|  |
| --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  **PHÒNG THÍ NGHIỆM EDABK**  Có thể là hình ảnh về văn bản cho biết '11101101 BK EDA-BK LABORATORY'  **HƯỚNG DẪN SỬ DỤNG MÃ NGUỒN**  **C++ ĐỂ MÔ PHỎNG VÀ KIỂM TRA KIẾN TRÚC RANC**  Người thực hiện: Nguyễn Lê Trung  Đặng Bá Luân  Giảng viên hướng dẫn: PGS. TS. Nguyễn Đức Minh  Hà Nội, 01-2022 |

**LỜI NÓI ĐẦU**

Tài liệu này được thành viên phòng thí nghiệm EDABK Viện Điện tử - Viễn thông, trường Đại học Bách Khoa Hà Nội soạn thảo và ban hành nhằm mục đích hướng dẫn sử dụng mã nguồn C++ để mô phỏng và kiểm tra kiến trúc RANC.

Chúng tôi xin chân thành cảm ơn PGS. TS. Nguyễn Đức Mình đã hướng dẫn chúng tôi hoàn thành báo cáo này.

**LỜI CAM ĐOAN**

Chúng tôi là Nguyễn Lê Trung, Đặng Bá Luân thành viên lab EDABK. Người hướng dẫn là PGS. TS. Nguyễn Đức Minh. Chúng tôi xin cam đoan toàn bộ nội dung được trình bày trong *Hướng dẫn sử dụng mã nguồn C++ để mô phỏng và kiểm tra kiến trúc RANC* là kết quả quá trình tìm hiểu và nghiên cứu của chúng tôi. Các dữ liệu được nêu trong bản hướng dẫn là hoàn toàn trung thực, phản ánh đúng kết quả đo đạc thực tế. Mọi thông tin trích dẫn đều tuân thủ các quy định về sở hữu trí tuệ; các tài liệu tham khảo được liệt kê rõ ràng. Chúng tôi xin chịu hoàn toàn trách nhiệm với những nội dung được viết trong đồ án này.

**MỤC LỤC**

[DANH MỤC HÌNH VẼ i](#_Toc92848236)

[CHƯƠNG 1. I/O FILES 1](#_Toc92848237)

[1.1 File JSON đầu vào (input file) 1](#_Toc92848238)

[1.1.1 Thông tin về các packet 1](#_Toc92848239)

[1.1.2 Thông tin về đầu ra 1](#_Toc92848240)

[1.1.3 Thông tin các core 2](#_Toc92848241)

[1.1.4 Thông tin các neuron 2](#_Toc92848242)

[1.1.5 File JSON chứa các thông tin cấp cao của quá trình giả lập (configuration file) 3](#_Toc92848243)

[1.2 File đầu ra (output file) 4](#_Toc92848244)

[1.2.1 Trace file 4](#_Toc92848245)

[CHƯƠNG 2. SƠ LƯỢC VỀ KIẾN TRÚC MẠNG 5](#_Toc92848246)

[2.1 Phương thức sinh kích thích đầu vào cho mạng 5](#_Toc92848247)

[2.2 Kiến trúc 6](#_Toc92848248)

[2.3 Giải mã đầu ra từ các kích thích thu được 6](#_Toc92848249)

[CHƯƠNG 3. TẠO FILE ĐẦU VÀO 8](#_Toc92848250)

[CHƯƠNG 4. BIÊN DỊCH CODE SIMULATOR 10](#_Toc92848251)

[CHƯƠNG 5. TIẾN HÀNH MÔ PHỎNG 11](#_Toc92848252)

[5.1 Mô phỏng bằng code C++ 11](#_Toc92848253)

[5.2 Kiểm tra kết quả mô phỏng 12](#_Toc92848254)

[THAM KHẢO 14](#_Toc92848255)

# DANH MỤC HÌNH VẼ

[*Hình 1: Quy trình nhận diện ảnh 5*](#_Toc92848256)

[*Hình 2: Quy trình mã hóa ảnh và kiến trúc mạng 6*](#_Toc92848257)

[*Hình 3: Dataflow 8*](#_Toc92848258)

[*Hình 4: Google Colaboratory 9*](#_Toc92848259)

[*Hình 5: Tạo thư mục 10*](#_Toc92848260)

[*Hình 6: Quá trình biên dịch 10*](#_Toc92848261)

[*Hình 7: Các chức năng của chương trình mô phỏng 11*](#_Toc92848262)

[*Hình 8: Quá trình mô phỏng 12*](#_Toc92848263)

[*Hình 9: Upload đầu ra file mô phrong lên Google Colab 12*](#_Toc92848264)

[*Hình 10: Kết quả so sánh 13*](#_Toc92848265)

# I/O FILES

Để tiến hành mô phỏng, cần phải có:

* File JSON đầu vào (input file).
* File JSON chứa các thông tin cấp cao của quá trình giả lập (configuration file).

