ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Lê Văn Tuệ - 22022127

TỐI ƯU TÍNH TOÁN SONG SONG CHO MÔ HÌNH NHẬN DIỆN THIẾT BỊ ĐIỆN TRONG MẠCH ĐIỆN DÂN DỤNG

ĐỒ ÁN NGÀNH ĐẠI HỌC CHÍNH QUY

Ngành: Kĩ thuật máy tính

HÀ NỘI – 2025

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Lê Văn Tuệ - 22022127

TỐI ƯU TÍNH TOÁN SONG SONG CHO MÔ HÌNH NHẬN DIỆN THIẾT BỊ ĐIỆN TRONG MẠCH ĐIỆN DÂN DỤNG

ĐỒ ÁN NGÀNH ĐẠI HỌC CHÍNH QUY

Ngành: Kĩ thuật máy tính

Cán bộ hướng dẫn: TS. Nguyễn Ngọc An

HÀ NỘI – 2025

TÓM TẮT

Đồ án này trình bày giải pháp triển khai Hyperdimensional Computing (HDC) — một mô hình tính toán lấy cảm hứng từ não bộ, trên nền tảng thiết bị nhúng tài nguyên hạn chế, cụ thể là vi điều khiển ESP32 dual-core, nhằm giải quyết bài toán giám sát tải không xâm nhập (NILM). NILM đòi hỏi khả năng xử lý chuỗi thời gian công suất/dòng điện phức tạp trong thời gian thực tại biên mạng (Edge AI). Do kiến trúc tính toán nhẹ và tính song song bẩm sinh của HDC, mô hình đã được tối ưu hóa bằng cách chia nhỏ pipeline xử lý thành hai tác vụ độc lập, ghim lần lượt lên Core 0 và Core 1 của ESP32 thông qua cơ chế FreeRTOS Task Pinning. Kết quả thực nghiệm cho thấy kiến trúc song song này giúp tăng tốc độ tính toán (Speedup S) lên gần gấp 2 lần ($S \approx 1.8 - 2.0$) so với chạy trên một lõi duy nhất, giảm đáng kể độ trễ xử lý (latency). Tỷ lệ phần việc song song hóa (P) của HDC pipeline đạt mức cao, khẳng định HDC là một giải pháp tối ưu về hiệu suất năng lượng và tốc độ cho các ứng dụng AI trên thiết bị nhúng đa nhân.

Từ khóa: *HDC, IoT, TinyML, ESP32 dual-core.*

Mục lục

Danh sách hình vẽ

Danh sách bảng

Các từ viết tắt

HDC: Hyperdimensions Computing – Tính toán nhiều chiều .

IOT: Internet of Things – Mạng của các thiết bị.

FreeRTOS: Free real-time operating system – Hệ điều hành thời gian thực

NILM: Non-Intrusive Load Monitoring – Giám sát tải không xâm nhập

Edge AI: Edge Artificial Intelligence – Trí tuệ nhân tạo tại thiết bị biên

Chương 1. Giới thiệu

1.1. Bối cảnh

Non-Intrusive Load Monitoring (NILM)[1] là kỹ thuật phân tích dữ liệu điện năng tổng hợp để xác định trạng thái và mức tiêu thụ của từng thiết bị riêng lẻ. Với sự bùng nổ của IoT, việc triển khai NILM trực tiếp trên các thiết bị biên (Edge Computing) [3] như ESP32 đang trở thành xu hướng tất yếu nhằm giảm tải cho server và bảo vệ quyền riêng tư người dùng.

Hyperdimensional Computing (HDC)[2] nổi lên như một hướng đi đầy hứa hẹn cho NILM tại biên. Dựa trên việc mô phỏng các vector số chiều lớn (Hypervectors), HDC cho phép thực hiện các tác vụ học máy bằng các phép toán logic đơn giản, tiêu tốn ít năng lượng nhưng vẫn đảm bảo khả năng kháng nhiễu cao – một yếu tố cực kỳ quan trọng trong môi trường tín hiệu điện phức tạp.

1.2. Động lực nghiên cứu

Mặc dù HDC được đánh giá là mô hình nhẹ, việc thực thi các phép toán trên Hypervector hàng nghìn chiều (ví dụ $D = 10{,}000$) trên vi điều khiển vẫn đối mặt với thách thức về độ trễ (latency). Các triển khai HDC thông thường trên ESP8266 hoặc ARM Cortex-M cấp thấp thường chỉ chạy tuần tự, chưa tận dụng được các đặc tính phần cứng tiên tiến. ESP32 sở hữu kiến trúc lõi kép (Dual-core) và tập lệnh 32-bit mạnh mẽ. Động lực của nghiên cứu này là tìm cách tăng tốc mô hình HDC bằng cách song song hóa quy trình và tối ưu hóa mức bit (bit-level optimization). Việc tăng tốc này không chỉ giúp nhận dạng thiết bị nhanh hơn mà còn cho phép triển khai các mô hình NILM phức tạp hơn (nhiều thiết bị hơn) ngay trên một vi điều khiển giá rẻ.

1.3. Các nghiên cứu liên quan

- **HDC trong nhận dạng mẫu**: Các phương pháp mã hóa dữ liệu chuỗi thời gian thành Hypervector đã được chứng minh hiệu quả trong các bài toán cảm biến[4].
- **NILM trên hệ thống nhúng**: Các nghiên cứu sử dụng CNN/LSTM nén (compressed) hoặc SVM trên MCU; tuy nhiên thường gặp khó khăn về tài nguyên bộ nhớ[5].
- **Tăng tốc thuật toán trên ESP32**: Các kỹ thuật sử dụng FreeRTOS để phân bổ tác vụ lên Core 0 và Core 1, hoặc sử dụng tập lệnh SIMD/Bit-manipulation để tối ưu hóa việc xử lý tín hiệu số.

– **Binary HDC**: Xu hướng chuyển đổi Hypervector sang dạng nhị phân để thay thế các phép toán số thực bằng phép XOR và Popcount, giúp tăng tốc độ xử lý lên nhiều lần.[6]

1.4. Thách thức và khoảng trống nghiên cứu

Một số khoảng trống trong nghiên cứu:

– Nghẽn cổ chai tính toán: Các phép toán Encoding (mã hóa tín hiệu điện) và Search (so sánh với thư viện mẫu) vẫn chiếm dụng nhiều chu kỳ CPU nếu thực hiện tuần tự.

– Khai thác phần cứng: Chưa có nhiều nghiên cứu tập trung vào việc tối ưu hóa đặc thù cho kiến trúc Dual-core của ESP32 trong bài toán NILM bằng HDC.

– Độ trễ thời gian thực: Trong NILM, việc phản hồi chậm có thể dẫn đến mất mát các sự kiện đóng/ngắt thiết bị diễn ra liên tục. Cần một cơ chế đảm bảo thời gian thực (real-time) chặt chẽ giữa việc lấy mẫu và suy luận.

1.5. Mục tiêu nghiên cứu

Nghiên cứu hướng đến:

– Phát triển mô hình Binary HDC tối ưu cho NILM: Chuyển đổi các đặc trưng dòng điện/điện áp thành các Hypervector nhị phân.

– Tăng tốc thực thi trên ESP32: Áp dụng kỹ thuật song song hóa (Parallelization) dựa trên Dual-core và tối ưu hóa phép toán Popcount mức bit.

– Xây dựng Pipeline tự động: Từ huấn luyện trên Python đến sinh mã nguồn C++ (C++ Code Generation) đã được tối ưu cho ESP-IDF/Arduino.

– Đánh giá hiệu năng: Đo lường tốc độ tăng tốc (Speed-up ratio), độ chính xác nhận dạng và mức tiêu thụ năng lượng so với các phương pháp truyền thống.

1.6. Đóng góp chính

– Đề xuất kiến trúc HDC song song: Chia tách nhiệm vụ lấy mẫu/tiền xử lý (Core 0) và mã hóa/suy luận (Core 1) để tối đa hóa throughput[7].

– Thực nghiệm NILM thực tế: Triển khai hệ thống nhận dạng trạng thái thiết bị (Binary-state) với độ trễ thấp hơn đáng kể so với các nghiên cứu trước đó.

– Phân tích sự đánh đổi: Cung cấp số liệu chi tiết về mối quan hệ giữa số chiều Vector, thời gian thực thi trên ESP32.

1.7. Cấu trúc luận văn

Phần còn lại của luận văn được tổ chức như sau:

- Chương 2 trình bày bài toán NILM, mô hình hóa hệ thống và kiến trúc tổng thể.
- Chương 3 mô tả quá trình lựa chọn đặc trưng và xây dựng mô hình.
- Chương 4 trình bày thiết lập thí nghiệm và các kịch bản kiểm thử.
- Chương 5 đánh giá kết quả thực nghiệm về độ chính xác, thời gian suy luận và mức sử dụng tài nguyên.
- Chương 6 tổng kết kết quả đạt được và đề xuất hướng phát triển trong tương lai.

Chương 2. Mô hình bài toán - Mô hình giải pháp

2.1. Mô hình bài toán

2.1.1. Nhiệm vụ đích: phân loại đa nhãn trạng thái thiết bị điện

Nhiệm vụ trọng tâm của nghiên cứu là nhận diện đồng thời trạng thái hoạt động của tổ hợp 7 thiết bị điện gia dụng (tập dữ liệu VNDALE-2). Thay vì chỉ phân loại bật/tắt đơn lẻ, hệ thống phải giải quyết bài toán phân loại đa nhãn với $2^7 - 1 = 127$ trạng thái tổ hợp khác nhau.Dữ liệu đầu vào bao gồm các đặc trưng điện toán ($I_{rms}, V_{rms}, P, Q, S, PF$) được trích xuất từ tín hiệu gốc có tần số 600 Hz. Thách thức lớn nhất nằm ở việc phân biệt các tổ hợp thiết bị có đặc trưng công suất tương đồng (ví dụ: sạc điện thoại kết hợp đèn LED so với máy lọc không khí ở chế độ thấp) trong điều kiện nhiễu của lưới điện thực tế tại Việt Nam.

