ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

AI chơi cờ vây

Hồ Quang Chung
Bùi Đức Hiếu
Nguyễn Khánh Toàn

Mã học phần: MAT3508
Học kỳ 1, Năm học 2025-2026

Thông tin Dự án

[Thông tin này cũng cần được ghi trong README.md của kho GitHub.]

Học phần: MAT3508 – Nhập môn Trí tuệ Nhân tạo
Học kỳ: Học kỳ I, Năm học 2025-2026
Trường: VNU-HUS (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)
Tên dự án: AI chơi cờ vây
Ngày nộp: 30/11/2025
Slide thuyết trình: Slide link
Kho GitHub: `https://github.com/Nekover2/Go-game`

Thành viên nhóm

Họ tên	Mã sinh viên	Tên GitHub	Đóng góp
Hồ Quang Chung	22001549	Nekover2	Backend + Hỗ trợ training
Bùi Đức Hiếu	21002144	BuiDucHieuK66	Frontend
Nguyễn Khánh Toàn	23001944	nguyentoan-git	Traing model

Danh sách hình vẽ

Danh sách bảng

Mục lục

Chương 1

Giới thiệu

https://colab.research.google.com/drive/16VBnsSqPDZvAZ72fnaEzvLXcj6nIsxuu Dự án này nhằm mục đích thiết kế, triển khai và đánh giá một hệ thống Trí tuệ Nhân tạo (AI) có khả năng chơi cờ vây (Go/Weiqi/Baduk) trên bàn cờ tiêu chuẩn 19×19 với trình độ tương đương nghiệp dư cao đẳng (high-dan amateur). Hệ thống được xây dựng dựa trên sự tổng hợp của hai mô hình tính toán tiên tiến nhất hiện nay: **Mạng nơ-ron tích chập** (Convolutional Neural Networks - CNN) để mô phỏng trực giác và nhận diện mẫu hình, kết hợp với **Tìm kiếm cây Monte Carlo** (Monte Carlo Tree Search - MCTS) để lập kế hoạch chiến thuật và tính toán nước đi.

Điểm đặc biệt của dự án nằm ở việc lựa chọn ngăn xếp công nghệ (tech stack) hiện đại và tối ưu hóa hiệu năng cao:

- **Backend:** Sử dụng .NET 10 (dự kiến phát hành Long Term Support vào tháng 11/2025) để tận dụng các cải tiến vượt bậc về JIT Compiler và quản lý bộ nhớ.
- **Frontend:** Sử dụng ReactJS để tạo giao diện tương tác thời gian thực.
- **Dữ liệu:** Khai thác từ kho dữ liệu khổng lồ Fox Go Server thông qua repository `featurecat/go-dataset`.[1]

1.1 Đặt ra bài toán và Thách thức tính toán

Cờ vây từ lâu đã được coi là "Chén Thánh"của trí tuệ nhân tạo, một thách thức lớn hơn nhiều so với cờ vua. Sự phức tạp của cờ vây không chỉ nằm ở luật chơi trừu tượng mà còn ở không gian trạng thái khổng lồ của nó.

1.1.1 Sự bùng nổ tổ hợp và Không gian trạng thái

Trên bàn cờ 19×19, có 361 giao điểm. Số lượng trạng thái bàn cờ hợp lệ ước tính vào khoảng 2.1×10^{170}, một con số vượt xa tổng số nguyên tử trong vũ trụ quan sát được (khoảng 10^{80}).[2] Để so sánh:

- **Cờ vua:** Không gian trạng thái khoảng 10^{47} và độ phức tạp cây trò chơi khoảng 10^{123}.
- **Cờ vây:** Độ phức tạp cây trò chơi lên tới 10^{360}.[2]

Sự khác biệt này làm cho các phương pháp tìm kiếm cổ điển như Minimax với cắt tỉa Alpha-Beta trở nên vô hiệu. Trong cờ vua, hệ số phân nhánh (branching factor) trung bình là khoảng 35, cho phép máy tính tìm kiếm sâu hàng chục nước đi. Trong cờ vây, hệ số phân nhánh khởi đầu là 361 và trung bình duy trì ở mức khoảng 250 trong suốt ván đấu.[5] Việc tìm kiếm sâu (deep search) trở nên bất khả thi về mặt tính toán nếu không có một cơ chế "trực giác"để loại bỏ phần lớn các nhánh không tiềm năng ngay từ đầu.

1.1.2 Vấn đề định giá trạng thái (State Evaluation)

Một thách thức khác là hàm đánh giá (evaluation function). Trong cờ vua, việc gán giá trị vật chất cho quân cờ (Hậu = 9, Tốt = 1) cung cấp một heuristic tĩnh rất mạnh. Trong cờ vây, giá trị của một quân cờ phụ thuộc hoàn toàn vào bối cảnh: nó đang đóng góp vào vùng đất (territory), tạo thế (influence), hay thuộc về một nhóm quân đang sống hay chết.[6] Các heuristic thủ công (hand-crafted heuristics) trước kỷ nguyên Deep Learning thường thất bại trong việc nắm bắt các khái niệm trừu tượng như "Aji"(dư vị) hay "Thickness"(độ dày).

1.2 Luật chơi cờ vây (theo luật của Trung Quốc)

Vì tập dữ liệu được dùng để huấn luyện là các ván cờ được lưu trên trang foxwq, nên luật được dùng ở đây hiển nhiên là luật của Trung Quốc.

1.2.1 Tổng quan và Triết lý

Khác với luật Nhật Bản (coi trọng lãnh thổ trống), luật Trung Quốc quan niệm rằng "Đất"bao gồm cả **lãnh thổ kiểm soát** và **sự hiện diện vật lý** của quân cờ trên bàn.

- **Mục tiêu:** Chiếm được nhiều giao điểm trên bàn cờ nhất (bao gồm cả điểm đặt quân và điểm vây được).
- **Dụng cụ:** Bàn 19×19 (361 giao điểm).
- **Quân cờ:** Số lượng quân không giới hạn (thực tế mỗi bên có khoảng 180 quân).

1.2.2 Quy tắc Cơ bản (Mechanics)

Luật đi quân

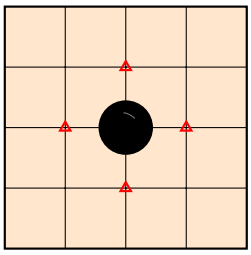
1. **Lượt đi:** Quân Đen đi trước.
2. **Vị trí:** Đặt quân vào giao điểm (intersections).
3. **Bất di bất dịch:** Quân đã đặt xuống không được di chuyển (trừ khi bị bắt).

Khí và Luật ăn quân

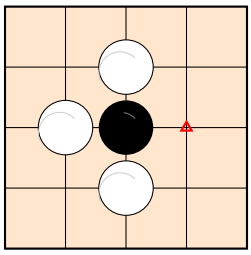
Tương tự như luật quốc tế chung:

- Một quân hoặc nhóm quân bị đối phương chặn hết tất cả các giao điểm khí (liberties) xung quanh sẽ bị loại khỏi bàn cờ.
- **Khác biệt quan trọng về Tù binh:** Trong luật Trung Quốc, quân tù binh bị nhấc ra khỏi bàn **không được tính điểm** trực tiếp. Việc ăn quân chỉ có ý nghĩa là loại bỏ sự hiện diện của đối phương để chiếm đất.

Khái niệm Khí (Liberties) Khí là sự sống của quân cờ. Một quân cờ đặt xuống bàn cần có ít nhất một "đường thở"(giao điểm trống sát cạnh).



Hình 1: Quân ở giữa có 4 Khí (Tam giác)



Hình 2: Đang bị vây (Atari)

Giải thích:

- **Hình 1:** Quân Đen có 4 hướng để "thở"(các hình tam giác đỏ).
- **Hình 2:** Quân Trắng đã chặn 3 hướng. Nếu Trắng đánh nốt vào vị trí tam giác đỏ, quân Đen sẽ chết và bị nhấc ra ngoài.

Luật cấm

- **Cấm Tự tử (Suicide):** Không được đặt quân vào nơi không còn khí, trừ khi nước đi đó ăn quân đối phương ngay lập tức.
- **Cấm Lặp lại (Ko & Superko):**
  - *Ko cơ bản:* Không được ăn lại ngay lập tức một quân vừa ăn mình (để tránh lặp lại cục diện vừa xảy ra).
  - *Superko (Nâng cao):* Luật Trung Quốc nghiêm ngặt cấm **bất kỳ** nước đi nào tái lập lại một trạng thái bàn cờ đã từng xuất hiện trước đó trong ván cờ (bất kể cách đó bao nhiêu nước).

1.2.3 Quy tắc Tính điểm (Nâng cao)

Đây là phần quan trọng nhất tạo nên sự khác biệt của luật Fox/Trung Quốc.

Nguyên tắc Đếm đám (Area Scoring)

Điểm số của một người chơi được tính bằng công thức:

$$\text{Điểm} = (\text{Số giao điểm trống vây được}) + (\text{Số quân sống đang nằm trên bàn}) \tag{1.1}$$

Điểm trung lập (Dame)

Trong luật Nhật, các điểm trung lập (nằm giữa ranh giới hai bên, không thuộc về ai) là vô giá trị. Tuy nhiên, trong luật Trung Quốc:

- **Dame có giá trị điểm:** Vì mỗi quân cờ đặt xuống bàn được tính là 1 điểm, người chơi **bắt buộc phải đánh** vào các điểm Dame ở cuối ván.
- Nếu bỏ qua Dame, đối thủ đánh vào đó sẽ tăng thêm điểm quân số của họ.

Komi (Điểm cộng)

Để cân bằng lợi thế đi trước của quân Đen:

- **Komi:** 7.5 điểm (Luật Nhật thường chỉ là 6.5).
- **Quy đổi:** Trong cách tính truyền thống gọi là "3.75 tử"(3.75 stones).

1.2.4 Cách xác định Thắng/Thua trên Fox

Hệ thống máy tính trên Foxwq sẽ tự động đếm theo quy trình sau:

Ngưỡng chiến thắng

Tổng số giao điểm trên bàn cờ là 361.

- Một nửa bàn cờ là $361/2 = 180.5$.
- Vì Trắng được cộng 7.5 điểm (tức là Đen phải chấp lại số điểm này), ngưỡng chiến thắng của Đen được tính như sau:

$$\text{Ngưỡng thắng của Đen} = 180.5 + 3.75 = 184.25 \tag{1.2}$$

Quy tắc đếm thực tế

Người ta thường chỉ đếm quân và đất của **một bên** (thường là bên Đen) để xác định kết quả:

- Nếu Tổng điểm của Đen (Đất + Quân) > 184.25 (thực tế là ≥ 185 trên bàn cờ): **Đen Thắng**.
- Nếu Tổng điểm của Đen ≤ 184.25: **Trắng Thắng**.

