LẬP TRÌNH SONG SONG ỨNG DỤNG

BÁO CÁO TIẾN ĐỘ

PLANT LEAF DISEASE DETECTION USING XGBOOST

Nhóm Double Slash

20120165 - Hồng Nhất Phương

19120522 - Phạm Quốc Hưng

01 Nhắc lại về XGBoost

02 Áp dụng các kĩ thuật cải tiến

03 Huấn luyện và đánh giá mô hình

04 Tổng kết

01

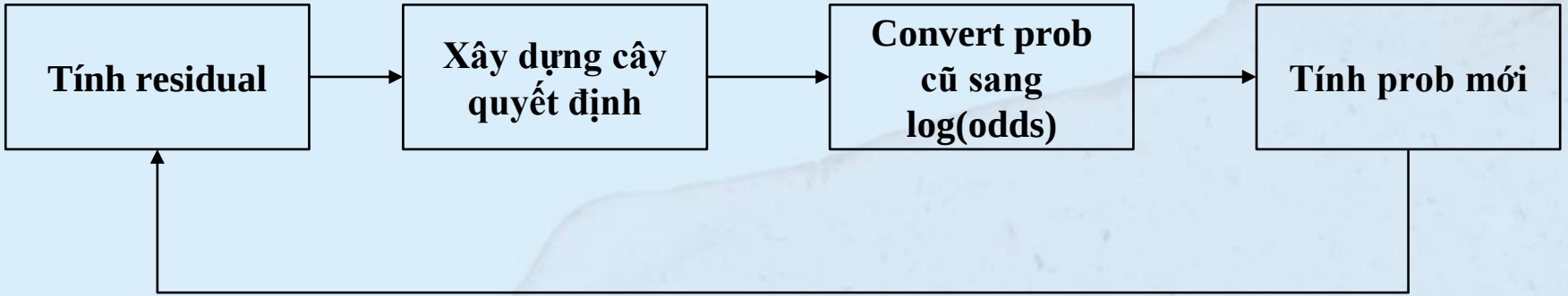
Nhắc lại về XGBoost

Sơ lược về XGBoost

XGBoost là một mô hình học máy sử dụng kĩ thuật Gradient Boosting để giải quyết các bài toán phân lớp và hồi quy.

Ý tưởng của Gradient Boosting là kết hợp nhiều weak models, mỗi model sẽ cố gắng giảm thiểu độ lỗi trong kết quả dự đoán của các mô hình trước đó, để có được kết quả dự đoán cuối cùng tốt hơn.

Tóm tắt các giai đoạn chính trong XGBoost



Bacterial spot

Early blight

Healthy

Late blight

Leaf mold

Septoria leaf spot

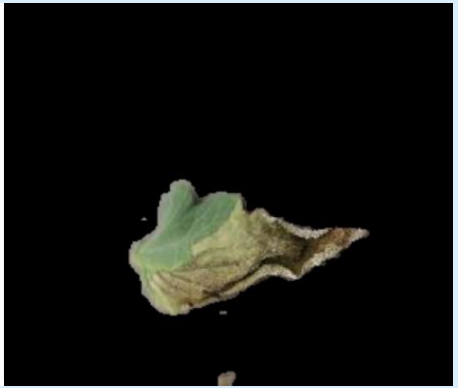
Spider mites

Target spot

Mosaic virus

Yellow leaf curl virus

Image segmentation



Color histogram

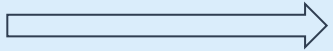
Haralick features

Hu moments

[color histogram haralick hu moments]

[color histogram haralick hu moments]
[color histogram haralick hu moments]
[color histogram haralick hu moments]
[color histogram haralick hu moments]
[color histogram haralick hu moments]
[color histogram haralick hu moments]

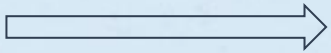
Min Max Scale



X

[healthy Leaf_mold ..... Early_blight]