Bảng 2.1. Các thiết bị điện được sử dụng trong tập dữ liệu VNDALE-2

Thiết bị	**Công suất định mức (W)**
Bóng đèn LED	7
Quạt điện	38
Tủ lạnh	130
Máy lọc không khí	25
Máy sấy tóc	370
Máy xay sinh tố	47
Sạc điện thoại	12

Dữ liệu điện được thu thập bởi cảm biến dòng và áp sử dụng chip đo lường BL0940 trong môi trường phòng thí nghiệm tại Việt Nam (điện áp lưới 220 V, nhiệt độ trung bình 29°C, độ ẩm 85%). Tần số lấy mẫu là 600 Hz. Các thiết bị được lựa chọn vì tính phổ biến trong hộ gia đình Việt Nam và có đặc trưng công suất rõ rệt. Khác với các bài toán NILM truyền thống chỉ phát hiện trạng thái bật/tắt của một thiết bị duy nhất, bài toán này yêu cầu phân biệt chính xác các tổ hợp thiết bị đồng thời hoạt động – một bài toán phân loại đa nhãn phức tạp hơn nhiều, đòi hỏi khả năng phân biệt chính xác giữa các lớp có đặc trưng công suất rất gần nhau.

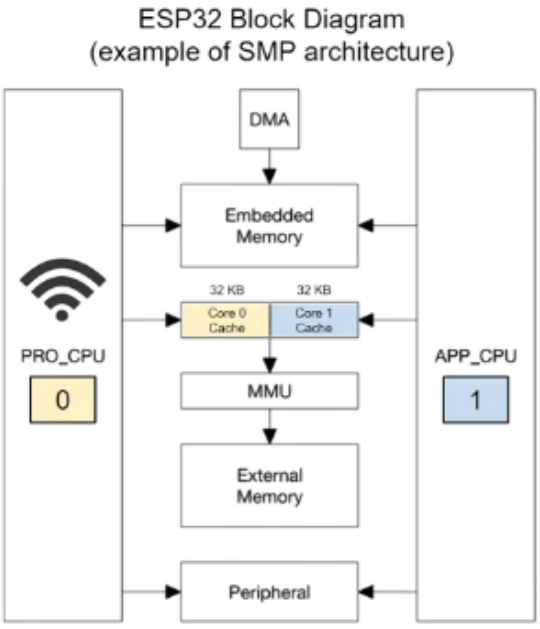
2.1.2. Ràng buộc triển khai: Vi điều khiển ESP32 và yêu cầu tăng tốc

Khác với các nghiên cứu trước đây trên ESP8266, nghiên cứu này sử dụng ESP32 để khai thác tối đa sức mạnh phần cứng cho việc tăng tốc thuật toán.

Bảng 2.2. Lợi thế phần cứng của ESP32 trong tăng tốc HDC

Thông số	ESP8266	ESP32 (Mục tiêu nghiên cứu)	Lợi ích cho HDC
Lõi CPU	Single-core	Dual-core LX6	Song song hóa Encoding và Search
SRAM	≈ 50 KB	≈ 320 KB	Lưu trữ nhiều Class Hypervectors hơn
Tập lệnh	Tiêu chuẩn	Hỗ trợ Bit-manipulation	Tối ưu phép toán XOR và Popcount
FPU	Không	Có	Tăng tốc tiền xử lý đặc trưng

Mặc dù ESP32 mạnh hơn, việc chạy mô hình HDC với số chiều lớn ($D > 10.000$) vẫn gây ra độ trễ đáng kể. Do đó, bài toán đặt ra là phải tận dụng kiến trúc đa nhân và tính toán mức bit để đưa thời gian suy luận xuống mức tối thiểu.



Hình 2.1. System structure của ESP32[9].

ESP32 là vi điều khiển hiệu năng cao của Espressif, nổi bật với khả năng xử lý song song nhờ kiến trúc SMP (Symmetric Multiprocessing). Gồm có các đặc điểm chính:

– Bộ xử lý hai lõi (Dual-core CPU): PRO_CPU (Core 0) (lõi chính, thường xử lý các tác vụ hệ thống); APP_CPU (Core 1) (lõi phụ, được sử dụng để xử lý các tác vụ ứng dụng, thuật toán hoặc các luồng song song).

– Bộ nhớ và quản lý bộ nhớ: cache riêng biệt cho từng lõi Core 0 và Core 1 đều là 32 KB.

– Khả năng chia sẻ tài nguyên: cả hai lõi CPU có thể truy cập chung các tài nguyên như bộ nhớ heap, biến toàn cục, thiết bị ngoại vi, timer, interrupt,... Việc chia sẻ này đòi hỏi cơ chế đồng bộ hóa để tránh xung đột và lỗi dữ liệu.

– Hỗ trợ hệ điều hành FreeRTOS: cho phép esp32 hoạt động ở chế độ dual-core scheduling, có gán task cho từng core cụ thể hoặc để hệ thống tự cân bằng, quản lý lịch trình và ưu tiên task, đồng bộ hóa giữa các task bằng mutex, semaphore, queue.

Phần kiến trúc này là nền tảng để triển khai mô hình HDC song song một cách hiệu quả, tận dụng tối đa khả năng xử lý song song của ESP32.

2.1.3. Thách thức hệ thống và mục tiêu tối ưu

Hệ thống cần đạt được các chỉ số hiệu năng sau:

– Độ chính xác: Duy trì $\geq 94\%$ nhờ khả năng xử lý số chiều lớn hơn mà không làm tăng độ trễ.

– Tốc độ suy luận: Giảm xuống dưới 2 ms (tăng tốc gấp 5 lần so với triển khai tuần tự).

– Hiệu quả tài nguyên: Tối ưu hóa việc phân bổ bộ nhớ động để tránh xung đột giữa hai nhân xử lý.

– Tính thời gian thực: Đảm bảo việc lấy mẫu dữ liệu tại Core 0 không bị gián đoạn bởi quá trình suy luận tại Core 1.

2.2. Mô hình giải pháp

2.2.1. Luồng thiết kế hệ thống

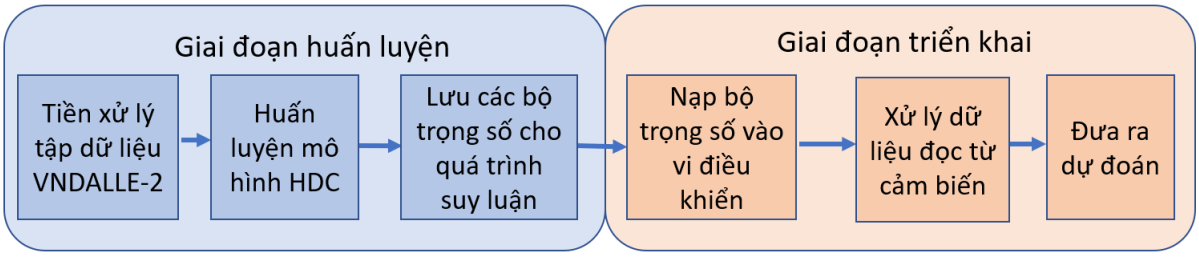
Quy trình phát triển hệ thống được minh họa trong Hình 2.1. Sau khi xác định rõ bài toán và các ràng buộc triển khai, hệ thống được thiết kế nhằm giải quyết 2 vấn đề:

– xây dựng một mô hình có khả năng nhận diện chính xác 127 tổ hợp trạng thái thiết bị từ tập dữ liệu VNDALE-2.

– Triển khai mô hình này trên vi điều khiển Esp32 tối ưu tính toán song song trên cả 2 cores.

Quy trình triển khai hệ thống được chia thành hai giai đoạn chính:

– Giai đoạn huấn luyện: thực hiện trên nền tảng Python nhằm tìm ra các prototypes hypervector tối ưu cho mỗi lớp và các tham số tiền xử lý phù hợp.

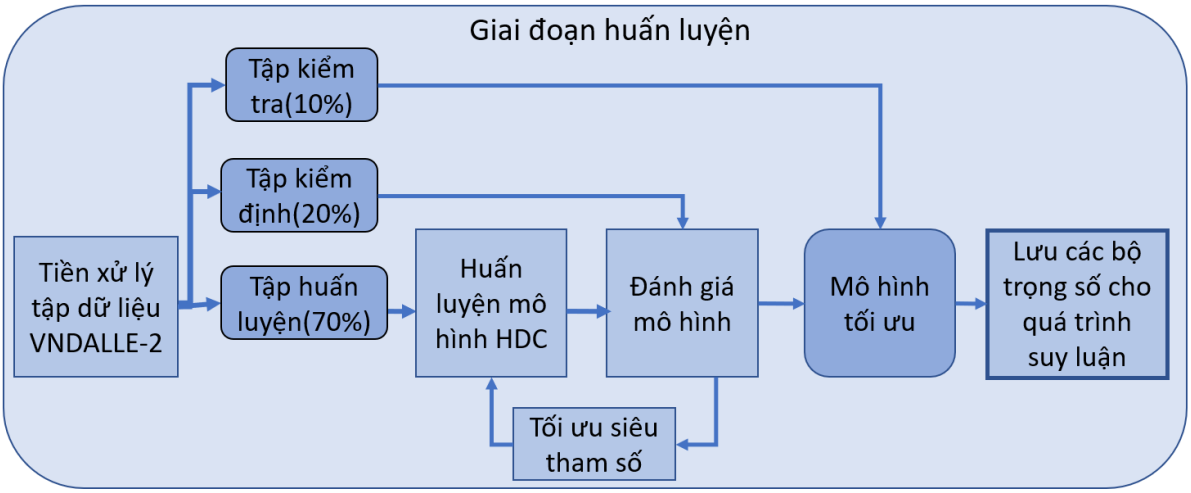


Hình 2.2. Luồng thiết kế hệ thống

– Giai đoạn triển khai: chuyển toàn bộ mô hình và tham số sang mã C, tối ưu hóa bộ nhớ và tốc độ để chạy trực tiếp trên 2 core của vi điều khiển ESP32. Mô hình hoàn thiện trên vi điều khiển được đánh giá độ chính xác, thời gian dự đoán và dung lượng bộ nhớ sử dụng.

2.2.2. Giai đoạn huấn luyện

Trước tiên, mô hình HDC được xây dựng và kiểm thử trên môi trường Python. Đây là bước quan trọng nhằm thiết kế đầy đủ các thành phần của hệ thống HDC, bao gồm: bộ vector cơ sở (item memory), bộ vector giá trị (value hypervectors), các phép toán mã hoá, cơ chế gộp thông tin và phương pháp phân loại.



