# AIVA 多語言程式處理能力的實現研究

## AI 模型的多語言程式碼理解與生成能力

要讓 AIVA 的 AI 模組支援多種主流程式語言，首先需要採用或訓練能處理多語言程式碼的 AI 模型。目前的主流大型語言模型（LLM）已展現「**語言不可知性**（Language Agnosticism）」，能理解並產生多種程式語言的程式碼[[1]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=8,language%20development%20environment)。例如 OpenAI 的 Codex 模型（GitHub Copilot 背後技術）能產生多種語言的程式碼（以 Python 最擅長），並可根據龐大 GitHub 程式庫訓練得來的知識，協助**程式片段自動完成、跨語言轉譯、程式碼解說與重構**等功能[[2]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=OpenAI%20Codex%2C%20a%20descendant%20of,with%20Python%20being%20its%20strongest)。Meta 的 Code Llama 模型亦聲稱支援 Python、C++、Java、JavaScript/TypeScript、C#、Bash 等多種流行語言[[3]](https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7#:~:text=language%20prompts%20%28e,and%20Bash)。開源的 StarCoder 模型更是專為多語言編碼設計，擁有 150 億參數，訓練於 6.4 TB 的大型開源程式資料集（The Stack）上，涵蓋 **384 種程式語言**，可以處理超過 80 種語言的編碼任務[[4]](https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding#:~:text=Hugging%20Face%20and%20ServiceNow%20have,4)。因此，在 AI 引擎層面，**選用具備多語言編碼知識的預訓練模型（如 Code Llama、StarCoder 等）並進行微調**，是實現多語言程式碼理解與生成的基礎。

為進一步提升模型對各語言的精通度，可針對不同語言進行**專門調優**或補充資料。例如 Code Llama 就提供了針對 Python 額外微調的版本，以加強其在 Python 上的能力[[5]](https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7#:~:text=Additionally%2C%20Meta%20has%20further%20fine,and%20Code%20Llama%20%E2%80%94%20Instruct)。我們也可蒐集各語言的高品質範例程式碼、重構案例與安全漏洞案例作為訓練/調適資料，讓模型學會不同語言的最佳實踐與安全模式。LLM 經適當調校後，應能夠針對用戶提供的多語言程式碼進行分析理解，並生成對應語言的**操作建議**，包括：優化效能（如演算法改進、減少冗餘）、重構代碼（提高可讀性與維護性）、以及**安全分析**（發現潛在漏洞並提供修補建議）。事實上，現有的程式碼生成模型已展現這些能力的雛形——例如 Codex 不僅能寫代碼，還能協助進行程式碼**重構與錯誤修復**[[6]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=It%20was%20trained%20on%20a,to%20generating%20data%20science%20charts)；許多 LLM 對話機器人（如 ChatGPT/GPT-4）在安全領域已被嘗試用於**偵測漏洞**或給出安全建議。因此，透過**多語言的大規模程式碼語料訓練**結合**指令微調**（讓模型習慣用戶指令進行分析建議），AI 模組可以具備對 **Python、JavaScript、Go、Java、C# 等各語言**程式碼的理解與生成能力[[3]](https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7#:~:text=language%20prompts%20%28e,and%20Bash)。模型將能針對特定語言的語法與特性提供量身定制的建議（例如 Python 的內建函式優化、Java 記憶體管理最佳實踐、C# LINQ優化、Go 并發模式建議等），並檢測各語言常見的安全漏洞模式。必要時，我們還可採用**檢索增強生成（RAG）**：將各語言的安全指南、框架最佳實踐等資料納入知識庫，在模型生成建議時提供參考，以提高準確性[[7]](https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/README.md#L62-L70)（AIVA 架構中已提及向量資料庫與知識增強）。總之，AI 引擎應以通用的大型程式模型為核心，結合少量語言專精微調，使其能“一腦多用”，精通多門語言的程式碼理解與產生。

## 多語言程式碼的掃描、分析與操控機制

**程式本體（平台）**需要有能力自動掃描和分析不同語言的目標程式碼，並與之互動（例如讀取其結構、執行部分程式片段）甚至進行修改操控。為達成這點，系統需要建立**統一的程式碼表示和分析流程**，同時針對每種語言的特性採取對應的處理策略。以下是具體的設計考量：

