫執行**方面，由 PlanExecutor 負責逐步執行攻擊計畫[[18]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L26-L34)。根據代碼清單，PlanExecutor.execute\_plan() 會循序執行每個 AttackStep，並內建依賴檢查、錯誤處理與重試機制，以確保執行的魯棒性[[19]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L28-L35)。同時，TraceLogger 會對整個執行過程進行記錄[[20]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L34-L39)。目前計畫編排與執行模組應能**正常運作**：計畫結構和執行器代碼均已完成，整體流程在核心模組連接測試中標記為「Scan Module 完全正常」「Core Module 基本正常」（僅路徑需確認）[[21]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/final_report.py#L50-L58)。這表示從計畫下達到掃描工具執行，再到結果收集的鏈路基本打通。

**問題分析：** 現有執行計畫/編排模組可能存在以下可改進空間：  
- **計畫生成智能度有限：** Orchestrator 將AI建議轉化為具體步驟時，可能主要依賴預定模板或簡單規則，尚未充分利用AI模型實時調整順序。攻擊計畫可能偏靜態，對目標的實際防禦狀況自適應不足。  
- **多階段依賴處理：** PlanExecutor 已實現依賴關係檢查 \_check\_dependencies()[[22]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L30-L34)，但需要確定 AttackStep 間依賴定義完備。例如某步需要前一步取得特定憑證才能運行，這類關係在計畫定義中是否被標註？若否，則執行器雖有檢查函數也無法真正生效。  
- **錯誤恢復與動態調整：** 當某步執行失敗時，目前 \_handle\_step\_error() 和 \_retry\_step() 機制提供了重試功能[[22]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L30-L34)。但若多次重試仍失敗，系統是否會動態修改計畫（例如略過此步或選擇其他途徑）？目前可能僅記錄錯誤但不會重新規劃。這限制了攻擊計畫的靈活性。

**增強建議：**  
- **動態計畫調整：** 強化 Orchestrator 使其能根據執行時反饋調整後續計畫。例如可在 PlanExecutor 偵測到某關鍵步驟連續失敗時，透過調用 Orchestrator 的介面生成替代步驟插入計畫（如改用另一種攻擊技術）。實現上，可增加一個如 Orchestrator.adjust\_plan(failed\_step) 方法，在 \_handle\_step\_error 中調用，以保持架構解耦。這樣當某步驟多次重試無果時，系統能嘗試替換其他策略而非終止計畫執行。  
- **強化依賴與條件判斷：** 確認 AttackStep 結構中包含**前置條件/後置結果**定義。例如在 AttackStep 增加字段如 prerequisite 或 depends\_on（引用另一步驟ID），讓 Orchestrator 在組織計畫時明確標註依賴。配合 PlanExecutor 現有的 \_check\_dependencies()，在執行前驗證先決條件是否滿足，未滿足則跳過或延後執行[[22]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L30-L34)。若目前 AttackPlan 未實現這種依賴關係標記，建議加以擴充並在 Orchestrator 產生計畫時賦值，以充分利用既有的依賴檢查邏輯。  
- **計畫優化與多路徑策劃：** 引入簡單的計畫優化步驟：在 Orchestrator 生成初步計畫後，使用 PlanComparator 或 RiskAssessmentEngine 對模擬執行結果進行評估比較，選擇預期收益最高的計畫路徑[[23]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L120-L128)[[24]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L340-L348)。雖然完整的多路徑搜索會增加複雜度，但可採用近似方法，如對關鍵步驟嘗試不同組合評分。這種優化可在不引入重型框架下由現有模組協作完成。例如，對兩套備選計畫用 PlanComparator 計算預期成功率，選擇分數較高者執行。

*(上述建議旨在保持模組原有的物件化設計，透過增添方法或擴充資料結構來提高靈活性，而非引入全新架構。)*

**（選用）補充實作：** 以下代碼片段展示了如何在 PlanExecutor 中結合 Orchestrator 實現動態計畫調整：

# 假設 Orchestrator 實現了一個調整計畫的方法  
from services.core.aiva\_core.planner import orchestrator  
  
class PlanExecutor:  
 def \_handle\_step\_error(self, step, error):  
 # ... 現有錯誤記錄處理 ...  
 if step.retry\_count >= self.max\_retries:  
 # 請 Orchestrator 嘗試生成替代步驟  
 alt\_step = orchestrator.generate\_alternative(step)  
 if alt\_step:  
 print(f"[PlanExecutor] 用替代步驟替換失敗步驟: {step} -> {alt\_step}")  
 return alt\_step # 返回新步驟以替換當前步驟  
 return None # 無替代，維持原流程