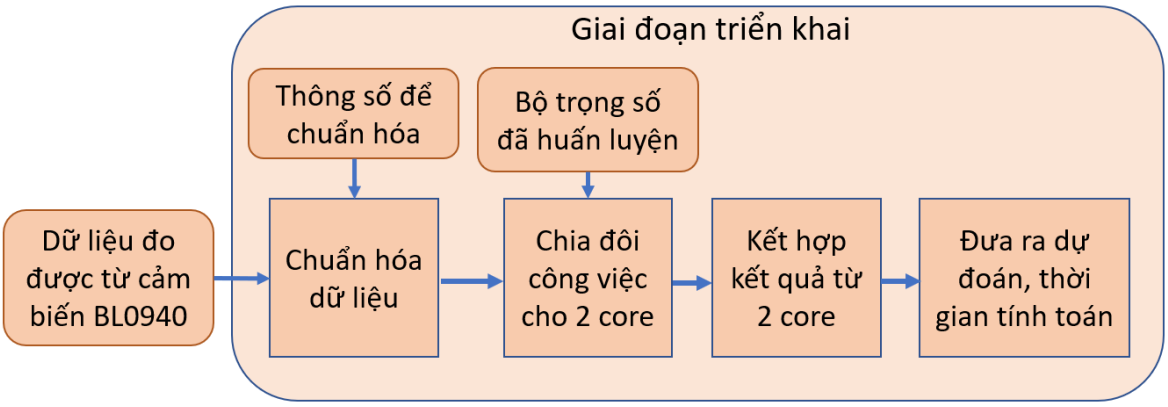
Hình 2.3. Giai đoạn huấn luyện

Sau khi mô hình được huấn luyện xong với tập huấn luyện, mô hình được đánh giá trên tập kiểm định từ đó đưa ra sự điều chỉnh về các siêu tham số, quá trình này dừng lại khi kết quả đánh giá mô hình trên tập kiểm định đủ cao. Tiếp đó thực hiện đánh giá mô hình này trên tập kiểm tra nhằm đưa ra đánh giá khách quan về khả năng của mô hình.

Toàn bộ tham số cần thiết được lưu lại để triển khai mô hình trên các vi điều khiển trong quá trình suy luận.

2.2.3. Giai đoạn triển khai

Quy trình tiếp theo là tối ưu hóa các tham số từ mô hình gốc và xây dựng lại thuật toán HDC bằng ngôn ngữ C/C++ để chạy trên vi điều khiển.



Hình 2.4. Giai đoạn triển khai

Quy trình được minh họa trong Hình 2.4 gồm có:

– Các thành phần đã thu được từ giai đoạn huấn luyện, bao gồm: bộ item memory, các value hypervectors, thông tin về bộ chuẩn hoá đặc trưng và các vector lớp (class hypervectors/ prototype) được lưu lại và gọi lúc mô hình suy luận.

– Sau khi xây dựng các hàm hỗ trợ suy luận mô hình, việc đưa ra dự đoán sẽ được thực hiện chia đôi và chạy song song trên dual-core.

– Kết quả tính toán của mỗi mô hình sẽ được kết hợp và đưa ra dự đoán cuối cùng.

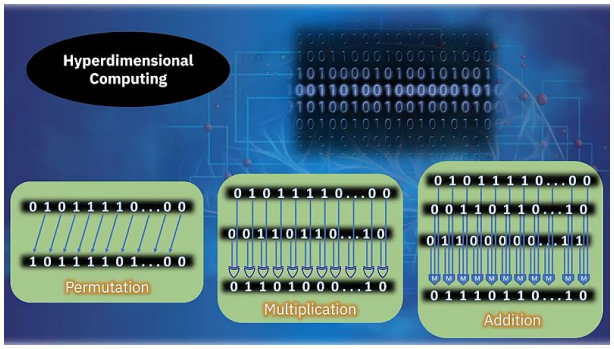
Trong giai đoạn này mô hình được đánh giá dựa trên các thông số về thời gian đưa ra dự đoán, mức độ tiêu thụ bộ nhớ tối đa trong quá trình đưa ra dự đoán trên mỗi core, và tổng thể. Các hàm chức năng như mã hóa, so sánh của mô hình sẽ được tối ưu để cải thiện tốc độ và bộ nhớ trong khi vẫn đảm khả năng dự đoán chính xác của mô hình ban đầu.

Chương 3. Đặc trưng và mô hình

3.1. Mô hình HDC

3.1.1. Kiến trúc mô hình Hyperdimensional Computing (HDC)

Mô hình Hyperdimensional Computing (HDC) được sử dụng trong hệ thống này dựa trên nguyên lý biểu diễn và xử lý thông tin bằng các vector siêu chiều (hypervectors). Khác với các mô hình học sâu truyền thống như mạng nơ-ron nhân tạo (ANN), HDC không dựa trên việc tối ưu các trọng số thông qua lan truyền ngược (backpropagation), mà khai thác các phép toán đại số trong không gian siêu chiều để mã hóa, kết hợp và suy luận dữ liệu.



Hình 3.1. Minh họa các phép toán cơ bản trong HDC[10]

Mô hình Tính toán Siêu chiều (Hyperdimensional Computing - HDC) được cấu thành từ ba khối chức năng chính: Bộ mã hóa (Encoder), Bộ nhớ lớp (Class Memory/Associative Memory) và Bộ phân loại (Classifier). Về mặt kiến trúc, HDC vận hành theo cơ chế lan truyền thuận (feedforward), trong đó dữ liệu được ánh xạ vào không gian vector siêu chiều (Hyperspace) và so sánh trực tiếp mà không cần các vòng lặp tối ưu hóa phức tạp như mạng nơ-ron truyền thống.Điểm ưu việt của HDC nằm ở việc sử dụng các toán tử đại số trên các vector nhị phân (binary vectors) hoặc vector nguyên (integer vectors) với số chiều lớn ($D \approx 10{,}000$). Các phép toán cốt lõi như Binding (XOR), Bundling (Cộng) và Permutation (Hoán vị) có thể được thực thi bằng các thao tác bit (bitwise operations) mức thấp. Đặc tính này loại bỏ sự phụ thuộc vào các đơn vị xử lý dấu phẩy động (FPU), giúp HDC trở thành giải pháp lý tưởng cho các vi điều khiển (MCU) và thiết bị biên (Edge devices) với tài nguyên hạn chế.

3.1.2. Quá trình xử lý tín hiệu trong HDC

Quy trình xử lý trong mô hình HDC được chia thành ba giai đoạn tuần tự: 1. Giai đoạn Mã hóa (Encoding Phase) Mục tiêu của giai đoạn này là ánh xạ một mẫu dữ liệu đầu vào $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ gồm n đặc trưng (features) thành một hypervector duy nhất $\mathbf{H} \in \{0, 1\}^D$. Quá trình này sử dụng hai bộ nhớ cơ sở:

– Item Memory (IM): Chứa các vector định danh $\mathbf{ID}_i$ đại diện cho vị trí hoặc tên của đặc trưng thứ i.

– Value Memory (VM): Chứa các vector mức $\mathbf{L}_v$ đại diện cho giá trị lượng tử hóa của dữ liệu.

Quá trình mã hóa thực hiện qua hai bước toán học:

– Phép Binding (Liên kết): Gán giá trị của đặc trưng vào vị trí của nó bằng phép XOR ($\oplus$).

$$\mathbf{H}_i = \mathbf{ID}_i \oplus \mathbf{L}_{x_i}$$

– Phép Bundling (Gộp): Tổng hợp thông tin của toàn bộ n đặc trưng bằng phép cộng vector, sau đó áp dụng hàm ngưỡng (thresholding) để đưa về dạng nhị phân.

$$\mathbf{H}_{sample} = \text{sgn}\left(\sum_{i=1}^{n} \mathbf{H}_i\right)$$

Trong đó hàm $\text{sgn}(\cdot)$ là hàm dấu (hoặc hàm ngưỡng đa số - majority vote) để chuẩn hóa kết quả về $\{0, 1\}$.

2. Giai đoạn Huấn luyện (Training Phase) Khác với phương pháp tĩnh (cộng gộp một lần), mô hình đề xuất áp dụng cơ chế huấn luyện lặp (Iterative Learning), trong đó các vector đại diện (Prototype Vectors) được cập nhật liên tục để tối ưu hóa khả năng phân loại. Quá trình này giúp mô hình ”tinh chỉnh” biên quyết định dựa trên các mẫu dữ liệu mới hoặc sửa chữa các sai sót trong quá trình học. Quy trình cập nhật cho một mẫu huấn luyện đầu vào **x** với nhãn đúng là y_{true} diễn ra như sau:

– Dự báo (Prediction): Mẫu **x** được mã hóa thành hypervector **H**. Hệ thống tính độ tương đồng của **H** với các vector lớp hiện tại để đưa ra nhãn dự đoán $\hat{y}$.

– Cập nhật trọng số (Weight Update): Dựa trên kết quả dự đoán, các vector lớp được cập nhật theo quy tắc học Perceptron (hoặc quy tắc Delta):

+ Nếu dự đoán đúng ($\hat{y} == y_{true}$): Giữ nguyên hoặc cập nhật nhẹ để củng cố bộ nhớ.

+ Nếu dự đoán sai ($\hat{y} \neq y_{true}$): Thực hiện cơ chế ”thưởng - phạt”.

– Công thức cập nhật toán học được biểu diễn như sau:

$$\mathbf{C}_{true}^{(new)} = \mathbf{C}_{true}^{(old)} + \alpha \cdot \mathbf{H}$$

$$\mathbf{C}_{wrong}^{(new)} = \mathbf{C}_{wrong}^{(old)} - \alpha \cdot \mathbf{H}$$

Trong đó, **C** là vector đại diện lớp (ở dạng nguyên - integer) và α là tốc độ học (learning rate), thường chọn $\alpha = 1$. Sau chu kỳ cập nhật, các vector lớp **C** sẽ được đưa qua hàm

ngưỡng (thresholding function) để chuẩn hóa về dạng nhị phân trước khi lưu vào bộ nhớ hoặc dùng cho suy luận tiếp theo:

$$\mathbf{C}_{final} = \text{sgn}(\mathbf{C}_{updated})$$

Cơ chế này cho phép mô hình HDC có khả năng thích nghi cao, sửa lỗi dần dần qua các vòng lặp (epochs) và dễ dàng cập nhật thêm tri thức mới mà không cần huấn luyện lại từ đầu.