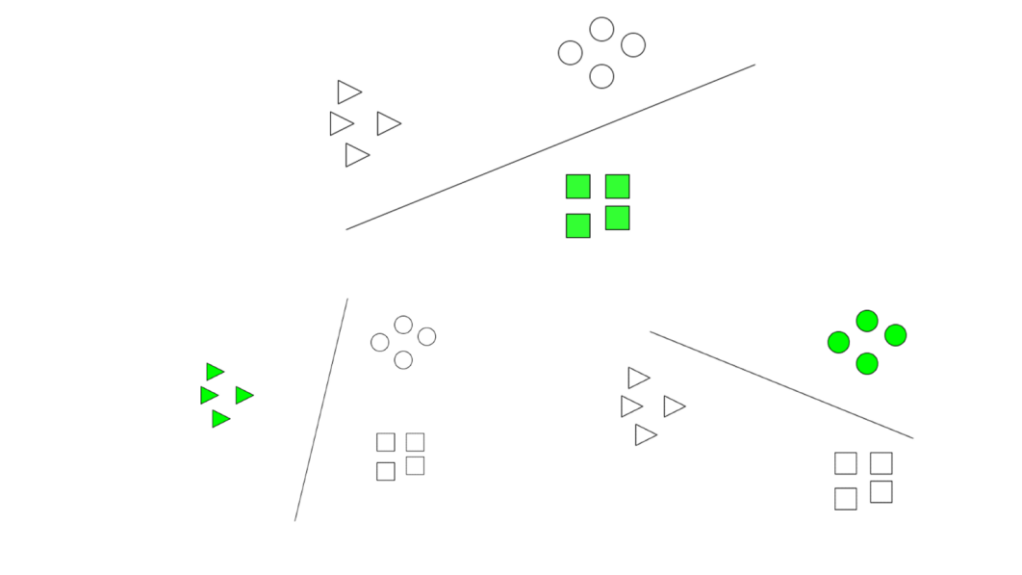
Label encoder



y

Multiclass problem

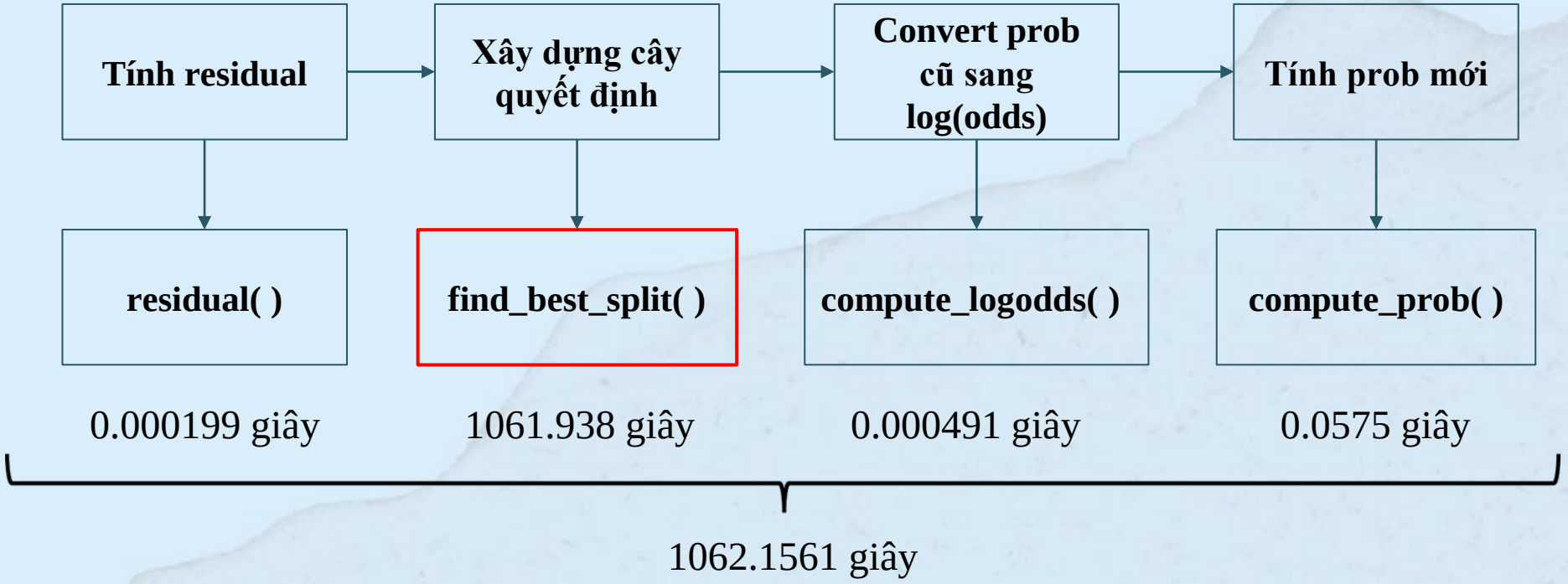
Sử dụng phương pháp One vs Rest (OvR) để giải quyết bài toán phân loại đa lớp
Chiến thuật của **OvR** là phân tách bài toán phân loại đa lớp ban đầu thành **n** bài toán phân loại hai lớp tương ứng với **n** lớp phải phân loại.



Bộ dữ liệu huấn luyện: bộ dữ liệu nhỏ 3000 mẫu (3 lớp)
Chiến lược: OvR

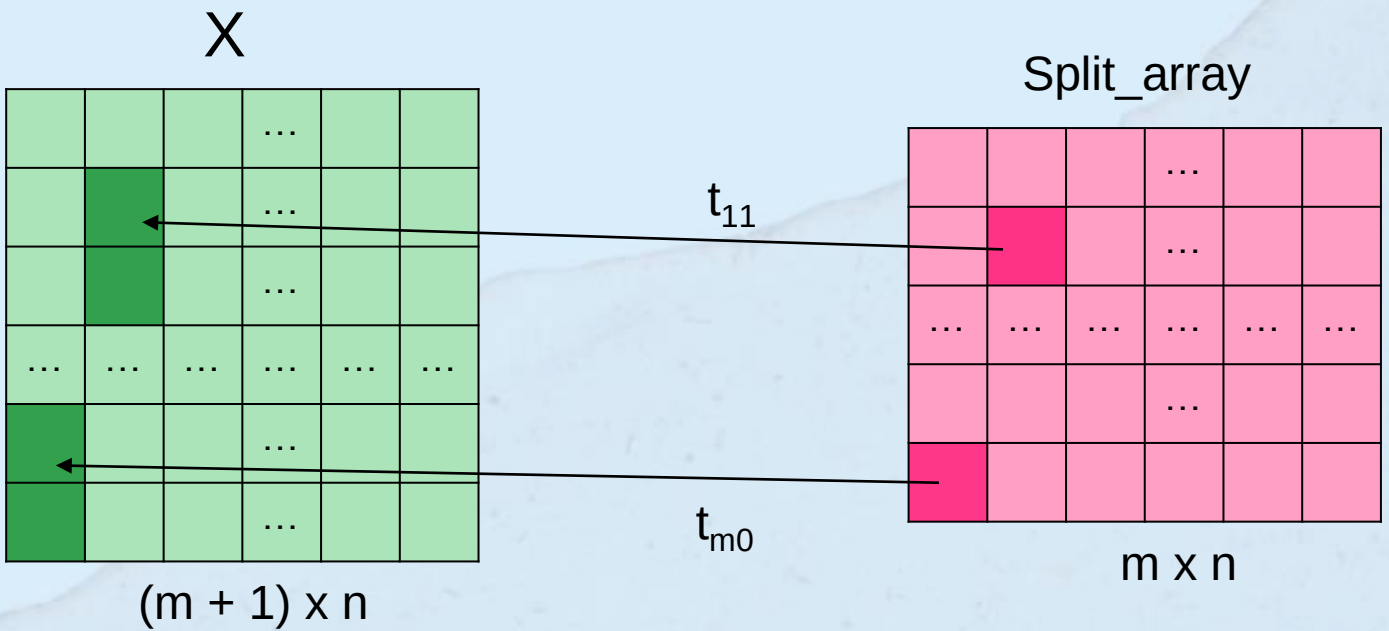
Số classes	Kích thước dữ liệu	Accuracy tuần tự	Thời gian chạy tuần tự
2	3000	0.95	350 – 370s
3	3000	0.946	1060 – 1150s

Khảo sát thời gian thực thi của các giai đoạn chính trong XGBoost



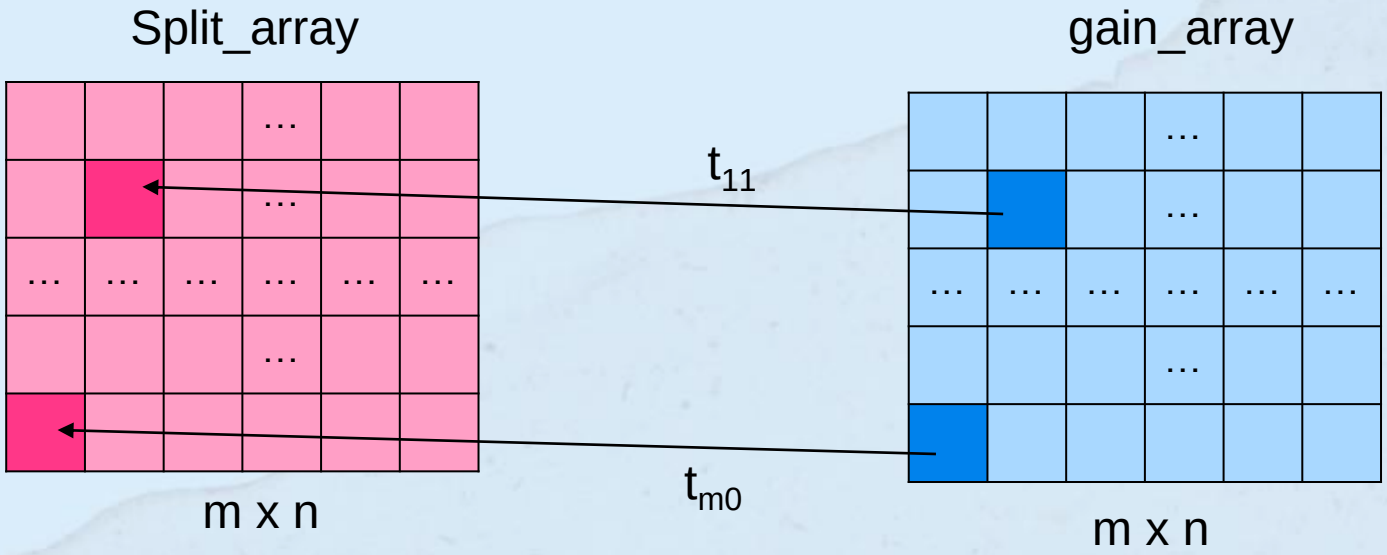
Song song hóa hàm find_best_split()

