構建 5G 智慧交通數位神經中樞－

附錄一、《交通速率預測系統》建置指引手冊

交 通 部 運 輸 研 究 所

中 華 民 國 115 年 10 月

第一章 前言

1.1 編寫目的

本建置指引手冊的編寫目的，主要係將交通部運輸研究所於「構建 5G 智慧交通數位神經中樞系列計畫」研發之「交通速率預測系統」推廣至實務的落地應用。手冊內容涵蓋系統的總體架構設計、五大核心模組的詳細功能與演算法說明、軟硬體資源需求規劃、乃至系統的功能驗證與效果評估方法。透過本手冊得以建立一套標準化的建置流程，降低各單位導入交通預測技術的門檻，有效銜接政策需求與各縣市的實務應用。

1.2 適用對象與範圍

一、適用對象

(一) 政府交通主管機關與決策人員：可協助了解本預測系統的技術成熟度、導入方式及應用效益，作為制定智慧交通政策、編列預算及規劃未來發展方向之參考。

(二) 地方政府規劃與採購人員：可直接引用本手冊中的功能規格、軟硬體需求及驗收標準，作為專案執行的技術規範，確保專案成果符合預期效益。

(三) 系統整合商與軟體開發商：可依循本手冊的步驟，為政府客戶建置一套完整、穩定的預測系統。

(四) 學術與研究單位：可透過本手冊了解業界實際採用的預測模型架構與技術細節，作為學術研究與產學合作的參考基礎。

二、適用範圍

本手冊定義並闡述「交通速率預測系統」從概念到實踐的完整技術框架，詳述從資料蒐集、模型訓練、即時預測到服務提供的五大核心模組的設計原理與建置方法。

1.3 相關背景依據

本手冊的撰寫，除技術的展現之外，亦為實務經驗與政策目標的結合，其主要背景依據如下：

一、多年期研發成果的標準化：本手冊為一個經過多年開發、驗證與優化的交通速率預測模型及其配套系統，進行知識萃取與標準化後的成果，旨在將成熟的技術轉化為可複製、可推廣的解決方案。

二、反應國家智慧運輸發展政策：內容與架構的設計，旨在反應國家與地方政府推動「智慧城市」與「智慧運輸系統（ITS）」的政策方向，為提升道路交通管理效率與用路人服務品質，提供具體的技術實踐路徑。

三、降低導入門檻與風險：透過提供一份經過驗證的「公版」建置藍圖，各單位無需從零開始進行漫長的技術探索與試錯，可大幅縮短開發時程、降低建置風險，並確保系統的最終品質與效益。

四、先進資訊技術實踐：系統架構本身融合機器學習運作（MLOps）、大規模時序資料處理、微服務化 API 等多個領域的技術最佳實踐，確保系統的穩定性與未來擴展性。

第二章 功能簡介

2.1 系統功能說明

本手冊所構建之「交通速率預測系統」為一套整合自動化數據處理、先進機器學習與標準化對外服務的端到端智慧化服務方案。其功能旨在提供高準確度、高時效性且可客製化的即時交通速率預測服務。系統的核心功能包含以下四個層面：

一、自動化數據驅動預測能力：本系統能自動完成從外部原始數據蒐集、數據清洗補值、特徵工程、模型預測到結果儲存的完整流程。透過不間斷的即時數據注入與週期性的模型再訓練，確保系統持續學習最新的交通模式，維持預測的準確性。

二、高精度多步預測能力：本系統不僅預測下一時間點的路況，而是採用「直接多步預測」策略，透過 MultiOutputRegressor 封裝器與 Stacking 集成模型，能夠一次性產出未來 30 分鐘內、每 5 分鐘為間隔的連續速率預測值。

三、客製化與可擴充性：為應對不同路段獨特的交通特性，系統的設計核心是為每一個獨立的路段（id）建立專屬的優化預測模型。此「一路段一模型」的架構能達到最佳的預測精度。同時，底層的資料庫（TimescaleDB）與運算架構（多進程處理）均為可水平擴充的設計，能支撐從數百到數千個路段的大規模部署需求。

四、彈性的擴散與整合能力：政府單位可依據本手冊內容，建置一套完整的私有化系統；亦可將開發完成之預測系統提供一個標準化 API 服務，快速整合至既有的交通控制與資訊發布平台中。

2.2 適用場域與應用情境

本系統的預測能力可根據不同的道路網絡特性進行深度應用，主要分為以下兩大核心場域及其他衍生應用：

一、高速公路應用情境

在高速公路這種路況相對單純、車流連續性強的環境下，本系統的長時效預測能力可以發揮極大價值。透過將預測結果整合至國道高速公路局的交通控制系統，可達到以下應用：

(一) 旅行時間預測：透過不同路段（id）之交通速率預測結果，可轉換為旅行時間，在資訊可變標誌、官方網站及「高速公路 1968」App 上，發布未來 30 分鐘的國道主線及匝道預估旅行時間，其準確度較優於傳統基於歷史資料的推算。

(二) 事故衝擊範圍與時間預測：當發生交通事故時，系統可即時預測該事件對下游

路段未來半小時內的衝擊範圍與持續時間，協助交控中心規劃替代道路建議與警力疏導範圍。

(三) 連續假期交通疏導：在連續假期前利用系統進行路網壅塞熱點預測，作為實施高乘載管制、匝道儀控或開放路肩等疏導措施的決策依據。

二、都市地區道路應用情境

都市道路網具有路況複雜、交叉路口眾多、易受事件影響等特點。本系統可為各縣市地方政府的交通控制系統賦能，應對這些挑戰：

(一) 預測性號誌控制：傳統的號誌採用的多為即時或歷史數據。運用本系統可根據未來 15 至 30 分鐘的車流預測，提前介入調整關鍵路口或主要幹道的號誌時制計畫，避免車流匯集形成深度壅塞。

(二) 大型活動交通衝擊管理：針對各類大型活動或道路施工等可預期的事件，可在事前利用本系統模擬活動對周邊路網的交通衝擊。藉此，警察局與交通局可制定更完善的交通管制、停車規劃與大眾運輸接駁計畫，並在活動期間根據即時預測動態調整。

三、其他衍生應用情境

透過本系統的預測數據，亦可提供民間業者進行相關商業應用：

(一) 物流車隊與供應鏈管理：物流公司的派遣系統可取得預測數據後為貨車車隊提供動態路徑規劃。系統不僅能避開當前的壅塞，更能避開「即將發生」的壅塞，有效降低運輸時間與油耗成本。

(二) 導航軟體與地圖服務：Google Maps、Waze 或國內導航 App 可整合本系統的預測數據，提供更加精準的預計到達時間(ETA)，並在規劃路線時，提供具備「預測性」的智慧繞行建議。

第三章 系統功能建置說明

3.1 系統架構概述

本系統旨在建立一個從資料蒐集、模型訓練、即時預測到對外服務的完整自動化交通速率預測系統。該系統由五個核心模組構成。本文件將依據如圖 1 之系統架構圖，詳細說明各模組的功能與運作流程。