3.1.3. Trích xuất đặc trưng

Quá trình trích xuất đặc trưng đóng vai trò quan trọng trong việc nâng cao hiệu năng của mô hình học máy, đặc biệt trong bối cảnh hệ thống NILM (Non-Intrusive Load Monitoring) – nơi nhiều thiết bị điện hoạt động đồng thời và tạo ra các mẫu tín hiệu chồng lấn phức tạp.

Trong hệ thống này, bộ dữ liệu (được thu thập bằng vi mạch BL0940) cung cấp ba đại lượng điện cơ bản:

- Dòng điện hiệu dụng (I_{rms}),
- Điện áp hiệu dụng (U_{rms}),
- Hệ số công suất (Pf).

Mặc dù ba thông số này mang thông tin quan trọng về trạng thái tiêu thụ điện, chúng thường chưa đủ khả năng phân biệt giữa nhiều tổ hợp thiết bị khác nhau. Thực tế cho thấy, với 127 tổ hợp thiết bị được xét trong nghiên cứu, nhiều tổ hợp có thể cho ra giá trị I_{rms}, U_{rms} và Pf khá tương đồng, gây khó khăn cho quá trình nhận dạng nếu chỉ dựa trên các đặc trưng cơ bản này.

Bổ sung đặc trưng dẫn xuất (Derived Features): Để tăng cường khả năng phân loại, hệ thống tính toán thêm các đặc trưng dẫn xuất từ ba thông số cơ bản trên, bao gồm:

- Công suất tác dụng (P),
- Công suất phản kháng (Q),
- Công suất biểu kiến (S).

Ba đặc trưng này là các đại lượng cơ bản trong hệ thống điện xoay chiều (AC), phản ánh mối quan hệ pha giữa dòng điện và điện áp. Chúng được xác định thông qua các công thức toán học sau:

$$\cos(\varphi) = Pf$$
$$S = U_{rms} \times I_{rms}$$
$$P = S \times \cos(\varphi) = U_{rms} \times I_{rms} \times Pf$$
$$Q = S \times \sin(\varphi) = U_{rms} \times I_{rms} \times \sqrt{1 - Pf^2}$$

Trong đó:

- φ: góc lệch pha giữa điện áp và dòng điện,
- U_{rms}: điện áp hiệu dụng (volt),
- I_{rms}: dòng điện hiệu dụng (ampe),
- Pf: hệ số công suất,
- S: công suất biểu kiến (VA),
- P: công suất tác dụng (W),
- Q: công suất phản kháng (VAR).

Góc φ giữa hai vector dòng và áp thể hiện tỉ lệ giữa năng lượng sử dụng hữu ích và năng lượng bị trễ pha trong hệ thống. Biểu diễn này không chỉ mang ý nghĩa vật lý mà còn giúp lý giải việc bổ sung các đặc trưng P, Q, và S vào để tăng khả năng phân biệt giữa các thiết bị điện.

Trong mô hình HDC, sáu đặc trưng điện được chọn (gồm I_{rms}, U_{rms}, P, Q, S, và Pf) tạo thành vector đặc trưng đầu vào của hệ thống. Mỗi đặc trưng sẽ được ánh xạ sang một hypervector ngẫu nhiên trong không gian siêu chiều thông qua hàm mã hóa (encoding function).

3.2. Tiền xử lý dữ liệu

Trong mô hình HDC, bước đầu tiên của quá trình mã hoá đặc trưng là đưa toàn bộ dữ liệu về cùng một thang đo. Do các đặc trưng I_{rms}, U_{rms}, P, Q, S, PF có biên độ rất khác nhau, việc chuẩn hoá là cần thiết để đảm bảo mỗi đặc trưng đóng góp công bằng vào quá trình sinh Hypervector.

Chuẩn hoá bằng Z-score

Với dãy dữ liệu thực:

$$x_1, x_2, \ldots, x_n.$$

Trung bình:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Độ lệch chuẩn:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}.$$

Giá trị chuẩn hoá (z-score):

$$z_i = \frac{x_i - \mu}{\sigma}.$$

Trong HDC, mỗi đặc trưng sau khi được chuẩn hoá sẽ được ánh xạ vào một mức lượng tử (quantized level), và mức này sẽ quyết định HV giá trị (*value HV*) được binding với HV chỉ mục (IM). Việc dùng z-score có những lợi ích quan trọng sau:

- Đưa các đặc trưng về cùng phân bố chuẩn hoá: Sau chuẩn hoá, toàn bộ đặc trưng đều có trung bình bằng 0 và độ lệch chuẩn bằng 1, giúp tránh tình trạng đặc trưng có biên độ lớn chi phối quá trình majority vote của HDC.
- Làm đồng đều thang đo trước khi lượng tử hoá: Vì mô hình HDC của chúng tôi chia khoảng $[-3, 3]$ thành 120 mức, chuẩn hoá z-score đảm bảo phần lớn dữ liệu nằm trong khoảng này. Điều này làm cho việc gán giá trị vào các mức lượng tử trở nên ổn định và nhất quán.
- Bảo toàn quan hệ tương đối giữa các giá trị: Z-score không làm thay đổi trật tự (ordering) giữa các mẫu trong cùng một đặc trưng, đảm bảo các giá trị lớn/nhỏ vẫn phản ánh đúng sự khác biệt của tín hiệu.

Dựa trên bất đẳng thức Chebyshev, ta có:

$$P(|Z| \leq 3) \geq 0.8889.$$

Điều này nghĩa là ít nhất 88.89% dữ liệu sau chuẩn hoá sẽ nằm trong đoạn $[-3, 3]$. Vì vậy việc chia đoạn này thành 120 mức là phù hợp phần lớn dữ liệu sẽ được mã hoá bằng các mức giá trị hợp lệ, giúp quá trình sinh Hypervector ổn định.

Các giá trị vượt ngoài $[-3, 3]$ được gán về biên:

$$z_i = \begin{cases} 3, & \text{khi } z_i > 3, \\ -3, & \text{khi } z_i < -3, \\ z_i, & \text{còn lại.} \end{cases}$$

Mỗi mức lượng tử sau đó được ánh xạ sang một **HV_value** tương ứng để chuẩn bị cho bước binding với IM.

Z-score đóng vai trò quan trọng trong pipeline HDC vì nó:

- chuẩn hoá giá trị đầu vào,

– đảm bảo phân bố ổn định cho lượng tử hoá,

– giúp các đặc trưng khác nhau đóng góp công bằng vào quá trình xây dựng Hypervector mẫu.

3.3. Kĩ thuật Memory Tiling song song quá trình suy luận của mô hình HDC

Đây là phương pháp ứng dụng để song song hóa thuật toán HDC, nhằm chia công việc cho 2 cores tính toán.

3.3.1. Chiến lược song song hóa dữ liệu (Memory Tiling)

Giải pháp cốt lõi của đồ án là chia nhỏ Hypervector (có độ dài D) thành hai phần (tiles) bằng nhau: D_0 và D_1 (với $D_0 = D_1 = D/2$). Kỹ thuật này cho phép tận dụng tối đa kiến trúc đa nhân của ESP32 để xử lý song song:

– Core 0 (Master Core): Thực hiện mã hóa nửa đầu của vector ($0 \rightarrow D/2 - 1$) và tính toán khoảng cách Hamming cục bộ với nửa đầu của tất cả 127 prototypes trong bộ nhớ.

– Core 1 (Worker Core): Đồng thời thực hiện mã hóa nửa sau của vector ($D/2 \rightarrow D-1$) và tính toán khoảng cách Hamming với nửa sau tương ứng của các prototypes.

Cụ thể, thực hiện chia vector D chiều thành hai phân mảnh (tiles) độc lập để xử lý song song. Các biến toàn cục được ánh xạ vào vùng nhớ dùng chung (Shared Memory) để cả hai lõi có thể truy xuất:

– **Phân mảnh dữ liệu:**Vector truy vấn (Query Hypervector) được chia đôi:

+ **Tile 0 (Core 0):** Quản lý các bit từ 0 đến $D/2 - 1$.

+ **Tile 1 (Core 1):** Quản lý các bit từ $D/2$ đến $D - 1$.

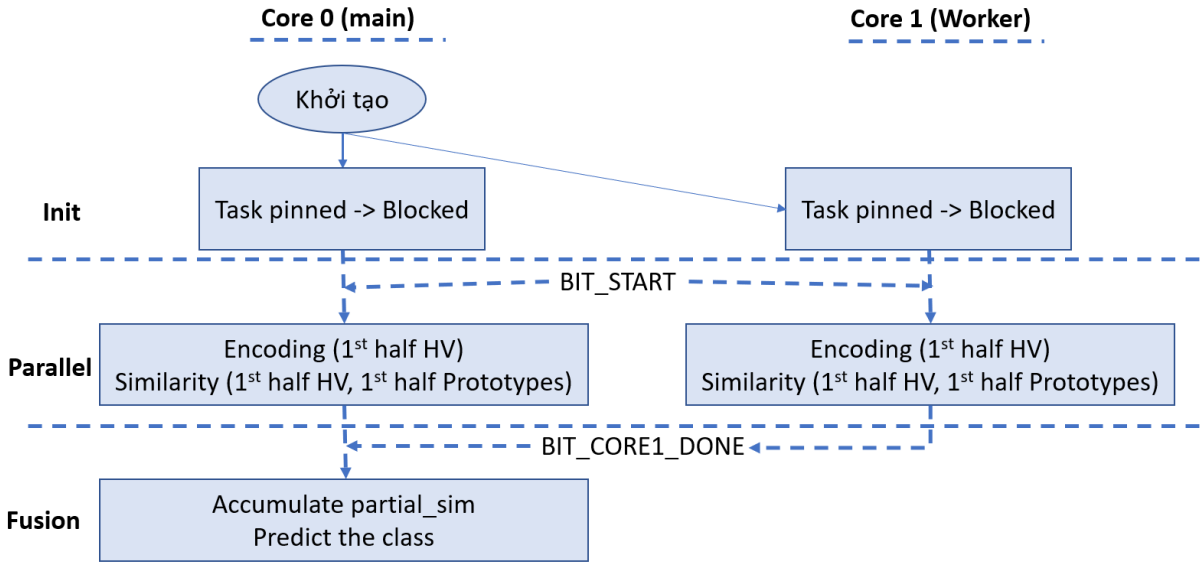
– **Kết quả cục bộ (Partial Results):** Thay vì một biến kết quả duy nhất, chúng tôi sử dụng mảng 2 chiều `partial_sim[2][NUM_CLASSES]` để lưu trữ điểm độ tương đồng cục bộ:

$$Sim_{total}(C_k) = \texttt{partial\_sim}[0][k] + \texttt{partial\_sim}[1][k] \quad (1)$$

Trong đó `partial_sim`[core_id][k] là khoảng cách Hamming tính được trên phân mảnh vector do lõi đó phụ trách.