Bước 1: Tính split_array

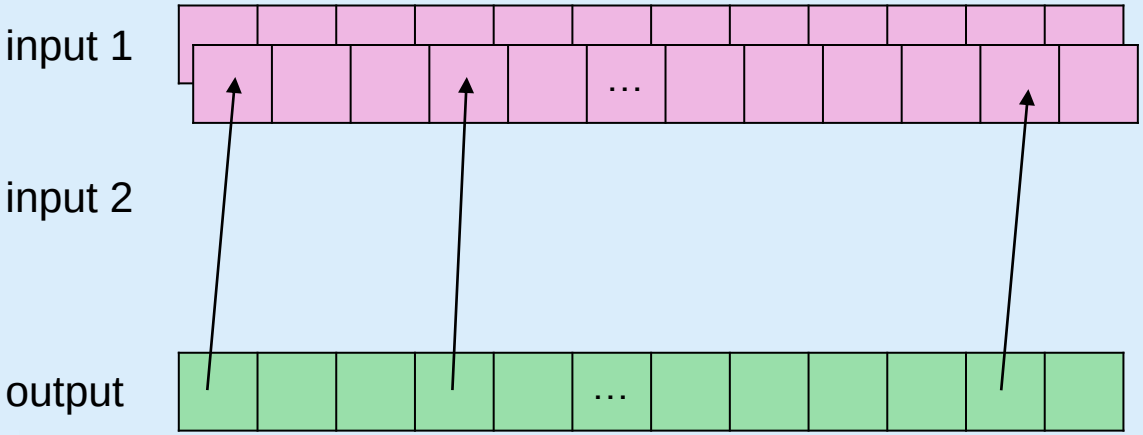


Song song hóa hàm find_best_split()

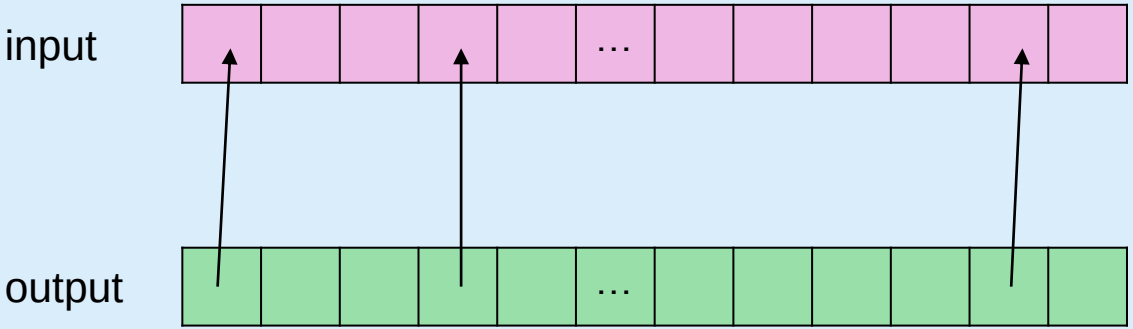
Bước 2: Tính gain_array



Song song hóa hàm residual()



Song song hóa hàm compute_logodds() và compute_prob()



Bộ dữ liệu huấn luyện giống với phiên bản tuần tự
Chiến lược: **OvR** giống với phiên bản tuần tự

Số classes	Kích thước dữ liệu	Accuracy tuần tự	Accuracy song song	Thời gian chạy tuần tự	Thời gian chạy song song
2	3000	0.95	0.95	350 – 370s	4 – 6s
3	3000	0.946	0.946	1060 – 1150s	9 – 11s

Bộ dữ liệu huấn luyện **10 lớp đầy đủ**
Chiến lược: OvR giống với phiên bản tuần tự

Số classes	Kích thước dữ liệu	Accuracy song song	Thời gian chạy song song	Note
2	3000	0.95	4 – 6s	
3	3000	0.946	9 – 11s	
10	10.000	0.891	2700 – 2800s	***tuần tự 3000 mẫu ~ 1200 giây***

Bộ dữ liệu huấn luyện **10 lớp đầy đủ**
Chiến lược: **OvR** và **under sampling** (40 : 60)

Số classes	Kích thước dữ liệu	Accuracy tuần tự	Thời gian chạy song song	Note
2	3000	0.95	4 – 6s	
3	3000	0.946	9 – 11s	
10	10.000	0.891	2700 – 2800s	
10	10.000	0.903	700 – 800s	Dùng under sampling

02

Áp dụng các kĩ thuật cải tiến

Các hàm được song song hóa

compute_split_value()

Tạo matrix split value

compute_gain()

Tạo matrix gain value

residual()

Tính sự chênh lệch giữa y thực tế và y dự đoán

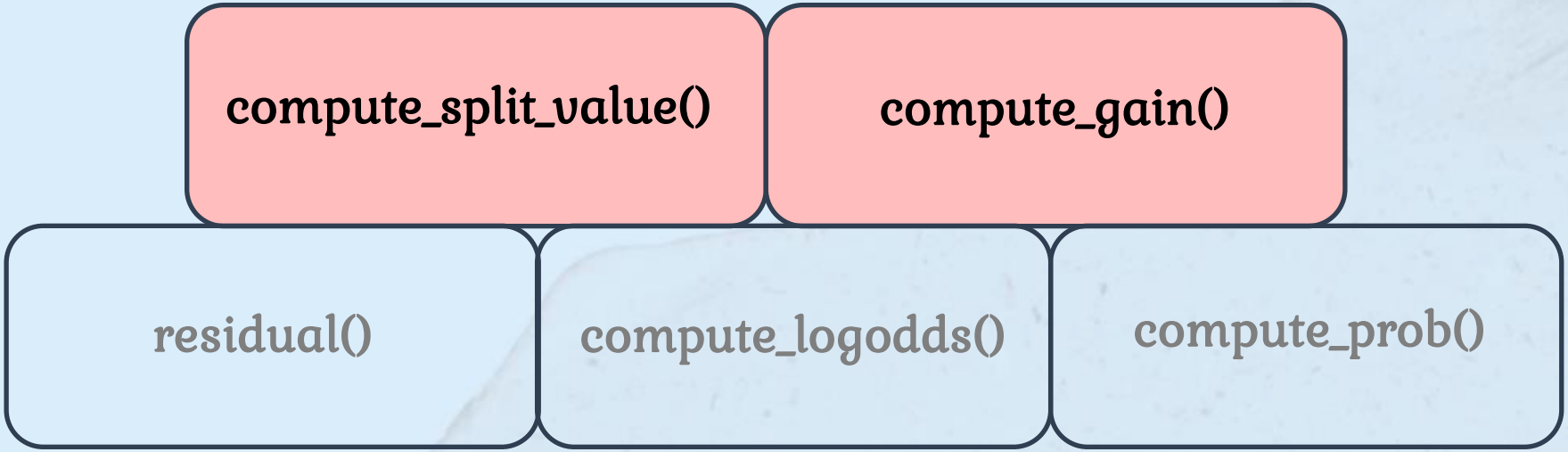
compute_logodds()

Chuyển giá trị xác suất p thành giá trị log(odds)

compute_prob()