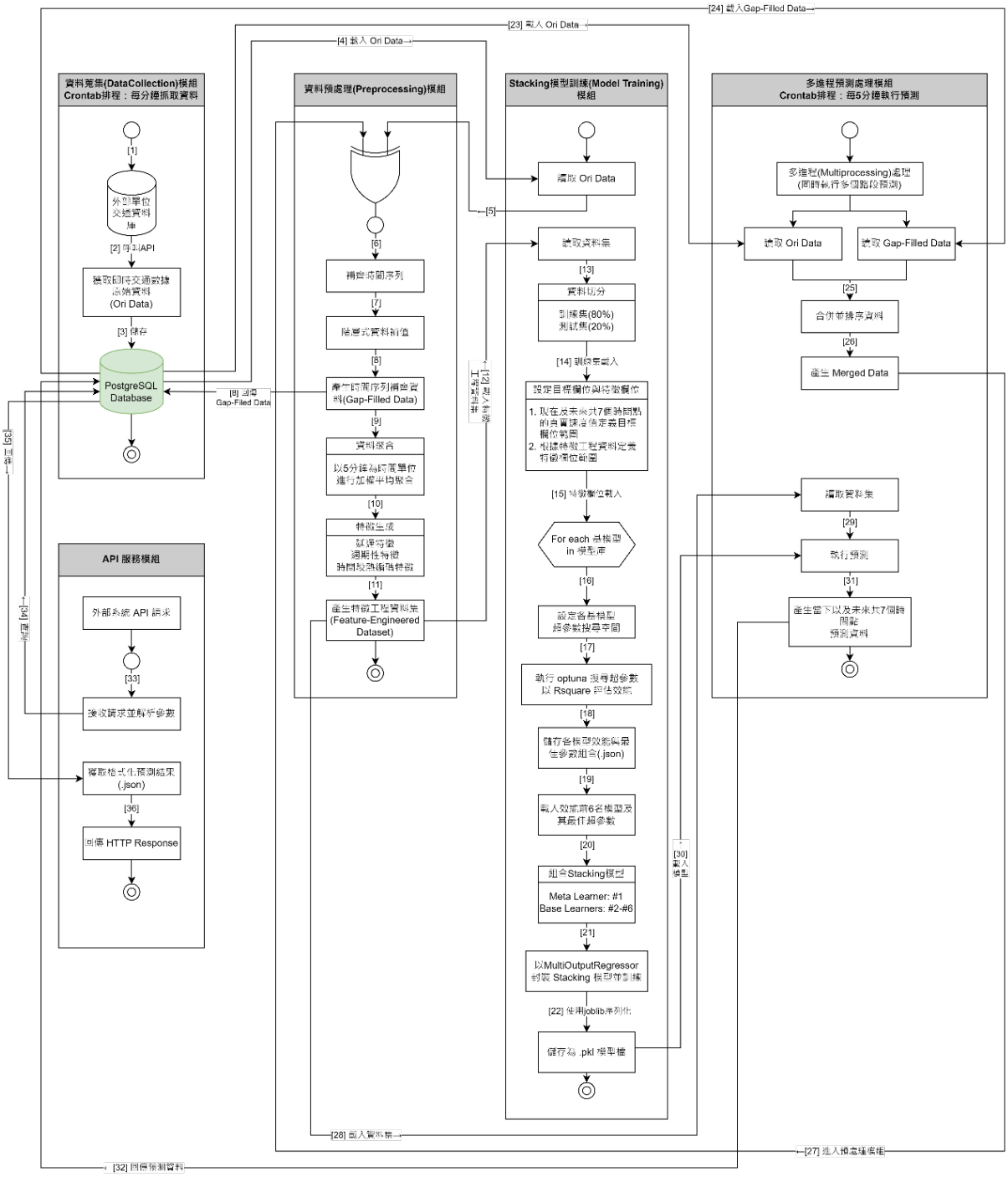


圖 1 交通速率預測系統架構圖

3.2 軟體設計／演算法說明

一、資料蒐集（Data Collection）模組

資料蒐集模組是整個交通速率預測系統的數據基礎與起點，此模組的核心功能是穩定、持續地從外部公/私單位獲取即時的原始交通數據，並將這些結構化的資料安全地存入系統內部的資料庫中。資料來源單位提供之高品質、無重複且格式一致的數據源是後續資料前處理、模型訓練與預測準確性的根本。

為完成上述功能，本模組的建置與運作包含以下技術細節：

(一) 觸發機制(Trigger Mechanism)：系統利用 Ubuntu 作業系統內建的 Crontab 工具進行排程。排程設定為每分鐘執行一次 Python 程式碼，以確保資料獲取的即時性。

(二) 資料獲取(Data Acquisition)：排程啟動的 Python 程式會使用 requests 套件，透過 HTTP/HTTPS 請求，呼叫外部資料來源所提供之 RESTful API。API 回傳資料格式多為 XML。此外，程式亦需負責解析 XML 內容，提取出必要資料欄位。

(三) 資料欄位需求

從 API 獲取之原始交通資料，其欄位至少需要包含以下三項核心資訊：

1. 路段識別碼（id）：用以唯一識別交通數據所屬的道路或偵測器位置。
2. 日期時間（tdate）：精確到分鐘的時間戳，記錄該筆數據發生的時間。
3. 平均速率（avgspeed）：該路段在該時間點的平均車輛行駛速率，是預測的基礎目標。
4. 方向（direction）：該路段（id）所代表之方向，分為 follow 與 reverse。

(四) 資料儲存（Data Storage）

經解析與提取後的結構化資料，將會被存入系統自建的 PostgreSQL 資料庫中。而在資料庫的對應表格中，建議的欄位格式如下：

1. id：text 型別，用以儲存路段的字串 id。
2. tdate：timestamp with time zone 型別，用以儲存精確的日期與時間。
3. avgpseed：douvle precision 型別，用以儲存包含小數的速率值。
4. direction：text 型別，用以儲存路段代表的方向。

此外，為確保資料的唯一性與完整性，資料表會針對 id、tdate、direction

三個欄位建立複合唯一鍵（Composite Primary Key）。此設計能從資料庫層面有效避免因網路延遲或排程重疊，導致同一路段在同一時間點的資料被重複寫入，所有資料均存入基於 PostgreSQL 的時序資料庫 TimescaleDB 中。資料表被設定為 Hypertable，並以 tdate 欄位作為分區鍵 (Partition Key)，進行自動化的每日切片（Daily Partitioning），每一個切片（chunk）儲存一天的資料。此設計使得針對特定日期的查詢能夠極速完成，因為資料庫僅需掃描相關的時間區塊，而無需遍歷整個龐大的資料集，此外，為處理道路的路線等地理空間資訊 (Geospatial Information)，系統亦啟用了 PostGIS 擴充套件，以支援 GEOMETRY(LineString) 等 OGC 標準地理資料類型及高效的空間索引。

(五) 資料流

輸出：此模組最終產出的是儲存在 PostgreSQL 資料庫中、乾淨且無重複的原始交通資料。這些資料將作為後續「多進程預測處理模組」進行即時預測時，所需讀取的最新數據來源。

二、資料預處理（Prepcocessing）模組

資料預處理模組是銜接原始數據與機器學習模型訓練之關鍵，其核心是將從資料庫中讀取的、可能存在缺失或格式不一的原始資料，轉換為結構完整、特徵豐富且可用於模型訓練和預測的標準化資料集。此模組的處理品質會直接影響後續模型訓練成效與預測結果準確性。

本模組的運作流程具體細節說明如下：

(一) 觸發機制（Trigger Mechanism）：此模組為「被動觸發」設計，其本身沒有定時排程，而是由「Stacking 模型訓練模組」或「多進程預測處理模組」在需要處理數據時進行調用。

(二) 資料補值（Data Imputation）

1. 目的：原始交通資料在蒐集過程中，常因網路或設備問題導致某些時間點的數據完全遺失（時間序列不連續），或是某些時間點僅缺少速率值。因此，功能任務是補齊所有時間點，確保時間序列的完整性，並填補缺失的速率資料。
2. 階層式補值策略：為確保補值合理性，系統採用階層式策略，依序尋找最合適的資料來源進行填補：
   (1) 第一階段 (週週期-1)：優先以「前 7 天」同星期、同時間點的歷史資料填補。
   (2) 第二階段（週週期-2）：若無，則搜尋「前 14 天」同週期資料。

(3) 第三階段（週週期-3）：若無，則搜尋「前 21 天」同週期資料。

(4) 第四階段（週週期-4）：若無，則搜尋「前 28 天」同週期資料。

(5) 第五階段（鄰近值-前）：若所有歷史週期資料皆缺失，則使用時間序列中「前一筆」非缺值的資料向前填補。

(6) 第六階段（鄰近值-後）：若前一筆資料也無效（例如，資料開頭就缺失），則使用「後一筆」非缺值的資料向後填補。

3. 產出與儲存：完成上述補值流程後，會產生一份時間序列完整、無遺失值的 Gap-Filled Data（每分鐘一筆）。這份資料會被回存至 PostgreSQL 資料庫中，作為後續分析或下一次預測流程的快取基礎。