Sau khi mô phỏng, thu được file đầu ra và có thể có trace file.

## File JSON đầu vào (input file)

Là một file JSON bao gồm thông tin về các packet sẽ được đưa vào mô phỏng, các tham số của các core và neuron sẽ dùng trong mô phỏng. Các thông tin sẽ được phân chia thành các object tương ứng.

File này sẽ được tạo ra từ “*tealayer*”. Chi tiết ở chương 03.

Một ví dụ về input file: [0]

### Thông tin về các packet

Thông tin về các packet được lưu trong object “*packets*”:

{

"destination\_core": một mảng gồm 2 số nguyên chỉ định tọa độ của destination core,

"destination\_axon": một số nguyên chỉ định địa chỉ destination axon,

"destination\_tick": một số nguyên chỉ định packet này sẽ đến vào tick nào

}

### Thông tin về đầu ra

Tiếp theo, thông tin về bus chứa các spike để đưa ra ngoài được lưu trong object “*output\_bus*”:

{

"coordinates": một mảng gồm 2 số nguyên chỉ định tọa độ của output\_bus

"num\_outputs": số lượng axon ở trong output\_bus (số lượng đầu ra)

}

Các neuron sẽ gửi spike đến “*output\_bus*” thông qua “*coordinates*”. “*num\_outputs*” chỉ định số lượng axon trong output bus này. Các neuron có thể gửi spike tới các axon này tương tự như đối với các core.

### Thông tin các core

Sau đó, thông tin của tất cả các core được lưu trong object “*cores*” dưới dạng mảng 1 chiều, trong đó mỗi phần tử của mảng bao gồm thông tin của một core:

{

"axons": là một mảng số nguyên 1 chiều chứa thông tin về các neuron instruction cho mỗi axon (chỉ định loại axon),

"neurons": là một mảng các neuron (giải thích bên dưới),

"connections": là một mảng 2 chiều chứa thông tin về các synaptic connection trong 1 core: mảng thứ i chứa thông tin về các synaptic connection cho neuron thứ i,

"coordinates": một mảng 2 chiều chứa tọa độ của core

}

### Thông tin các neuron

Thông tin của các neuron được lưu trong object “*neurons*” nằm trong object “*cores*”, nó bao gồm:

{

"reset\_potential": là một số nguyên biểu diễn reset potential của neuron,

"weights": là một mảng 1 chiều chứa các số nguyên bao gồm các weight của các loại axon,

"leak": là một số nguyên biểu diễn leak của neuron,

"positive\_threshold": là một số nguyên biểu diễn positive threshold của neuron,

"negative\_threshold": là một số nguyên biểu diễn negative threshold của neuron,

"destination\_core\_offset": là một mảng chứa 2 số nguyên biểu diễn tọa độ (dx, dy) của core đích đến của spike bắn ra từ neuron này, [0, 0] ám chỉ rằng spike bắn ra từ neuron này sẽ quay lại chính nó,

"destination\_axon": là một số nguyên biểu diễn index của axon đích đến của spike bắn ra từ neuron này,

"destination\_tick": là một số nguyên biểu diễn index của destination tick của spike bắn ra từ neuron này,

"current\_potential": là một số nguyên biểu diễn điện áp ban đầu của neuron,

"reset\_mode": là một số nguyên chỉ định mode reset của neuron (giải thích bên dưới)

}

Có 02 mode reset:

* 0: Absolute Reset:

Nếu điện áp hiện tại (*current\_potential*) lớn hơn hoặc bằng ngưỡng dương (*positive\_threshold*), nó sẽ reset về “*reset\_potential*”. Nếu điện áp hiện tại nhỏ hơn ngưỡng âm (*negative\_threshold*), nó sẽ reset về “*-reset\_potential*”.