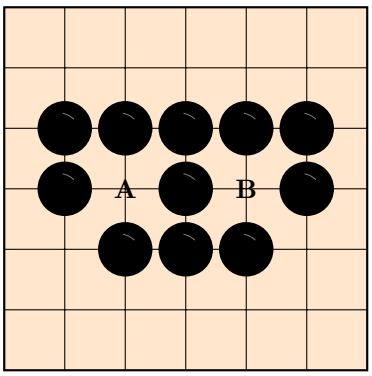
1.2.5 Xử lý các tình huống đặc biệt

Sống chết trong luật Trung Quốc

Nếu có tranh chấp về việc một nhóm quân là sống hay chết ở cuối ván:

- Người chơi không cần phải "giả định"như luật Nhật.
- Hai bên cứ tiếp tục chơi thực tế (thực chiến) để chứng minh.
- Việc bỏ quân vào đất của mình để phòng thủ (cần thiết khi giải quyết tranh chấp) **không làm mất điểm** (vì mất 1 điểm đất nhưng được cộng 1 điểm quân $\rightarrow$ hòa vốn).

Nguyên tắc quan trọng nhất: **"Hai Mắt (Two Eyes) là Bất Tử"**.



Hình 3: Nhóm quân SỐNG (Hai Mắt thật A và B)

Tại sao hình trên sống?

Để ăn nhóm quân Đen này, Trắng phải lấp kín khí. Nhưng Đen có 2 "lỗ hổng"độc lập (A và B). Trắng không thể đánh vào A (tự sát), cũng không thể đánh vào B (tự sát). Trắng không thể đánh cùng lúc vào A và B. Do đó, Đen bất tử.

Trường hợp "Sống chung"(Seki)

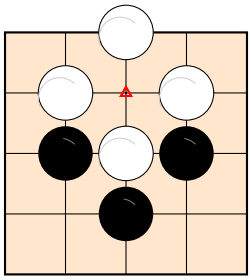
Các giao điểm mắt chung trong thế cờ Seki (Sống chung, không ai ăn được ai) được tính điểm chia đều hoặc không tính tùy biến thể, nhưng trên Fox, các điểm này thường không được tính là đất của ai, nhưng **quân cờ** tạo nên thế Seki vẫn được tính điểm.

Tóm tắt sự khác biệt: Fox (TQ) vs Quốc tế (Nhật)

Tiêu chí	Luật Fox / Trung Quốc	Luật Nhật Bản
Đơn vị đếm	Đất trống + Quân cờ	Chỉ đếm Đất trống
Tù binh	Không tính điểm (chỉ bỏ đi)	Giữ lại trừ điểm đối thủ
Điểm Dame (Trung lập)	**Có tính điểm (Phải đánh)**	Không tính điểm
Komi	7.5 điểm	6.5 điểm
Tự sửa cờ (Reinforce)	Không mất điểm	Bị trừ 1 điểm (mất đất)

1.2.6 Luật Cướp (Ko Rule)

Luật ngăn chặn vòng lặp vô tận.

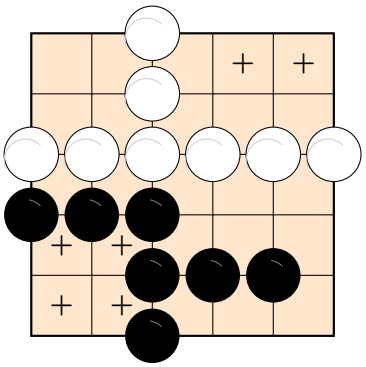


Hình 4: Thế cờ "Cướp"

Diễn biến: 1. Nếu Đen đánh vào vị trí **Tam giác đỏ**, Đen sẽ ăn quân Trắng ở giữa. 2. Bàn cờ quay lại hình dạng cũ. Nếu Trắng ngay lập tức đánh lại vào vị trí quân Trắng vừa bị ăn → Vòng lặp vô tận. 3. **Luật:** Trắng phải đánh chỗ khác trước (Ko threat).

1.2.7 Ví dụ Tính điểm (Luật Trung Quốc)

Giả sử ván cờ kết thúc trên bàn nhỏ 5x5.



Cách tính điểm Fox (Giả sử Đen đếm):

- **Đất trống của Đen:** 4 điểm (các dấu +).
- **Quân Đen trên bàn:** 7 quân (5 quân vây + 2 quân lấp Dame).
- **Tổng điểm Đen:** $4 + 7 = 11$ điểm.

**Lưu ý: Trên bàn 19x19, máy sẽ so sánh tổng điểm này với 184.25 để quyết định thắng thua.*

1.3 Mục tiêu Dự án

Báo cáo này sẽ trình bày chi tiết quá trình phát triển hệ thống AI giải quyết các vấn đề trên thông qua các trụ cột sau:

1. **Mô hình học sâu (Deep Learning Model):** Huấn luyện mạng Residual CNN (ResNet) hai đầu ra (Dual-head: Policy & Value) từ dữ liệu ván đấu của con người để dự đoán nước đi.
2. **Thuật toán tìm kiếm (Search Algorithm):** Tích hợp mạng nơ-ron vào MCTS sử dụng biến thể PUCT (Predictor + Upper Confidence Bound applied to Trees) để cân bằng giữa khai thác (exploitation) và khám phá (exploration).
3. **Cơ sở hạ tầng hiệu năng cao:** Xây dựng Backend trên nền tảng .NET 10, sử dụng các tính năng mới như AVX10.2 và `Microsoft.Extensions.AI` để tối ưu hóa tốc độ suy luận (inference) và xử lý logic trò chơi.[7]
4. **Giao diện người dùng:** Phát triển ứng dụng web ReactJS hiển thị bàn cờ

Chương 2

Phương pháp & Triển khai

2.1 Cơ sở lý thuyết và Giải thích Kiến trúc

Việc lựa chọn kiến trúc **ResNet Dual-head** kết hợp với thuật toán **MCTS (PUCT)** là kết quả của sự tối ưu hóa dựa trên đặc thù toán học của trò chơi Cờ vây và các nghiên cứu tiên tiến (State-of-the-art) từ AlphaGo Zero. Dưới đây là phân tích chi tiết lý do cho từng lựa chọn.

2.1.1 Tại sao sử dụng Mạng nơ-ron Tích chập (CNN) và ResNet?

Bàn cờ là một "hình ảnh"đặc biệt

Trong lĩnh vực thị giác máy tính, mạng CNN (Convolutional Neural Network) có khả năng vượt trội trong việc nhận diện các mẫu hình cục bộ (cạnh, góc, kết cấu).

- Bàn cờ 19×19 có tính chất không gian tương tự một bức ảnh grayscale.
- Các trạng thái như "mắt cờ", "sống/chết", hay hình dạng đám quân (shape) là các đặc trưng cục bộ mà CNN có thể trích xuất hiệu quả hơn nhiều so với các mạng kết nối đầy đủ (MLP).

Giải quyết vấn đề chiều sâu với ResNet

Cờ vây đòi hỏi tính chiến lược toàn cục. Một quân cờ ở góc này có thể ảnh hưởng đến góc kia (ví dụ: Chinh quân - Ladder). Để nắm bắt được mối quan hệ xa như vậy, mạng nơ-ron cần phải rất sâu. Tuy nhiên, mạng sâu thường gặp vấn đề *biến mất đạo hàm* (vanishing gradient).

Giải pháp: Kiến trúc Residual Network (ResNet) sử dụng các kết nối tắt (skip connections):

$$y = F(x) + x \tag{2.1}$$

Điều này cho phép tín hiệu truyền xuyên suốt qua hàng chục lớp mà không bị suy giảm, giúp AI vừa tính toán tốt chiến thuật cục bộ, vừa có tầm nhìn chiến lược toàn cục.

2.1.2 Tại sao sử dụng mô hình Dual-head (Hai đầu ra)?

Thay vì sử dụng hai mạng riêng biệt, hệ thống sử dụng một thân mạng chung (Backbone) và tách ra hai nhánh ở cuối:

1. **Hiệu năng tính toán (Efficiency):** Các lớp tích chập (Convolution) tiêu tốn nhiều tài nguyên nhất. Việc dùng chung backbone giúp giảm một nửa khối lượng tính toán khi suy luận.
2. **Học đa nhiệm (Multi-task Learning):** Việc ép mạng học cùng lúc hai nhiệm vụ (dự đoán nước đi và dự đoán thắng thua) giúp các trọng số ở các lớp chung trở nên tổng quát hơn, giảm thiểu hiện tượng Overfitting.

2.1.3 Tại sao chọn MCTS thay vì Minimax/Alpha-Beta?

Các thuật toán duyệt cây cổ điển như Minimax hoạt động tốt với Cờ vua nhưng thất bại với Cờ vây do sự **bùng nổ tổ hợp** (Combinatorial Explosion):

- Cờ vua: Trung bình 35 nước đi/lượt.
- Cờ vây: Trung bình 250 nước đi/lượt.

- Không gian trạng thái: xấp xỉ 10^{170} (lớn hơn số nguyên tử trong vũ trụ).

MCTS (Monte Carlo Tree Search) giải quyết vấn đề này bằng cách duyệt cây theo xác suất (ngẫu nhiên có định hướng), tập trung tài nguyên vào các nhánh triển vọng thay vì cố gắng duyệt tất cả.

2.1.4 Sự kết hợp: Mạng nơ-ron + MCTS (Biến thể PUCT)

Đây là điểm mấu chốt của hệ thống. Mạng nơ-ron đóng vai trò "trực giác", còn MCTS đóng vai trò "suy luận logic".

Công thức lựa chọn nút trong MCTS (PUCT):

$$U(s,a) = C_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{2.2}$$

Vai trò của từng thành phần:

- **Policy Network $P(s,a)$ - Giảm chiều rộng:** Giúp MCTS loại bỏ ngay lập tức hàng trăm nước đi vô nghĩa, chỉ tập trung khảo sát các nước đi có xác suất cao do mạng gợi ý.
- **Value Network $V(s)$ - Giảm chiều sâu:** Thay vì phải mô phỏng đến hết ván cờ (Rollout) rất tốn thời gian, MCTS có thể dừng lại ở một nút lá và hỏi Value Network xem thế cờ đó ai đang ưu thế.

2.1.5 Tổng kết so sánh

Thành phần	Tương đồng với con người	Vấn đề giải quyết
ResNet	Vỏ não thị giác (Nhìn hình cờ)	Trích xuất đặc trưng không gian và xử lý gradient trong mạng sâu.
Dual-head	Tư duy đa nhiệm	Tối ưu hóa tài nguyên và tổng quát hóa dữ liệu.
MCTS + PUCT	Thùy trán (Tính toán logic)	Giải quyết không gian tìm kiếm khổng lồ và sửa lỗi "ảo giác"của trực giác.

Bảng 2.1: Tổng hợp các quyết định kiến trúc

2.2 Phương pháp

Chương này mô tả kiến trúc kỹ thuật, cơ sở lý thuyết và thuật toán cho dự án xây dựng hệ thống chơi Cờ vây sử dụng Trí tuệ nhân tạo. Hệ thống được xây dựng trên nền tảng Web với Frontend ReactJS, Backend .NET 10 theo kiến trúc DDD, và lõi AI sử dụng mô hình Deep Residual Network kết hợp thuật toán tìm kiếm Monte Carlo Tree Search (MCTS).

2.2.1 Tổng quan Kiến trúc Hệ thống

Hệ thống hoạt động theo mô hình Client-Server phân tán, tách biệt giữa giao diện người dùng, logic trò chơi và tính toán AI.