Chuyển giá trị log(odds) thành giá trị xác suất p

Sử dụng shared memory



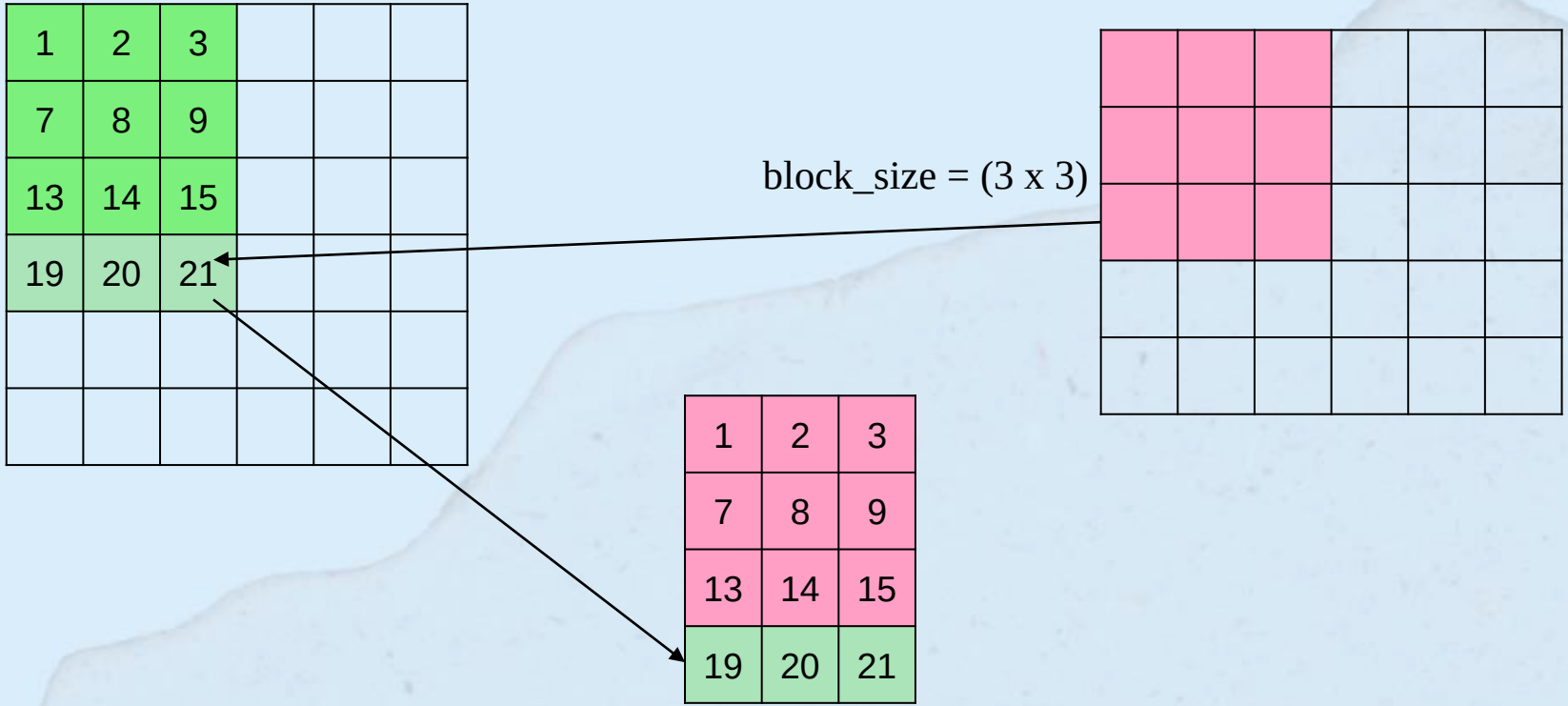
compute_split_value()

Dữ liệu của sample X nằm ở bộ nhớ GMEM và mỗi lần đọc dữ liệu của X thì các thread đều phải chạy xuống GMEM.

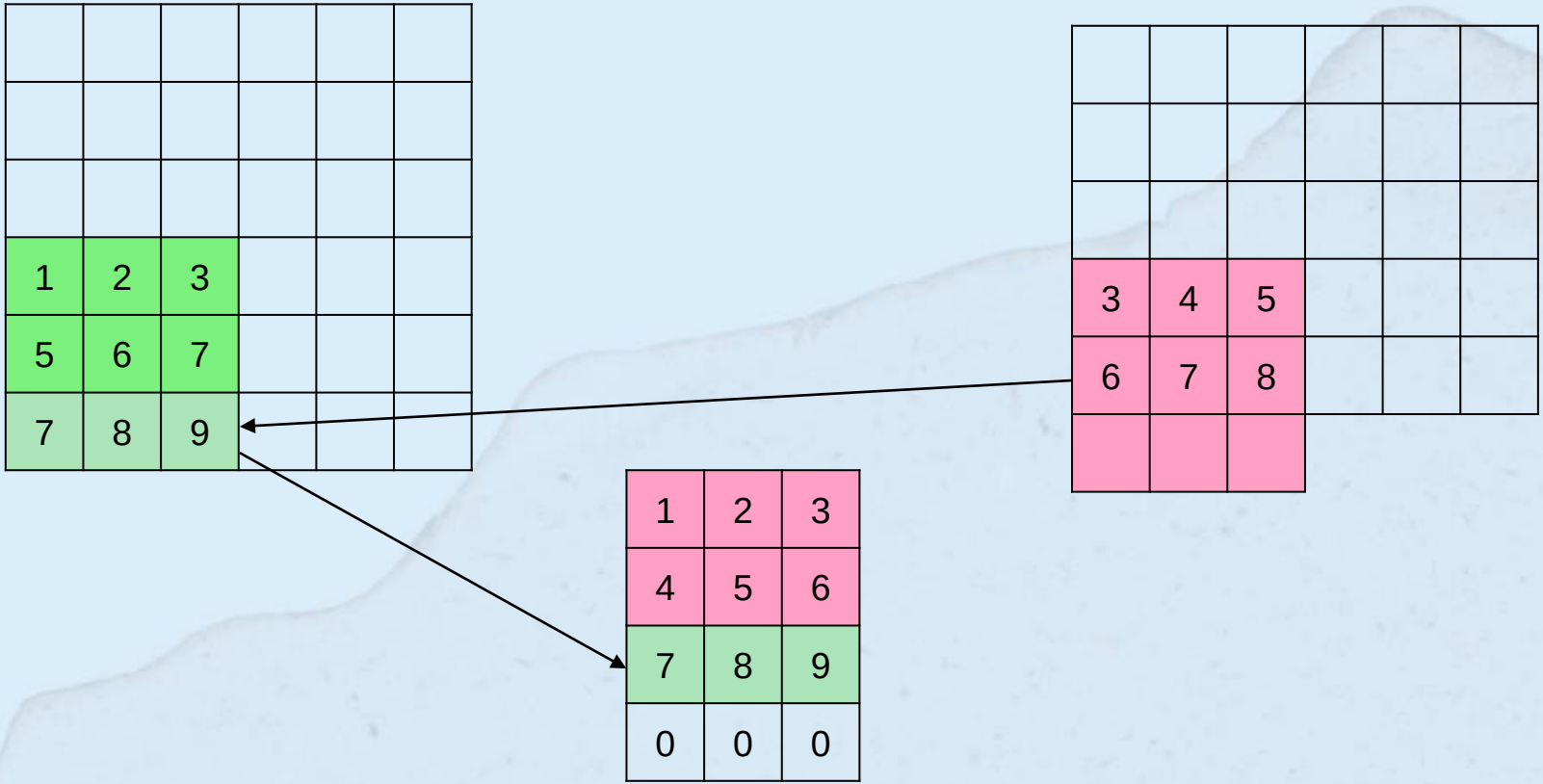
Vì vậy sử dụng SMEM chứa phần dữ liệu mong muốn để tính split_value.