* 1: Linear Reset:

Nếu điện áp hiện tại (*current\_potential*) lớn hơn hoặc bằng ngưỡng dương (*positive\_threshold*), nó sẽ reset về “*positive\_threshold - current\_potential*”. Nếu điện áp hiện tại nhỏ hơn ngưỡng âm (*negative\_threshold*), nó sẽ reset về “*negative\_threshold + current\_potential*”

### File JSON chứa các thông tin cấp cao của quá trình giả lập (configuration file)

Là một file JSON chứa các thông tin bậc cao về quá trình giả lập, nó bao gồm:

{

"num\_neurons": số lượng neuron trong 1 core,

"num\_axons": số lượng axon trong 1 core,

"num\_cores\_x": số lượng core theo chiều x,

"num\_cores\_y": số lượng core theo chiều y,

"num\_weights": số lượng weight chứa trong 1 neuron,

"max\_tick\_offset": số lượng tick offset tối đa trong scheduler,

"neuron\_block\_trace\_verbosity": giải thích bên dưới,

"core\_controller\_trace\_verbosity": giải thích bên dưới,

"scheduler\_trace\_verbosity": giải thích bên dưới

}

Chú ý: Khi tiến hành giả lập, sẽ có tổng cộng num\_cores\_x\*num\_cores\_y sẽ được triển khai. Tuy nhiên, những core không được khai báo trong file input sẽ không kết nối với bất kì core nào khác và các neuron trong core này sẽ mang threshold bằng 1, đồng nghĩa những neuron trong những core này sẽ không bao giờ sinh ra spike.

Một ví dụ về file config: [1]

## File đầu ra (output file)

File đầu ra sẽ chứa thông tin về các spike bắn ra từ output\_bus cho mỗi tick. Cụ thể nó sẽ là một ma trận, trong đó các hàng chỉ các tick, các cột chỉ các đầu ra của output\_bus.

Ví dụ tại hàng 2 cột 3 có “1” nghĩa là vào thời điểm tick thứ 2, tại đầu ra thứ 3 sẽ có 1 spike.

### Trace file

Ghi lại tất cả những gì diễn ra trong quá trình mô phỏng, đối với mỗi mode trace (có thể chỉnh trong config file) sẽ cho ra một trace file:

* neuron\_block\_trace\_verbosity:
* 1: Ghi lại quá trình leaky-integrate-and-fire của mỗi neuron tại mỗi tick.
* 0: Ghi lại các axon có kết nối và nhận spike từ mỗi neuron xác định tại mỗi tick.
* core\_controller\_trace\_verbosity:
* 1: Ghi lại các spike nhận được tại mỗi core ở mỗi tick thành các giá trị nhị phân.
* scheduler\_trace\_verbosity:
* 1: Ghi lại quá trình sram hoạt động mỗi khi nó được cập nhật.
* 0: Ghi “1” mỗi khi có dữ liệu mới đưa vào hoặc dữ liệu được đưa ra

# SƠ LƯỢC VỀ KIẾN TRÚC MẠNG

Tập dataset MNIST sẽ được sử dụng để tiến hành mô phỏng.

## Phương thức sinh kích thích đầu vào cho mạng

Đầu vào của mạng được lấy từ 10.000 ảnh trong tập test của MNIST dataset. Kích thước mỗi ảnh là 28x28, thể hiện một số viết tay bất kì từ 0 đến 9 (10 class).

Diagram

Description automatically generated

Hình : Quy trình nhận diện ảnh

Các ảnh chữ số viết tay được mã hóa trở thành các spike như sau:

* Các pixels có giá trị lớn hơn hoặc bằng 126 được chuẩn hóa về giá trị 1.
* Các pixels có giá trị nhỏ hơn 126 được chuẩn hóa về giá trị 0.
* Những vị trí có giá trị 0 được coi là không có kích thích và bị bỏ qua.
* Những vị trí có giá trị 1 được mã hóa thành các gói kích thích có độ rộng 30 bits, gửi vào mạng SNN ngay khi tín hiệu tick được kích hoạt.
* Nội dung của các gói kích thích phụ thuộc vào số lượng Core đầu vào và việc chia ảnh theo các Core đầu vào này.

## Kiến trúc

Diagram

Description automatically generated

Hình : Quy trình mã hóa ảnh và kiến trúc mạng

Để mã hóa ảnh MNIST 28x28, trong bài sử dụng mô hình mạng SNN gồm 2 tầng, trong đó tầng đầu gồm 4 Core, tầng thứ hai gồm 1 Core:

* 4 Core đầu vào: mỗi Core gồm 256 Axons, mỗi Axons sẽ nhận kích thích ứng với một pixel trên ảnh. Như vậy một Core chỉ có thể nhận được tối đa 1 ảnh 16x16 pixels. Vì vậy để có thể đọc toàn bộ 1 ảnh 28x28, ta cần sử dụng 4 Core quét qua toàn bộ ảnh như trên Hình 2.
* Mỗi Core đầu vào nhận kích thích từ 256 Axons, sau đó tính toán và đưa kích thích ra từ 64 Neurons của mình. Ở đây mỗi Core tại tầng 1 chỉ sử dụng tới 64 Neurons để phóng kích thích thay vì dùng toàn bộ 256 Neurons. Lý do vì tầng 2 chỉ gồm 1 Core với 256 Axons đầu vào, nghĩa là tầng 2 có thể nhận tối đa 256 kích thích từ tầng 1 (64 Neurons x 4 Cores = 256).