3.3.2. Quy trình thực thi song song (Parallel Execution Flow)

Khác với cơ chế Semaphore đơn lẻ, hệ thống sử dụng FreeRTOS Event Groups để quản lý trạng thái phức tạp của nhiều luồng.



Hình 3.2. Luồng xử lý song song

Quy trình thực thi theo mô hình Fork-Join được mô tả chi tiết như sau:

- **Khởi tạo (Fork):** Hai tác vụ (Tasks) được gán cố định (pinned) vào Core 0 và Core 1. Cả hai đều ở trạng thái chờ (Blocked) tại sự kiện `BIT_START`.
- **Kích hoạt:** Vòng lặp chính (Main Loop) gửi tín hiệu `BIT_START` thông qua hàm `xEventGroupSetBits`. Cả hai lõi cùng thức dậy và bắt đầu xử lý song song.
- **Xử lý song song (Parallel Processing):**
  + **Core 0 (Master Worker):** Thực hiện mã hóa cho nửa đầu vector. Sau đó, tính toán khoảng cách Hamming giữa nửa vector này với phần tương ứng của toàn bộ 127 vector mẫu (Prototypes).
  + **Core 1 (Slave Worker):** Đồng thời thực hiện mã hóa cho nửa sau vector và tính toán khoảng cách Hamming tương ứng. Sau khi hoàn tất, Core 1 bật cờ sự kiện `BIT_CORE1_DONE` và quay lại trạng thái chờ.
- **Đồng bộ và Hợp nhất (Join & Fusion):** Core 0 sau khi hoàn thành phần việc của mình sẽ chờ tín hiệu `BIT_CORE1_DONE` từ Core 1. Khi cả hai lõi đã hoàn tất, Core 0 thực hiện bước hợp nhất (Fusion): cộng dồn các kết quả `partial_sim` để tìm ra lớp có độ tương đồng tổng thể cao nhất.

3.3.3. Lợi ích của mô hình giải pháp

- Giảm độ trễ tuyến tính: Về lý thuyết, thời gian thực hiện phép so sánh giảm đi gần 50% so với việc chạy trên một nhân duy nhất.

– Locality of Reference: Core 0 luôn truy cập nửa đầu của bộ nhớ, Core 1 luôn truy cập nửa sau giúp tốc độ đọc dữ liệu nhanh hơn.

– Cân bằng tải (Load Balancing): Vì hai nửa vector có độ dài bằng nhau ($D/2$), khối lượng tính toán trên mỗi nhân là tương đồng, giúp tối thiểu hóa thời gian chờ (idle time) của CPU.

– Độ chính xác không đổi: Vì đây là phép toán song song hóa dữ liệu thuần túy (không phải xấp xỉ), kết quả cuối cùng hoàn toàn trùng khớp với thuật toán tuần tự nhưng với tốc độ cao hơn.

3.4. Xây dựng hàm khi triển khai xuống vi điều khiển

3.4.1. Hàm Encode

Chức năng cốt lõi của phân hệ Mã hóa (Encoder) là thực hiện phép ánh xạ dữ liệu từ miền không gian thực R^N sang miền không gian Hamming $\{0,1\}^D$. Tại đây, mỗi mẫu dữ liệu bao gồm nhiều đặc trưng (features) riêng biệt được hợp nhất thành một vector siêu chiều duy nhất (Hypervector), đảm bảo bảo toàn các tính chất thống kê và cấu trúc của tín hiệu gốc.

Quá trình này sử dụng hai bộ nhớ cơ sở (Base Memories) trực giao được khởi tạo ngẫu nhiên trong quá trình huấn luyện:

– **Item Memory (IM):** Chứa các vector định danh (**ID**$_i$) đại diện cho vị trí hoặc tên của đặc trưng thứ i.

– **Value Memory (VM):** Chứa các vector mức (**L**$_v$) đại diện cho giá trị lượng tử hóa của tín hiệu.

Quá trình mã hóa giống trong các mô hình HDC trước đây áp dụng:

1. Binding: Với mỗi đặc trưng f_i, vector ngữ nghĩa tương ứng IM[i] được binding với vector giá trị HV_value[v_i] thông qua phép XOR:

$$HV(IMvalue_i) = IM_i \oplus HV\_value(v_i)$$

Algorithm 1 Bind Hypervectors

```
function Bind(HV_a, HV_b)
    for i = 0 to D − 1 do
        HV_out[i] ← HV_a[i] ⊕ HV_b[i]
    end for
    return HV_out
end function
```

2. Rotation (Xoay): Sau khi binding, mỗi $HV(IMvalue_i)$ được xoay vòng (circular shift) theo số bước tỉ lệ với thứ tự đặc trưng:

$$HV(IMvalue_i) = Rotate(HV(IMvalue_i),\ shift = i)$$

Việc xoay giúp đảm bảo rằng thứ tự các đặc trưng được mã hóa khác biệt trong

Algorithm 2 Circular Shift of Hypervector

function Roll($HV, shift$)
    $shift \leftarrow shift$ mod D
    if $shift < 0$ **then**
        $shift \leftarrow shift + D$
    end if
    for $i = 0$ to $D - 1$ **do**
        $HV_{out}[i] \leftarrow HV[(i - shift + D)$ mod $D]$
    end for
    return HV_{out}
end function

không gian siêu chiều.

3. Bundling (Tổng hợp HV mẫu): Tất cả các $HV(IMvalue_i)$ được kết hợp lại để tạo ra HV đại diện cho toàn bộ mẫu dữ liệu. Quá trình này sử dụng phép majority vote:

$$HV(sample)_j = \begin{cases} 1, & \text{nếu hơn 50\% giá trị tại vị trí } j \text{ trong các } HV(IMvalue_i) \text{ là 1} \\ 0, & \end{cases}$$

Algorithm 3 Bundle Hypervectors using Majority Vote

```
function Bundle(HV_list)
    for i = 0 to D − 1 do
        count ← 0
        for each HV in HV_list do
            if HV[i] = 1 then
                count ← count + 1
            end if
        end for
        if count ≥ |HV_list|/2 then
            HV_out[i] ← 1
        else
            HV_out[i] ← 0
        end if
    end for
    return HV_out
end function
```

Kết quả thu được là một vector siêu chiều duy nhất HV(sample) đại diện cho đặc trưng tổng hợp của mẫu dữ liệu.

Các hàm hỗ trợ việc mã hóa bên trên sẽ được sử dụng trong hàm encode dưới đây để tạo ra vector biểu diễn giá trị đầu vào.

Algorithm 4 Mã giả cho hàm Encode

function EncodeSample($sample$)
    Initialize empty list HV_list
    for $i = 0$ to $NUM\_FEATURES - 1$ **do**
        $value \leftarrow sample[i]$
        $level \leftarrow \left\lfloor \frac{value + 3}{6} \times (NUM\_LEVELS - 1) \right\rfloor$
        if $level < 0$ **then**
            $level \leftarrow 0$
        else if $level \geq NUM\_LEVELS$ **then**
            $level \leftarrow NUM\_LEVELS - 1$
        end if
        $hv \leftarrow Bind(IM[i], VALUE\_HV[level])$
        $hv \leftarrow Roll(hv, i \times \lfloor D/NUM\_FEATURES \rfloor)$
        Append hv to HV_list
    end for
    $HV\_out \leftarrow Bundle(HV\_list)$
    return HV_out
end function

Hàm `Encode` thực hiện mã hóa 1 nửa vector HV để chia công việc cho mỗi core và thực hiện bước tính toán khoảng cách với các prototypes đại diện cho các classes.

3.4.2. Hàm Hamming Similarity

Hàm hammingSim() được sử dụng trong giai đoạn suy luận nhằm đo mức độ tương đồng giữa hypervector đầu vào (HV_sample) và các hypervector đại diện của từng lớp (prototype vectors). Trong mô hình HDC, độ tương đồng giữa vector HV càng cao chứng tỏ mẫu dữ liệu càng gần với lớp tương ứng. Khoảng cách Hamming đo số lượng bit khác

Algorithm 5 Hamming Similarity giữa hai Hypervector

function HammingSim(HV_a, HV_b)
    $sim \leftarrow 0$
    for $i = 0$ to $SIZE\_BYTES - 1$ **do**
        $x \leftarrow HV_a[i] \oplus HV_b[i]$
        $diff \leftarrow$ PopCount(x)
        $sim \leftarrow sim + (8 - diff)$
    end for
    return sim
end function

nhau giữa hai vector, do đó số bit giống nhau có thể được dùng làm thước đo tương đồng. Quy trình tính toán được thực hiện theo ba bước chính:

1. Thực hiện phép XOR giữa hai hypervector
   Hai vector nhị phân HV_{sample} và HV_{proto} được thực hiện phép XOR từng bit:

   $$diff = HV_{\text{sample}} \oplus HV_{\text{proto}}$$

   Vector $diff$ sẽ có bit bằng 1 tại những vị trí mà hai hypervector khác nhau.

2. Đếm số bit khác nhau bằng phép popcount
   Hàm built-in `__builtin_popcount()` được sử dụng để đếm tổng số bit bằng 1 trong vector $diff$:

   $$d = \text{popcount}(diff)$$

   Giá trị d biểu thị số lượng bit khác nhau giữa hai hypervector.

3. Tính số bit giống nhau và trả về độ tương đồng
   Với hypervector có chiều D, số bit giống nhau được xác định bởi:

   $$\text{similarity} = D - d$$

   Đây là giá trị được hàm hammingSim() trả về để so sánh mức tương đồng giữa mẫu đầu vào và các prototype vector.

Cách tính toán này tận dụng các phép toán bit-level nên rất hiệu quả khi triển khai trên vi điều khiển, hỗ trợ quá trình suy luận nhanh và tiết kiệm tài nguyên.