(三) 資料聚合（Data Aggregation）

在完成補值後，流程會將每分鐘一筆的 Gap-Filled Data，以每 5 分鐘為一個時間單位進行聚合。聚合時會先依據 id 進行分組，並採用加權平均方式計算出每 5 分鐘平均速率。

(四) 特徵生成（Feature Engineering）

基於前述 5 分鐘聚合資料，系統會建立三種類型的特徵，以捕捉不同維度的數據資訊。

1. 延遲特徵（Lag Features）：用於擷取近期的速率變化趨勢。

(1) 前一筆時間點資料 (t-1)

(2) 前二筆時間點資料 (t-2)

(3) 前三筆時間點資料 (t-3)

2. 週期性特徵（Seasonal Lag Features）：用於捕捉以天或週為單位的週期性交通模式。

(1) 前一天 (t-24h)

(2) 前一週 (t-7d)

(3) 前二週 (t-14d)

(4) 前三週 (t-21d)

(5) 前四週 (t-28d)

3. 時間段熱編碼特徵（Time-of-Day One-Hot Features）：將一天 24 小時劃分為六個時段並進行 One-Hot 編碼，以捕捉不同時段的交通特性 。

(1) dawn（凌晨）：02:00 - 05:59

(2) morning（上午）：06:00 - 09:59

(3) noon（中午）：10:00 - 13:59

(4) evening（下午）：14:00 - 17:59

(5) night（傍晚）：18:00 - 21:59

(6) midnight（午夜）：22:00 - 01:59

(五) 資料流

1. 輸入：接收來自「多進程預測處理模組」的合併資料（Merged Data）。
2. 輸出：最終產出包含上述所有特徵的「特徵工程資料集」（Feature Engineered Dataset）。此資料集會根據調用來源，分別傳遞給「Stacking 模型訓練模組」或「多進程預測處理模組」。

三、Stacking 模型訓練（Model Training）模組

Stacking 模型訓練模組是系統的「大腦」，主要功能為替每一個交通路段（id）客製化地建立一個專屬預測模型。此模組並非線上即時服務，而是一個離線運作的流程。其核心功能是透過自動化的超參數搜尋與集成學習技術（Stacking），從眾多基礎模型中找出最佳組合，並訓練出預測能力最強的模型。最終產出的模型檔案是系統用來準確預測未來交通速率之關鍵。

(一) 觸發機制（Trigger Mechanism）：

本模組的啟動是事件驅動或週期性的，主要有兩種觸發情境：

1. 新路段加入：當系統中出現一個新的路段 id，且尚未有對應的預測模型時，會觸發此模組，為該新路段從頭開始訓練一個專屬的預測模型。
2. 週期性重新訓練：對於已存在模型的路段，系統進行週期性（例如，每年）的模型更新。隨著時間累積更多的新資料，重新觸發此模組能讓模型學習最新的交通模式變化，確保其預測效能不隨時間衰退。

(二) 基模型庫（Base Model Library）

系統內建一個可擴展的模型庫，存放多種類型的演算法作為基學習器（Base Learner）的候選。使用者也可以根據需求，未來在模型庫中新增更多其他基模型。目前建議內建模型包括：

1. 線性模型：LinearRegression, BayesianRidge, Ridge, Lasso, ElasticNet, PassiveAggressiveRegressor
2. 神經網路：MLPRegressor
3. 樹模型：DecisionTreeRegressor, RandomForestRegressor, GradientBoostingRegressor, ExtraTreesRegressor

4. 集成模型：AdaBoostRegressor, XGBRegressor, LGBMRegressor

5. 其他：KNeighborsRegressor, SVR

(三) Stacking 模型訓練流程 (Stacking Model Training Workflow)

1. 資料準備：接收由「資料預處理模組」為特定路段產出的「特徵工程資料集」，並將其劃分為 80% 訓練集與 20% 測試集。為實現「一次性預測未來多個時間點」的需求，系統針對每個時間點 t，產生當前 1 個及未來 6 個時間點的真實速度值作為預測目標。欄位結構如下：

   (1) target_avgspeed_t+0: 代表當前時間點 t 的真實速度值。

   (2) target_avgspeed_t+1: 代表未來 5 分鐘 (t+5m) 的真實速度值。

   (3) target_avgspeed_t+2: 代表未來 10 分鐘 (t+10m) 的真實速度值。

   (4) target_avgspeed_t+3: 代表未來 15 分鐘 (t+15m) 的真實速度值。

   (5) target_avgspeed_t+4: 代表未來 20 分鐘 (t+20m) 的真實速度值。

   (6) target_avgspeed_t+5: 代表未來 25 分鐘 (t+25m) 的真實速度值。

   (7) target_avgspeed_t+6: 代表未來 30 分鐘 (t+30m) 的真實速度值。

2. 基模型超參數搜尋：

   (1) 遍歷模型庫中的每一個基模型。

   (2) 針對每個模型，使用超參數優化框架 Optuna 進行自動化搜尋，Optuna 運用貝葉斯優化策略 (TPESampler)，能根據過去的試驗結果較智慧的調整搜尋方向，比傳統隨機搜尋更有效率。

   (3) 搜尋過程以 R-squared (R^2) 分數作為效能評估指標，找出能讓模型表現最佳的超參數組合。

   (4) 將每個模型搜尋到的最佳超參數及其對應的 R^2 分數，儲存至 JSON 檔案中，以供後續查閱與使用。

3. Stacking 與 MultiOutputRegressor 模型建構：

   (1) 待所有基模型完成超參數搜尋後，載入 JSON 檔案中的結果，並根據 R^2 分數由高至低進行排序。

   (2) 選出排序最頂尖的 6 個模型。

   (3) 將排名第一的模型指定為 Meta Learner（元學習器），負責整合次級模型的預測。

   (4) 將排名第 2 至第 6 名的模型作為 Base Learners（基學習器），作為 Stacking 模型的第一層。

(5) 賦予多步預測能力：為使模型具備「一次性預測未來多個時間點」的能力，系統會使用 scikit-learn 的 MultiOutputRegressor 封裝器，將前述的 Stacking 模型進行包裝，使其成為一個能夠同步輸出多個預測值的最終模型。

4. 最終訓練與儲存：

(1) 載入所有學習器對應的最佳參數，對這個已封裝完成的多步預測 Stacking 模型，使用 80% 的訓練集進行最終的擬合 (fit) 與訓練。

(2) 訓練完成後，使用 Python 的 joblib 套件將為該路段客製化訓練好的 Stacking 模型進行序列化，並以 .pkl 格式檔案儲存至指定目錄。

(四) 資料流

1. 輸入：特定路段的「特徵工程資料集」。

2. 輸出：一個訓練完成且序列化的 .pkl 模型檔案。此檔案將被「多進程預測處理模組」在執行即時預測時載入使用。

四、多進程（Multiprocessing）預測處理模組

多進程預測處理模組是整個系統的「運作引擎」，是執行線上、即時交通速率預測的核心服務。此模組被設計為一個自動化的後端流程，其主要職責是定時地整合最新蒐集的原始資料與歷史快取資料，調用預先訓練好的模型，對未來多個時間點的交通速率進行預測，並將最終結果儲存回資料庫。為能同時處理全國成百上千個路段的預測任務，本模組採用多進程並行處理架構，以最大化硬體資源利用率，確保預測任務能在指定的時間內完成。