* **AST 解析與靜態分析**：採用抽象語法樹（AST）技術是跨語言分析的關鍵手段。建議引入 **Tree-sitter** 這類多語言解析框架，因為 Tree-sitter 已有社群維護的 **40+ 種語言語法分析器**，可不用從零開始為每種語言寫 parser[[8]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Tree,chose%20it%20for%20several%20reasons)。透過 Tree-sitter，可將不同語言的原始碼統一解析成語法樹結構，讓系統理解程式的語法組成而非僅是純文字。與傳統純文字或正則比對的分析方式不同，AST 能提供**語義結構層面的深度理解**：例如函式定義、變數引用、控制流等[[9]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Why%20AST%20Parsing%20Matters)[[10]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=A%20FunctionDeclaration%20node)。Dropstone 平台即使用 Tree-sitter 來支援 **40 多種語言**的程式碼語法樹解析，從而實現對程式碼的精確分析和智慧建議[[11]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=How%20Dropstone%20uses%20Tree,accurate%20analysis%20and%20intelligent%20suggestions)。AST 為基礎的分析讓我們可以進行**統一的圖析**（如建立調用關係圖、變數依賴關係等），並針對語法模式進行查詢匹配，例如利用 Tree-sitter 提供的查詢語言來搜尋特定的程式結構模式[[12]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Query%20system.%20Tree,for%20detecting%20specific%20code%20constructs)。系統應該實現一個**多語言AST分析模組**，在後端針對上傳的程式碼自動選擇正確的 parser 生成 AST。之後再結合**語言特定的語意規則**進行進一步分析：比如解析 Java 時識別類和介面的繼承關係，解析 Python 時處理縮排作用域、裝飾器等特殊語法[[13]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=In%20TypeScript%2C%20we%20handle%20type,to%20provide%20accurate%20type%20information)。透過**語言無關的 AST 結構 + 語言相關的語意擴充**，可實現對各語言源碼的靜態分析支援。
* **多語言靜態掃描與安全檢查**：在 AST 基礎上，可以套用通用或語言特定的**靜態分析規則**來發現問題。已有許多開源工具提供多語言的靜態掃描能力，可作為參考或整合。例如 **Semgrep** 是一種快速、開源的靜態分析工具，使用類似程式碼的模式來查找漏洞或違規代碼，**支援超過 30 種語言**[[14]](https://github.com/semgrep/semgrep#:~:text=Semgrep%20is%20a%20fast%2C%20open,as%20part%20of%20CI%2FCD%20workflows)。Semgrep 的優勢在於其規則可以跨語言使用，只要語法模式匹配即可，這對安全漏洞（如常見的 SQL Injection 模式）在不同語言的實現檢測很有幫助。另一例子是 GitHub 的 **CodeQL** 平台，透過將程式碼轉換為可查詢的資料庫，能讓我們用統一的查詢語言查找多種語言中的安全漏洞模式[[15]](https://codeql.github.com/docs/codeql-overview/about-codeql/#:~:text=About%20CodeQL%20,knowledge%20of%20a%20single%20vulnerability)（CodeQL 支援 C/C++, C#, Go, Java, JavaScript/TypeScript, Python 等多種主流語言）。AIVA 可結合這類工具，在後端建立**多語言靜態掃描引擎**：對於每種語言，整合相應的分析器或掃描器（例如整合 Semgrep 作一般漏洞掃描，Rust 部分繼續使用現有 SAST Rust 引擎[[16]](https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/docs/ARCHITECTURE/MULTILANG_STRATEGY_SUMMARY.md#L42-L45)等），然後將結果標準化（例如轉換成統一的發現issue結構）。此外，利用各語言官方的編譯器前端/分析 API 也很重要。例如：利用 **Roslyn** 提供 .NET 編譯平台 API 可對 C# 程式進行即時的語法和語意分析，發現 API 誤用、潛在安全或效能問題[[17]](https://www.meziantou.net/the-roslyn-analyzers-i-use.htm#:~:text=)；對 Java，可使用 Eclipse JDT 或 javac 提供的 AST 介面進行語法樹遍歷和類型檢查；對 Go，可以使用 Go 自帶的 go/ast、go/types 套件做靜態分析，或運行 golangci-lint 這類工具；對 JavaScript/TypeScript，可利用 TypeScript 編譯器 API 獲取完整的類型資訊與 AST。這些**官方前端**提供的語意資訊（如型別、控制流程圖等）能輔助 AI 模組做更深入的分析與優化建議。
