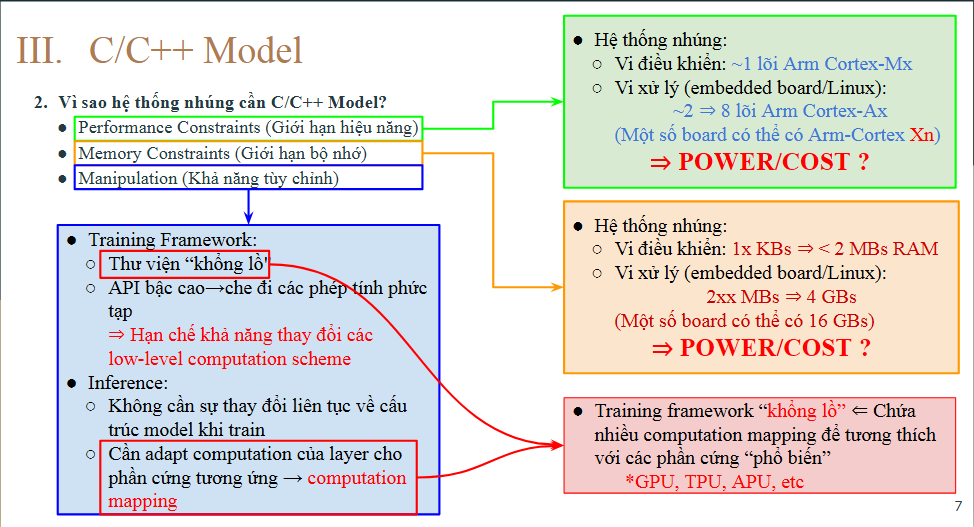
# **Vì sao hệ thống nhúng cần C/C++ Model?**

****

## **1. Ràng buộc về hiệu năng và bộ nhớ**

**(Performance & Memory Constraints):**

* Hệ thống nhúng thường chạy trên:
  + Vi điều khiển (MCU): chỉ ~1 lõi Arm Cortex-Mx.
  + Vi xử lý nhúng (SBC): 2–8 lõi Arm Cortex-Ax (nhiều nhất cũng chỉ ~16 GB RAM).
* Điều này giới hạn:
  + **Tốc độ xử lý** → không đủ cho các mô hình deep learning lớn.
  + **Bộ nhớ** → không tải nổi các framework AI khổng lồ.

## **2. Khả năng tùy chỉnh và điều khiển thấp của các framework huấn luyện:**

**(Memory Constraints)**

* Các thư viện AI như TensorFlow chứa nhiều API bậc cao → khó can thiệp vào tầng tính toán thấp (low-level computation).
* Khi đưa mô hình xuống thiết bị nhúng, không thể thay đổi cấu trúc như lúc huấn luyện.
* Việc cần làm là **mapping lại computation của từng layer** sao cho tương thích với phần cứng cụ thể → điều này cần **C/C++** để kiểm soát chi tiết.

## **3.** **Inference khác với training:**

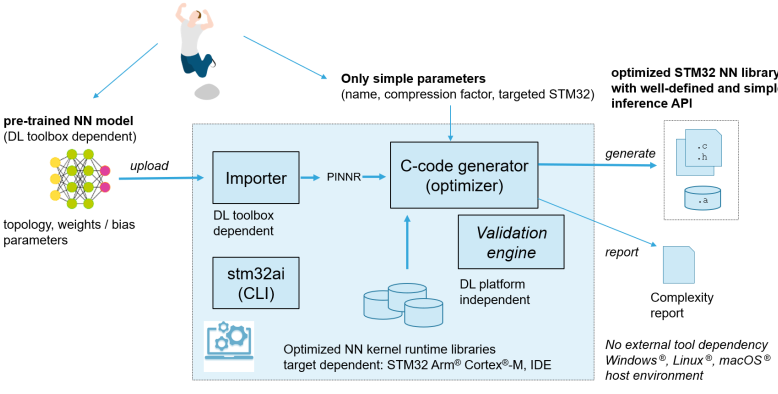
**(Manipulation)**

* Huấn luyện: cần GPU/TPU mạnh, API trừu tượng.
* Suy luận (inference): yêu cầu nhẹ, cần tối ưu **tốc độ + tiêu thụ điện năng** → chỉ thực hiện trên phần cứng có giới hạn (Cortex-M, Cortex-A).
* ⇒ Cần mô hình nhẹ (quantized, pruned), viết bằng C/C++ để tối ưu.

# **Phương pháp chuyển đổi X-CUBE-AI**

Cấu hình MPU để đảm bảo tính toàn vẹn của data – khi dùng chung với DMA: [[**Link**](https://tapit.vn/tim-hieu-memory-protection-unit-mpu-vi-xu-ly-arm-cortex-m/)]

**Quy trình chuyển đổi mô hình (X-CUBE-AI Flow)**



## **1. Original Model**

Là mô hình đã được **huấn luyện trên máy tính**, sử dụng các framework phổ biến như:

* **Keras/TensorFlow (.h5, .pb)**
* **ONNX, TFLite**

Bao gồm:

* Cấu trúc mô hình (topology)
* Trọng số (weights)
* Tham số bias, các hàm kích hoạt…

## **2. Conversion Engine**

Thực hiện bởi công cụ **STM32AI CLI** hoặc qua giao diện đồ họa của **STM32CubeMX**.

Chức năng:

* **Phân tích mô hình gốc**.
* **Chuyển topology sang code C/C++ tối ưu hóa** tương thích với nhân xử lý STM32.
* **Tự động mapping computation** từ các layer (Conv, ReLU, Dense...) sang thư viện ARM CMSIS-NN hoặc thư viện riêng của ST.
* Hỗ trợ **quantization**, chọn kiểu dữ liệu phù hợp (float32 → int8).

**Kết quả**: sinh ra **mã C/C++ và cấu hình cần thiết** để đưa vào project STM32.

## **3. Output Library**

Là **thư viện nhúng được sinh tự động** từ Conversion Engine.

Bao gồm:

**Thư viện C/C++** có thể chèn trực tiếp vào firmware STM32.

**Hàm inference (STM32 NN API)** đơn giản, dễ tích hợp.

**Báo cáo chi tiết** về:

* Mức sử dụng bộ nhớ (RAM/Flash)
* Thời gian suy luận
* Số chu kỳ CPU
* Độ chính xác suy luận (validation)

Đây là phần **cuối cùng bạn tích hợp vào chương trình chính (main.c / app.c)** của thiết bị.

**Các tính năng nâng cao đi kèm:**

* Quantization: Giảm kích thước mô hình, tăng tốc độ, giảm RAM (dùng int8 thay vì float32).
* Memory allocation (5 vùng ngoài, 6 vùng trong): cho phép cấu hình cách cấp phát bộ nhớ phù hợp với phần cứng.
* Dynamic/static memory trade-off: lựa chọn giữa hiệu năng cao và sử dụng bộ nhớ tối ưu.