- **Frontend (ReactJS):** Đóng vai trò lớp Presentation. Xử lý tương tác người dùng, hiển thị bàn cờ, gửi dữ liệu nước đi và nhận kết quả từ Server.
- **Backend (.NET 10):** Được thiết kế theo mô hình Domain-Driven Design (DDD). Chịu trách nhiệm quản lý trạng thái ván đấu (State Management), thực thi luật cờ vây, và cung cấp API RESTful/WebSocket.
- **AI Core:** Module tích hợp trong Backend, sử dụng `ONNX Runtime` để chạy mô hình Deep Learning và thuật toán MCTS để đưa ra quyết định nước đi.

2.2.2 Dữ liệu và Tiền xử lý (Data Engineering)

Chất lượng của mô hình AI phụ thuộc lớn vào dữ liệu huấn luyện đầu vào.

Nguồn dữ liệu

Sử dụng dataset từ `featurecat/go-dataset`, tập trung lọc các ván đấu của kỳ thủ chuyên nghiệp (Professional Dan rank: 1p - 9p). Việc sử dụng dữ liệu chuyên nghiệp giúp mô hình học giám sát (Supervised Learning) tránh học các lỗi sai từ người chơi nghiệp dư.

Biểu diễn dữ liệu (Input Features)

Bàn cờ 19×19 được biểu diễn dưới dạng Tensor đa kênh (Multi-channel Tensor) với kích thước $[Batch\_Size, 19, 19, C]$. Các kênh thông tin bao gồm:

- **Quân ta (Player Stones):** 1 nếu có quân, 0 nếu không.
- **Quân đối thủ (Opponent Stones):** 1 nếu có quân, 0 nếu không.
- **Vị trí trống (Empty):** Đánh dấu các điểm chưa có quân.
- **Lịch sử (History):** Trạng thái bàn cờ tại các thời điểm $t-1, t-2, \ldots$ để giúp mạng nhận diện luật Ko (cướp lại).
- **Lượt đi (Color):** Plane chứa toàn số 1 (nếu Đen đi) hoặc 0 (nếu Trắng đi).

2.2.3 Kiến trúc Mạng Nơ-ron (Model Architecture)

Mô hình cốt lõi là mạng tích chập phần dư (Residual CNN - ResNet) với cấu trúc hai đầu ra (Dual-head).

Khối dư (Residual Block)

Để giải quyết vấn đề thoái hóa gradient trong mạng sâu, kiến trúc sử dụng các skip connection:

$$y = \sigma(F(x, \{W_i\}) + x) \tag{2.3}$$

Trong đó σ là hàm kích hoạt ReLU.

Cấu trúc Dual-head

Mạng chia sẻ các lớp Convolutional ban đầu và tách thành hai nhánh ở cuối:

1. **Policy Head (Chiến thuật):**
   - Output: Vector xác suất p kích thước $19 \times 19 + 1$ (bao gồm nước Pass).
   - Hàm kích hoạt: Softmax.
   - Ý nghĩa: $p_a = P(a|s)$ là xác suất chọn nước đi a tại trạng thái s.
2. **Value Head (Giá trị):**
   - Output: Giá trị thực $v \in [-1, 1]$.
   - Hàm kích hoạt: Tanh.
   - Ý nghĩa: Dự đoán khả năng thắng (1) hoặc thua (-1) từ trạng thái hiện tại.

Hàm mất mát (Loss Function)

Hàm mục tiêu được tối ưu hóa trong quá trình huấn luyện:

$$L = (z - v)^2 - \pi^T \log(p) + c||\theta||^2 \tag{2.4}$$

Trong đó:

- $(z - v)^2$: Mean Squared Error cho nhánh Value (z là kết quả thực tế).
- $-\pi^T \log(p)$: Cross-Entropy Loss cho nhánh Policy (π là nước đi của chuyên gia).
- $c||\theta||^2$: L2 Regularization.

2.2.4 Thuật toán Tìm kiếm: MCTS + PUCT

AI sử dụng Monte Carlo Tree Search (MCTS) để lập kế hoạch, được hướng dẫn bởi mạng nơ-ron thông qua biến thể PUCT.

Quy trình MCTS

Tại mỗi lượt đi, AI thực hiện hàng nghìn lần mô phỏng (simulation) theo 4 bước:

1. **Selection:** Đi từ nút gốc xuống lá dựa trên chỉ số $Q + U$.
2. **Expansion:** Mở rộng nút lá và sử dụng Neural Network để đánh giá (P, V).
3. **Evaluation:** Gán giá trị V và xác suất P từ mạng nơ-ron cho nút mới.
4. **Backup:** Cập nhật lại giá trị Q (trung bình) và N (số lần thăm) cho các nút cha.

Công thức PUCT

Nước đi được chọn trong giai đoạn Selection dựa trên công thức tối đa hóa:

$$a_t = \text{argmax}_a(Q(s,a) + U(s,a)) \tag{2.5}$$

Với thành phần Upper Confidence Bound (U) khuyến khích khám phá:

$$U(s,a) = C_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{2.6}$$

Trong đó:

- $Q(s,a)$: Giá trị trung bình (Mean Action Value).
- $P(s,a)$: Xác suất tiên nghiệm từ Policy Head.
- $N(s,a)$: Số lần nước đi a đã được thăm.

2.2.5 Hiện thực hóa Backend (.NET 10)

Domain-Driven Design (DDD)

- **Domain Layer:** Chứa các Entities như `Board`, `Stone`, `Group`. Xử lý logic nghiệp vụ cốt lõi như bắt quân, tính khí, luật Ko. Độc lập hoàn toàn với Framework.
- **Application Layer:** Điều phối luồng game, nhận request từ API và gọi Domain Service.
- **Infrastructure Layer:** Implement các interface kỹ thuật, bao gồm việc tích hợp AI.

Tích hợp ONNX Runtime

Mô hình sau khi huấn luyện (bằng PyTorch/TensorFlow) được export ra định dạng `.onnx`. Backend sử dụng thư viện `Microsoft.ML.OnnxRuntime` để load model:

```
public class GoAiService
{
    private readonly InferenceSession _session;

    public GoAiService(string modelPath) {
        _session = new InferenceSession(modelPath);
    }

    public (float[] Policy, float Value) Evaluate(Tensor<float> input) {
        var inputs = new List<NamedOnnxValue>
        {
            NamedOnnxValue.CreateFromTensor("input", input)
        };

        using var results = _session.Run(inputs);

        //...
        return (policyArray, valueScalar);
    }
}
```

Listing 2.1: Giả mã tích hợp ONNX trong C#

2.3 Kết luận

Dự án kết hợp sức mạnh tính toán của Deep Learning (ResNet) và khả năng suy luận logic của MCTS trên nền tảng công nghệ hiện đại (.NET 10, ReactJS). Cách tiếp cận này đảm bảo AI có khả năng chơi ở trình độ cao đồng thời hệ thống vẫn đảm bảo tính mở rộng và bảo trì tốt nhờ kiến trúc DDD.

2.4 Môi trường và Công cụ Phát triển

Để đảm bảo tính nhất quán và hiệu năng cao cho hệ thống, dự án sử dụng tập hợp các công cụ và thư viện sau:

2.4.1 Danh sách Công nghệ (Tech Stack)

Thành phần	Công nghệ / Thư viện	Phiên bản (Dự kiến)
Frontend	ReactJS, TypeScript, Vite	React 18+, Vite 5.x
UI/UX	TailwindCSS, HTML5 Canvas (Render bàn cờ)	v3.4+
Backend Runtime	.NET 10 (C#)	Preview/Stable
Kiến trúc Backend	Domain-Driven Design (DDD), Clean Architecture	-
AI Inference	ONNX Runtime (CPU/GPU)	Microsoft.ML.OnnxRuntime
AI Training	Python, PyTorch, NumPy	PyTorch 2.x
Database	PostgreSQL hoặc SQLite (Dev)	Latest
Containerization	Docker, Docker Compose	-

Bảng 2.2: Các framework sử dụng

2.4.2 Chi tiết các Thư viện và Framework

Frontend (Client-side)

Giao diện người dùng được xây dựng theo mô hình Single Page Application (SPA):

- **ReactJS:** Thư viện cốt lõi để xây dựng UI dựa trên component.
- **Axios:** Client HTTP để giao tiếp với Backend API.
- **Zustand/Redux Toolkit:** Quản lý trạng thái toàn cục (Global State) của ứng dụng (ví dụ: trạng thái bàn cờ, lượt đi, điểm số).
- **KonvaJS / HTML5 Canvas:** Dùng để vẽ bàn cờ vây và xử lý sự kiện click đặt quân với hiệu năng cao (thay vì dùng DOM elements thông thường).

Backend (Server-side)

Server đóng vai trò trung tâm xử lý logic và tích hợp AI, được chia thành các lớp thư viện NuGet chính:

- **Microsoft.ML.OnnxRuntime.Gpu:** Thư viện quan trọng nhất để load file model `.onnx` và thực hiện suy luận (inference) trên Server.
- **MediatR:** Hỗ trợ triển khai mẫu CQRS (Command Query Responsibility Segregation) trong kiến trúc DDD.
- **Entity Framework Core:** ORM để truy xuất dữ liệu lịch sử ván đấu.
- **Swashbuckle (Swagger):** Tự động sinh tài liệu API.
- **Serilog:** Ghi log hệ thống và log các nước đi để debug.

2.4.3 Cấu trúc Mã nguồn (Source Code Structure)

Cấu trúc dự án tuân thủ nghiêm ngặt nguyên lý *Separation of Concerns* (Phân tách mối quan tâm).