上述伪碼中，當某步驟重試多次失敗後，調用 Orchestrator 的 generate\_alternative 產生一個替代的 AttackStep。PlanExecutor 可將此替代步驟插入剩餘計畫序列中繼續執行。這樣無需中斷整個攻擊計畫流程。此實作風格與原架構一致（均為 Python 模組間調用），透過少量新方法的引入即可顯著提升計畫執行的成功率和靈活性。

## 任務執行器 (TaskExecutor)

**目前狀況：** TaskExecutor 模組旨在執行單個攻擊任務或步驟。代碼結構上存在 execution\_tracer/task\_executor.py 檔案[[25]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L143-L151)，但在架構文檔和流程圖中並未強調此模組，暗示其目前功能可能較弱或尚未實質投入使用。在當前設計中，**任務的執行主要由其他機制承擔**：PlanExecutor 將計畫步驟發送給 TaskDispatcher，經由 RabbitMQ 對應的功能模組（如各漏洞掃描worker）來執行，再由 ResultCollector 收集結果[[26]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L318-L325)[[27]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L322-L327)。因此，TaskExecutor 可能僅在某些情境下使用（例如無消息隊列時的本地執行、或訓練模式下模擬執行）。目前推測 TaskExecutor **尚未完整實作**：可能僅包含一個類或函數框架，內部為TODO，或者僅作為執行追蹤的佔位元。換言之，在默認部署中並不直接由TaskExecutor執行具體攻擊步驟。

**問題分析：** TaskExecutor 模組存在的問題在於：  
- **未充分集成：** 現有架構主要依賴消息系統調度外部工具執行任務，TaskExecutor 的角色不明顯。如果未來需要在無MQ情況下運行（如單機測試），TaskExecutor 的缺失會成為短板。  
- **模擬與真實執行割裂：** 在強化學習訓練時，也許需要一個模擬環境執行攻擊步驟並產生獎勵。若無TaskExecutor，則難以在不連接真實工具的情況下模擬步驟效果。  
- **代碼風格不一致風險：** 若日後補上TaskExecutor，需注意與PlanExecutor、TaskDispatcher的介面協調。目前可能缺少這部分設計，例如如何從PlanExecutor切換至直接由TaskExecutor執行等。

**增強建議：**  
- **完善 TaskExecutor 實作：** 建議為 TaskExecutor 補充完整功能，使其能在不使用RabbitMQ時直接執行任務。可透過查找 AttackStep 所對應的工具或函式，直接在本地調用相應的檢測函數。例如，對於 Python 實現的功能模組（如 IDOR、PostEx 等），TaskExecutor 可透過導入其 worker.py 或核心函數直接執行。[[28]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L279-L288)[[29]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L244-L252)這需要在 AttackStep 中有標識使用何種工具模組的字段（如枚舉或類型），TaskExecutor 根據此路由到正確的函數。  
- **支援訓練模式模擬：** 在強化學習訓練或測試場景下，引入**模擬執行**選項。TaskExecutor 可內建一個 *simulate* 開關，當啟用時，不真正發出網路請求或危害操作，而是依據預定邏輯產生結果。例如，可根據AttackStep類型隨機或根據難度產生「成功/失敗」結果，並返回給 TraceLogger 和 ExperienceManager 用於更新。[[26]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L318-L325)這樣可在無需大量真實目標的情況下訓練AI決策。  
- **與現有調度協調：** 保持 TaskExecutor 與 TaskDispatcher 的介面一致性。可考慮讓 PlanExecutor 在初始化時根據配置決定**執行路徑**：若偵測到 RabbitMQ 可用，則使用 TaskDispatcher 發送任務；否則調用 TaskExecutor 直接本地執行。這種設計確保變更最小化：通過配置切換，而非改動主要流程。同時，在 TaskExecutor 執行完任務後，應手動呼叫 ResultCollector 彙總結果，以模擬消息系統的行為，保持後續流程一致。