3.4.3. Dự đoán

Hàm predict() chịu trách nhiệm xác định lớp (tổ hợp thiết bị) tương ứng với một mẫu dữ liệu mới dựa trên hypervector đầu vào. Sau khi mẫu được mã hóa thành một hypervector HV_{sample} thông qua hàm encode(), quá trình dự đoán được thực hiện bằng cách so sánh mức độ tương đồng giữa hypervector mẫu và các prototype vectors đã được xây dựng trong giai đoạn huấn luyện.

Cụ thể, hàm predict() thực hiện các bước sau:

1. Tính Hamming Similarity
   Với mỗi prototype vector $HV_{\text{proto}}^{(c)}$ ứng với lớp c, giá trị độ tương đồng được tính bằng hàm hammingSim():

   $$sim(c) = \text{hammingSim}\left(HV_{\text{sample}}, HV_{\text{proto}}^{(c)}\right)$$

   Độ tương đồng càng cao chứng tỏ mẫu càng gần với lớp đó trong không gian siêu chiều.

Algorithm 6 HDC Inference Function

```
function Predict(sample)
    HV ← EncodeSample(sample)
    bestClass ← 0
    bestSim ← −∞
    for c = 0 to NUM_CLASSES − 1 do
        proto ← PROTOTYPES[c]
        sim ← HammingSim(HV, proto)
        if sim > bestSim then
            bestSim ← sim
            bestClass ← c
        end if
    end for
    return bestClass
end function
```

2. Duyệt qua toàn bộ các lớp
   Với tổng số lớp được lưu trong PROTOTYPES, hàm sẽ duyệt qua toàn bộ các hypervector lớp để tìm giá trị tương đồng lớn nhất:

$$c^* = \arg\max_c \ sim(c)$$

3. Chọn lớp có similarity lớn nhất
   Chỉ số lớp c^* có giá trị similarity cao nhất được chọn làm kết quả dự đoán vì đây là lớp có prototype vector gần nhất với hypervector đầu vào.

Phương pháp phân loại sử dụng mô hình HDC chỉ yêu cầu các phép toán bit-level (XOR, popcount) và một vòng lặp tuyến tính qua các lớp. Nhờ đó, predict() có thể được thực thi nhanh chóng và hiệu quả trên các vi điều khiển có tài nguyên hạn chế.

Chương 4. Thực nghiệm

Các thí nghiệm trong chương này được thực hiện nhằm đánh giá hiệu năng về tốc độ, dung lượng, độ chính xác của mô hình Hyperdimensional Computing (HDC) triển khai trên 2 core và so sánh với mô hình HDC, ANN trên 1 core khi triển khai trên cùng bộ dữ liệu VNDALE-2 trong điều kiện độ chính xác của 2 mô hình xấp xỉ nhau và trên vi điều khiển esp32.

4.1. Tập dữ liệu kiểm thử

VNDALE-2 là bộ dữ liệu bao gồm 128 tổ hợp thiết bị, được tạo thành từ 7 thiết bị điện gia dụng khác nhau. Mỗi tổ hợp cung cấp các thông số điện đặc trưng bao gồm:

- Dòng điện hiệu dụng (I_{rms}),
- Điện áp hiệu dụng (U_{rms}),
- Dòng điện tức thời (I_n),
- Điện áp tức thời (U_n),
- Hệ số công suất (Power factor – PF).

Dữ liệu được thu thập bằng hệ thống Raspberry Pi 4 kết hợp với mạch đo điện BL0940. Các mẫu tín hiệu được lấy với tần số 600 Hz, cho phép trích xuất các đặc trưng tần số cao của tín hiệu điện, phục vụ cho bài toán nhận dạng và phân loại tổ hợp thiết bị điện.

Trước khi đưa vào mô hình HDC, dữ liệu được chuẩn hóa bằng phương pháp StandardScaler. Phương pháp này đảm bảo mỗi đặc trưng có giá trị trung bình bằng 0 và độ lệch chuẩn bằng 1, giúp quá trình lượng tử hóa và ánh xạ sang các mức giá trị trong Value Memory được thực hiện ổn định và nhất quán.

4.2. Huấn luyện và đánh giá mô hình trên phần mềm

Đối với việc huấn luyện và đánh giá mô hình chưa triển khai trên vi điều khiển, thực hiện huấn luyện mô hình HDC với số chiều khác nhau và đánh giá độ chính xác của các mô hình trước khi triển khai. Các chỉ số đánh giá bao gồm Accuracy, Precision, Recall và F1-score.

4.2.1. Thực hiện tiền xử lý dữ liệu

Dữ liệu bao đầu sẽ thực hiện chuẩn hóa bằng phương pháp StandardScaler, với công thức ở trên. Phương pháp này đảm bảo giá trị của mỗi đặc trưng có trung bình bằng 0 và độ lệch chuẩn bằng 1, điều này giúp quá trình lượng tử hóa và ánh xạ sang các mức

trong Value Memory được thực hiện ổn định, đồng thời giúp mô hình không bị ”thiên vị (bias)” đối với các đặc trưng có giá trị lớn hơn nhiều so với các đặc trưng còn lại.

Tập dữ liệu VNDALE-2 được chia thành ba phần:

- Tập huấn luyện (Training set): Chiếm 70% tổng số mẫu, trong đó mỗi nhãn có 4000 mẫu dùng để huấn luyện mô hình.
- Tập xác thực (Validation set): Chiếm 20% tổng số mẫu, dùng để đánh giá và điều chỉnh mô hình trong quá trình huấn luyện.
- Tập kiểm thử (Test set): Chiếm 10% tổng số mẫu, dùng để đánh giá hiệu năng cuối cùng của mô hình.

4.2.2. Huấn luyện mô hình và đánh giá

Để đảm bảo tính khách quan và hiệu quả của mô hình khi triển khai thực tế, quy trình thực nghiệm được tiến hành nghiêm ngặt qua ba giai đoạn riêng biệt trên các tập dữ liệu đã phân chia:

- **Giai đoạn 1: Huấn luyện (Training Phase):** Sử dụng **Tập huấn luyện (70%)** để xây dựng các vector đại diện (Class Hypervectors). Quá trình này thực hiện mã hóa và tính toán các vector Prototypes, được sử dụng cho bước suy luận tiếp theo.
- **Giai đoạn 2: Tinh chỉnh tham số (Validation & Tuning Phase):** Sử dụng **Tập xác thực (20%)** để đánh giá mô hình với các cấu hình siêu tham số khác nhau. Tại bước này,thực hiện thay đổi số chiều D (từ 64 đến 2048) và số mức lượng tử hóa Q để tìm ra các siêu tham số phù hợp – cấu hình mang lại sự cân bằng tốt nhất giữa độ chính xác và tài nguyên tính toán (bộ nhớ, thời gian xử lý) trước khi chốt phương án triển khai.
- **Giai đoạn 3: Kiểm thử (Testing Phase):** Sau khi xác định được bộ tham số tối ưu từ giai đoạn 2, mô hình hoàn chỉnh được đánh giá lần cuối trên **Tập kiểm thử (10%)** – tập dữ liệu hoàn toàn mới chưa từng tham gia vào quá trình huấn luyện hay tinh chỉnh. Các chỉ số hiệu năng cuối cùng (Accuracy, Precision, Recall, F1-Score) được ghi nhận từ giai đoạn này để đảm bảo kết quả phản ánh trung thực khả năng tổng quát hóa của hệ thống.

4.3. Cấu hình phần cứng cho thí nghiệm

Hệ thống được triển khai và đánh giá trực tiếp trên nền tảng vi điều khiển nhúng, đại diện cho các node IoT trong thực tế.

- Vi xử lý trung tâm (SoC):

+ Nhóm thực hiện sử dụng vi điều khiển ESP32-Wroom-32. Đây là dòng chip SoC giá rẻ nhưng hiệu năng đáng kể, đặc trưng cho kiến trúc Dual-core Xtensa 32bit LX6.

+ Tốc độ xung nhịp: Được cấu hình cố định ở mức 240MHz để tối đa hóa khả năng tính toán.

+ Bộ nhớ: Thiết bị có 520KB SRAM nội (Internal RAM) và 4MB Flash. Việc quản lý bộ nhớ SRAM là yếu tố then chốt do kích thước lớn của các hypervector trong thuật toán HDC.

– Môi trường phát triển:

+ Mã nguồn được phát triển bằng ngôn ngữ C/C++ trên nền tảng ESP-IDF (Espressif IoT Development Framework), chạy trên ứng dụng VsCode

+ Hệ điều hành thời gian thực FreeRTOS được sử dụng để quản lý tác vụ (task scheduling). Điều này cho phép chúng tôi gán cứng (pinning) các tác vụ tính toán cụ thể vào từng lõi (Core 0 và Core 1) để kiểm soát chính xác tài nguyên tính toán song song.

– Phương pháp đo lường:

+ Để đo lường độ trễ suy luận (Inference Latency) một cách chính xác nhất, chúng tôi không sử dụng các hàm thời gian của hệ thống tệp mà sử dụng trực tiếp Timer phần cứng độ phân giải cao (esp_timer_get_time()).

+ Đơn vị đo lường là micro-seconds (μs).

4.4. Kịch bản kiểm thử

Để có cái nhìn toàn diện về hiệu quả của giải pháp đề xuất, thực hiện thiết kế 2 nhóm thực nghiệm chính, tập trung vào khía cạnh kiến trúc xử lý và tối ưu hóa thời gian xử lý.

4.4.1. Đánh giá hiệu năng đa lõi

Mục tiêu của thí nghiệm nhằm định lượng mức độ cải thiện tốc độ xử lý khi chuyển từ lõi đơn sang đa lõi(cụ thể là 2) và hiệu quả của kĩ thuật tối ưu bộ nhớ. Các cấu hình được so sánh:

– Single core: toàn bộ quá trình từ chuẩn hóa, mã hóa đến dự đoán được triển khai trên cùng 1 core. Đây là cơ sở để đối sánh.

– Dual core: thực hiện triển khai mô hình trên 2 lõi, tích hợp kĩ thuật Memory tiling. Các phép toán trên hypervector kích thước lớn được chia nhỏ thành các ”tile” nhằm tối ưu tốc độ truy cập trong bộ nhớ cache của esp32. Kịch bản này nhằm chứng minh

việc kết hợp giữa tính toán song song và quản lý bộ nhớ cục bộ mang lại hiệu quả đáng kể.