Giả sử kích thước block là 3 x 3, thì kích thước share là 4 x 3 (vì 1 split_value cần 2 giá trị x). Các thread sẽ lấy phần dữ liệu ứng với chỉ số ***(row, col)*** của chúng và ghi vào SMEM, đối với phần dữ liệu còn lại sẽ do ***thread cuối*** xử lý

threadIdx.y == (blockDim.y – 1) *or row == (split_arr.shape[0] – 1)*



threadIdx.y == (blockDim.y – 1) or ***row == (split_arr.shape[0] – 1)***

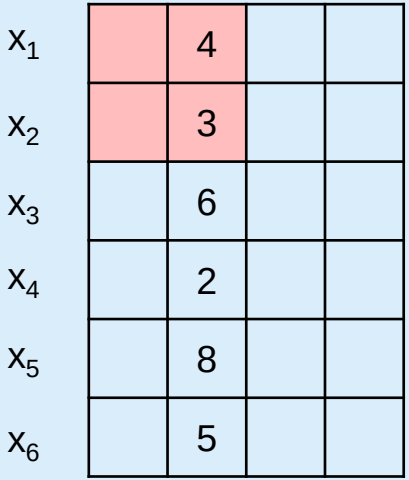


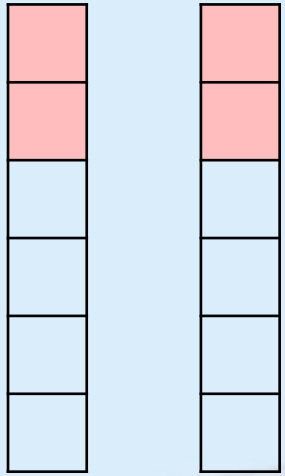
compute_gain()

Với root gain, ta sẽ để thread có chỉ số ***(tx = 0, ty = 0)*** load dữ liệu vào SMEM.

Với các dữ liệu còn lại, ta sẽ chia phần dữ liệu mà block cần thành nhiều phần nhỏ:

- Ở lần xử lý đầu tiên, block sẽ load một phần nhỏ của dữ liệu từ GMEM vào SMEM, rồi mỗi thread trong block đọc dữ liệu ở SMEM để tính một phần kết quả
- Ở lần xử lý kế, block sẽ load phần nhỏ tiếp theo và tính tiếp từ kết quả đang tính trước đó, tiếp tục cho đến hết.





Split_arr

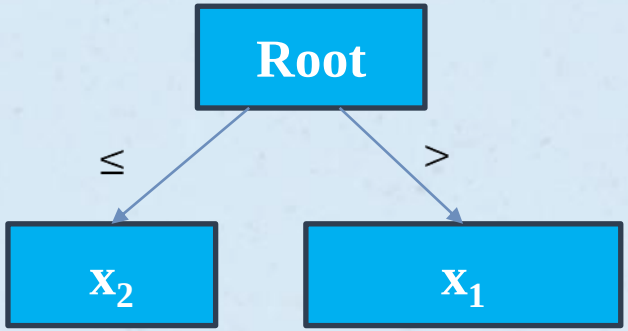
	3		

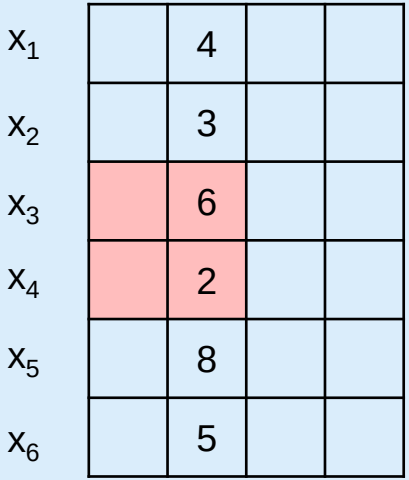
nu_left = ….

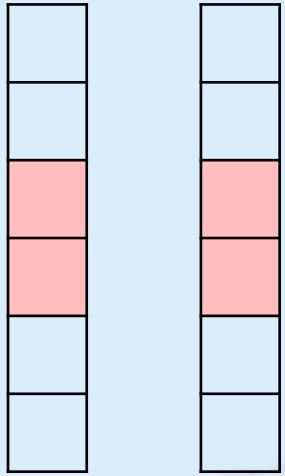
de_left = …

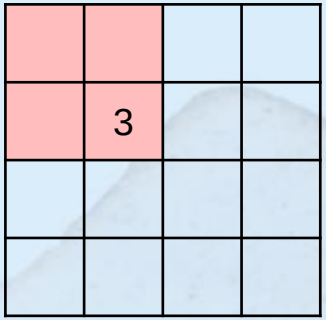
nu_right = …

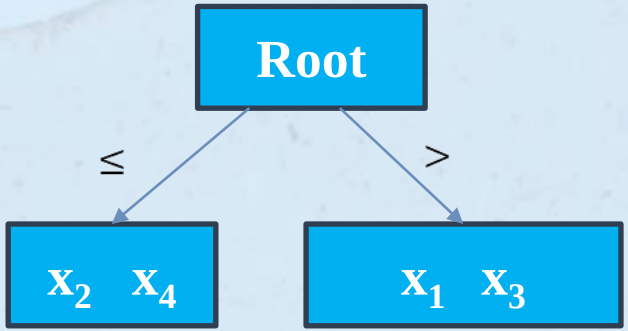
de_right = …









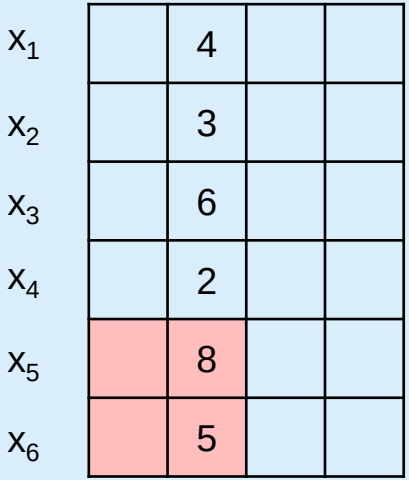


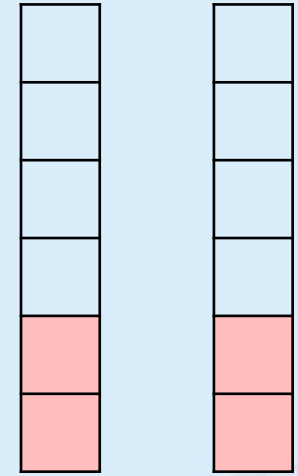
nu_left = ….