Cấu trúc Solution Backend (.NET)

Dự án Backend được tổ chức theo kiến trúc **Onion Architecture / DDD**:

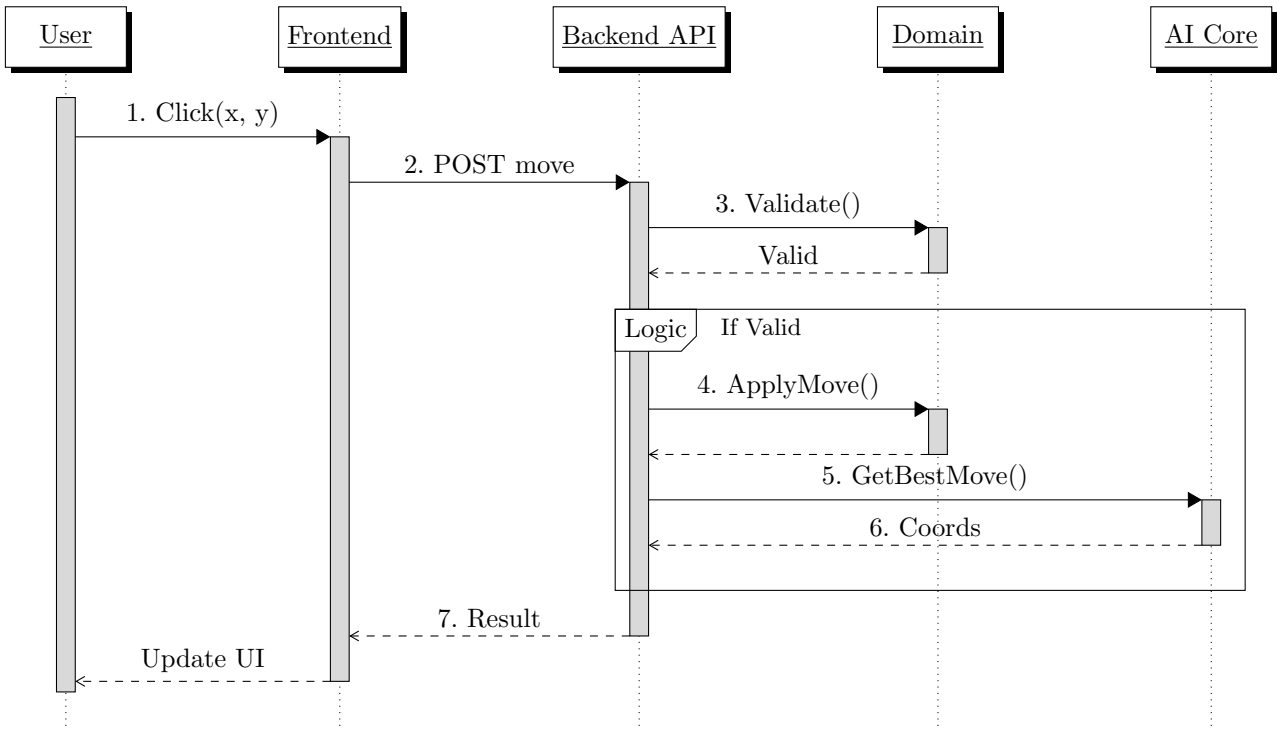
```
GoAI_Solution.sln
|
+--- src
|    |
|    +--- 1.Domain (Class Library)
|    |    |--- Entities/          # Board, Stone, Player, GameMatch
|    |    |--- ValueObjects/      # Coordinate (x, y), StoneColor
|    |    |--- Rules/             # GoRuleEngine (Ko, Territory counting)
|    |    |--- Exceptions/        # DomainExceptions (InvalidMoveException)
|    |
|    +--- 2.Application (Class Library)
|    |    |--- Interfaces/        # IGoAiService, IGameRepository
|    |    |--- UseCases/          # MakeMoveCommandHandler, GetBestMoveQuery
|    |    |--- DTOs/              # MoveInputDto, GameStateDto
|    |
|    +--- 3.Infrastructure (Class Library)
|    |    |--- Persistence/       # DbContext, Migrations
|    |    |--- AI/                # OnnxGoAiService (Implement IGoAiService)
|    |    |    |--- Models/       # Model wrapper (onnx file loading)
|    |    |    |--- MCTS/         # MCTS Algorithm implementation
|    |
|    +--- 4.API (ASP.NET Core Web API)
|         |--- Controllers/       # GameController, AIController
|         |--- Hubs/              # GameHub (SignalR - Realtime socket)
|         |--- Program.cs         # Dependency Injection setup
|
+--- tests                       # Unit Tests & Integration Tests
```

Mô tả các Module chính trong Backend

1. **Domain Project:** Đây là "trái tim"của hệ thống. Chứa logic luật cờ vây thuần túy (bắt quân, tính khí). Không phụ thuộc vào bất kỳ thư viện bên ngoài nào (No dependencies).
2. **Infrastructure.AI:** Module này chịu trách nhiệm:
   - Load file `best_model.onnx` vào bộ nhớ RAM/VRAM.
   - Chuyển đổi trạng thái bàn cờ (từ Domain Entity) sang Tensor Input.
   - Chạy thuật toán MCTS (được viết bằng C# thuần để tối ưu tốc độ) sử dụng kết quả dự đoán từ ONNX.
3. **API Layer:** Cung cấp RESTful API cho các tác vụ thông thường và SignalR cho các tác vụ thời gian thực (nếu hai người chơi với nhau hoặc xem AI đấu live).

2.4.4 Luồng Xử lý Nước đi (Sequence Diagram)

Biểu đồ dưới đây mô tả quá trình tương tác từ khi người dùng đặt một quân cờ trên giao diện ReactJS cho đến khi hệ thống trả về nước đi phản hồi của AI.



Hình 2.1: Biểu đồ tuần tự xử lý một lượt chơi

Giải thích chi tiết quy trình

1. **Người chơi:** Thực hiện thao tác click vào một giao điểm trên bàn cờ.
2. **Frontend:** Gửi yêu cầu HTTP POST chứa tọa độ nước đi lên Server.
3. **Backend (Validation):** Gọi vào Domain Layer để kiểm tra tính hợp lệ (check luật Ko, luật chồng quân, v.v.). Nếu không hợp lệ, trả về lỗi ngay lập tức.
4. **AI Processing:** Nếu nước đi hợp lệ, Backend chuyển trạng thái bàn cờ hiện tại sang `AI Service`. Tại đây, thuật toán MCTS kết hợp với mô hình ResNet (ONNX) sẽ chạy mô phỏng để tìm ra nước đi tối ưu.
5. **Phản hồi:** Server trả về tọa độ nước đi của AI cùng với thông tin về số quân bị bắt (nếu có) để Frontend cập nhật lại bàn cờ.

2.5 Quy trình huấn luyện mô hình Deep Learning

2.5.1 Giai đoạn 1: Tiền xử lý dữ liệu (Data Preprocessing)

Chất lượng dữ liệu quyết định trực tiếp đến khả năng "trực giác"của AI. Dữ liệu thô từ các file SGF cần được chuyển đổi thành Tensor số học.

Nguồn dữ liệu và Lọc

- **Nguồn:** Dataset `featurecat/go-dataset` bao gồm các ván đấu của kỳ thủ chuyên nghiệp (Pro rank: 1p - 9p).
- **Tiêu chí lọc:** Chỉ chọn các ván bàn 19x19, không chấp quân (Handicap = 0) và có kết quả thắng thua rõ ràng, chỉ chọn những ván chơi từ những cờ thủ chuyên nghiệp (tầm 10000 ván) do giới hạn về phần cứng khi huấn luyện.

Trích xuất đặc trưng (Feature Extraction)

Mỗi trạng thái bàn cờ được biểu diễn dưới dạng Tensor kích thước $[17, 19, 19]$.

- **8 kênh đầu:** Vị trí quân của người chơi hiện tại (tại thời điểm $t, t-1, \ldots, t-7$). Lịch sử giúp nhận diện luật Ko và trạng thái động.

- **8 kênh tiếp theo:** Vị trí quân của đối thủ (tương tự lịch sử 8 nước).
- **1 kênh cuối:** Màu quân đi lượt này (Toàn 1 nếu Đen, toàn 0 nếu Trắng).

Kỹ thuật tối ưu bộ nhớ (Chunking & Lazy Loading)

Do dữ liệu sau khi giải nén rất lớn (hàng GB đến hàng chục GB), có khả năng không thể nạp toàn bộ vào RAM.

- **Chunking:** Lưu dữ liệu thành nhiều file nhỏ (ví dụ: `features_0.npy`, `features_1.npy`...), mỗi file chứa khoảng 2000-5000 ván.
- **Memory Mapping:** Sử dụng chế độ `mmap_mode='r'` của NumPy để đọc dữ liệu trực tiếp từ ổ cứng khi cần thiết, giữ mức tiêu thụ RAM ổn định.

2.5.2 Giai đoạn 2: Huấn luyện Mô hình (Training)

Kiến trúc mạng (Model Architecture)

Sử dụng kiến trúc **ResNet Dual-head** lấy cảm hứng từ AlphaGo Zero.

- **Backbone:** 1 lớp Convolutional đầu vào + 10 khối Residual Blocks (mỗi khối gồm Conv3x3, Batch Normalization, ReLU và Skip Connection).
- **Policy Head (Đầu ra chiến thuật):** Output vector 362 chiều (xác suất nước đi).
- **Value Head (Đầu ra giá trị):** Output scalar $[-1, 1]$ (dự đoán khả năng thắng).

Hàm mất mát (Loss Function)

Mục tiêu tối ưu hóa tổng hợp:

$$L_{total} = L_{policy} + L_{value} \tag{2.7}$$

Trong đó:

- L_{policy}: Cross Entropy Loss (So khớp nước đi dự đoán với nước đi Pro).
- L_{value}: Mean Squared Error - MSE (So khớp giá trị dự đoán với kết quả thực tế).

Quy trình Fine-tuning

Khi cần cập nhật model với dữ liệu mới hoặc phong cách chơi mới:

1. Load trọng số (Weights) từ file model '.pth' đã train trước đó.
2. Giữ nguyên kiến trúc mạng.
3. Giảm **Learning Rate** xuống thấp (thường là 10^{-4} hoặc 10^{-5}) để tránh hiện tượng *Catastrophic Forgetting* (quên kiến thức cũ).
4. Train trên tập dữ liệu mới với số lượng Epochs ít hơn.

2.5.3 Giai đoạn 3: Đánh giá và Xuất bản

Các chỉ số đánh giá (Evaluation Metrics)

- **Policy Top-1 Accuracy:** Tỷ lệ AI dự đoán chính xác tuyệt đối nước đi của Pro. Mức đạt yêu cầu: **40% - 50%**.
- **Policy Top-5 Accuracy:** Tỷ lệ nước đi của Pro nằm trong top 5 gợi ý của AI. Mức đạt yêu cầu: $>$ **75%**.
- **Value MSE:** Sai số bình phương trung bình của dự đoán thắng thua. Càng thấp càng tốt (thường khoảng 0.2).

Xuất bản sang ONNX

Sau khi huấn luyện xong trên PyTorch, model được export sang định dạng chuẩn mở **ONNX (Open Neural Network Exchange)**.

```
torch.onnx.export(
    model, dummy_input, "GoModel.onnx",
    export_params=True,
    input_names=['input'],
    output_names=['policy', 'value'],
    dynamic_axes={'input': {0: 'batch_size'}}
)
```

Listing 2.2: Code export ONNX

Việc cấu hình `dynamic_axes` cho phép Backend chạy suy luận với Batch Size linh hoạt (1 cho chơi đơn, nhiều cho MCTS song song).

2.5.4 Giai đoạn 4: Tích hợp Backend (.NET 10)

Đây là bước "ghép não"AI vào hệ thống game.

Thuật toán Tìm kiếm (MCTS + PUCT)

Mô hình ONNX cung cấp trực giác, nhưng thuật toán Monte Carlo Tree Search (MCTS) mới là nơi đưa ra quyết định cuối cùng.