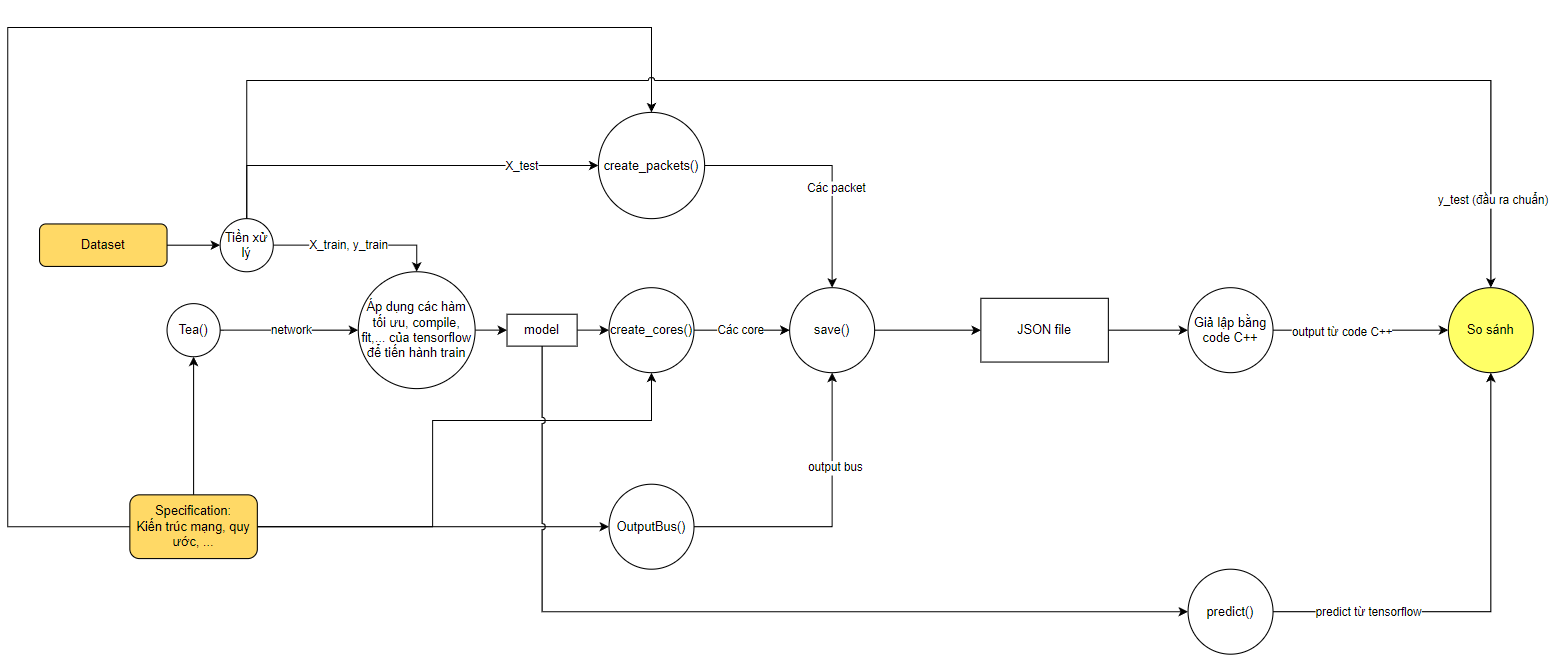
Lưu ý: 1 pixel có thể gửi gói tới nhiều hơn 1 Core đầu vào, nếu các Core này quét qua phần ảnh trùng lấn với nhau.

## Giải mã đầu ra từ các kích thích thu được

Tại Core đầu ra, ta cũng chỉ sử dụng 250 Neurons phóng kích thích ra ngoài mạng thay vì 256 Neurons, để phù hợp với số lượng class (10 class ứng với các số từ 0 đến 9, và 250 chia hết cho 10). Trong đó 25 Neurons đầu tiên sẽ vote cho số 0, 25 Neurons tiếp theo vote cho số 1,… tương tự cho các số (class) còn lại. Mỗi kích thích được phóng ra từ 1 Neuron đầu ra sẽ được tính là 1 vote.