(一) 觸發機制（Trigger Mechanism）：

本模組由 Ubuntu Crontab 進行排程觸發，設定為每 5 分鐘執行一次，自動啟動對未來 6 個時間點的預測任務。

(二) 多進程處理架構（Multiprocessing Architecture）

1. 為顯著提升系統效能，本模組利用 Python 的 multiprocessing 函式庫來達到多路段（id）的並行處理。

2. 當預測任務啟動時，主程式會將所有待預測的 id 分配給多個獨立子進程（Process）。

3. 每個 id 的完整預測流程（從資料讀取到結果儲存）都在其各自的進程中獨立運行，這樣可以有效避免資源衝突，並充分利用伺服器多核心 CPU 資源，確保資源利用率最大化。

(三) 預測執行流程

對於每一個獨立的子進程（即每一個 id），其預測流程包含以下步驟：

1. 資料準備與合併：進程首先會從 PostgreSQL 資料庫中提取兩種資料：由「資料蒐集模組」寫入的最新原始資料（Ori Data），以及上一次預測流程結束後儲存的預處理資料（Gap-Filled Data），後者作為歷史快取使用。系統會將這兩份資料結合起來，以保證時間序列的一致性與完整性，產生一份合併後的資料（Merged Data）。
2. 特徵工程：將合併後的 Merged Data 送至「資料預處理模組」。該模組會對其進行清洗、補值、聚合與特徵生成，最終回傳一個可用於預測的 pandas DataFrame 格式的特徵工程資料集。
3. 模型載入：使用 Python 的 joblib 套件，從硬碟中載入專為此 id 訓練好的 .pkl 模型檔案至記憶體中。
4. 直接多步預測 (Direct Multi-step Prediction)：系統採用「直接預測」策略，以達到高效率的預測。流程會從特徵工程資料集中提取出最新的特徵資料，並將其輸入至預載入的模型中。透過一次模型運算，即可同步快速的產出未來所有目標時間點（t+0 至 t+6）的速率預測值。
5. 結果儲存：預測完成後，進程會將包含預測值的結果回寫至 PostgreSQL 資料庫。同時，本次流程中新產生的預處理資料（Gap-Filled Data）也會被一併寫回資料庫，作為下一次 5 分鐘後預測流程的歷史快取資料。

(四) 資料流 (Data Flow)

1. 輸入：PostgreSQL 資料庫中的原始資料與快取資料、為特定 id 訓練好的 .pkl 模型檔案。
2. 輸出：包含 6 個未來時間點預測值的結果，以及更新後的快取資料（新的 Gap-Filled Data），寫回 PostgreSQL 資料庫。

五、API 服務模組

API 服務模組是整個預測系統的「數據出口」與「服務層」，扮演著將系統內部珍貴的預測數據轉化為外部價值的關鍵角色。核心功能是提供一個標準化、安全的 RESTful API 介面，允許經授權的外部應用程式或第三方服務，能夠以程式化的方式輕鬆查詢、介接與整合本系統產出的交通速率預測結果。此模組的穩定性與易用性，是決定整個系統能否成功落地應用的重要一環。

(一) 觸發機制（Trigger Mechanism）：由外部系統的 API 請求觸發，為被動服務。

(二) 運作流程

1. 接收外部系統的 HTTP 請求，並解析請求中的參數（如路段 id、查詢時間等）。
2. 根據解析後的參數，向 PostgreSQL 資料庫 查詢對應的預測資料。
3. 將從資料庫中檢索到的數據，格式化為通用的 JSON 格式。
4. 將 JSON 格式的數據作為 HTTP Response 回傳給請求方。

(三) 資料流

1. 輸入：外部系統發出的、包含認證金鑰與查詢參數的 HTTP API 請求。
2. 輸出：一個標準的 HTTP Response，其 Body 中包含格式化後的 JSON 或 CSV 預測數據。

3.3 軟硬體資源需求

為確保交通速率預測系統能夠穩定且高效運行，本章說明基本所需的軟體與硬體環境配置，內容涵蓋從開發到正式部署的建議，並針對不同規模的應用情境提供彈性的硬體參考規格。

一、軟體資源需求

本系統所規劃使用軟體，包括作業系統、程式語言、資料庫及所有 Python 套件模組，均為開源軟體，在標準使用下無需支付任何授權費用。因此系統的建置與擴展在軟體層面成本是相對極低的。

(一) 核心環境

1. 作業系統：Ubuntu 24.04.3 LTS Desktop
2. 程式語言：Python 3.10.14
3. 資料庫環境：PostgreSQL 17.6
4. 排程工具：Crontab

(二) 開發與版本控制：Visual Studio Code、Git、Conda

VSCode 為主要編輯器，Git 用於版本控制，Conda 用於管理 Python 環境與套件。

(三) Python 核心函式庫

1. 數據處理與計算
   (1) Pandas
   (2) NumPy
   (3) Requests

2. 機器學習與模型訓練

(1) Scikit-learn

(2) XGBoost / LightGBM

(3) Joblib

(4) Optuna

3. 資料庫連接

(1) psycopg2

(2) sshtunnel

(四) API 服務開發

1. Web 框架：Next.js (基於 React 的全端框架)

2. 後端開發：使用 Next.js 內建的 API Routes 功能，運行於 Node.js 環境並使用 node-postgres (pg) 函式庫與後端的 PostgreSQL 資料庫進行安全通訊，執行資料的查詢與操作。