**補充實作：** 以下提供一個簡化的 TaskExecutor 實現範例，以展示其本地執行與調度邏輯：

# 簡化的 TaskExecutor 類實作  
from services.core.aiva\_core.messaging.task\_dispatcher import TaskDispatcher  
from services.core.aiva\_core.messaging.result\_collector import ResultCollector  
  
class TaskExecutor:  
 def \_\_init\_\_(self, use\_message\_queue: bool = True):  
 self.use\_message\_queue = use\_message\_queue  
 if use\_message\_queue:  
 self.dispatcher = TaskDispatcher() # 使用MQ調度  
 else:  
 self.dispatcher = None  
 self.result\_collector = ResultCollector()  
  
 def execute\_task(self, attack\_step):  
 """  
 執行單個 AttackStep。若配置使用MQ，則透過TaskDispatcher發送；  
 否則直接調用對應功能模組執行。  
 """  
 if self.use\_message\_queue:  
 # 經由訊息佇列的標準流程  
 self.dispatcher.dispatch(attack\_step)  
 else:  
 # 本地直接執行模式  
 tool = attack\_step.tool # 假設 AttackStep 有 tool 欄位標明所需工具  
 result = None  
 if tool == "SQLI":  
 # 導入並呼叫本地 SQLi 檢測模組  
 from services.function.function\_sqli.aiva\_func\_sqli import worker  
 result = worker.run(attack\_step.payload)  
 elif tool == "XSS":  
 from services.function.function\_xss.aiva\_func\_xss import worker  
 result = worker.run(attack\_step.payload)  
 # ... 其他工具類型 ...  
 else:  
 print(f"[TaskExecutor] 未知的工具類型: {tool}")  
 # 將結果提交給結果收集器  
 if result is not None:  
 self.result\_collector.collect(attack\_step.id, result)

上述實作展示了 TaskExecutor 在兩種模式下的行為：當使用消息隊列時，直接委派給現有的 TaskDispatcher；當不使用時，則依據 AttackStep 所包含的工具類型，本地調用對應的功能模組代碼（此處假設每個模組提供一個簡單的 worker.run 接口）。執行完畢後，通過 ResultCollector.collect() 提交結果，使後續流程（如TraceLogger記錄、ExperienceManager更新等）無縫銜接。此設計遵循原有的結構和風格，使用條件判斷替代框架，引入 minimal 的改動即可讓 TaskExecutor 發揮作用。需要注意實際代碼中 AttackStep 結構和各功能模組接口名稱，需與專案保持一致，上例僅作概念示範。

## 經驗管理 / 訓練管理 (Experience & Training Manager)

**目前狀況：** AIVA 在**經驗積累**與**模型訓練**方面已搭建明確的模組架構。ExperienceManager 負責在執行過程中收集並管理經驗樣本，提供添加樣本、篩選高質量樣本、更新標註以及導出訓練集等功能[[30]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L63-L71)。而 ModelTrainer 則承擔訓練AI模型的工作，具備監督學習 (train\_supervised)、強化學習 (train\_reinforcement)、模型評估 (evaluate\_model) 和部署 (deploy\_model) 等方法[[31]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L72-L77)。從代碼與文檔來看，這兩個模組應該**基本完成實作並可運行**：ExperienceManager 能將 TraceLogger 記錄的執行過程轉化為經驗樣本存入經驗庫，並能按照質量評分提取出優質樣本供訓練使用[[32]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L66-L70)；ModelTrainer 則可以基於收集的資料對 BioNeuron 模型進行迭代優化（監督學習用於初始訓練，強化學習閉環用於持續改進）[[33]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L72-L76)。實施進度報告顯示這部分功能已標記為完成✅，在最終完成報告中也無明顯TODO項。因此，目前經驗/訓練管理模組能正常運作，支撐整個AI閉環學習。然後，現階段訓練效果距離理想仍有差距（如前述模型通過率僅80%），顯示改進空間。