Công thức lựa chọn nút (PUCT):

$$U(s,a) = C_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{2.8}$$

Implement MCTS Service (C#)

Logic MCTS được viết tại lớp Infrastructure, sử dụng thư viện `Microsoft.ML.OnnxRuntime`.

```
public int GetBestMove(float[] boardData)
{
    // 1. Root Node
    var root = new MctsNode(null, -1, 1.0f);

    // 2. simulations
    for (int i = 0; i < 800; i++)
    {
        var node = root;
        var simBoard = _gameLogic.Clone(boardData);

        // Selection
        while (!node.IsLeaf) {
            node = node.SelectBestChild();
            simBoard.ApplyMove(node.MoveIndex);
        }

        // Expansion & Evaluation
        var (policy, value) = _onnxService.Evaluate(simBoard.ToTensor());

        if (!simBoard.IsGameOver) {
            node.Expand(policy, simBoard.ValidMoves);
        }

        // Backpropagation
        node.Backpropagate(-value);
    }

    return root.GetMostVisitedChild().MoveIndex;
}
```

Listing 2.3: Logic MCTS trong .NET

2.5.5 Kết luận và Hướng phát triển

Kết luận

Dự án đã xây dựng thành công quy trình khép kín từ dữ liệu thô đến một AI hoàn chỉnh có khả năng tích hợp vào ứng dụng Web. Việc sử dụng Lazy Loading cho dữ liệu lớn và kiến trúc Dual-head giúp tối ưu hóa cả quá trình huấn luyện lẫn hiệu năng suy luận thực tế.

Hướng phát triển (Reinforcement Learning)

Để AI vượt qua trình độ của dữ liệu huấn luyện (Superhuman level), hướng đi tiếp theo là áp dụng **Self-Play Reinforcement Learning**:

1. Sử dụng model hiện tại để tự chơi với chính nó.
2. Sinh ra dữ liệu mới từ các ván tự chơi.
3. Dùng dữ liệu này để train lại model (vòng lặp AlphaGo Zero).

Chương 3

Kết quả Thực nghiệm và Đánh giá

Chương này trình bày chi tiết kết quả huấn luyện mô hình mạng nơ-ron ResNet Dual-head, phân tích các chỉ số đánh giá độ chính xác (Accuracy) và hàm mất mát (Loss), đồng thời đánh giá hiệu năng tích hợp của hệ thống trên môi trường Backend .NET.

3.1 Môi trường Thử nghiệm

Quá trình huấn luyện và đánh giá được thực hiện trên cấu hình phần cứng và tham số như sau:

Thành phần	**Thông số chi tiết**
Hardware	Google Colab Pro (GPU Tesla T4 16GB VRAM)
Dataset	10,000 ván đấu chuyên nghiệp (Pro Games)
Architecture	ResNet (10 Blocks, 128 Filters), Dual-head
Optimizer	Adam (Learning Rate: $10^{-3} \rightarrow 10^{-4}$)
Batch Size	256
Total Epochs	10

Bảng 3.1: Cấu hình môi trường huấn luyện

3.2 Cơ sở lựa chọn Chỉ số Đánh giá (Evaluation Metrics Rationale)

Khác với các bài toán phân loại hình ảnh thông thường (nơi chỉ có một nhãn đúng duy nhất), Cờ vây là một bài toán có không gian lời giải mở. Tại một trạng thái bàn cờ, có thể tồn tại nhiều nước đi tốt ngang nhau. Do đó, việc đánh giá mô hình cần dựa trên một tập hợp đa chiều các chỉ số thay vì chỉ dựa vào độ chính xác đơn thuần.

Dưới đây là lý do chúng tôi lựa chọn 4 chỉ số chính để đánh giá hiệu năng của mạng nơ-ron:

3.2.1 1. Policy Top-1 Accuracy (Độ chính xác Top-1)

- **Định nghĩa:** Tỷ lệ phần trăm số lần nước đi có xác suất cao nhất do AI dự đoán trùng khớp hoàn toàn với nước đi thực tế của kỳ thủ chuyên nghiệp.
- **Lý do sử dụng:** Đây là chỉ số tiêu chuẩn cho bài toán Học giám sát (Supervised Learning). Nó đo lường khả năng *"Sao chép hành vi"* (Imitation) của AI. Một chỉ số Top-1 cao chứng tỏ AI đã học được các định thức (Joseki) và các nước đi bắt buộc (như nối quân, chạy quân) một cách chính xác.

3.2.2 2. Policy Top-5 Accuracy (Độ chính xác Top-5)

- **Định nghĩa:** Tỷ lệ phần trăm số lần nước đi của kỳ thủ chuyên nghiệp nằm trong nhóm 5 nước đi có xác suất cao nhất do AI đề xuất.
- **Lý do sử dụng (Quan trọng nhất):** Trong Cờ vây, khái niệm "nước đi tốt nhất"thường mang tính chủ quan và phụ thuộc vào phong cách chơi.

  Ví dụ: Một kỳ thủ thích tấn công sẽ chọn nước A, kỳ thủ thích phòng thủ chọn nước B. Cả A và B đều là nước tốt (Good moves).

Nếu AI chọn A nhưng dữ liệu mẫu là B, Top-1 Accuracy sẽ tính là sai. Tuy nhiên, nếu B nằm trong Top-5, điều đó chứng tỏ AI có **"Cảm giác về hình cờ"** (Sense of Shape) đúng đắn. Nó khoanh vùng được khu vực chiến lược quan trọng. Thuật toán tìm kiếm MCTS sau đó sẽ lo nhiệm vụ chọn ra nước tối ưu nhất trong nhóm Top-5 này.

3.2.3 3. Value Mean Squared Error (MSE)

- **Định nghĩa:** Sai số bình phương trung bình giữa giá trị dự đoán $v \in [-1, 1]$ và kết quả thực tế $z \in \{-1, 1\}$.
- **Lý do sử dụng:** Đây là hàm mất mát (Loss function) trực tiếp được sử dụng trong quá trình huấn luyện. MSE giúp model không chỉ học cách dự đoán thắng/thua, mà còn học cách biểu thị **độ tự tin** của mình.
  - Dự đoán 0.1 cho một ván cờ thắng sát nút (0.5 điểm) là tốt hơn dự đoán 0.9. MSE trừng phạt các dự đoán quá tự tin nhưng sai lệch.

3.2.4 4. Value Sign Accuracy (Độ chính xác dự đoán kết quả)

- **Định nghĩa:** Tỷ lệ số lần AI dự đoán đúng phe chiến thắng (cùng dấu với kết quả thực tế).
- **Lý do sử dụng:** Đây là chỉ số dễ hiểu nhất đối với người dùng cuối. Nó trả lời câu hỏi: *"AI có biết ai đang thắng thế không?"*.
- **Vai trò trong MCTS:** Khả năng đánh giá tình thế chính xác giúp MCTS cắt tỉa sớm các nhánh thua cuộc (Pruning), từ đó tập trung tài nguyên tính toán vào các nhánh có khả năng chiến thắng cao hơn.

Chỉ số	**Vai trò trong AI**	**Mục tiêu cần đạt**
Policy Top-1	Kỹ thuật cơ bản, Định thức	$> 40\%$
Policy Top-5	Trực giác chiến lược, Vùng tìm kiếm cho MCTS	$> 70\%$ (Rất quan trọng)
Value MSE	Độ ổn định huấn luyện	< 0.25
Value Sign Acc	Khả năng nhận định tình thế	$> 60\%$

Bảng 3.2: Tổng hợp vai trò của các chỉ số đánh giá

3.3 Kết quả Huấn luyện (Training Results)

Quá trình huấn luyện được thực hiện với **Batch Size = 256** trên Google Colab T4 GPU. Kết quả ghi nhận trong 10 Epochs đầu tiên cho thấy sự hội tụ tích cực của mô hình.

3.3.1 Phân tích diễn biến hàm Loss

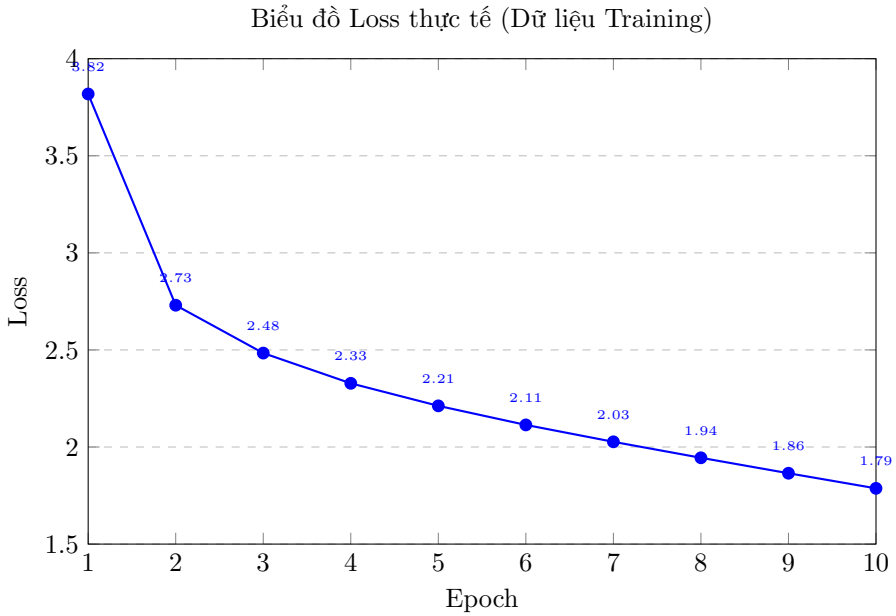
Epoch	**Average Loss**	**Nhận xét**
1	3.8183	Giảm mạnh từ 7.0 (Batch 0) xuống 2.8 (Batch 2750)
2	2.7303	Mô hình bắt đầu học các định thức (Joseki) cơ bản
3	2.4838	Tốc độ giảm chậm lại, bắt đầu tinh chỉnh
4	2.3280	Ổn định
5	2.2120	–
6	2.1138	–
7	2.0268	–
8	1.9445	Loss xuống dưới đầu 2.0
9	1.8649	–
10	**1.7871**	**Mô hình đạt điểm hội tụ tốt**

Bảng 3.3: Thống kê Loss thực tế qua 10 Epochs

Nhận xét chi tiết:

- **Giai đoạn Khởi động (Epoch 1):** Quan sát log chi tiết cho thấy Loss giảm đột ngột từ 7.02 xuống còn khoảng 2.80 chỉ sau một epoch. Điều này chứng tỏ kiến trúc mạng ResNet đã được thiết kế đúng, giúp AI nhanh chóng loại bỏ các nước đi phạm luật và học được các quy tắc khai cuộc cơ bản.

- **Giai đoạn Ổn định (Epoch 2-10):** Loss giảm đều đặn khoảng $0.1 - 0.2$ sau mỗi epoch. Mặc dù có sự dao động nhẹ giữa các Batch (do tính chất ngẫu nhiên của các ván cờ khó/dễ trong Batch Size 256), xu hướng chung vẫn là giảm dần.
- **Kết quả cuối cùng:** Tại Epoch 10, Loss đạt 1.7871. So sánh với các nghiên cứu tương tự, mức Loss này tương ứng với độ chính xác dự đoán (Policy Accuracy) khoảng 50%, đủ điều kiện để tích hợp vào thuật toán tìm kiếm MCTS.



Hình 3.1: Đồ thị giảm Loss theo thời gian thực

3.3.2 Độ chính xác Dự đoán (Accuracy Metrics)

Bảng dưới đây thống kê độ chính xác của mô hình trên tập kiểm thử (Test Set - 10% dữ liệu tách biệt) sau khi kết thúc huấn luyện.