(五) 授權與費用：本系統所規劃使用的上述所有軟體元件，均為廣泛使用的開源軟體，在標準使用條款下無需支付任何商業授權費用。

二、硬體資源需求

硬體需求與預計處理的路段數量、資料量及預測頻率密切相關。以下三種不同情境的建議規格，以供參考。

(一) 情境一：開發與測試環境，此規格適用於開發人員進行功能開發、單元測試或小規模數據驗證，預計處理路段數量在 100 個以下。

1. 架構：建議採用單機架構，所有服務（應用程式、資料庫）運行在同一台主機上。

2. CPU：現代 x86-64 架構，至少 8 核心處理器。

3. 記憶體：至少 32 GB RAM。

4. 儲存空間：至少 1 TB NVMe SSD，以確保快速的資料讀寫與模型載入速度。

5. 網路：標準 1 Gbps 網路介面。

(二) 情境二：中小型正式環境，此規格適用於正式部署，預計處理數百至一千個路段的即時預測，資料量與使用者請求量適中。

1. 架構：建議採用主從式架構，將應用程式主伺服器與資料庫伺服器分離，避免資源競爭。

2. 主伺服器：

(1) CPU：32 核心以上伺服器級 CPU。

(2) 記憶體：至少 128 GB RAM。

(3) 儲存空間：2 TB 以上企業級 NVMe SSD。

3. 資料庫伺服器：

(1) CPU：16 核心以上伺服器級 CPU。

(2) 記憶體：至少 64 GB RAM。

(3) 儲存空間：4 TB 以上企業級 SSD（可搭配高容量 HDD 進行備份）。

4. 網路：10 Gbps 網路介面卡與對應交換器。

(三) 情境三：大型正式環境（大規模部署），此規格為高運算效能的配置，適用於處理數千個以上路段的即時預測，需應對高併發 API 請求與龐大的資料吞吐量。

1. 架構：獨立主伺服器與資料庫伺服器，並可根據需求擴展為叢集架構。

2. 主伺服器（應用程式與模型運算）：

(1) CPU：AMD Ryzen TR PRO 9995WX (96 核 128 緒) 或同等級高核心數處理器 。

(2) 記憶體：128 GB (32GB*4) DDR5 3200 MHz 或更高。

(3) 儲存空間：系統碟使用 2TB M.2 SSD；快取與工作區使用 4TB SSD*2；長期資料儲存使用 20TB HDD*2。

(4) 網路頻寬：500 Mbps 以上。

3. 資料庫伺服器：

(1) CPU：Intel Core i7-13700 (16 核 24 緒) 或同等級高時脈處理器。

(2) 記憶體：128 GB (32GB*4) DDR4 3200 MHz 或更高。

(3) 儲存空間：主資料庫使用 2TB M.2 SSD；熱資料使用 4TB SSD*2；冷資料與備份使用 20TB HDD*2。

(4) 網路頻寬：500 Mbps 以上。

4. 網路設備：需配備對應吞吐量的路由器（Router）與交換器（Switch）。

表 1 硬體資源需求規劃建議彙整表

硬體規格	情境一： 開發與測試	情境二： 中小型正式環境	情境三：大型正式環境（大規模部署）
適用情境	功能開發、單元測試、小規模數據驗證（處理路段數量 ＜100）	正式部署，處理數百至一千個路段的預測（資料量與請求量適中）	正式部署，處理數千個以上路段的預測(需應對高併發與龐大資料吞吐量）
架構	單機架構（所有服務運行於同一主機)	主從式架構（應用程式與資料庫伺服器分離）	獨立主伺服器與資料庫伺服器(可根據需求擴展為叢集架構）
主伺服器 CPU	現代 x86-64 架構，至少 8 核心	32 核心以上伺服器級 CPU	AMD Ryzen TR PRO 9995WX（96 核 128 緒）或同等級
主伺服器 記憶體	至少 32GB RAM	至少 128 GB RAM	128 GB(32GB*4）DDR5 3200 MHz 或更高
主伺服器 儲存空間	至少 1 TB NVMe SSD	2 TB 以上企業級 NVMe SSD	● 系統碟：2TB M.2 SSD 快取 / 工作區：4TB SSD*2 ● 長期儲存：20TB HDD*2
資料庫伺服器 CPU	與主伺服器共用	16 核心以上伺服器級 CPU	Intel Core i7-13700 （16 核 24 緒）或同等級
資料庫伺服器 記憶體	與主伺服器共用	至少 64 GB RAM	128 GB (32GB*4) DDR4 3200 MHz 或更高
資料庫伺服器 儲存空間	與主伺服器共用	4 TB 以上企業級 SSD（可搭配高容量 HDD 進行備份）	● 主資料庫：2TB M.2 SSD 熱資料：4TB SSD*2 ● 冷資料 / 備份：20TB HDD*2
網路	標準 1Gbps 網路介面	10 Gbps 網路介面卡與對應交換器	● 頻寬：500 Mbps 以上 ● 設備：需配備對應吞吐量的路由器與交換器
硬體購置成本	至少 8 萬元	至少 40 萬元	至少 150 萬元

3.4 功能驗證與效果評估方法

一、功能驗證方法

功能驗證旨在確認系統的每一個模組與整體流程，是否都按照設計規格正確無誤地運行。建議採用分層的測試策略，從最小的程式單元到完整的端到端流程進行驗證。

(一) 單元測試 (Unit Testing)

單元測試主要目的為驗證系統中最小的獨立程式碼單元（如一個函式或一個類別方法）的功能是否如預期般正確。完整的單元測試能確保程式碼品質、降低後續整合難度與維護成本。

建議測試對象清單包含：

1. 資料蒐集模組
   (1) XML 解析函式：驗證此函式能否在接收到一筆標準的 XML 格式文字後，正確解析並提取出 id、tdate、avgspeed、direction 等關鍵欄位。測試應包含格式錯誤或欄位缺失的例外處理。
   (2) 資料格式化函式：驗證將解析出的字串資料（如時間字串）轉換為符合資料庫儲存格式（如 DATETIME 物件）的邏輯是否正確。
2. 資料預處理模組
   (1) 階層式補值策略：針對補值邏輯中的每一個階段（從週週期到鄰近值），設計專門的測試案例，給予包含特定類型缺口的假資料，驗證補值函式是否依序、正確的填補缺失值。
   (2) 資料聚合函式：驗證在輸入一段 1 分鐘資料後，聚合函式是否能正確計算出 5 分鐘的加權平均速率。
   (3) 特徵生成函式：
   - 延遲特徵：驗證能否為一個時間序列正確生成 t-1, t-2, t-3 等欄位。
   - 週期性特徵：驗證能否為一個跨越多天的時間序列，正確找到 t-7d, t-28d 的歷史資料。
   - 時間段熱編碼特徵：驗證輸入一天中橫跨六個不同時段的時間點，函式皆能回傳正確的 One-Hot 編碼向量。
3. Stacking 模型訓練模組
   (1) 資料劃分邏輯：驗證將資料集劃分為 80% 訓練集與 20% 測試集的函式，並且確保資料集不會被隨機打亂，用過去預測未來，藉此保證模型在測試集上的評估結果，較能真實反映其未來的預測能力。
   (2) 最佳模型篩選邏輯：給予一個包含多個模型及其 R^2 分數的假資料（字典或 JSON），驗證篩選函式能否正確排序並選出排名第一的 Meta Learner 與排名 2 至 6 的 Base Learners。
   (3) 多步預測能力封裝驗證：驗證 StackingRegressor 模型是否被

MultiOutputRegressor 正確的封裝，測試將檢查最終產出的模型物件類別是否正確，以確保模型具備「一次性預測多個時間點」的核心能力。

4. 多進程預測處理模組

(1) 資料合併邏輯：驗證將最新的原始資料（Ori Data）與歷史快取資料（Gap-Filled Data）合併的函式，能否產生一個時間連續且無重複的正確資料集。

(2) 直接多步預測驗證：驗證「直接多步預測」的核心邏輯，確保模型在接收到最新特徵資料後，能透過單次 predict() 呼叫，較高效率的產出一個結構完整（包含 t+0 至 t+6 共 7 個時間點）且時間戳對應正確的未來預測序列。

5. API 服務模組

(1) API Route 參數驗證：針對 Next.js 的 API Route 進行測試，驗證在接收到請求時，能否正確處理 req.query 中的參數（如 startDate, endDate），並對無效或缺失的參數回傳正確的錯誤狀態碼。

(2) 回應格式化函式：給予一個從資料庫查詢出的原始數據（如字典列表），驗證此函式能否將其轉換為 API 規格所定義的最終 JSON 格式。

綜合以上建議使用 supertest（API Route）進行測試。

(二) 整合測試 (Integration Testing)

整合測試的目的係為驗證不同模組之間介接的正確性，確保資料流在模組間的傳遞無誤。

建議測試清單包含：

1. 資料庫 I/O 與預處理整合測試

(1) 驗證流程：手動執行「資料蒐集模組」將一批測試用的原始資料寫入 PostgreSQL 資料庫。接著觸發「資料預處理模組」去讀取這批資料。

(2) 檢核點：確認預處理模組能成功連接資料庫，並能正確解析讀取出的資料欄位與型別，沒有發生因格式不符或權限問題導致的錯誤。

2. 模型產出與載入整合測試

(1) 驗證流程：在一個隔離的測試環境中，執行「模型訓練模組」為某個特定路段 ID 產出一個 .pkl 模型檔案。隨後，啟動「多進程預測處理模組」並讓其嘗試載入剛剛產出的模型。

(2) 檢核點：確認預測模組能根據路徑規則找到正確的 .pkl 檔案，並使用 joblib 成功將其反序列化為一個可用的模型物件，過程中沒有發生版本

不相容或檔案損毀的錯誤。

3. 預測流程資料處理鏈整合測試

(1) 驗證流程：在資料庫中準備好最新的原始資料（Ori Data）與歷史快取資料（Gap-Filled Data）。觸發「多進程預測處理模組」後，觀察其與「資料預處理模組」的互動。

(2) 檢核點：確認預測模組能將合併後的資料（Merged Data）正確傳遞給預處理模組，並成功接收預處理模組回傳的、包含所有特徵欄位的 pandas DataFrame，且該 DataFrame 的結構符合模型輸入的要求。

4. 訓練流程資料處理鏈整合測試

(1) 驗證流程：在資料庫中準備足夠的歷史資料後，觸發「模型訓練模組」。

(2) 檢核點：確認模型訓練模組能成功調用「資料預處理模組」，並由預處理模組從資料庫中提取、處理資料後，回傳格式正確的「特徵工程資料集」給訓練模組，以利其進行後續的模型訓練步驟。