4.4.2. Khảo sát ảnh hưởng của số chiều đến khả năng song song

Trong Hyperdimensional Computing, số chiều D là siêu tham số quan trọng nhất, ảnh hưởng trực tiếp đến độ chính xác và tài nguyên tiêu thụ. Khảo sát này đánh giá khả năng song song của mô hình khi số chiều thay đổi.

Thí nghiệm thực hiện:

- Thay đổi số chiều D của hypervector theo tập giá trị: $64, 128, 256, 512, 1024, 2048$ và đánh giá kết quả và tốc độ suy luận của mô hình.
- Các chỉ số đo lường: Accuracy trên tập test, thời gian suy luận (từ khi nhận dữ liệu đến khi đưa ra kết quả dự đoán), và dung lượng bộ nhớ tiêu thụ.

Chương 5. Đánh giá

5.1. Đánh giá mô hình HDC trong môi trường phát triển

Mục tiêu của bài báo cáo nhằm hướng đến tăng tốc độ xử lý nên thực hiện chọn số chiều hypervector D = 256, và số bậc trong mã hóa giá trị là 64 có kết quả đánh giá của mô hình tương tự với mô hình ANN (với cấu trúc 2 lớp ẩn 64 và 128 nút) đã huấn luyện trước đây[8].

Bảng 5.1. Đánh giá kết quả huấn luyện mô hình trên tập kiểm tra (test)

Metric	**Tiny ANN**	**HDC**
Accuracy	0.9429	0.9485
Precision	0.9470	0.9602
Recall	0.9429	0.9485
F1 Score	0.9416	0.9461

Dựa trên số liệu tổng hợp tại Bảng Bảng 5.2, kết quả thực nghiệm cho thấy mô hình HDC với cấu hình tối giản ($D = 256$) không chỉ hoàn thành tốt mục tiêu tăng tốc độ xử lý mà còn thể hiện hiệu năng phân loại vượt trội so với mô hình Tiny ANN tham chiếu. Cụ thể, dù hoạt động trong không gian vector giới hạn, HDC vẫn đạt độ chính xác (Accuracy) là 94.85% nhỉnh hơn so với mức 94.29% của ANN, đồng thời ghi nhận mức cải thiện đáng kể về độ chính xác dự báo dương (Precision) khi đạt 0.9602 so với 0.9470. Chỉ số F1-Score đạt 0.9461 tiếp tục khẳng định tính cân bằng và ổn định của giải pháp đề xuất, chứng minh rằng việc giảm số chiều xuống 256 kết hợp với mã hóa 64 bậc không làm suy giảm khả năng học đặc trưng dữ liệu, qua đó xác lập HDC là một giải pháp thay thế hiệu quả, nhẹ nhàng hơn về mặt tính toán nhưng vẫn đảm bảo độ tin cậy cao hơn so với kiến trúc mạng nơ-ron truyền thống.

5.2. Đánh giá về độ trễ và thời gian suy luận, và tài nguyên sử dụng

Thực hiện triển khai mô hình HDC trên vi điều khiển và đánh giá tốc độ suy luận, lượng RAM, Flash tiêu thụ và so sánh giữa việc khi triển khai trên 1 core, 2 core và mô hình ANN trên vi điều khiển esp32-wroom-32.

Số liệu thực nghiệm từ Bảng Bảng 5.2 cung cấp cái nhìn rõ nét về hiệu năng vượt trội của kiến trúc HDC đa lõi trên nền tảng nhúng ESP32, cụ thể như sau:

Bảng 5.2. Đánh giá hiệu suất của các mô hình khi triển khai trên vi điều ESP32

Mô hình	**Tiny ANN**	**HDC (single-core)**	**HDC (dual-core)**
Thời gian suy luận (ms)	2.472	2.502	1.319
RAM (byte)	71876	11396	12708
Flash (byte)	238340	187785	190749

– Hiệu quả tăng tốc nhờ xử lý song song (Parallelism Speedup): Kết quả ấn tượng nhất nằm ở khả năng mở rộng hiệu năng của thuật toán HDC. Khi chuyển từ chế độ đơn lõi (Single-core) sang đa lõi (Dual-core), thời gian suy luận giảm mạnh từ 2.502 ms xuống còn 1.319 ms.

+ Hệ số tăng tốc (Speedup) đạt xấp xỉ 1.897 lần, tiệm cận mức lý tưởng (2.0 lần).

+ Điều này khẳng định kiến trúc tính toán của HDC (dựa trên các phép toán độc lập trên hypervector) cực kỳ tương thích với phần cứng đa lõi, cho phép tận dụng triệt để tài nguyên của ESP32 mà Tiny ANN khó đạt được do phụ thuộc tuần tự giữa các lớp mạng.

– Ưu thế vượt trội về tài nguyên bộ nhớ (Memory Efficiency): So với Tiny ANN, mô hình HDC thể hiện sự tối ưu hóa tài nguyên cực lớn, yếu tố sống còn cho các thiết bị biên (Edge devices):

+ Về RAM: HDC chỉ tiêu thụ khoảng 11-12 KB RAM, thấp hơn 6 lần so với mức 71 KB của Tiny ANN. Việc tiết kiệm hơn 80% dung lượng RAM giúp vi điều khiển có thêm không gian bộ nhớ dồi dào cho các tác vụ khác như ngăn xếp WiFi, Bluetooth hoặc xử lý giao diện người dùng mà không lo tràn bộ nhớ (Stack/Heap Overflow).

+ Về Flash: HDC cũng giúp giảm khoảng 20% dung lượng lưu trữ chương trình (190 KB so với 238 KB), do không cần lưu trữ các ma trận trọng số số thực phức tạp và các thư viện hỗ trợ tính toán dấu phẩy động nặng nề.

– Sự đánh đổi chấp nhận được (Performance Trade-off): Khi so sánh giữa HDC Single-core và Dual-core, ta thấy lượng RAM tiêu thụ có tăng nhẹ (từ 11396 lên 12708 bytes, tăng khoảng 1.3 KB). Sự gia tăng này là không đáng kể và hoàn toàn dễ hiểu, xuất phát từ chi phí quản lý tác vụ (Task overhead), ngăn xếp riêng cho lõi thứ hai (Stack memory) và các cơ chế đồng bộ hóa của FreeRTOS. Tuy nhiên, việc hy sinh một lượng nhỏ RAM để đổi lấy tốc độ xử lý nhanh gấp đôi là một sự đánh đổi hoàn toàn xứng đáng trong các ứng dụng thời gian thực.

Với thời gian suy luận chỉ 1.3 ms và mức tiêu thụ RAM cực thấp, mô hình HDC đa lõi là giải pháp tối ưu nhất cho bài toán giám sát năng lượng trên vi điều khiển ESP32,

vượt trội hoàn toàn so với giải pháp mạng nơ-ron truyền thống (Tiny ANN) về mặt hiệu quả tài nguyên và tốc độ phản hồi.

5.2.1. Phân tích chi tiết phân bố thời gian thực thi

Để hiểu rõ hơn về đặc tính tính toán của thuật toán HDC trên kiến trúc đa lõi, tiến hành đo đạc chi tiết thời gian của hai giai đoạn chính: Mã hóa (Encoding) và Truy vấn tương đồng (Similarity Search). Kết quả đo đạc thực tế từ log hệ thống (System Log) được thể hiện như sau:

Bảng 5.3. Phân bố thời gian xử lý trên từng lõi (Core)

Giai đoạn	**Core 0 (μs)**	**Core 1 (μs)**	**Trung bình (μs)**
Encoding	266	200	233
Similarity	1028	1022	1025
Tổng cộng*	**1294**	**1222**	**—**

**Lưu ý: Tổng thời gian suy luận thực tế (Total Inference) là 1318 μs, bao gồm thời gian xử lý của lõi chậm nhất (Core 0) cộng với chi phí đồng bộ (overhead) khoảng 24 μs.*

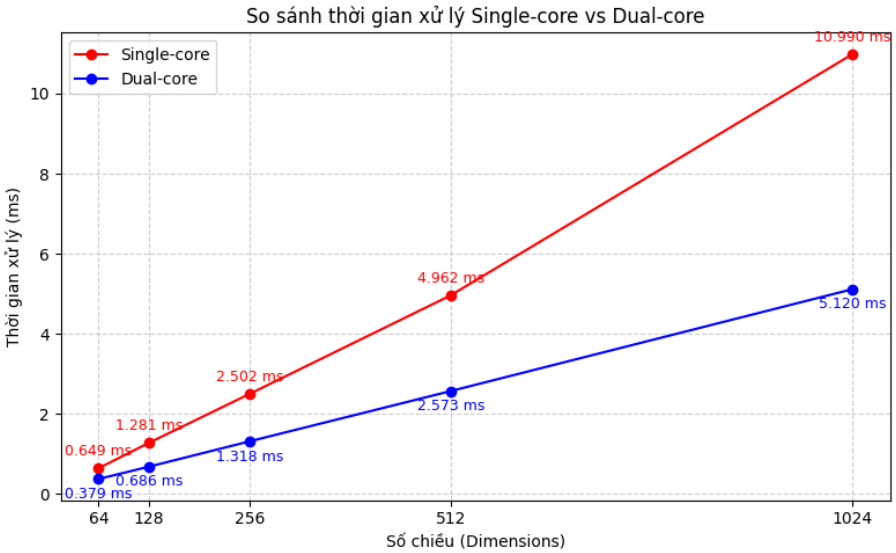
Nhận xét:

- Cân bằng tải (Load Balancing): Khối lượng tính toán được phân chia khá đồng đều giữa hai lõi, đặc biệt là trong giai đoạn Similarity Check (chênh lệch chỉ 6 μs).
- Ảnh hưởng của tác vụ nền: Quan sát cho thấy Core 0 mất nhiều thời gian hơn cho việc Encode (266 μs) so với Core 1 (200 μs). Nguyên nhân là do Core 0 trên ESP32 phải đảm nhiệm thêm các ngắt hệ thống (System Interrupts) và duy trì kết nối Wi-Fi/Bluetooth, trong khi Core 1 được dành riêng (dedicated) cho tác vụ tính toán.
- Xác định điểm nghẽn: Giai đoạn Similarity chiếm khoảng 78% tổng thời gian thực thi ($1028/1318\mu s$). Điều này gợi mở hướng nghiên cứu tiếp theo: để giảm độ trễ xuống dưới 1ms, cần tập trung tối ưu hóa thuật toán tính khoảng cách Hamming hoặc sử dụng các chỉ thị SIMD (nếu phần cứng hỗ trợ) cho giai đoạn này.