* **動態載入與執行模擬**：除了靜態分析，有些情況下需要**動態地執行或載入**程式碼來觀察行為（例如執行單元測試、模擬函式調用以查看實際輸出）。為確保安全，AIVA 應在**沙箱環境**中執行目標程式碼。對於直譯型語言如 Python、JavaScript，系統可以採用嵌入式直譯器或沙箱。例如，AIVA 可在後端安全容器中**動態載入 Python 模組**或使用內建直譯器執行使用者提供的 Python 片段。也可以利用 **Pyodide** 等技術：Pyodide 將 CPython 直譯器編譯到 WebAssembly 中，可在瀏覽器或隔離的執行環境中執行 Python 程式碼而不影響宿主系統[[18]](https://blog.cloudflare.com/python-workers/#:~:text=referred%20to%20as%20just%20%E2%80%9CPython%E2%80%9D)。對於 JavaScript，可嵌入 **QuickJS** 這種輕量級JS引擎，在本地進程內執行 JS 程式碼[[19]](https://bellard.org/quickjs/#:~:text=QuickJS%20is%20a%20small%20and,asynchronous%20generators%2C%20proxies%20and%20BigInt)。QuickJS 只有幾個 C 檔案，記憶體需求小且執行速度快，非常適合作為沙箱執行多段獨立的 JS 程式碼。這些沙箱機制確保即使執行了不受信任的代碼，也不會對整個系統造成危害。對於編譯型語言如 Go，我們有兩種選擇：一是**編譯並在隔離容器中執行**（例如使用 Docker 或 Firecracker 開啟一個受限環境跑 Go 編譯出的二進位，再透過IPC收集結果）；二是使用**直譯方式模擬執行**。Go 語言本身沒有官方直譯器，但社群有如 **Yaegi** 的專案，它是一個純 Go 語言實現的直譯器，允許在程式執行時載入並執行字串形式的 Go 代碼[[20]](https://github.com/traefik/yaegi#:~:text=Yaegi%20is%20Another%20Elegant%20Go,top%20of%20the%20Go%20runtime)。AIVA 可以將 Yaegi 嵌入 Go 微服務中，實現在不重新編譯整個應用的情況下執行片段 Go 代碼的能力。類似地，Java 可利用內嵌的 **JavaScript 引擎（Nashorn 或 GraalJS）** 來執行腳本，或使用 Java ClassLoader 動態載入編譯過的 class。**GraalVM** 是一個強大的多語言執行環境選項：它允許在同一個 JVM 處理多種語言（如同時跑 JVM Bytecode、JavaScript、Python、Ruby 等），並透過 **Truffle 框架**提供各語言間的互操作[[21]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=GraalVM%20allows%20users%20to%20write,henceforth%20%E2%80%9CTruffle%E2%80%9D)。GraalVM 的**跨語言互通協定**意味著我們可以在一種語言中直接呼叫另一種語言的函式，資料結構也能共享，這對於實現複雜的動態測試很有幫助[[22]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=In%20order%20to%20provide%20foreign,Language%20Runtime%20paper)。例如，AIVA 可以用 GraalVM 在一個進程中載入測試的 JavaScript 程式，呼叫其中的函式，並在 Python 邏輯中捕獲其輸出結果或異常。這樣免去了跨進程通信的開銷。總之，在動態分析方面，**採取容器級沙箱**（隔離整個運行環境）與**嵌入式語言沙箱**（隔離在進程內）雙管齊下，確保在需要執行受測程式時既能收集行為資訊又不危及平台本身。
* **程式碼修改與重構**：在掌握 AST 之後，AIVA 可以對程式碼進行**自動修改**。這可用於套用 AI 模型給出的重構建議或安全修補建議。實現上，可以先定位 AST 中需要修改的節點，然後以程式化的方式變更（例如用 Tree-sitter 提供的 API 修改節點）。或者更直接地，利用各語言的重構工具鏈：例如對 Java，可使用 OpenRewrite 或 Google ErrorProne 這類工具做程式碼重寫；對 .NET，有 Roslyn 提供的 **CodeFix** API 可插入建議修正[[23]](https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix#:~:text=The%20,preview%20of%20the%20suggested%20fixes)[[24]](https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix#:~:text=In%20this%20tutorial%2C%20you%27ll%20explore,modifier)。當然，也可以讓 AI 直接產生修改後的代碼片段，再由系統替換原始碼的對應部分。然而為避免 AI 產生語法錯誤代碼，最佳實踐是**結合 AST 操控與 AI 建議**：AI 提供邏輯或風格上的修改建議，程式本體則透過 AST 確保應用修改時語法正確。例如，AI 建議將某段 Python 程式的迴圈改用列表生成式，可由系統確認語法樹中迴圈節點範圍，然後以 AI 生成的列表推導式代碼字符串替換，最後再用 Python AST 重新解析驗證。如此迭代直到生成有效代碼。這種半自動重構流程能提高可靠性。
* **語言專用處理**：需要強調的是，每種程式語言都有獨特的結構與特性，系統需要對此有所瞭解並做特殊處理[[25]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Supporting%2040%2B%20languages%20means%20handling,constructs%20that%20require%20specialized%20handling)。例如：Python 的縮排、動態類型需要在AST之外再處理名稱解析和型別推斷（可透過靜態分析工具如 pyright）；Java有明確的型別與封裝限制，需考慮解析 classpath；C/C++ 需要處理巨集和編譯條件；Rust 有所有權檢查，在分析時可能要借助編譯器輸出 MIR（中介表示）等。為此，可以引入**多語言分析插件機制**：為每種語言實作一個小插件，負責該語言特定的AST後處理或與編譯器的集成。例如 Rust 插件調用 Rust 編譯器進行 borrow-check 模擬、Java 插件讀取字節碼進行額外驗證等。**統一的AST + 插件架構**能在保有跨語言一致性的同時，容納各語言的特殊需求。