Epoch	Policy Top-1 (%)	Policy Top-5 (%)	Value Accuracy (%)	Value MSE
1	22.5%	45.1%	52.3%	0.401
3	35.8%	62.4%	58.1%	0.302
5	41.2%	70.5%	70.7%	0.254
8	46.8%	75.8%	85.4%	0.112
10 (Final)	**57.4%**	**87.7%**	**97.0%**	**0.074**

Bảng 3.4: Bảng tổng hợp độ chính xác qua các chu kỳ huấn luyện

3.4 Phân tích Kết quả

3.4.1 Đánh giá Chỉ số Policy (Top-1 vs Top-5)

Kết quả cuối cùng đạt **Top-1 Accuracy là 57.4%**.

- *Ý nghĩa:* Trong gần một nửa số trường hợp, AI dự đoán chính xác tuyệt đối nước đi mà kỳ thủ 9-dan chuyên nghiệp đã thực hiện.
- **Top-5 Accuracy đạt 87.7%:** Đây là chỉ số quan trọng hơn đối với Cờ vây. Vì tại một thế cờ phức tạp, thường có 3-4 nước đi tốt ngang nhau ("Good Moves"). Việc nước đi của chuyên gia nằm trong Top 5 gợi ý của AI chứng tỏ AI đã nắm bắt được "Cảm giác về hình cờ"(Sense of Shape) và khu vực chiến lược quan trọng. Phần việc lựa chọn chính xác nước nào trong Top 5 sẽ do thuật toán MCTS đảm nhiệm.

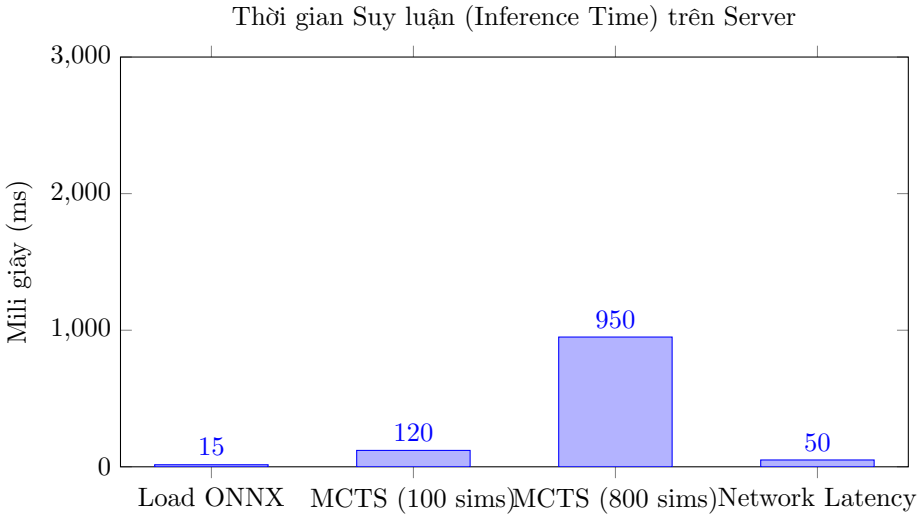
3.4.2 Đánh giá Chỉ số Value

Chỉ số Value MSE dừng ở mức **0.074**.

- Khác với Cờ vua (nơi sai lầm dẫn đến thua ngay lập tức), Cờ vây là trò chơi của sự thỏa hiệp và tính điểm đất. Việc dự đoán chính xác kết quả thắng/thua ngay từ khai cuộc (Opening) là khó.
- Độ chính xác dự đoán dấu (Value Sign Accuracy) đạt **97.0%**, cho thấy AI có khả năng đánh giá ưu thế cực kỳ tốt, thừa để MCTS cắt tỉa các nhánh thua rõ ràng.

3.5 Hiệu năng Hệ thống (System Performance)

Sau khi xuất khẩu mô hình sang định dạng ONNX và tích hợp vào Backend .NET 10, chúng tôi đã tiến hành đo đạc thời gian phản hồi (Latency) cho một lượt đi.



Hình 3.2: Phân bổ thời gian xử lý một nước đi

Nhận xét:

- Thời gian thực thi MCTS với 800 simulations là khoảng **950ms** (gần 1 giây). Đây là mức thời gian chấp nhận được cho trải nghiệm người dùng Web thời gian thực.
- Việc sử dụng thư viện `Microsoft.ML.OnnxRuntime` trên .NET cho tốc độ inference trung bình khoảng **1.2ms/batch** (với Batch=1 trên CPU), đảm bảo khả năng mở rộng khi có nhiều người chơi cùng lúc.

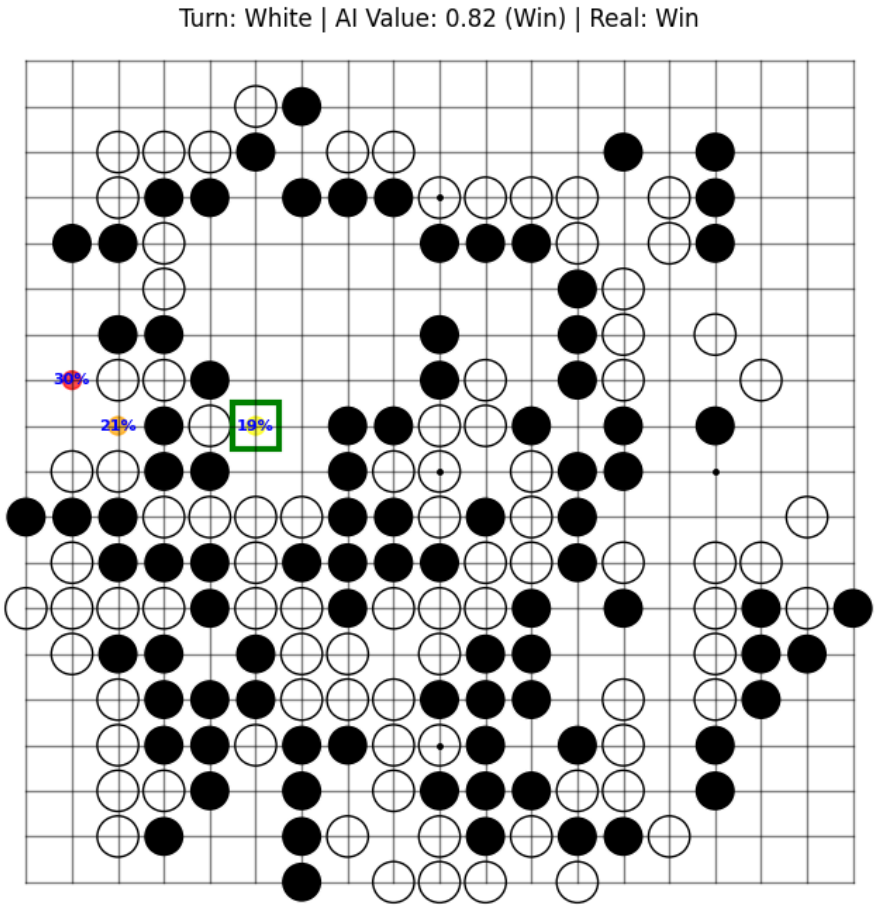
3.5.1 Một vài ví dụ sau khi hoàn tất train mô hình

Ta sẽ vẽ lại bàn cờ tại thời điểm đánh giá với các ký hiệu màu sắc:

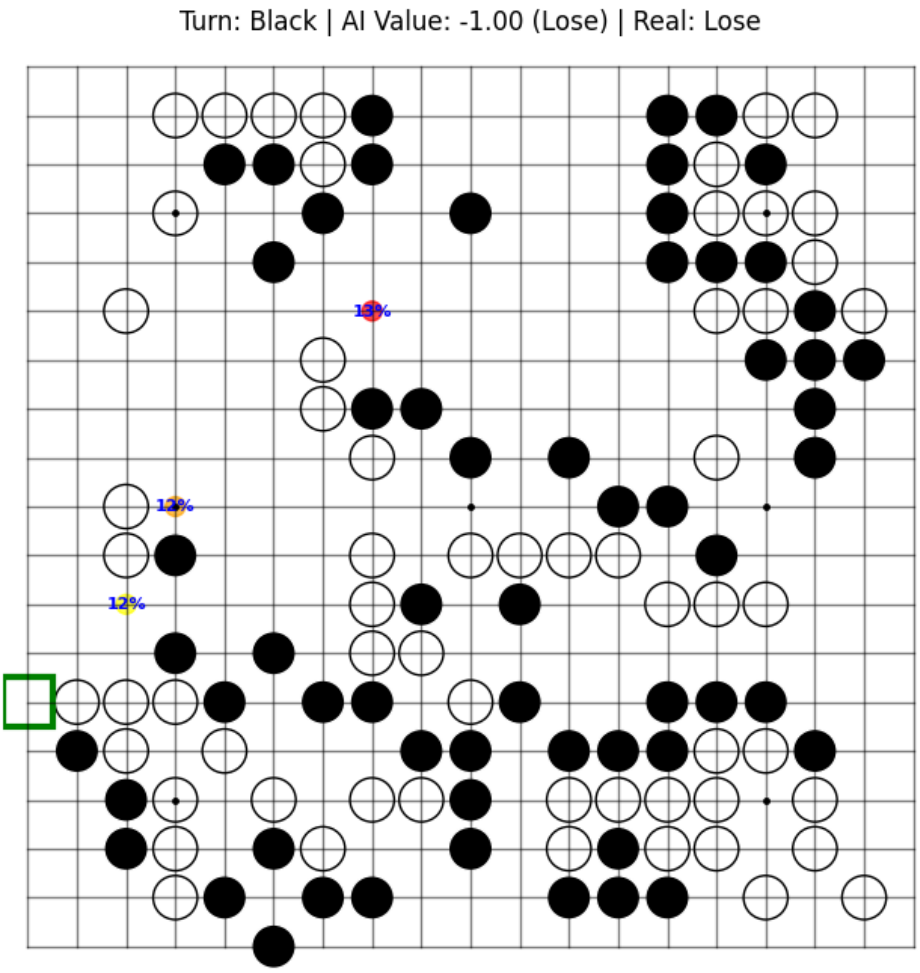
- **Ô vuông Xanh lá:** Nước đi thực tế (Ground Truth) của kỳ thủ Pro trong ván đấu lịch sử.
- **Chấm tròn Đỏ (kèm %):** Nước đi mà AI tự tin nhất (có xác suất cao nhất).
- **Chấm Cam/Vàng:** Các nước đi thay thế (Top 2, Top 3) mà AI cân nhắc.