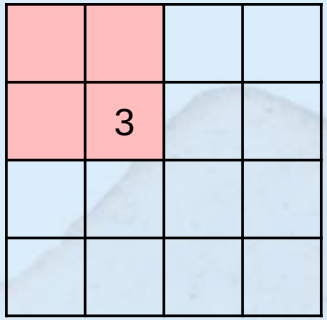
de_left = …

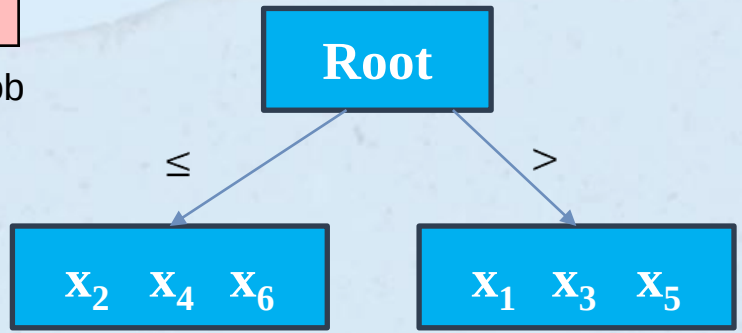
nu_right = …

de_right = …









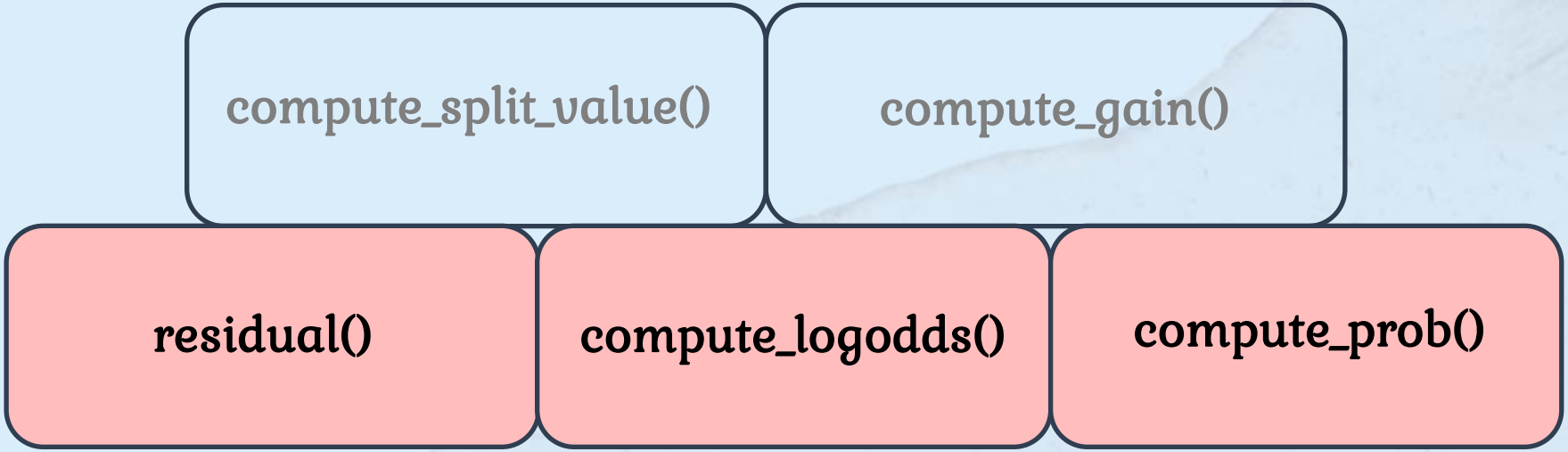
nu_left = ….

de_left = …

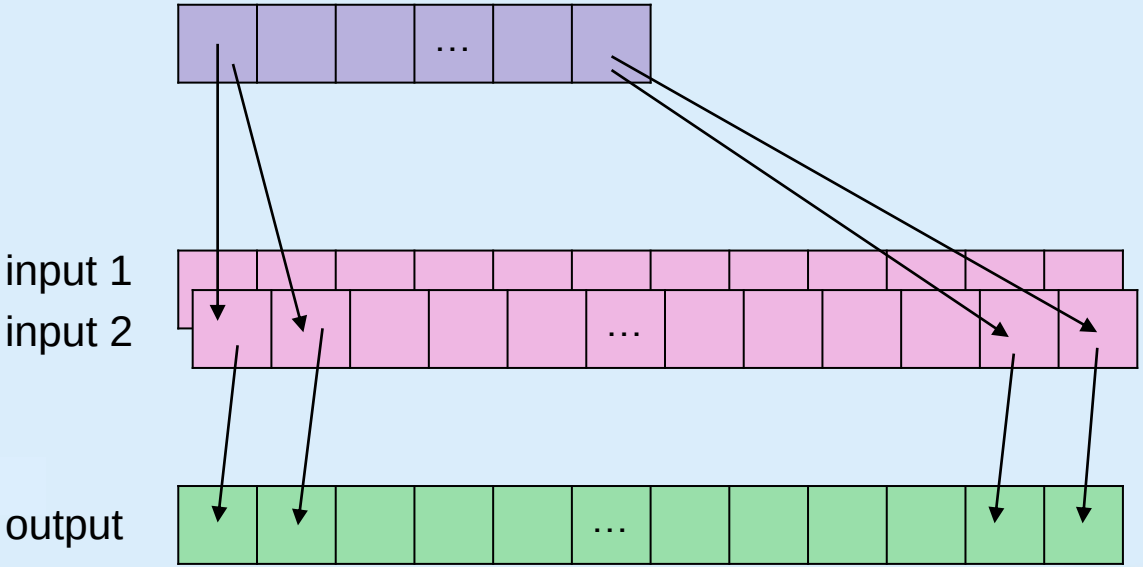
nu_right = …

de_right = …

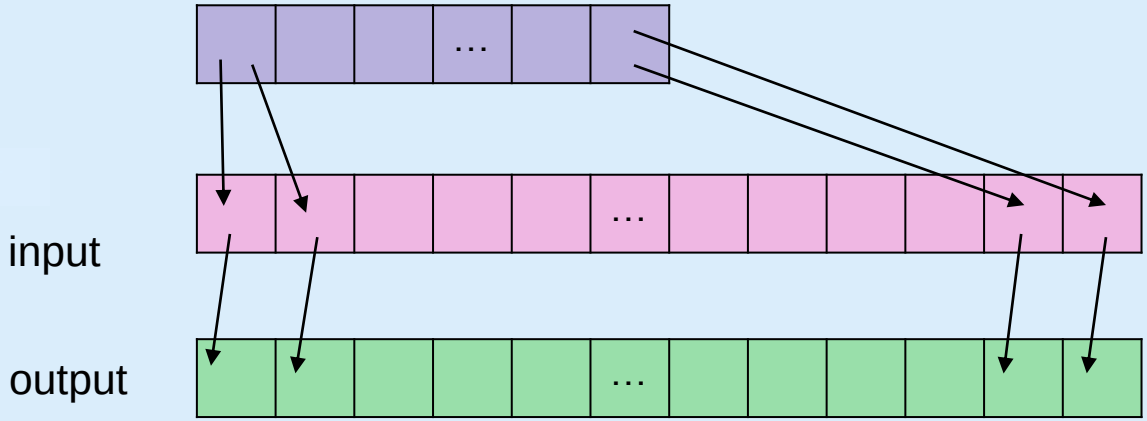
1 thread xử lý 2 phần tử



Ý tưởng cải tiến hàm residual()



Song song hóa hàm compute_logodds() và compute_prob()



