ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO DỰ ÁN

Dự án: Cải tiến OCR chữ viết tay tiếng Việt dựa trên Kiến trúc Transformer và Chiến lược Dữ liệu (Data-Centric)

Tác giả: Nguyễn Hoàng Quân

Tóm tắt đóng góp (dự án cá nhân):

- Xây dựng pipeline dữ liệu theo hướng data-centric (thu thập, làm sạch, stitching/chuẩn hóa) cho OCR chữ viết tay tiếng Việt.
- Huấn luyện/tinh chỉnh mô hình OCR dựa trên kiến trúc Transformer (TrOCR) và tối ưu hóa quá trình suy luận.
- Tích hợp mô hình ngôn ngữ KenLM cho hậu xử lý/giải mã; thiết kế đánh giá thực nghiệm và phân tích lỗi; hoàn thiện demo và báo cáo kỹ thuật.

Mục lục

TÓM TẮT

Nhận dạng chữ viết tay tiếng Việt (Vietnamese Handwritten OCR) là một bài toán quan trọng trong bối cảnh chuyển đổi số, nhằm số hoá dữ liệu từ tài liệu giấy sang văn bản có thể lưu trữ, tìm kiếm và phân tích. Bài toán này đặc biệt thách thức do sự đa dạng lớn của nét chữ, điều kiện chụp/scan không đồng nhất, đồng thời tiếng Việt có hệ thống dấu thanh và dấu phụ khiến sai khác nhỏ ở ký tự có thể làm thay đổi nghĩa từ. Bên cạnh đó, dữ liệu chữ viết tay tiếng Việt chất lượng cao thường khan hiếm, khiến các mô hình nhận dạng truyền thống khó đạt hiệu năng tốt và khó tổng quát hoá trong môi trường thực tế.

Trong đồ án này, tôi xây dựng một hệ thống OCR theo hướng **end-to-end** bằng cách kết hợp các mô-đun mạnh sẵn có và tập trung tối ưu mô-đun nhận dạng. Cụ thể, hệ thống sử dụng **DBNet** cho bài toán phát hiện vùng văn bản trên ảnh tài liệu, sau đó thực hiện hậu xử lý để sắp xếp và cắt ảnh theo từng dòng, và áp dụng mô hình nhận dạng dựa trên **Transformer (TrOCR)** để sinh chuỗi ký tự. Để tăng độ hợp lệ ngôn ngữ trong giai đoạn suy luận, hệ thống tích hợp **KenLM** trong quá trình giải mã bằng beam search. Theo hướng tiếp cận *data-centric*, tôi triển khai quy trình huấn luyện theo ba giai đoạn: (i) **pre-train** trên hơn **1 triệu** ảnh dữ liệu bao gồm cả chữ in và chữ viết tay tiếng Anh lẫn tiếng Việt để mô hình nhận diện được nét chữ, (ii) **fine-tune** tiếp trên tập **1 triệu** ảnh ở giai đoạn 1 và kết hợp thêm vài chục nghìn ảnh chữ viết tay do tôi tự viết và áp dụng kỹ thuật **stitching** để tăng đa dạng độ dài/chuỗi ký tự, và (iii) **fine-tune chuyên biệt** trên tập thuần chữ viết tay tiếng Việt do tôi tổng hợp từ các nguồn và của cả chính tôi tự tạo nhằm thích nghi miền dữ liệu mục tiêu.

Kết quả thực nghiệm trên tập kiểm thử ảnh 1 dòng cho thấy mô hình TrOCR sau tối ưu đạt **WER 26.9%** (tương ứng **CER 10.88%** và **SER 76.02%**), cải thiện rõ rệt so với mô hình baseline CRNN đạt **WER 52.21%**. Các benchmark hiệu năng bổ sung cũng cho thấy hệ thống có thể vận hành ở chế độ suy luận nhanh (greedy) hoặc chế độ chất lượng cao (beam search kết hợp KenLM), phù hợp với các yêu cầu triển khai khác nhau trong thực tế.

CHƯƠNG 1: GIỚI THIỆU

1.1 Lý do chọn đề tài (Motivation)

1.1.1 Bối cảnh và nhu cầu trong chuyển đổi số

Trong chuyển đổi số, rất nhiều thông tin vẫn đang nằm trên giấy dưới dạng sổ tay, biểu mẫu điền tay, đơn từ hoặc ghi chú. Nếu không chuyển những nội dung này sang văn bản số thì việc lưu trữ, tìm kiếm và khai thác dữ liệu sẽ phải làm thủ công, tốn thời gian và dễ sai sót. Vì vậy, OCR (nhận dạng ký tự quang học) là một công cụ quan trọng để tự động hoá bước nhập liệu từ tài liệu giấy sang tài liệu số.