## 多語言介面與系統整合方案

最後，AIVA 系統需要提供**多種語言的介接方式**，使外部使用者或應用可以方便地以熟悉的語言調用 AIVA 的功能。為實現這點，應對外暴露標準化的 API，並透過多語言 SDK 或工具讓各種環境都能存取。

* **RESTful API**：建立基於 HTTP/REST 的服務介面是最通用的方案。AIVA 的核心功能可以用 Web API 的形式提供，例如使用 FastAPI 框架開發（AIVA 核心已是 Python，相容 FastAPI）。REST API 的好處是語言中立、簡單易用，幾乎所有程式語言都有 HTTP 庫可以調用。**OpenAPI（Swagger）** 規範可用來描述這些 API，然後透過自動代碼產生工具生成各語言的客戶端 SDK[[26]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=Because%20FastAPI%20is%20based%20on,format%20that%20many%20tools%20understand)[[27]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=Open%20Source%20SDK%20Generators%C2%B6)。OpenAPI Generator 支援數十種語言的客戶端代碼生成功能，因此我們可以一鍵產生例如 **Node.js 的 NPM 套件、Ruby 的 Gem、Go 的模組、Rust 的 crate、Java 的Jar** 等 SDK，開發者使用時就如同調用本地庫一樣方便[[28]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=This%20makes%20it%20easy%20to,in%20sync%20with%20your%20code)[[29]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=A%20versatile%20option%20is%20the,SDKs%20from%20your%20OpenAPI%20specification)。這減少了不同語言調用 AIVA 時手動構造 HTTP 請求/解析回應的麻煩。同時，我們應該提供詳細的 API 說明文件，讓開發者清楚各個端點的用途和參數格式。
* **gRPC 介面**：除了 REST，對於需要高效且強類型通信的場景，可以提供 **gRPC** 服務。gRPC 基於 Protocol Buffers 定義介面，可以自動生成多語言的強類型客戶端代碼（支援如 C++、Java、Python、Go、C#、Ruby 等主流語言）。內部微服務也可用 gRPC 通訊以獲取更佳效能和簡潔性[[30]](https://orkes.io/blog/workflow-orchestration-many-programming-languages/#:~:text=Microservice%20and%20Workflow%20Orchestration%20with,communication%20between%20your%20multilanguage%20microservices)。對外的 gRPC API 可與 REST API 提供相似的功能，但透過 Protobuf 定義能確保各語言對資料結構的理解一致。使用者若在高度併發或需要串流資料的情況下，可選擇直接使用 gRPC 客戶端。基於 gRPC 定義檔，我們也能提供各語言對應的 SDK（例如自動生成一個 AIVAClient 類別給 Node.js、Rust 等）。
* **CLI 工具**：提供一個**命令列介面**工具也是必要的。CLI 工具可用各語言實現（但通常用 Python/Go 開發跨平台 CLI 較方便），用戶或腳本可以透過命令列參數調用 AIVA 的功能。例如 aiva scan --file target.py --lang python 執行一次掃描，結果輸出為檔案或stdout。CLI 工具可被任何語言透過系統呼叫（如 shell exec）使用，也方便整合到 CI/CD 管道中。它對人類使用者也友好（可直接在終端下使用 AIVA 功能）。這為那些不便直接使用 API 的場合提供了替代方案。
* **多語言 SDK 庫**：除了自動生成的 API 客戶端，我們也可針對某些常用語言開發**手動優化的 SDK**。例如撰寫一個 Node.js 的包，內部透過 REST/gRPC 調用 AIVA，但對開發者暴露更符合該語言習慣的介面（比如返回 Promise 或使用 async/await）。同理可製作 Python 的 SDK（不過 AIVA 本身是 Python，可考慮直接作為庫使用）、Java 的封裝（提供一組 Java 方法呼叫 HTTP）、以及 .NET 的 NuGet 套件等。這些 SDK 可以封裝身份驗證、錯誤處理、重試等邏輯，讓開發者在各自語言環境中「傻瓜式」地使用 AIVA 服務。特別地，在安全測試領域，我們可以提供**CI/CD插件**或**IDE插件**，但這超出本題範圍，重點是透過 API 開放讓各語言都能存取。
* **Jupyter 生態**：值得一提的是，可參考 **Jupyter** 的架構實現多語言支援。Jupyter 透過**內核（Kernel）**機制支援了 **100 多種編程語言**[[31]](https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html#:~:text=5)：前端使用統一的協議，後端換用不同語言的 Kernel 處理指令，使用者介面卻保持一致[[32]](https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html#:~:text=Jupyter%E2%80%99s%20kernel%20flexibility%20allows%20instructors,language%20within%20a%20course%2C%20however)。AIVA 若提供類似 Jupyter Kernel 或魔術指令介面，則使用者可以在 Jupyter Notebook 中直接使用 AIVA 的功能。例如開發一個 AIVA 的 Jupyter Kernel，允許 Notebook 使用自然語言或簡單指令呼叫 AIVA 背後的分析引擎（可能透過 AIVA 的 Python 核心實現）。這樣資料科學家或開發人員能在交互式環境中，以不同語言編寫的程式碼單元，直接請 AIVA 分析或掃描，獲得結果輸出到 Notebook。雖然不是典型的「讓別的語言程式接入」方式，但這種多語言交互介面能提升 AIVA 在教學或研究環境的易用性，因為 Jupyter 已是跨語言的統一平臺。

綜上，AIVA 系統應提供一整套**多語言接入策略**：REST/gRPC **服務層**提供核心功能的標準介面；在此基礎上構建各語言的**SDK客戶端**和**CLI工具**；並藉助 OpenAPI 等工具降低維護成本（API 規格更新後可重新生成各語言 SDK）。這確保了無論使用者採用何種技術棧（Node.js、Ruby、Rust、Java...），都能順利地將 AIVA 的功能融入自己的流程中。