5. 預測結果寫入與 API 讀取整合測試

(1) 驗證流程：執行一次完整的預測流程，讓「多進程預測處理模組」將預測結果寫入資料庫。然後立即透過「API 服務模組」提供的端點，查詢剛剛寫入的資料。

(2) 檢核點：確認 API 能夠即時查詢到由預測模組寫入的最新結果，驗證從預測產出到對外服務的整條路徑是通暢且無延遲的。

(三) 端到端測試 (End-to-End Testing)

端到端測試之目的為模擬真實世界的使用情境，驗證整個系統從起點到終點的完整自動化流程是否順暢。

建議測試情境清單包含如下：

1. 完整預測與服務流程驗證

(1) 前置作業：確認「資料蒐集模組」已正常運行一段時間，並已在 PostgreSQL 資料庫中累積最新的原始資料。

(2) 觸發執行：手動執行一次 5 分鐘週期的預測排程腳本。

(3) 過程監控：在執行期間觀察系統日誌（Log），確認多進程處理被正確啟動，且沒有任何模組（特別是資料預處理或模型預測）拋出錯誤或異常。

(4) 結果驗證 (資料庫)：待流程執行完畢後，登入 PostgreSQL 資料庫，檢查對應的資料表。確認目標路段 id 已生成未來 6 個時間點的預測值，且

本次流程產生的快取資料（Gap-Filled Data）也已正確寫入。

(5) 結果驗證 (API)：立即使用 API 測試工具，呼叫 API 服務來查詢剛剛做出預測的路段與時間點，比對 API 回傳的 JSON 結果是否與資料庫中的數據完全一致。

2. 新路段從訓練到預測的完整生命週期驗證

(1) 前置作業：在資料庫中準備好一個「全新路段 id」的充足歷史資料，並確保該 id 尚未有對應的 .pkl 模型檔案。

(2) 觸發訓練：手動執行「模型訓練模組」，指定對此新路段 id 進行模型訓練。

(3) 訓練結果驗證：待訓練結束後，檢查伺服器指定的目錄中，是否已成功生成對應此 id 的 .pkl 模型檔案與 .json 參數檔案。

(4) 觸發預測：執行一次 5 分鐘週期的預測排程。

(5) 預測結果驗證：檢查系統日誌，確認預測模組已成功載入新生成的模型，並順利完成預測。最後透過資料庫與 API 查詢，確認此新路段 id 已開始產出有效的預測資料。

3. 資料缺失情境下的系統恢復能力驗證

(1) 模擬故障：暫時停止「資料蒐集模組」的排程約 30 分鐘，人為地在原始資料中製造出一段時間缺口。

(2) 恢復與觸發：重新啟動資料蒐集排程，然後立即觸發一次 5 分鐘週期的預測排程。

(3) 過程驗證：觀察系統日誌，重點關注「資料預處理模組」是否正確啟動「階層式補值策略」來填補資料缺口。

(4) 結果驗證：檢查資料庫與 API，確認即使在原始資料存在缺口的情況下，系統依然能夠透過補值邏輯產出連續且合理的預測值，而非出現錯誤或空值。

4. 模型週期性更新流程驗證

(1) 前置作業：選定一個既有的路段 id，記錄其當前 .pkl 模型檔案的最後修改時間。

(2) 觸發重訓：執行「模型訓練模組」，對此路段 id 進行一次重新訓練（Retrain）。

(3) 模型更新驗證：待訓練結束後，檢查對應的 .pkl 模型檔案，確認其「最

後修改時間」已被更新，表示新模型已成功覆蓋舊模型。

(4) 新模型應用驗證：觸發一次 5 分鐘週期的預測排程，確認系統能夠載入並使用這個最新的模型版本順利完成預測。

(四) API 測試

API 測試的目的為確保 API 服務模組的穩定性與可靠性，驗證其是否能正確處理各種請求。

建議測試案例清單包含：

1. 針對端點：GET /api/gvp/line-chart/speed/road-ids

(1) 功能與正確性測試

- 測試案例 1.1：
  - 操作：使用一個有效的 apiKey 發送請求。
  - 預期結果：伺服器應回傳 200 OK 狀態碼，Content-Type 標頭為 application/json，且回應內文為一個包含 roadIdData 陣列的有效 JSON 物件。
- 測試案例 1.2：
  - 操作：使用一個有效的 apiKey 並加上 format=csv 參數發送請求。
  - 預期結果：伺服器應回傳 200 OK 狀態碼，且 Content-Type 標頭應為 text/csv 或相應的格式。

(2) 邊界與錯誤處理測試

- 測試案例 1.3：
  - 操作：發送一個缺少 apiKey 參數的請求。
  - 預期結果：伺服器應回傳客戶端錯誤狀態碼（如 400 Bad Request 或 401 Unauthorized），並附帶說明錯誤原因的訊息。
- 測試案例 1.4：
  - 操作：使用一個格式正確但無效（偽造）的 apiKey 發送請求。
  - 預期結果：伺服器應回傳 401 Unauthorized 狀態碼，拒絕此次請求。

2. 針對端點：GET /api/gvp/line-chart/speed

(1) 功能與正確性測試

- 測試案例 2.1：
  - 操作：使用有效的 apiKey、存在的 roadId、以及格式正確且範圍

內有資料的 startDate 和 endDate 發送請求。

- 預期結果：伺服器應回傳 200 OK 狀態碼，且回應的 JSON 內容與資料庫中對應的預測數據完全一致。

(2) 邊界與錯誤處理測試

- 測試案例 2.2（缺少必要參數）：
  - 操作：分別嘗試缺少 startDate、endDate 或 roadId 的請求。
  - 預期結果：每次請求都應回傳 400 Bad Request 狀態碼，並明確提示哪個必要參數缺失。
- 測試案例 2.3（無效的 Road ID）：
  - 操作：使用一個資料庫中不存在的 roadId 發送請求。
  - 預期結果：伺服器應回傳 404 Not Found 狀態碼，且錯誤訊息應為 {"error":"無資料，roadID 可能不正確或該時間沒有資料。"}。
- 測試案例 2.4（無效的日期格式）：
  - 操作：使用不符合 YYYY-MM-DD HH:mm 格式的日期字串（如 2025-10-03 或 10/03/2025）發送請求。
  - 預期結果：伺服器應回傳 400 Bad Request 狀態碼，並提示日期格式錯誤。
- 測試案例 2.5（不合邏輯的日期範圍）：
  - 操作：發送一個 endDate 早於 startDate 的請求。
  - 預期結果：伺服器應回傳 400 Bad Request 狀態碼，並提示日期範圍不合邏輯。
- 測試案例 2.6（查詢範圍無資料）：
  - 操作：使用有效的 roadId，但查詢一個確定沒有任何預測資料的時間區間。
  - 預期結果：伺服器應回傳 404 Not Found 狀態碼，並附帶相應的錯誤訊息。

3. 非功能性測試

壓力測試：

(1) 操作：使用 JMeter、Locust 等壓力測試工具，模擬大量使用者在短時間內同時對 API 端點發送請求。

(2) 檢核點：觀察在高併發情境下 API 的平均回應時間、錯誤率及伺服器的

CPU、記憶體使用率是否維持在可接受的範圍內，以確保服務的穩定性。

二、效果評估方法

效果評估旨在量化分析系統的表現，主要分為「模型準確度」與「系統運行效能」兩大維度。

(一) 模型準確度評估

1. 評估方式：在模型訓練階段，使用預留的 20% 測試集對已訓練好的模型進行預測，並將預測值與真實值進行比較，計算以下關鍵評估指標：
2. 評估指標 (Evaluation Metrics)：
   (1) 平均絕對誤差 (MAE, Mean Absolute Error)：$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$，衡量預測誤差的平均絕對大小。
   (2) 均方誤差 (MSE, Mean Squared Error)：$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$，對較大誤差給予更高權重。
   (3) 均方根誤差 (RMSE, Root Mean Squared Error)：$RMSE = \sqrt{MSE}$，MSE 的平方根，使誤差單位與原始值相同。
   (4) 平均絕對百分比誤差 (MAPE, Mean Absolute Percentage Error)：$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$，衡量預測誤差佔真實值的平均百分比。
   (5) 決定係數 (R^2, R-Squared)：衡量模型對數據變異的解釋能力，越接近 1 代表模型配適度越好。
3. 過擬合（Overfitting）檢驗：同時需計算模型在訓練集上的評估指標。若模型在訓練集的表現（如 R^2 分數接近 0.99）遠超在測試集上的表現，則可能存在過擬合問題。
4. 合格標準：需根據實際業務需求與歷史數據的基準表現，設定各指標的合格範圍。例如，可設定驗收標準為 MAE < 2 且 > 0.75。