Như vậy, sau khi hoàn thành xử lý cho một ảnh, đầu ra sẽ thu được các lượt votes ứng với các class, dựa trên số Neurons phóng kích thích. Mỗi class sẽ có tối đa 25 lượt votes (đạt giá trị này khi cả 25 Neurons ứng với class này đều phóng kích thích). Class nào có số lượt votes lớn nhất sẽ được chọn là class dự đoán của mạng tại tick đó.

# TẠO FILE ĐẦU VÀO



Hình 3: Dataflow

Tiến hành giả lập với 10000 ảnh trong tập MNIST.

Sao chép file này về:



File này được chỉnh sửa lại từ “MNIST Example using tealayer2.0” [2]

Truy cập: <https://colab.research.google.com/>

Tiến hành đăng nhập bằng tài khoản Google.

File → Upload notebook → Choose File → Tiến hành upload file vừa tải

Graphical user interface, text, application, email, Teams

Description automatically generated

Hình : Google Colaboratory

Icon

Description automatically generated with medium confidenceThực hiện như hướng dẫn trong file. Để chạy từng đoạn code, nháy vào biểu tượng

# BIÊN DỊCH CODE SIMULATOR

Giải nén file này:

[3]

Trong môi trường linux, truy cập vào folder vừa giải nén, tạo folder “build”

Graphical user interface, application

Description automatically generated

Hình : Tạo thư mục

Truy cập folder build → Chuột phải → Open In Terminal

Gõ “cmake ..” → “make” để tiến hành biên dịch code:

Text

Description automatically generated

Hình : Quá trình biên dịch

# TIẾN HÀNH MÔ PHỎNG

## Mô phỏng bằng code C++

Tiến hành paste file “mnist\_config.json” vừa tải về và configuration file [1] vào folder “build” này.

Trong cửa sổ terminal tại folder “build”, tiến hành gõ “./ranc\_sim -h” để mở mục “help”

Text

Description automatically generated

Hình : Các chức năng của chương trình mô phỏng

Đây chính là các chế độ của chương trình, giả sử chạy giả lập với 10000 ảnh thuộc tập MNIST, tiến hành gõ:

“./ranc\_sim -i ./mnist\_config.json -o ./simulator\_output.txt -c ./config.json 10002”

Sau đó nhấn Enter

Lưu ý: Tại sao lại là con số 10002, đó là do kiến trúc tạo ra lưu trong file “mnist\_config.json” bao gồm 2 layer, layer đầu có 4 core, layer sau có 1 core. Khi tick đầu tiên đến, layer đầu tiên mới bắt đầu nhận các packet, layer thứ 2 vẫn chưa nhận được gì, khi tick thứ 2 đến, layer thứ 2 mới nhận được các packet từ layer đầu tiên, và phải đến tick thứ 3 đầu ra mới xuất hiện. Do vậy cần phải chạy mô phỏng trong 10002 tick.

Text

Description automatically generated

Hình : Quá trình mô phỏng

Sau khi quá trình giả lập kết thúc, thu được file “simulator\_output.txt”

## Kiểm tra kết quả mô phỏng

Tiến hành upload file output vừa sinh ra lên Files của Google colab ở chương 03.

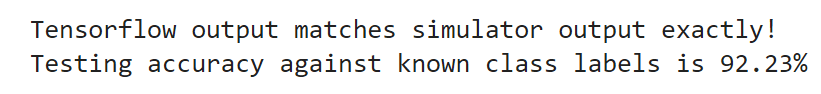
Lưu ý: Xóa đi 2 dòng đầu của file “simulator\_output.txt” trước khi update, 2 dòng này là output do trễ 2 tick như đã giải thích ở trên.

Graphical user interface, text, application

Description automatically generated

Hình : Upload đầu ra file mô phrong lên Google Colab

Chạy đoạn code cuối cùng “SO SÁNH ĐẦU RA TỪ SIMULATOR VỚI ĐẦU RA CỦA TENSORFLOW”



Hình : Kết quả so sánh

Có thể thấy, kết quả giả lập từ code C++ hoàn toàn giống với predict của tensorflow.

# THAM KHẢO

[0] [RANC/input.json at master · UA-RCL/RANC (github.com)](https://github.com/UA-RCL/RANC/blob/master/simulator/data/example/input.json)

[1] [RANC/config.json at master · UA-RCL/RANC (github.com)](https://github.com/UA-RCL/RANC/blob/master/simulator/config.json)

[2] [RANC/README.md at master · UA-RCL/RANC (github.com)](https://github.com/UA-RCL/RANC/blob/master/software/tealayers/README.md)

[3] [RANC/simulator at master · UA-RCL/RANC (github.com)](https://github.com/UA-RCL/RANC/tree/master/simulator)