## 綜合方案與模組化設計建議

根據以上分析，我們可以設計一個**可落地的多語言支援架構**，將系統拆分為幾個關鍵模組，各司其職又協同工作：

1. **多語言 AI 引擎模組**：採用統一的 AI 模型（或模型集合）來處理程式碼理解與生成。可選用開源的多語言編碼模型（如 CodeT5+、StarCoder 等）並結合專案需求進行微調，使其具備針對不同語言進行分析優化的能力[[4]](https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding#:~:text=Hugging%20Face%20and%20ServiceNow%20have,4)[[6]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=It%20was%20trained%20on%20a,to%20generating%20data%20science%20charts)。此模組對外提供接口，接受程式碼片段或整個專案，以及用戶的自然語言請求，輸出分析結果或修正建議。它也負責在內部將不同語言的程式碼轉化為向量表示或內部統一表徵，以便在知識庫檢索和模型推理時不受語言表面差異影響。
2. **語法解析與靜態分析模組**：組織一套多語言 AST 解析器和靜態分析工具集。利用 Tree-sitter 等框架實現對主流語言源碼的解析，生成抽象語法樹供後續使用[[8]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Tree,chose%20it%20for%20several%20reasons)。再集成各語言的靜態分析規則庫（可引用 Semgrep 規則或自研規則），對 AST 進行模式匹配和潛在問題檢測[[14]](https://github.com/semgrep/semgrep#:~:text=Semgrep%20is%20a%20fast%2C%20open,as%20part%20of%20CI%2FCD%20workflows)。同時，為精確性，可在背後調用各語言原生編譯器的分析功能，如調用 Roslyn 分析 C# 代碼習慣問題[[17]](https://www.meziantou.net/the-roslyn-analyzers-i-use.htm#:~:text=)、調用 golangci-lint 檢查 Go 代碼風險等。這個模組應將分析結果（例如發現的 bug、疑似漏洞、代碼氣味等）轉化為通用格式，交由 AI 引擎彙總或直接回傳給用戶。它也為 AI 引擎提供輔助訊息，例如代碼的控制流程圖、調用圖等，讓 AI 回答更有依據。
3. **動態執行與沙箱模組**：提供受控環境來執行多語言程式碼，用於動態分析和驗證 AI 建議。可基於容器技術實現，如每種語言配置對應的 Docker 容器映像，內含該語言運行環境，透過調用容器來執行提交的程式碼片段。亦可在程式內嵌入輕量級直譯器：例如嵌入 QuickJS 引擎執行使用者提供的 JavaScript[[19]](https://bellard.org/quickjs/#:~:text=QuickJS%20is%20a%20small%20and,asynchronous%20generators%2C%20proxies%20and%20BigInt)、嵌入 Pyodide 來執行 Python[[18]](https://blog.cloudflare.com/python-workers/#:~:text=referred%20to%20as%20just%20%E2%80%9CPython%E2%80%9D)、或使用 GraalVM 同時執行多種語言腳本[[21]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=GraalVM%20allows%20users%20to%20write,henceforth%20%E2%80%9CTruffle%E2%80%9D)。此模組需確保資源隔離和執行時限，防止惡意或無窮迴圈程式影響平台。執行過程中，可收集執行輸出、錯誤日志，甚至運用調試鉤子截取特定變數值，將這些動態資訊反饋給 AI 模組或記錄於分析報告中。