**問題分析：**  
- **樣本品質評估單一：** ExperienceManager 的 \_calculate\_quality\_score() 用於給每個樣本打分[[34]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L67-L71)。若當前評分機制過於簡單（例如僅根據漏洞是否發現或步驟成功與否），可能無法全面衡量樣本價值。例如，一些失敗的嘗試也許包含重要訊息，可用於改進決策，但可能在簡單評分中被忽略。  
- **樣本多樣性與遺忘問題：** 經驗庫隨著時間增大，如何避免模型只記住高分樣本而忽視多樣性？目前 get\_high\_quality\_samples() 著重選出優質樣本，但可能導致訓練集中於相似場景，缺乏對未見情況的泛化。反之亦然，如果缺乏機制淘汰陳舊或低相關性的樣本，經驗庫膨脹會影響訓練效率。  
- **強化學習細節挑戰：** ModelTrainer 的 train\_reinforcement() 需要設計合理的強化學習演算法。若目前僅粗略實現，例如每回合根據 PlanComparator 分數調整一次模型參數，可能收斂緩慢或不穩定。強化學習還涉及**探索與利用平衡**，現有架構中未明示探索策略，可能預設隨機或ε-greedy，但具體效果未知。

**增強建議：**  
- **改進樣本評分機制：** 在 ExperienceManager 中融合更多維度計算質量分數。例如可考慮：漏洞嚴重性（根據CVSS分數加權）[[17]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L10-L18)、步驟效率（是否以最少步驟達成目的）、以及創新性（是否為以前未出現的策略）。可以將這些因子線性組合到 \_calculate\_quality\_score() 中，保持實現簡單的同時提高評分區分度。這樣高質量樣本不再僅限於「成功樣本」，也包含對未來決策有啟發性的失敗樣本。  
- **維持經驗多樣性：** 在 get\_high\_quality\_samples() 選取樣本時，引入一定隨機性或分組策略。例如先按場景類型分組，再在每組取高分樣本，最後合併，以確保不同類型攻擊經驗都有覆蓋。此外，可定期對經驗庫做**遺忘策略**：淘汰最舊或質量評分長期偏低的部分樣本，控制庫大小，防止模型訓練被陳舊經驗干擾。這些策略可通過簡單的清單切片和排序實現，無需複雜演算法。  
- **強化學習參數與策略調優：** 在 ModelTrainer 中，細化強化學習的實作。可考慮引入**折扣因子γ**來考量長期獎勵，確保模型著眼於整體計畫成功而非單步得失。同時，實現 ε-貪婪 (ε-greedy) 策略在訓練時讓AI有一定機率嘗試新策略，而不只是沿用既有經驗，提升策略探索性。由於未引入外部框架，可自行在訓練迴圈中加入epsilon值，對決策採樣進行控制。在監督學習方面，確保資料標註充分利使用（如利用 ExperienceManager 的標註更新功能提高標準答案品質）。  
- **評估與反饋閉環：** 加強 evaluate\_model() 部分，使其不只產出整體準確率，還能分析各類場景下模型表現，將此反饋給 ExperienceManager/KnowledgeBase。比如發現模型對某類漏洞策略成功率低，可標記需要更多該類經驗。在不引入新框架下，可透過記錄不同類型攻擊的成功率並生成簡單報表[[35]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md#L144-L152)實現，指導下一階段經驗收集重點。

*(以上建議保持使用既有的 Python/Numpy 手段來實現，避免引入複雜的第三方強化學習庫，以維持專案風格。)*

**（選用）補充實作：** 下面是一段示意代碼，展示如何在 ExperienceManager 中實現多因素的樣本質量評分：

# ExperienceManager 類中的質量評分函數示意  
def \_calculate\_quality\_score(self, sample: ExperienceSample) -> float:  
 score = 0.0  
 # 因子1: 是否成功完成任務  
 if sample.outcome == "success":  
 score += 50  
 # 因子2: 攻擊步驟數量（越少越好）  
 score += max(0, 20 - sample.steps) # 用步驟數的反比作為分  
 # 因子3: 漏洞嚴重性權重  
 score += sample.impact\_score \* 10 # 例如 CVSS 基本分\*10  
 # 因子4: 創新性（經驗庫中該策略出現頻率的倒數）  
 rarity = 1.0 / (1 + self.strategy\_count.get(sample.strategy\_signature, 0))  
 score += rarity \* 20  
 return score

此實作綜合考慮了成功與否、效率、影響力和稀有度等要素來給經驗樣本打分。strategy\_signature 可視為對樣本中所採用策略的唯一標識，用於衡量該策略的新穎性。代碼中使用了純 Python 計算和簡單算術，符合項目對於程式碼風格的要求。儘管這只是示例，實際實現時可根據專案需求調整各因子的權重。透過這種方式，ExperienceManager 能夠選出更能豐富模型決策能力的經驗集，進一步提升訓練效果。