Chỉ copy phần dữ liệu cần thiết

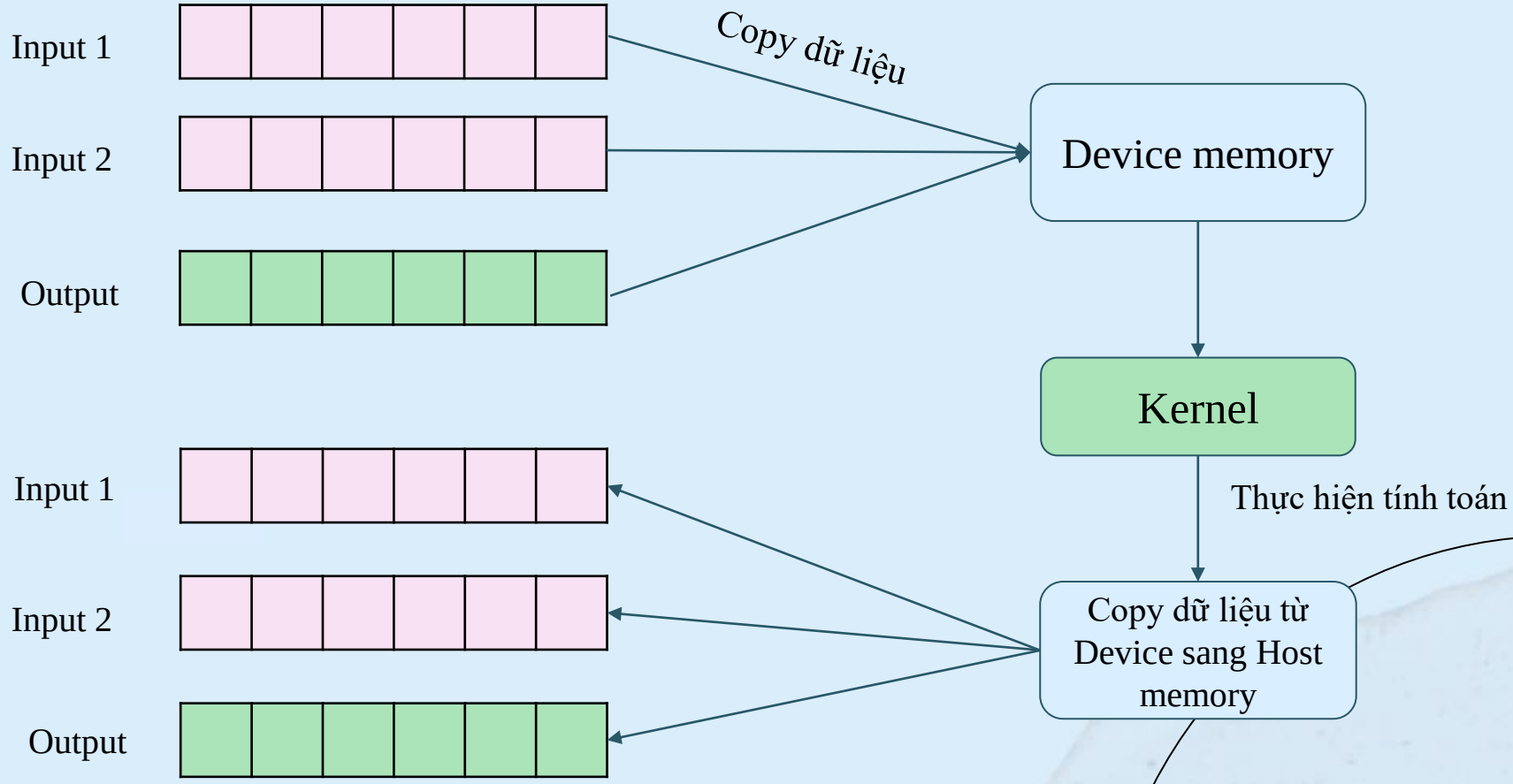
compute_split_value()

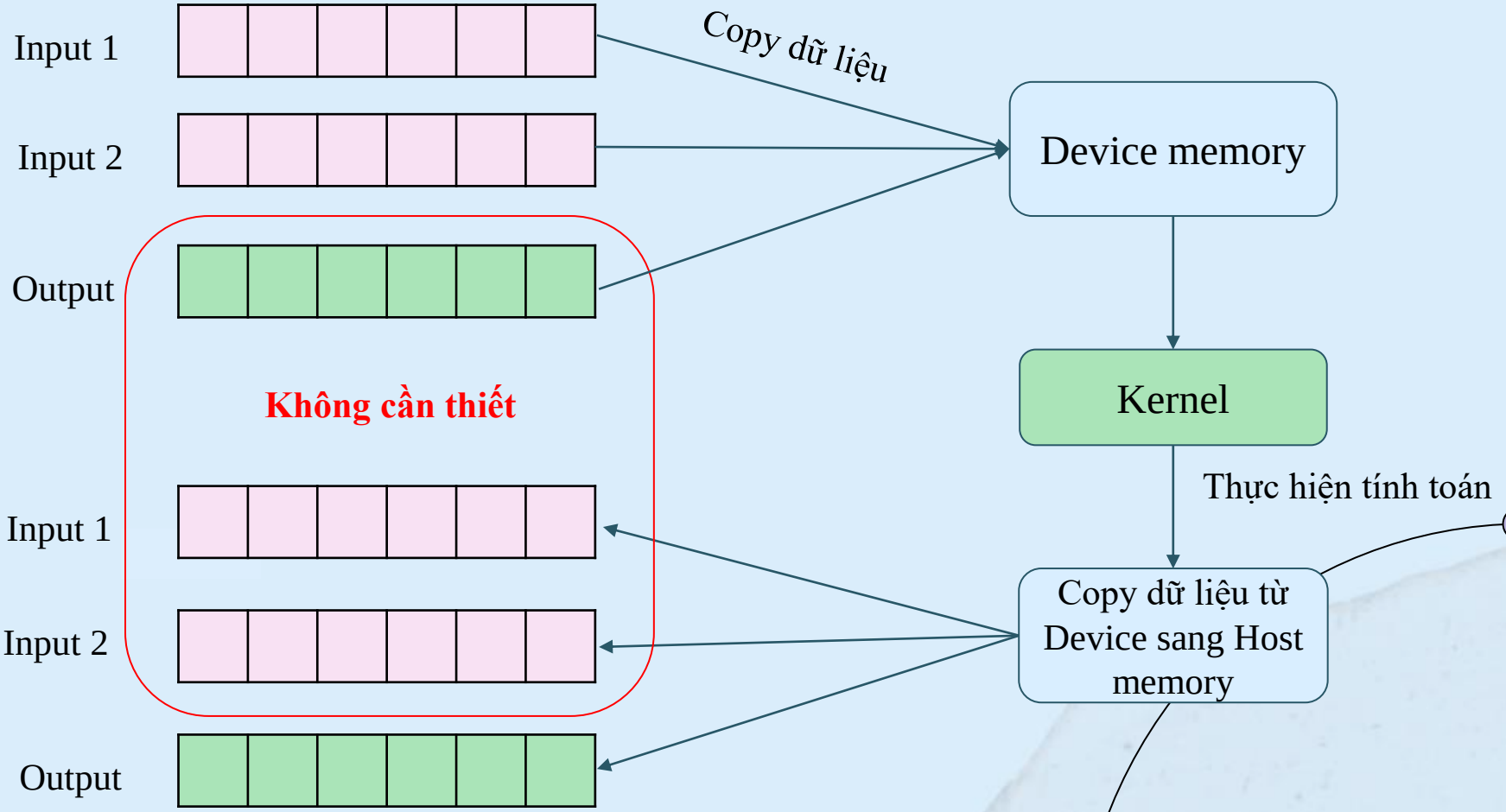
compute_gain()