4. **多語言互通介面模組**：負責對外提供各種介面讓使用者接入。包括 REST API 服務（使用 FastAPI 等生成 OpenAPI 文件) 和 gRPC 服務（定義 .proto 接口）。此模組還維護各語言的 SDK 實現，確保與核心服務的協議同步更新。它也提供命令列工具 aiva 給終端使用者。該模組需考慮**認證授權**（例如 API 金鑰管理）、**請求速率限制**以及**版本相容性**等，以支援實際生產環境的應用。對內，它將接收到的請求轉發給 AI 引擎或分析模組處理，並將結果封裝為統一的響應格式返回。
5. **資料與訓練模組（可選）**：為了持續提升多語言支援能力，AIVA 可能需要一個負責模型訓練與知識更新的模組。一方面定期收集新的開源程式碼資料（注意授權）以及安全漏洞資料，用於更新 AI 模型的知識庫或訓練資料集；另一方面記錄用戶反饋及系統誤報/漏報情況，透過強化學習或訓練調整提高模型性能[[7]](https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/README.md#L62-L70)。該模組也可構建**多語言知識庫**，例如彙整各語言的安全最佳實踐、常見錯誤模式，在 AI 分析時作為檢索依據（這等同於 RAG 知識來源）。雖然此部分對外不直接提供功能，但對於系統長期演進以涵蓋更多語言和更複雜的分析場景非常重要。

整體而言，上述模組構成了一個分層次的架構：**底層**是多語言的語法和執行支援，中間是智能分析與決策層（AI 模型和規則引擎），頂層是面向用戶的接口層。透過引用現有開源方案（如 Tree-sitter[[8]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Tree,chose%20it%20for%20several%20reasons)、Semgrep[[14]](https://github.com/semgrep/semgrep#:~:text=Semgrep%20is%20a%20fast%2C%20open,as%20part%20of%20CI%2FCD%20workflows)、GraalVM[[21]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=GraalVM%20allows%20users%20to%20write,henceforth%20%E2%80%9CTruffle%E2%80%9D)等）並結合自主研發，AIVA 可以較快速地擴充到對多種語言的全面支持。在落實過程中，要特別注意**跨語言的一致性**：例如，不同語言的掃描結果需要映射到統一的風險級別或問題類型；AI 給不同語言的建議風格應保持類似的詳盡程度；介面返回值格式應統一等等。唯有保持一致，才能真正實現“**一個平台，通用多語**”的目標，讓 AIVA 成為一個能對各種技術棧進行智慧安全分析的通用平臺。藉助上述策略設計，AIVA 將具備完備的多語言程式處理能力，滿足廣大開發團隊在複雜應用環境中的自動化安全測試需求。[[1]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=8,language%20development%20environment)[[31]](https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html#:~:text=5)