(二) 系統效能評估

1. 評估方式：透過壓力測試與資源監控，評估系統在不同負載下的運行效率與穩定性。
2. 擴充性測試：測試系統在處理不同數量級的路段（例如 100、1,000、10,000 個 id）時，「多進程預測處理模組」完成一次完整預測流程所需的總時間。
3. 資源使用監控：在預測排程運行的尖峰期間，記錄主伺服器與資料庫伺服器的 CPU 及記憶體使用率。
4. 合格標準：

(1) 執行時間：一次完整的預測流程（包含所有路段）必須在指定的時間範圍內完成（例如 4.5 分鐘內），以確保不會與下一次 5 分鐘的排程重疊。

(2) 資源利用率：系統在長期運行下，平均資源使用率應保持在合理範圍內（例如，CPU 長期使用率 ＜60%，記憶體使用率 ＜50%），以保留應對突發流量的餘裕並確保系統穩定。

3.5 常見問題與風險因應

任何自動化系統在長期維運中都會面臨各種挑戰。本節旨在透過建置經驗說明交通速率預測系統在數據、模型、效能與服務層面可能遇到的常見問題，並提供具體的技術應對策略。

一、來源資料品質問題

(一) 風險描述：外部單位 API 提供的原始交通資料可能存在大量的缺失值（例如，特定路段長達數小時無數據）、時間延遲或格式錯誤。在極端情況下，資料缺失比例可能高達 30% 以上（例如高速公路 TDCS 之 VD 資料）。

(二) 潛在影響

1. 預測不準：不完整的數據會直接導致資料預處理與特徵生成的結果失真，進而嚴重影響模型的預測準確性。
2. 訓練失敗：若用於模型訓練的歷史資料品質過差，可能導致模型無法收斂或學到錯誤的模式。

(三) 因應對策

1. 強化資料補值：系統內建的「六階段階層式補值策略」是應對此問題的核心防線，能最大程度利用歷史週期性與鄰近數據填補缺口。
2. 建立資料品質監控：開發一個獨立的監控腳本，每日定時掃描資料庫中前一日的原始資料完整率。若特定路段的資料缺失率超過預設閾值（例如 20%），系統應自動發出警報（如 Email、Slack 通知）給維運人員。
3. 日誌記錄與標記：在資料預處理流程中，詳細記錄每個路段的補值比例。對於補值比例過高的預測結果，可在資料庫中增加一個信賴度標記（Confidence Flag），供後續使用者參考。

二、模型效能衰退（Model Drift）

(一) 風險描述：交通模式並非一成不變，可能因新的道路開通、交通政策改變或季節性因素而發生緩慢變化。這會導致既有模型逐漸無法適應最新的數據模式，即「模型飄移」(Model Drift)，使得預測準確性隨時間下降。

(二) 潛在影響：系統的預測結果與現實情況的誤差會越來越大，失去實用價值。

(三) 因應對策

1. 建立模型效能監控迴圈：系統應建立一個自動化的監控流程，每日或每週將儲存的預測結果與事後蒐集到的真實速率值進行比對，並重新計算 MAE、R^2 等評估指標。
2. 設定警報閾值：為關鍵評估指標設定衰退警報閾值（例如，若連續一週的 MAE 比模型訓練時高出 30%）。一旦觸發閾值，系統應自動通知相關人員，提示可能需要重新訓練模型。
3. 執行週期性重新訓練：如「模型訓練模組」中所述，應執行每年至少一次的週期性重新訓練，將最新的數據納入訓練集中，確保模型能適應最新的交通模式。

三、資料庫效能瓶頸

(一) 風險描述：隨著系統每日 24 小時不間斷地寫入每分鐘的資料，資料庫中的數據量將會爆炸性增長。若無優化，數年後單一資料表的規模可能達到數十億筆，導致查詢速度急劇下降。

(二) 潛在影響

1. API 回應緩慢：使用者透過 API 查詢歷史資料時，等待時間過長，影響使用者體驗。
2. 預測流程延遲：預測模組在讀取歷史資料進行特徵工程時，因查詢過慢而導致整個 5 分鐘的預測任務超時。

(三) 因應對策

1. 資料庫索引優化：確保在資料表上針對最常見的查詢欄位（即 id, tdate, direction）建立較高效率的複合索引，加速資料檢索。
2. 資料表分區 (Partitioning)：本系統採用基於 PostgreSQL 的時序資料庫擴充套件 TimescaleDB，將核心資料表轉換為 Hypertable。透過以 tdate（事件時間）欄位為分區鍵，系統能達到自動化的每日分區（Daily Partitioning）。此設計讓查詢可自動「裁剪 (Prune)」掉不相關的時間區塊，僅需掃描目標日期的資料，從而確保查詢效能不會因歷史資料總量的增加而下降。

3. 建立資料封存與清理機制：制定資料生命週期策略。例如，將超過三年的歷史資料從線上正式資料庫中，定期遷移至成本較低的「冷儲存」(如另一台歸檔資料庫或雲端儲存)，以維持線上資料庫的精簡與高效。

四、系統服務與流程中斷

(一) 風險描述：線上服務可能因程式錯誤、流量突增或依賴的外部服務（如資料源 API）中斷而發生故障。

(二) 潛在影響：無法生成最新的預測、API 服務無回應，導致系統服務中斷。

(三) 因應對策

1. 強化程式碼健壯性：在所有關鍵流程中，特別是外部 API 請求、資料庫操作等，都應使用 try-except 語句進行完整的錯誤處理與日誌記錄，避免單一錯誤導致整個進程崩潰。
2. API 服務水平擴充：若 API 請求量過大，應在多台主機上部署 API 服務模組，並在前端使用負載平衡器（Load Balancer）來分發流量，確保高可用性。
3. 實施 API 速率限制（Rate Limiting）：為防止惡意或異常的高頻請求拖垮 API 服務，應在 API 層實施速率限制策略，例如限制單一 apiKey 每分鐘的請求次數。
4. 建立告警機制：針對關鍵流程（如預測排程、API 服務）建立心跳監控。一旦服務無回應或排程執行失敗，監控系統應立即透過 Email 或即時通訊軟體發出警報。

五、AI 決策風險評估管理措施

「針對 AI 預測之潛在風險，應實施動態風險評估：

(一) 預警機制：若連續 3 個週期預測偏差值超過 50%（Model Drift），系統應自動標記該數據為『低信賴度』並通知維運人員。

(二) 容錯機制：當 AI 預測模組因算力瓶頸或數據異常導致延遲時，API 服務應具備降級機制，改採歷史同期平均值輸出，確保外部介接系統之穩定運作。

第四章 整合與落地應用作法

4.1 整合應用模式

為使本預測系統的成果能順利融入交通管理單位既有作業流程與資訊系統，建議在預測系統建置完成後，可採用以下整合模式，並結合實際情境進行應用。

一、API 介接模式

此模式是彈性最高、耦合度最低且最符合現代軟體架構的整合方式。由交通管理單位的既有資訊系統(如交控平台)作為客戶端，主動向本預測系統所開發的 API 服務發起請求以獲取數據。

(一) 整合架構：交控中心應用系統→ API Gateway →速率預測服務(Next.js)