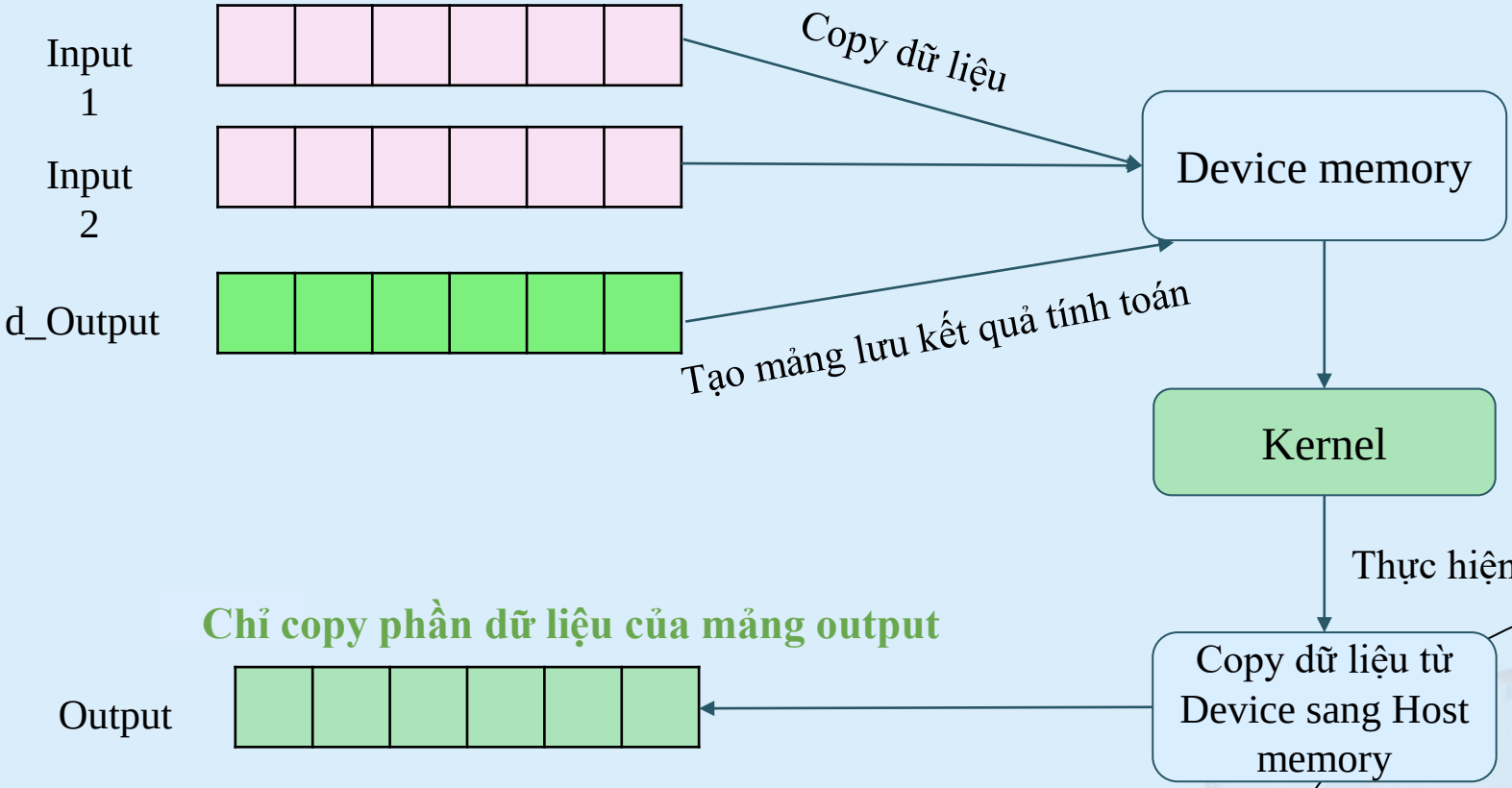
residual()

compute_logodds()

compute_prob()


Input 1
Input 2
Output
Copy dữ liệu
Device memory
Kernel
Thực hiện tính toán
Input 1
Input 2
Output
Copy dữ liệu từ Device sang Host memory


Input 1
Input 2
Output
Copy dữ liệu
Device memory
Kernel
Thực hiện tính toán
Không cần thiết
Input 1
Input 2
Copy dữ liệu từ Device sang Host memory
Output


Input 1
Input 2
d_Output
Copy dữ liệu
Device memory
Tạo mảng lưu kết quả tính toán
Kernel
Thực hiện tính toán
Chỉ copy phần dữ liệu của mảng output
Output
Copy dữ liệu từ Device sang Host memory

03

Huấn luyện và đánh giá mô hình

Bộ dữ liệu huấn luyện giống với phiên bản tuần tự
Chiến lược: **OvR** giống với phiên bản tuần tự

Số class	Kích thước	Accuracy tuần tự	Accuracy song song	Accuracy optimize	Thời gian chạy tuần tự	Thời gian chạy song song	Thời gian chạy optimize
2	3000	0.95	0.95	0.95	350 – 370s	4 – 6s	2 – 3s
3	3000	0.946	0.946	0.946	1060 – 1150s	9 – 11s	6 – 8s

Bộ dữ liệu huấn luyện **10 lớp đầy đủ**
Chiến lược: OvR giống với phiên bản tuần tự

Số class	Kích thước	Accuracy song song	Accuracy optimize	Thời gian chạy song song	Thời gian chạy optimize
2	3000	0.95	0.95	4 – 6s	2 – 3s
3	3000	0.946	0.946	9 – 11s	6 – 8s
10	10.000	0.891	0.891	2700 – 2800s	2500 – 2650s

Bộ dữ liệu huấn luyện **10 lớp đầy đủ**
Chiến lược: **OvR** và **under sampling** (40 : 60)

Số class	Kích thước	Accuracy song song	Accuracy optimize	Thời gian chạy song song	Thời gian chạy optimize	Note
2	3000	0.95	0.95	4 – 6s	2 – 3s	
3	3000	0.946	0.946	9 – 11s	6 – 8s	
10	10.000	0.891	0.891	2700 – 2800s	2500 – 2650s	
10	10.000	~ 0.91	~ 0.91	700 – 800s	500 – 650s	***under sampling***

04

Tổng kết

- ✓ Xây dựng thành công mô hình XGBoost tuần tự.
- ✓ Xây dựng thành công mô hình XGBoost song song mà vẫn đảm bảo độ chính xác và thời gian thực thi được cải thiện.
- ✓ Áp dụng các kĩ thuật cải tiến lên mô hình XGBoost mà vẫn đảm bảo độ chính xác và thời gian thực thi được cải thiện.
- ✓ Sử dụng chiến lược OvR và under sampling nhằm phân loại đa lớp mà vẫn đảm bảo thời gian thực thi.

- Áp dụng nhiều kĩ thuật cải tiến hơn
- Thử xây dựng mô hình phân loại đa lớp trực tiếp mà không thông qua OvR
- Thử áp dụng mô hình lên bộ dữ liệu lớn và phức tạp hơn

THE END