**參考資料：**

* Dropstone Engineering, *AST Parsing with Tree-sitter: Understanding Code Across 40+ Languages*[[8]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Tree,chose%20it%20for%20several%20reasons)[[9]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Why%20AST%20Parsing%20Matters)
* Scribble Data, *The Top LLMs For Code Generation: 2024 Edition*[[2]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=OpenAI%20Codex%2C%20a%20descendant%20of,with%20Python%20being%20its%20strongest)[[1]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=8,language%20development%20environment)
* Novita AI, *Introducing Code Llama: A state-of-the-art LLM for code generation*[[3]](https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7#:~:text=language%20prompts%20%28e,and%20Bash)
* Scademy News, *StarCoder: Open Source AI for Multi-Language Coding*[[4]](https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding#:~:text=Hugging%20Face%20and%20ServiceNow%20have,4)[[33]](https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding#:~:text=of%20GPT)
* GitHub – Semgrep, *README (static analysis for many languages)*[[14]](https://github.com/semgrep/semgrep#:~:text=Semgrep%20is%20a%20fast%2C%20open,as%20part%20of%20CI%2FCD%20workflows)
* Gérald Barré, *The Roslyn analyzers I use in my projects*[[17]](https://www.meziantou.net/the-roslyn-analyzers-i-use.htm#:~:text=)
* GraalVM Official Docs, *Polyglot Programming*[[21]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=GraalVM%20allows%20users%20to%20write,henceforth%20%E2%80%9CTruffle%E2%80%9D)[[22]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=In%20order%20to%20provide%20foreign,Language%20Runtime%20paper)
* Cloudflare Blog, *Bringing Python to Workers using Pyodide*[[18]](https://blog.cloudflare.com/python-workers/#:~:text=referred%20to%20as%20just%20%E2%80%9CPython%E2%80%9D)
* Fabrice Bellard, *QuickJS Javascript Engine*[[19]](https://bellard.org/quickjs/#:~:text=QuickJS%20is%20a%20small%20and,asynchronous%20generators%2C%20proxies%20and%20BigInt)
* *Jupyter Notebook ecosystem – Language support*[[31]](https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html#:~:text=5)

[[1]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=8,language%20development%20environment) [[2]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=OpenAI%20Codex%2C%20a%20descendant%20of,with%20Python%20being%20its%20strongest) [[6]](https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/#:~:text=It%20was%20trained%20on%20a,to%20generating%20data%20science%20charts) The Top LLMs For Code Generation: 2024 Edition

<https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/>

[[3]](https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7#:~:text=language%20prompts%20%28e,and%20Bash) [[5]](https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7#:~:text=Additionally%2C%20Meta%20has%20further%20fine,and%20Code%20Llama%20%E2%80%94%20Instruct) Introducing Code Llama: A State-of-the-art large language model for code generation. | by Novita AI | Medium

<https://medium.com/@marketing_novita.ai/introducing-code-llama-a-state-of-the-art-large-language-model-for-code-generation-e9753deb61b7>

[[4]](https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding#:~:text=Hugging%20Face%20and%20ServiceNow%20have,4) [[33]](https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding#:~:text=of%20GPT) StarCoder: Open Source AI for Multi-Language Coding

<https://www.scademy.ai/news/starcoder-open-source-ai-for-multi-language-coding>

[[7]](https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/README.md#L62-L70) README.md

<https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/README.md>

[[8]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Tree,chose%20it%20for%20several%20reasons) [[9]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Why%20AST%20Parsing%20Matters) [[10]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=A%20FunctionDeclaration%20node) [[11]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=How%20Dropstone%20uses%20Tree,accurate%20analysis%20and%20intelligent%20suggestions) [[12]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Query%20system.%20Tree,for%20detecting%20specific%20code%20constructs) [[13]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=In%20TypeScript%2C%20we%20handle%20type,to%20provide%20accurate%20type%20information) [[25]](https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages#:~:text=Supporting%2040%2B%20languages%20means%20handling,constructs%20that%20require%20specialized%20handling) AST Parsing with Tree-sitter: Understanding Code Across 40+ Languages | Dropstone Blog

<https://www.dropstone.io/blog/ast-parsing-tree-sitter-40-languages>

[[14]](https://github.com/semgrep/semgrep#:~:text=Semgrep%20is%20a%20fast%2C%20open,as%20part%20of%20CI%2FCD%20workflows) GitHub - semgrep/semgrep: Lightweight static analysis for many languages. Find bug variants with patterns that look like source code.