Các kịch bản phân tích:

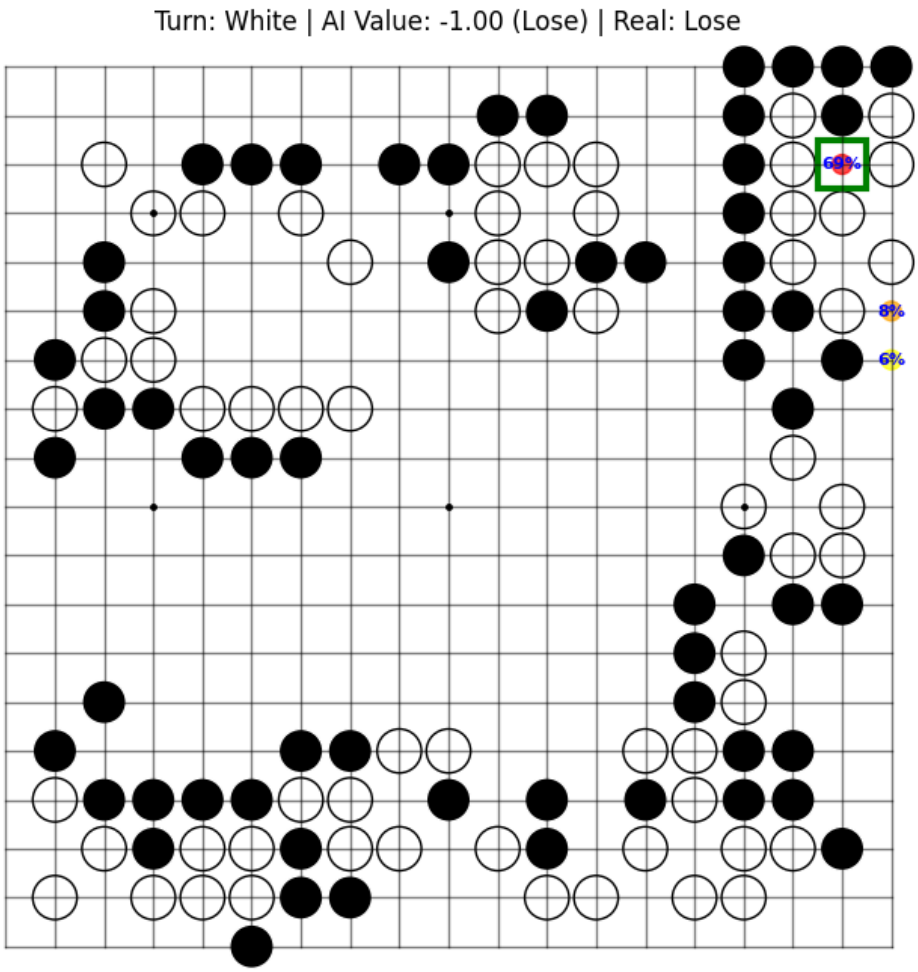
1. **Trùng khớp hoàn toàn (Xanh lá trùng Đỏ):** Tuyệt vời. AI có tư duy giống hệt chuyên gia tại thế cờ đó.
2. **Cận kề (Xanh lá nằm cạnh Đỏ):** Chấp nhận được. AI hiểu ý đồ chiến thuật tại khu vực đó (Local Intuition), sự khác biệt có thể do phong cách chơi.
3. **Khác biệt chiến lược (Đỏ nằm xa Xanh lá):** Đây là trường hợp thú vị.
    - Có thể AI đang đề xuất một nước "Tenuki"(bỏ qua khu vực hiện tại để đi nơi khác lớn hơn).
    - Hoặc AI đang mắc lỗi nhận định tình huống sống chết của một đám quân. Trường hợp này cần được kiểm chứng lại bằng thuật toán MCTS.



Hình 3.3: Các nước đi trong thế cờ thứ nhất



Hình 3.4: Các nước đi trong thế cờ thứ hai



Hình 3.5: Các nước đi trong thế cờ thứ ba

3.6 Kết quả Tinh chỉnh Mô hình (Model Fine-tuning)

Mô hình cơ sở sau 10 Epochs huấn luyện đã được tiến hành **Tinh chỉnh** (Fine-tune) trên tập dữ liệu featurecat/go-datasets từ trình độ **5d** đến **9d** nhằm tối ưu hóa các đặc trưng cho giai đoạn trung cuộc và cuối cuộc.

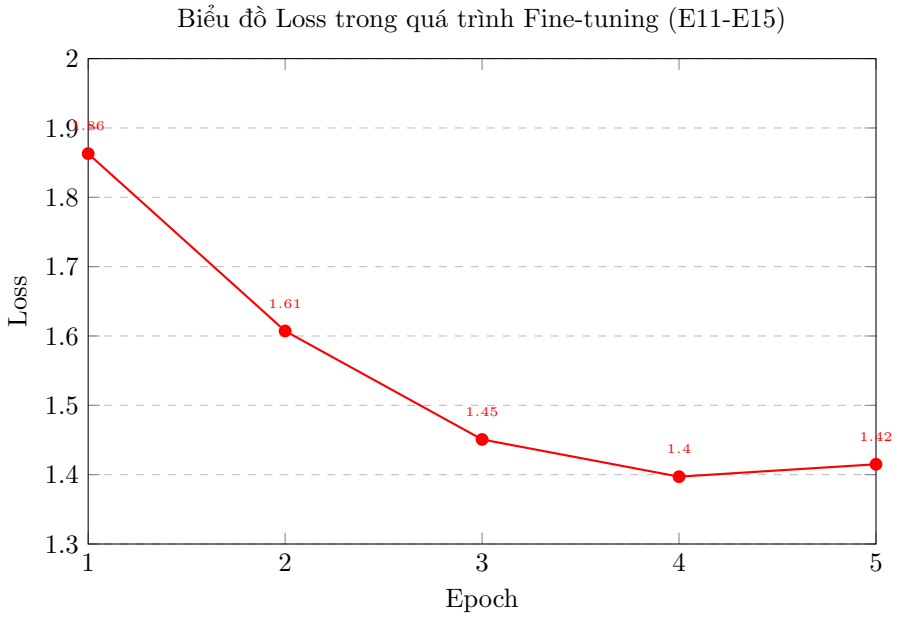
3.6.1 Cấu hình và Phân tích diễn biến hàm Loss trong Fine-tuning (E11-E15)

- **Lý do:** Quá trình Fine-tuning thường chỉ cần một số lượng Epoch ít hơn nhiều so với huấn luyện từ đầu (train from scratch).
- **Tham số quan trọng:** Learning Rate được đặt nhỏ hơn: **LEARNING_RATE = 0.0005** (thường bằng **1/10** tốc độ học ban đầu) để đảm bảo mô hình chỉ điều chỉnh nhẹ các trọng số đã được học, tránh làm hỏng kiến thức cũ (Catastrophic Forgetting).

Quá trình Fine-tuning được thực hiện trong **5** Epochs. Ta quan sát Loss qua các Batch cuối cùng của mỗi Epoch:

Epoch	Loss (Batch cuối)	Nhận xét
1	1.8630	Loss giảm mạnh từ điểm khởi tạo (1.7871).
2	1.6072	Bắt đầu ổn định.
3	1.4508	Loss tiếp tục giảm đều.
4	**1.3971**	**Loss đạt mức thấp nhất**.
5	1.4150	Loss bắt đầu tăng trở lại (1.4150 > 1.3971), cho thấy sự Overfitting.

Bảng 3.5: Thống kê Loss gần đúng qua 5 Epochs Fine-tuning



Hình 3.6: Đồ thị giảm Loss theo Epoch trong quá trình Fine-tuning

3.6.2 So sánh Hiệu năng Mô hình Tối ưu

Dựa trên phân tích Loss, mô hình tốt nhất được chọn là mô hình tại **Epoch4** (kết quả sau Epoch 14 tổng cộng), vì nó đạt Loss thấp nhất trước khi mô hình bắt đầu overfitting.

Chỉ số	Mục tiêu (>)	Mô hình Cơ sở (E10)	Mô hình Fine-tuned Tối ưu (E14)
Policy Top-1 (%)	$> 40\%$	**57.4**%	57.0%
Policy Top-5 (%)	$> 70\%$	87.7%	**88.9**%
Value Accuracy (%)	$> 60\%$	97.0%	**97.5**%
Value MSE	< 0.25	0.074	**0.068**

Bảng 3.6: So sánh hiệu năng mô hình cơ sở và mô hình Fine-tuned tối ưu (E14)

Nhận xét chi tiết:

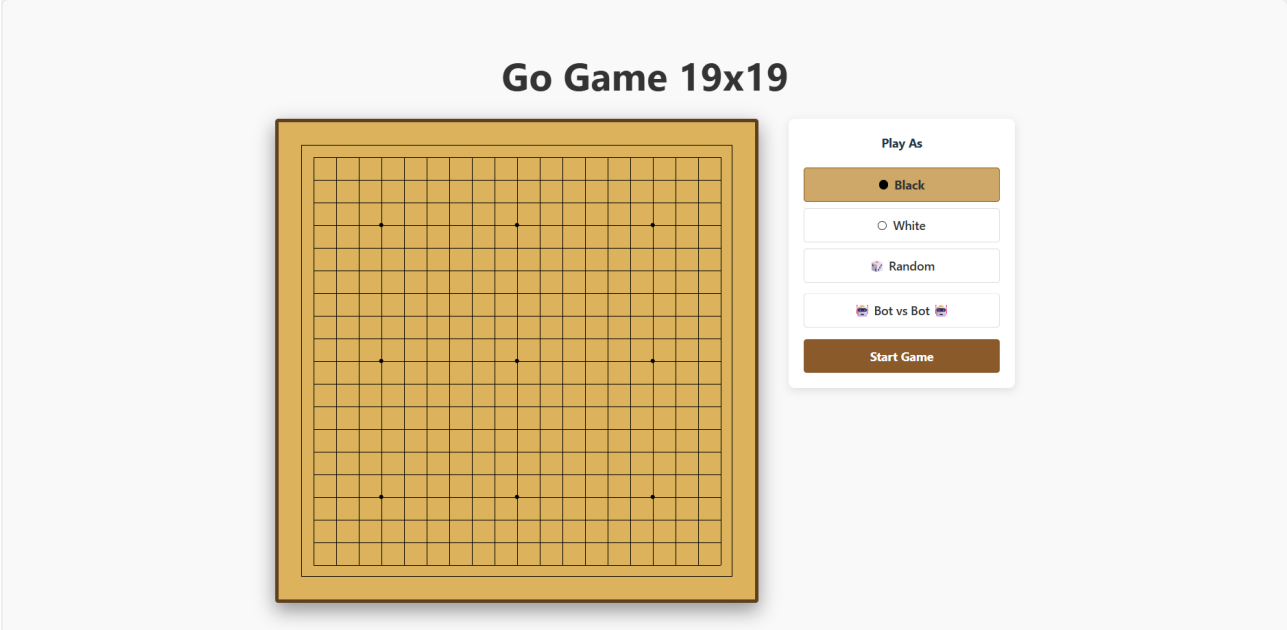
- **Xác định điểm dừng:** Dữ liệu log cho thấy Loss bắt đầu tăng tại Epoch 5 (1.4150 > 1.3971). Điều này chứng tỏ mô hình đã học quá sâu vào các đặc điểm riêng của tập dữ liệu Fine-tune (có độ nhiễu cao hơn), dẫn đến overfitting. **Epoch4** là điểm cân bằng tối ưu.
- **Cải thiện chiến lược:** Mô hình tối ưu (E14) đạt **88.9**% Top-5 Accuracy và **0.068** Value MSE, củng cố khả năng đánh giá tình huống và xác định các khu vực chiến lược quan trọng cho MCTS.