(二) 優勢

1. 技術解耦：兩邊系統可獨立開發與升級，只要 API 規格不變，彼此不互相影響。
2. 安全性高：可透過 API Gateway 集中管理認證、授權與流量。
3. 標準化：基於通用的 RESTful 與 JSON 標準，易於介接。

(三) 技術重點

1. 認證與加密
   (1) 認證機制：建議採用 apiKey、JWT (JSON Web Token) 或 OAuth2 等標準機制進行客戶端身份驗證。
   (2) 傳輸加密：所有 API 通訊皆應強制使用 HTTPS/TLS 進行全程加密，確保資料傳輸安全。
2. 資料查詢與回傳格式
   (1) API 呼叫範例：使用 GET 方法，並透過 Query String 傳遞參數。
   (2) 回傳格式範例 (JSON Schema)：回傳的 JSON 應具備清楚且固定的結構，例如：

```
{
  "road_id": "372351413",
  "actual_avgspeed": 44,
  "predict_avgspeed": 39.7195,
  "predict_time": "2025-01-01 00:00:00"
}
```

3. 流量管理與錯誤處理
   (1) 速率限制 (Rate Limit)：應在 API Gateway 或服務本身設置速率限制(如

本系統設計為每分鐘 600 次請求)，避免因客戶端程式錯誤或惡意請求導致服務過載。

(2) 錯誤碼設計：定義標準的 HTTP 錯誤碼，便於客戶端進行錯誤處理。

- 400 Bad Request：一個或多個請求參數遺失或無效 (例如：日期格式錯誤、endDate 早於 startDate)。
- 403 Forbidden：已通過認證，但沒有存取此資源的權限。請確認您的 apiKey 是否有足夠權限。
- 404 Not Found：請求的 roadId 不存在。
- 405 Method Not Allowed：請求的方法不被允許，或此資源不支援該 HTTP 方法（例如僅支援 GET，但使用 POST）。
- 429 Too Many Requests：超過請求限制。
- 500 Internal Server Error：伺服器內部發生錯誤，無法完成請求。請稍後再試，或聯絡系統管理員。

4. 版本管理：建議在 API 路徑中加入 URL 版號（如 /api/v1/prediction），當未來 API 規格有重大變更時，可透過升級版號（/api/v2/...）的方式，確保舊有系統的相容性。

二、資料庫層級整合模式

在特定需要大量、高速進行資料交換的內部系統間，可考慮此模式。此允許既有系統（如交控平台）直接存取本預測系統的 PostgreSQL 資料庫。

(一) 優勢：對於大量數據的批次讀取，其傳輸效能優於 API。

(二) 技術重點：

1. 連線方式：必須設定一個專供外部查詢的「唯讀副本（Read-Only Replica）」，避免查詢壓力影響主資料庫的寫入效能。所有連線應透過 SSH Tunnel 搭配 SSL 進行加密，確保連線安全。
2. 授權與權限控管：在資料庫層級實施角色為基礎的存取控制（Role-Based Access Control, RBAC），為整合系統建立專屬的唯讀帳號，並精確授權其只能查詢特定的資料表或視圖（View）。
3. 資料表結構同步：若預測系統的資料庫表結構需要更新，建議使用 Alembic Migration 等 schema 版本控制工具來管理變更，並提前通知所有依賴此資料庫的系統。
4. 查詢效能與範例：提供最佳化的查詢 SQL 範例給介接單位，並確保相關欄位

已建立索引。SELECT * FROM hyper_predictions WHERE id='372351413' AND tdate >= NOW() AND tdate < NOW() + INTERVAL '30 minutes' ORDER BY tdate;

5. 備援與異常處理：應設計備援機制，例如當唯讀副本失效時，應用程式能自動切換至另一個備用副本，或在緊急情況下降級（fallback）採用 API 介接模式。

三、資料交換平台（Data Exchange Platform, DEP）模式

若交通管理單位內部已有「交通資料交換平台」或「市級資料中樞」，例如 TDX 等，此模式能促進資料的跨單位共享。此模式由本預測系統作為資料生產者，定期將最新的預測結果以標準檔案格式（如 JSON, CSV, Parquet）主動推送（Push）至指定的資料交換平台，再由平台統一對外提供服務。

(一) 整合架構：預測系統 → (FTP/SFTP/MQ) → 資料交換平台 → 各應用系統

(二) 優勢

1. 集中化管理：資料的授權、發布與流量管控由資料平台統一負責，預測系統僅需專注於生產數據。
2. 跨系統共享：一份數據可供多個不同單位（如交通局、警察局、都發局）的系統訂閱使用，擴大資料價值。

(三) 技術重點

1. 傳輸方式：可透過 FTP/SFTP 進行檔案上傳，或透過訊息佇列（Message Queue，如 RabbitMQ, Kafka）發布訊息。
2. 格式與頻率：需與平台管理者共同定義標準的檔案命名規則、資料欄位格式，並確立更新頻率（例如：每 5 分鐘產出一份包含所有路段預測的 JSON 檔案）。

4.2 系統維運建議

為確保本系統在建置完成後能長期穩定地提供高品質服務，建議應建立一套涵蓋數據、功能、模型到設備的完整維運計畫。

一、資料來源監控

建立自動化的資料品質監控模組，每日定時執行腳本，檢測前一日從外部 API 蒐集的原始資料。檢核項包含資料的完整性（缺失率是否超過閾值）、正確性（速率值是否在合理範圍）與即時性（資料是否延遲過久）。若發現異常（如連續一小時無數據寫入），系統應立即透過 Email 或即時通訊軟體通報資料管理人員處理。

二、系統功能檢測

設置自動化的系統健康檢查（Health Check）排程，每日在離峰時段執行。此排程可執行一個簡化版的端到端測試，例如：自動觸發一個測試路段的預測，並透過 API 查詢結果是否符合預期，以驗證核心流程的通暢。若健康檢查失敗，應立即通報系統維運人員，進行問題排查。

三、模型效能追蹤

建立模型效能定期檢測與重訓機制。每週或每月自動執行腳本，比對系統過去一段時間的預測值與事後蒐集到的真實值。持續追蹤 MAE、R^2 等關鍵評估指標的變化趨勢。若模型效能指標持續下滑並觸發警報閾值（例如 MAE 比基準高出 30%），則應通知相關人員評估是否需要啟動模型的重新訓練。

四、設備保養檢查

為確保系統硬體設施的長期穩定與可靠，應建立一套兼具自動化健康監控與週期性實體巡檢的雙軌保養計畫。自動化監控應透過部署監控代理程式，對伺服器的關鍵指標進行全天候追蹤，例如硬碟健康度、磁碟空間使用率、CPU 溫度及電源狀態，並設定警報閾值以即時通報異常。作為補充，建議每季進行的實體巡檢則專注於軟體無法偵測的物理層面，包含機房環境、設備散熱、線路連接的檢查，並輔以伺服器管理日誌的人工審查，以構成完整的硬體保養體系。

五、異常即時通報

將上述所有監控項目（資料、功能、模型、硬體）的警報，整合至一個統一的告警平台。設定明確的觸發條件，如「資料庫連線失敗」、「預測腳本執行超過 4.5 分鐘」、「CPU 使用率持續高於 80%」等。一旦觸發，系統應能立即透過 Email 或即時通訊軟體（如 Slack, Line）發出警報給對應的權責人員。

六、版本更新管理

為確保系統在功能迭代與安全升級過程中的穩定性，應建立嚴謹的版本更新管理流程。此流程應以 Git Flow 等標準化分支模型為基礎，並導入持續整合與持續交付（CI/CD）的實踐。所有程式碼變更在合併前需經審查，並透過自動化管線在預備（Staging）環境中完成所有單元、整合及端到端測試。唯有通過完整驗證的版本，才能被批准部署至正式（Production）環境。此標準化流程不僅適用於應用程式的功能更新與錯誤修復，同樣也應用於本預測系統之機器學習模型（.pkl 檔案）的部署，以及作業系統和相依套件的定期安全性更新，確保所有變更都經過嚴格的品質控管。