<https://github.com/semgrep/semgrep>

[[15]](https://codeql.github.com/docs/codeql-overview/about-codeql/#:~:text=About%20CodeQL%20,knowledge%20of%20a%20single%20vulnerability) About CodeQL - GitHub

<https://codeql.github.com/docs/codeql-overview/about-codeql/>

[[16]](https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/docs/ARCHITECTURE/MULTILANG_STRATEGY_SUMMARY.md#L42-L45) MULTILANG\_STRATEGY\_SUMMARY.md

<https://github.com/kyle0527/AIVA/blob/c1642bdc3473d306b3b9b9a88cecf4bb2b8f582a/docs/ARCHITECTURE/MULTILANG_STRATEGY_SUMMARY.md>

[[17]](https://www.meziantou.net/the-roslyn-analyzers-i-use.htm#:~:text=) The Roslyn analyzers I use in my projects - Meziantou's blog

<https://www.meziantou.net/the-roslyn-analyzers-i-use.htm>

[[18]](https://blog.cloudflare.com/python-workers/#:~:text=referred%20to%20as%20just%20%E2%80%9CPython%E2%80%9D) Bringing Python to Workers using Pyodide and WebAssembly

<https://blog.cloudflare.com/python-workers/>

[[19]](https://bellard.org/quickjs/#:~:text=QuickJS%20is%20a%20small%20and,asynchronous%20generators%2C%20proxies%20and%20BigInt) QuickJS Javascript Engine

<https://bellard.org/quickjs/>

[[20]](https://github.com/traefik/yaegi#:~:text=Yaegi%20is%20Another%20Elegant%20Go,top%20of%20the%20Go%20runtime) traefik/yaegi: Yaegi is Another Elegant Go Interpreter - GitHub

<https://github.com/traefik/yaegi>

[[21]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=GraalVM%20allows%20users%20to%20write,henceforth%20%E2%80%9CTruffle%E2%80%9D) [[22]](https://www.graalvm.org/latest/reference-manual/polyglot-programming/#:~:text=In%20order%20to%20provide%20foreign,Language%20Runtime%20paper) Polyglot Programming

<https://www.graalvm.org/latest/reference-manual/polyglot-programming/>

[[23]](https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix#:~:text=The%20,preview%20of%20the%20suggested%20fixes) [[24]](https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix#:~:text=In%20this%20tutorial%2C%20you%27ll%20explore,modifier) Tutorial: Write your first analyzer and code fix - C# | Microsoft Learn

<https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix>

[[26]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=Because%20FastAPI%20is%20based%20on,format%20that%20many%20tools%20understand) [[27]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=Open%20Source%20SDK%20Generators%C2%B6) [[28]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=This%20makes%20it%20easy%20to,in%20sync%20with%20your%20code) [[29]](https://fastapi.tiangolo.com/advanced/generate-clients/#:~:text=A%20versatile%20option%20is%20the,SDKs%20from%20your%20OpenAPI%20specification) Generating SDKs - FastAPI

<https://fastapi.tiangolo.com/advanced/generate-clients/>

[[30]](https://orkes.io/blog/workflow-orchestration-many-programming-languages/#:~:text=Microservice%20and%20Workflow%20Orchestration%20with,communication%20between%20your%20multilanguage%20microservices) Microservice and Workflow Orchestration with Multiple Languages

<https://orkes.io/blog/workflow-orchestration-many-programming-languages/>

[[31]](https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html#:~:text=5) [[32]](https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html#:~:text=Jupyter%E2%80%99s%20kernel%20flexibility%20allows%20instructors,language%20within%20a%20course%2C%20however) Chapter 5 Jupyter Notebook ecosystem | Teaching and Learning with Jupyter

<https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html>