Nhìn chung Mô hình HDC có khả năng song song hóa cao, thời gian xử lý đồng bộ overhead chỉ khoảng $24\mu s$, cho thấy khả năng tăng tốc hơn nữa khi số lượng core tăng lên.

5.3. Đánh giá thời gian suy luận khi thực hiện tăng số chiều

Để đánh giá thời gian suy luận khi triển khai mô hình trên 2 cores so với một core, thực hiện thực nghiệm thời gian suy luận 1 mẫu trên vi điều khiển esp32-wroom với 2 cách tiếp cận triển khai trên 1 core và 2 cores.



Hình 5.1. Ảnh hưởng của số chiều đến tốc độ song song của mô hình HDC

Quan sát biểu đồ cho thấy thời gian xử lý của cả hai kịch bản (Single-core và Dual-core) đều tăng tuyến tính theo số chiều D (độ phức tạp $O(D)$). Tuy nhiên, hệ số góc (slope) của đường Dual-core thấp hơn đáng kể so với Single-core:

– Tại $D = 256$: Thời gian giảm từ 2.502 ms xuống 1.318 ms.

– Tại $D = 1024$: Thời gian giảm từ 10.990 ms xuống 5.120 ms.

Điều này cho thấy khi không gian bài toán càng lớn (tăng D), lợi ích của việc song song hóa càng trở nên rõ rệt. Hệ thống đa lõi duy trì được mức tăng tốc (Speedup) ổn định xấp xỉ 2.1 lần ở các chiều cao ($D = 1024$), thậm chí vượt nhẹ mức lý thuyết (2.0x) do giảm tải áp lực lên hàng đợi xử lý của từng lõi đơn lẻ.

Trong quá trình thực nghiệm. Tại mức siêu tham số $D = 2048$:

– Kịch bản Single-core: Hệ thống gặp lỗi Stack Overflow và khởi động lại. Nguyên nhân là do toàn bộ các biến cục bộ, bộ đệm trung gian (intermediate buffers) và ngăn xếp gọi hàm đều dồn lên một tác vụ (Task) duy nhất. Khi D lớn, dung lượng bộ nhớ stack được cấp phát cho tác vụ này bị quá tải.

– Kịch bản Dual-core: Hệ thống vẫn hoạt động ổn định và trả về kết quả chính xác.

Kiến trúc song song giúp phân tán áp lực bộ nhớ (Memory Pressure Distribution). Thay vì dồn gánh nặng lên một ngăn xếp (Stack) của Core 1, dữ liệu và quá trình tính toán được chia nhỏ và xử lý trên hai không gian ngăn xếp riêng biệt của hai tác vụ (Task

0 trên Core 0 và Task 1 trên Core 1). Điều này cho phép hệ thống xử lý được các vector có kích thước lớn hơn gấp đôi so với giới hạn của phương pháp đơn lõi mà không cần thay đổi tổng dung lượng RAM vật lý của vi điều khiển.

Chương 6. Kết luận và phương hướng phát triển

6.1. Kết luận

Nghiên cứu đã trình bày một phương pháp triển khai HDC hiệu năng cao cho bài toán NILM trên vi điều khiển lõi kép. Bằng cách kết hợp Song song hóa dữ liệu (Dimension Splitting) và Tối ưu hóa bộ nhớ (Memory Tiling), đã song song hóa công việc tính toán của mô hình HDC giúp giảm thời gian suy luận.

6.2. Các phát hiện chính

Nghiên cứu này đã triển khai và đánh giá thành công mô hình Tính toán Siêu chiều (Hyperdimensional Computing - HDC) trên nền tảng vi điều khiển ESP32 cho bài toán giám sát tải không xâm nhập (NILM). Các kết quả thực nghiệm dẫn đến những kết luận quan trọng sau:

- Tính khả thi và Hiệu năng vượt trội: Chúng tôi chứng minh rằng HDC là một giải pháp thay thế đầy tiềm năng cho các mạng nơ-ron nhân tạo (ANN) trên các thiết bị biên. Với cấu hình tối ưu ($D = 256$, lượng tử hóa 64 mức), mô hình HDC đạt độ chính xác 94.85%, vượt qua mô hình Tiny ANN tham chiếu (94.29%) trong khi tiêu thụ lượng tài nguyên RAM ít hơn 6 lần (11KB so với 71KB).
- Hiệu quả của kiến trúc Đa lõi: Việc áp dụng kỹ thuật song song hóa trên 2 lõi Xtensa LX6 giúp giảm thời gian suy luận xuống còn 1.319 ms (tăng tốc xấp xỉ 2 lần so với đơn lõi). Quan trọng hơn, kiến trúc này giúp phân tán áp lực bộ nhớ (Memory Pressure), cho phép hệ thống vận hành ổn định ở số chiều cao ($D \geq 2048$) mà không gặp lỗi tràn ngăn xếp (Stack Overflow) như phương pháp đơn lõi truyền thống.
- Sự đánh đổi linh hoạt: Nghiên cứu chỉ ra mối quan hệ tuyến tính giữa độ phức tạp và tài nguyên. Người dùng có thể linh hoạt cấu hình số chiều D: chọn $D = 256$ cho các ứng dụng yêu cầu phản hồi cực nhanh (<1.5ms) hoặc $D = 10000$ cho các ứng dụng ưu tiên độ bền vững (robustness) tối đa.

6.3. Hạn chế

Mặc dù đạt được những kết quả khả quan, hệ thống vẫn tồn tại một số hạn chế cần được ghi nhận:

- Nút thắt cổ chai tại khâu so khớp (Similarity Check): Phân tích thời gian thực thi cho thấy quá trình tính khoảng cách Hamming chiếm tới 78% tổng thời gian suy

luận. Hiện tại, quá trình này vẫn được thực hiện bằng phần mềm thuần túy mà chưa tận dụng được các chỉ thị phần cứng chuyên biệt.

– Phụ thuộc vào dữ liệu sạch: Các thực nghiệm chủ yếu được tiến hành trên bộ dữ liệu VN_DALLE2 với một số thiết bị đơn giản. Hiệu năng của mô hình trong môi trường thực tế với nhiễu điện lưới cao và sự xuất hiện của các thiết bị lạ (unseen appliances) chưa được đánh giá toàn diện.

– Chi phí đồng bộ hóa: Dù đạt tốc độ tăng tốc tốt, việc quản lý đồng bộ giữa hai lõi (Core affinity, Mutex) làm tăng độ phức tạp của mã nguồn và tiêu tốn thêm một lượng nhỏ tài nguyên hệ thống (1.3KB RAM overhead). Điều này cũng dẫn đến việc vi điều khiển sẽ tiêu tốn nhiều năng lượng hơn.

6.4. Đề xuất và hướng phát triển

Dựa trên các phân tích trên, chúng tôi đề xuất các hướng nghiên cứu tiếp theo để hoàn thiện giải pháp:

– Tối ưu hóa mức thấp (Low-level Optimization): Tập trung giải quyết nút thắt cổ chai tại khâu Similarity Check bằng cách sử dụng các chỉ thị SIMD (nếu chuyển sang ESP32-S3) hoặc viết lại các hàm tính toán cốt lõi bằng Assembly để tối đa hóa tốc độ xử lý bit.

– Học trực tuyến trên thiết bị (On-device Learning): Tận dụng ưu điểm lớn nhất của HDC là khả năng học nhanh (one-shot learning). Phát triển tính năng cho phép người dùng gán nhãn dữ liệu mới ngay trên thiết bị để mô hình tự cập nhật mà không cần huấn luyện lại từ đầu trên máy chủ.

– Mở rộng tập dữ liệu: Kiểm thử mô hình trên các bộ dữ liệu đa dạng hơn (ví dụ: UK-DALE) và triển khai hệ thống phần cứng thực tế (Smart Plug) để đánh giá độ trễ và độ chính xác trong điều kiện vận hành thời gian thực tại các hộ gia đình.

Tài liệu tham khảo

[1] G. Tanoni, E. Principi, and S. Squartini, “Non-Intrusive Load Monitoring in industrial settings: A systematic review,” *Renewable and Sustainable Energy Reviews*, vol. 202, p. 114703, Sep. 2024. doi: 10.1016/j.rser.2024.114703.

[2] M. Stock, W. Van Criekinge, D. Boeckaerts, S. Taelman, M. Van Haeverbeke, P. Dewulf, and B. De Baets, “Hyperdimensional computing: A fast, robust, and interpretable paradigm for biological data,” *PLOS Computational Biology*, vol. 20, no. 9, Sep. 2024, Art. no. e1012426. doi: 10.1371/journal.pcbi.1012426.

[3] M. Satyanarayanan, “The Emergence of Edge Computing,” *IEEE Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017. doi: 10.1109/MC.2017.9.

[4] A. Rahimi, P. Kanerva, J. M. Rabaey, “Efficient Biosignal Processing Using Hyperdimensional Computing,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 6, pp. 1469–1478, Dec. 2017.

[5] A. Ashfahani et al., “An Application of Machine Learning in Energy Disaggregation,” *Energy and Buildings*, vol. 166, pp. 351–360, 2018.

[6] M. Imani, J. H. Kim, D. Rosing, “Efficient Hyperdimensional Computing Using Binary Spatter Codes,” in *Proc. IEEE International Conference on Rebooting Computing (ICRC)*, 2017.

[7] A. Ducasse, A. Roux, M. Arzel, and S. Sentieys, “Scalable and High-Throughput Hyperdimensional Computing Inference on Multi-Core CPUs,” in *Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2022, pp. 1–8. doi: 10.1109/ASAP54787.2022.00014.

[8] N. N. Son, N. K. Son, and N. N. An, “Real-time NILM using Tiny-ANN on Sub-dollar Edge Device under Extreme Resource Constraints,” in *Proc. IEEE International Conference on Advanced Technologies for Communications (ATC)*, 2025, pp. 1–6.

[9] Espressif Systems, *ESP32 Technical Reference Manual*, Espressif Systems, 2022.

[10] TechieTonics, *Introduction to Hyperdimensional Computing*, TechieTonics Blog, 2020.