## 僅強化工具功能模組對整體能力的影響評估

單純加強漏洞掃描、漏洞檢測等**功能模組**的能力，對 AI 整體攻擊與決策能力的提升是有幫助但**相對有限**的。功能模組提升主要體現在**偵測深度與準確性**上：更強的 SAST/DAST/漏洞掃描引擎將產生更豐富、更可靠的安全資訊[[36]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/README.md#L67-L73)。這些資訊確實是 AI 決策的重要依據，因而**工具模組越強，AI 所掌握的環境狀態就越準確**，可以發現更多攻擊途徑。然而，AI 的最終攻擊效果還取決於其如何利用這些資訊進行策略規劃和動作決策。若核心決策代理和學習模組未相應加強，AI 可能無法充分利用新增的漏洞信息。換言之，**資訊輸入提升需要配合決策輸出優化**才能真正轉化為整體能力的提高。

從架構上看，AIVA 的 Core AI 與 Scan/Function 模組通過消息與契約交互[[37]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/README.md#L187-L194)。只加強後端掃描工具（比如提高SQLi檢測的靈敏度），會讓 Core AI 獲取更多漏洞發現結果，短期內決策命中率可能提升，因為之前可能漏報的漏洞現在能被發現。但如果 Core AI (BioNeuron + DecisionAgent) 沒有升級，它**仍以原有邏輯行動**：不會主動設計新策略去觸發改進後的掃描能力。例如，工具變強後或許可以檢測更隱蔽的XSS，但AI若未學會在適當時機啟用XSS掃描，則增強的工具價值無法充分發揮。再者，強化工具主要影響已知攻擊技術的偵察深度，對 AI **探索新攻擊面**的能力沒有直接幫助。AI 的創新策略來自於強化學習和經驗積累，而非工具本身。因此，僅提升功能模組帶來的是**漸進式改善**（更多漏洞被識別，降低誤報漏報），但要實現突破性的整體攻擊決策提升，仍需同步改進AI核心模組的策略生成和學習能力。

綜上所述，強化功能模組對整體能力有正面影響，但效果有限且可能受核心決策能力瓶頸制約。**最佳方案**是**核心AI**與**功能模組**協同增強：一方面提升掃描工具品質，提供更好的環境反饋；另一方面升級BioNeuron決策網路和DecisionAgent，使其能夠針對更豐富的資訊做出更明智的攻擊行動。只有這兩方面相輔相成，才能充分發揮AIVA架構中各模組的潛能，真正大幅提升整體攻擊與決策能力。

[[1]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L140-L148) [[4]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L142-L146) [[5]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L48-L56) [[6]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L144-L151) [[7]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L92-L101) [[8]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L102-L110) [[10]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L148-L154) [[11]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L162-L170) [[12]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L156-L165) [[13]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L272-L278) [[14]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L62-L70) [[15]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L100-L105) [[16]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L12-L20) [[17]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L10-L18) [[18]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L26-L34) [[19]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L28-L35) [[20]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L34-L39) [[22]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L30-L34) [[26]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L318-L325) [[27]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L322-L327) [[30]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L63-L71) [[31]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L72-L77) [[32]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L66-L70) [[33]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L72-L76) [[34]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/docs/ARCHITECTURE/AI_SYSTEM_OVERVIEW.md#L67-L71) AI\_SYSTEM\_OVERVIEW.md

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

[[2]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md#L120-L128) [[3]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md#L146-L154) [[35]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md#L144-L152) AI\_TRAINING\_SOP.md

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/AI_TRAINING_SOP.md>

[[9]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/services/core/aiva_core/ai_engine/__init__.py#L14-L21) \_\_init\_\_.py

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

[[21]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/final_report.py#L50-L58) final\_report.py

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/final_report.py>

[[23]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L120-L128) [[24]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L340-L348) [[25]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L143-L151) [[28]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L279-L288) [[29]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt#L244-L252) project\_tree\_latest.txt

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/_out/project_tree_latest.txt>

[[36]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/README.md#L67-L73) [[37]](https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/README.md#L187-L194) README.md

<https://github.com/kyle0527/AIVA/blob/91a5bebb9173886fec580c3b39f22254b4321919/README.md>