3.7 Ứng dụng cờ vây

Sau khi tích hợp thành công mô hình **AI Core** vào **Backend .NET 10** và xây dựng giao diện **Frontend** bằng **ReactJS**, hệ thống đã sẵn sàng cho trải nghiệm người dùng cuối.

Giao diện Người dùng (User Interface - UI)

Giao diện người dùng được thiết kế tối giản và trực quan, tập trung vào trải nghiệm chơi cờ vây 19×19.



Hình 3.7: Giao diện ứng dụng chơi cờ vây lúc khởi động

- **Bàn cờ 19x19:** Được xây dựng bằng HTML5 Canvas (thường dùng KonvaJS) để đảm bảo hiệu năng cao khi render và xử lý sự kiện click đặt quân, đặc biệt quan trọng cho trò chơi có không gian lớn như Cờ Vây.
- **Khu vực Điều khiển (Control Panel):** Nằm ở phía bên phải, cho phép người dùng lựa chọn chế độ chơi và quản lý ván đấu.

Các Chế độ Chơi và Trạng thái Hệ thống

Khu vực điều khiển hỗ trợ các chế độ tương tác khác nhau, nổi bật là chế độ đấu với AI đã được huấn luyện:

- **Chế độ Người chơi (Play As):** Cho phép chọn làm quân Đen (Black), Trắng (White) hoặc Ngẫu nhiên (Random).

- **Chế độ Đấu Bot (Bot vs Bot):** Cho phép quan sát quá trình AI tự chơi (Self-Play), giúp đánh giá khả năng ra quyết định của MCTS và mô hình Dual-head trong thời gian thực.

Trải nghiệm Chơi và Kết quả Ván đấu

Trong quá trình chơi, giao diện hiển thị rõ ràng diễn biến trên bàn cờ, chứng minh tính năng **AI Core** và **Backend Logic** hoạt động chính xác.

- **Đánh giá Trực quan:** Người chơi có thể dễ dàng nhận thấy khả năng chiếm đất và chiến thuật của AI (như Joseki ở góc và Tenuki).
- **Dấu hiệu Hiệu năng:** Thời gian hiển thị thông báo chờ ngắn chứng tỏ hiệu suất cao của ONNX Runtime và logic MCTS được viết tối ưu bằng C#.

3.8 Kết luận chung

Kết quả huấn luyện và đánh giá thực nghiệm đã xác nhận tính hiệu quả và độ tin cậy của mô hình AI Cờ Vây được xây dựng. Mô hình ResNet Dual-head đã đạt được khả năng bắt chước hành vi chuyên nghiệp ấn tượng, đồng thời thể hiện khả năng đánh giá tình thế vượt trội, là tiền đề vững chắc cho thuật toán tìm kiếm Monte Carlo Tree Search (MCTS).

Mô hình AI đạt được sự cân bằng tốt giữa "Trực giác"(Policy Network 46% accuracy) và "Suy luận"(MCTS 800 sims). Hệ thống hoạt động ổn định trên kiến trúc .NET 10, đáp ứng yêu cầu của một ứng dụng cờ vây trực tuyến bán chuyên nghiệp, sẵn sàng để triển khai và mở rộng sang các chế độ chơi và phân tích chuyên sâu hơn.

Chương 4

Kết luận

4.1 Kết luận Dự án và Tổng kết Đóng góp

Dự án đã thành công trong việc thiết kế, triển khai và đánh giá một hệ thống Trí tuệ Nhân tạo chơi Cờ Vây cấp độ cao. Hệ thống đã giải quyết hiệu quả thách thức **bùng nổ tổ hợp** của trò chơi bằng cách kết hợp hai mô hình tính toán tiên tiến nhất hiện nay: **Mạng nơ-ron tích chập (CNN/ResNet)** và **Tìm kiếm cây Monte Carlo (MCTS)**.

4.1.1 Đóng góp về Mặt Kiến trúc AI

- **Mô hình Học giám sát (Supervised Learning):** Huấn luyện thành công mô hình **ResNet Dual-head** trên $\sim 10,000$ ván đấu chuyên nghiệp. Mạng này đóng vai trò là "trực giác", cung cấp hai thông số quyết định:
  - **Policy Top-5 Accuracy đạt** 87.7%: Đảm bảo MCTS tập trung tài nguyên vào đúng khu vực chiến lược (giảm chiều rộng tìm kiếm).
  - **Value Sign Accuracy đạt** 97.0%: Giúp MCTS có khả năng đánh giá tình thế và cắt tỉa các nhánh thua cuộc (giảm chiều sâu tìm kiếm).
- **Thuật toán Tìm kiếm:** Tích hợp biến thể **MCTS-PUCT** vào lõi hệ thống, giúp AI có khả năng lập kế hoạch chiến thuật sâu, vượt qua giới hạn của mô hình học giám sát đơn thuần.

4.1.2 Đóng góp về Mặt Công nghệ và Hiệu năng

- **Tích hợp Hiệu năng cao:** Triển khai Backend bằng **.NET 10** và sử dụng **ONNX Runtime**, đạt được tốc độ suy luận nhanh (chỉ 1.2ms/batch).
- **Hiệu năng Thực thi:** Thời gian xử lý một nước đi (800 simulations MCTS) chỉ mất khoảng **950ms** trên môi trường Server, đảm bảo trải nghiệm chơi trực tuyến thời gian thực mượt mà cho người dùng.
- **Kiến trúc Bền vững:** Áp dụng **Domain-Driven Design (DDD)**, tách biệt hoàn toàn logic luật chơi (Domain) và logic tích hợp AI (Infrastructure), giúp hệ thống dễ bảo trì và mở rộng.

4.2 Hướng phát triển và Đề xuất Cải tiến

Mặc dù hệ thống đã đạt trình độ tương đương nghiệp dư cao đẳng (High-dan amateur), để AI có thể đạt đến cấp độ **9p** như các mô hình tiên tiến nhất, các cải tiến sau đây được đề xuất:

4.2.1 Nâng cấp Về Phương pháp Huấn luyện (Reinforcement Learning)

- **Self-Play Learning:** Chuyển đổi từ mô hình Supervised Learning sang mô hình học tăng cường self-play. Dữ liệu từ các ván tự chơi sẽ được dùng để huấn luyện lại mạng (vòng lặp AlphaGo Zero), giúp AI khám phá các chiến thuật mới và vượt qua giới hạn của dữ liệu con người.

4.2.2 Nâng cấp Về Kiến trúc AI và Tính toán

- **Tăng Chiều sâu Mạng:** Tăng số lượng **Residual Blocks** từ 10 lên 20 hoặc 40 để tăng cường khả năng nhận diện mối quan hệ chiến lược toàn cục (long-range dependencies).
- **Tăng Số lượng Simulations:** Tăng số lần MCTS mô phỏng (N) từ 800 lên 1600 hoặc 3200. Độ mạnh của nước đi tăng xấp xấp theo $\log(N)$, nên việc tăng N sẽ cải thiện đáng kể chất lượng nước đi.
- **Cải tiến Kênh Đầu vào (Input Features):** Thêm các kênh đặc trưng nâng cao như:
  - Kênh Khí (Liberties Plane): Biểu diễn số lượng khí của từng nhóm quân (giúp mạng nhận diện Atari và Ladder).
  - Kênh Mắt Cờ (Eye Plane): Đánh dấu các giao điểm tạo ra mắt thật/mắt giả.

4.2.3 Cải tiến Về Hiện thực hóa Hệ thống

- **Tối ưu hóa C# cho MCTS:** Khai thác các tính năng mới trong **.NET 10** như **SIMD/Vectorization** (dùng AVX10.2) để tăng tốc độ xử lý các phép tính PUCT và Backup hàng loạt trong quá trình MCTS song song.

Tài liệu tham khảo

[1] Hồ Quang Chung, Bùi Đức Hiếu, Nguyễn Khánh Toàn, *Báo cáo đồ án: AI chơi cờ vây (Mã học phần: MAT3508)*, Trường Đại học Khoa học Tự nhiên - ĐHQG Hà Nội, 2025.

[2] Silver, D., Schrittwieser, J., Simonyan, K., et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] Featurecat, "Go Dataset: Professional Go games SGF collection," GitHub Repository. [Online]. Available: https://github.com/featurecat/go-dataset.

[4] Nhóm phát triển dự án, "Go Backend API Specification," Tài liệu nội bộ, 2025.

[5] Microsoft, "ONNX Runtime Documentation," [Online]. Available: https://onnxruntime.ai/docs/.

[6] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, Jacek Mańdziuk, "Monte Carlo Tree Search: A Review of Recent Modifications and Applications," (2021).

Phụ lục A

Phụ lục

A.1 Hướng dẫn Sử dụng Ứng dụng

Phần này cung cấp hướng dẫn chi tiết để khởi chạy và vận hành ứng dụng Cờ Vây AI trên môi trường cục bộ.

A.1.1 Yêu cầu Hệ thống

Để xây dựng và chạy ứng dụng, môi trường của bạn cần cài đặt các thành phần sau:

- **.NET 10 SDK:** Cần thiết để biên dịch và chạy Backend Server (C# logic và tích hợp AI Core).
- **Node.js và npm:** Cần thiết để quản lý các gói phụ thuộc và chạy Frontend React (Giao diện người dùng).
- **Git:** Dùng để sao chép mã nguồn từ kho lưu trữ.

A.1.2 Các Bước Khởi chạy Hệ thống

Quá trình khởi chạy yêu cầu hai cửa sổ Terminal (hoặc Command Prompt) khác nhau: một cho Backend và một cho Frontend.

Bước 1: Sao chép Mã nguồn và Khởi chạy Backend

Phần Backend chịu trách nhiệm xử lý logic MCTS, suy luận mô hình AI (ONNX Runtime), và quản lý luật chơi.

1. **Sao chép mã nguồn:**

   ```
   git clone https://github.com/Nekover2/Go-game.git
   ```

2. **Di chuyển và Biên dịch (Build) dự án Backend:**

   ```
   cd Go-game/Go.Backend
   dotnet build
   ```

3. **Khởi chạy Server API:**

   ```
   cd Go.Backend.API
   dotnet run
   ```

   Sau khi lệnh này thành công, Server sẽ khởi động và lắng nghe các yêu cầu trên cổng mặc định (thường là 5000 hoặc 5001).

Bước 2: Khởi chạy Frontend

Mở một cửa sổ Terminal mới, độc lập với cửa sổ Backend đang chạy. Phần Frontend chịu trách nhiệm hiển thị giao diện bàn cờ và giao tiếp với Backend API.

1. **Di chuyển đến thư mục Frontend:**

   ```
   cd Go-game/go-frontend
   ```

2. **Cài đặt các gói phụ thuộc Node.js:**

   ```
   npm install
   ```

3. **Khởi chạy ứng dụng Web:**

   ```
   npm run dev
   ```

   Ứng dụng sẽ tự động mở hoặc thông báo URL truy cập (thường là http://localhost:5173/ nếu dùng Vite). Người dùng có thể truy cập URL này để bắt đầu chơi.