Tuy nhiên, OCR chữ viết tay khó hơn nhiều so với chữ in vì nét chữ mỗi người mỗi khác, thường có hiện tượng dính nét, thiếu nét, nghiêng hoặc cong dòng. Khi chụp/scan tài liệu còn có thêm nhiễu, mờ, bóng đổ và nền giấy không đều. Với tiếng Việt, độ khó càng lớn hơn do có dấu thanh và dấu phụ, chỉ cần sai dấu là có thể đổi nghĩa của từ. Do đó, xây dựng hệ thống OCR chữ viết tay tiếng Việt có độ chính xác cao và hoạt động ổn định trên ảnh tài liệu là một nhu cầu rõ ràng và có giá trị ứng dụng lớn.

1.1.2 Hạn chế của các phương pháp truyền thống và động lực áp dụng Transformer

Trong các hệ thống OCR hiện đại, bài toán thường được phân rã thành hai thành phần chính: **phát hiện vùng chữ (text detection)** và **nhận dạng nội dung (text recognition)**. Ở mức nhận dạng, các phương pháp phổ biến trước đây như CNN+RNN+CTC (điển hình là CRNN) đã chứng minh hiệu quả trên nhiều tập dữ liệu. Tuy nhiên, khi áp dụng cho chữ viết tay tiếng Việt, các hướng tiếp cận dựa nhiều vào cơ chế tuần tự (RNN/CTC) có thể bộc lộ một số hạn chế:

- **Khả năng mô hình hoá ngữ cảnh hạn chế**: Trong CRNN+CTC, mô hình thường xử lý ảnh dòng chữ bằng cách biến đổi ảnh thành một *dãy đặc trưng theo chiều ngang* (mỗi "cột" đặc trưng tương ứng một vị trí từ trái sang phải). CTC giả định các dự đoán tại từng vị trí trong dãy này gần như độc lập có điều kiện và sử dụng cơ chế căn chỉnh (alignment) để ghép dãy dự đoán thành chuỗi ký tự cuối. Vì vậy, khi dòng chữ dài, ký tự dính nét hoặc khoảng trắng không rõ, việc căn chỉnh trở nên khó ổn định và mô hình khó tận dụng ngữ cảnh xa để sửa các nhầm lẫn cục bộ. [11, 12]

- **Độ nhạy với biến dạng và nhiễu**: CRNN thường "đọc" ảnh theo hướng trái→phải bằng cách coi trục ngang của ảnh như trục tuần tự (sequence axis). Nói cách khác, ảnh được nén theo chiều cao và chuyển thành một chuỗi vector đặc trưng $\{f_1, f_2, \ldots, f_T\}$ với T xấp xỉ tỉ lệ với **chiều rộng** của ảnh sau backbone CNN. Khi chữ viết tay bị nghiêng, co giãn, cong dòng, hoặc ký tự bị dính/đứt nét, các vùng nét không còn thẳng hàng theo trục ngang, làm cho một vector f_t có thể chứa thông tin trộn lẫn của nhiều ký tự hoặc chỉ chứa

một phần ký tự. Điều này khiến mô hình dự đoán sai ở một số vị trí và lỗi dễ lan truyền qua bước căn chỉnh CTC. [4]

- **Khó tận dụng tri thức ngôn ngữ**: Với CTC, có thể bổ sung mô hình ngôn ngữ ở bước giải mã (ví dụ lexicon hoặc LM) để cải thiện chuỗi đầu ra. Tuy nhiên, do CTC dự đoán theo "khung vị trí" (frame-wise) và phụ thuộc mạnh vào alignment, việc kết hợp ngôn ngữ thường kém linh hoạt hơn so với mô hình seq2seq/Transformer, nơi ngôn ngữ được tích hợp tự nhiên trong quá trình sinh chuỗi (autoregressive decoding) và có thể khai thác ngữ cảnh dài hạn tốt hơn. [11]

Ngược lại, kiến trúc **Transformer** với cơ chế **self-attention** có khả năng mô hình hoá phụ thuộc dài hạn và biểu diễn ngữ cảnh tốt hơn. Trong bài toán OCR theo hướng TrOCR, mô hình thường tổ chức theo dạng encoder-decoder, cho phép sinh chuỗi ký tự một cách tự hồi quy và dễ tích hợp các kỹ thuật giải mã nâng cao (beam search, language model). Vì vậy, Transformer được kỳ vọng cải thiện đáng kể chất lượng nhận dạng, đặc biệt trong các tình huống chữ khó đọc hoặc cần ngữ cảnh để giảm nhầm lẫn.

Tuy nhiên, Transformer cũng đi kèm thách thức: **nhu cầu dữ liệu lớn** và **chi phí suy luận** cao hơn so với các kiến trúc truyền thống. Do đó, động lực quan trọng của đồ án không chỉ nằm ở việc "thay thế CRNN bằng Transformer", mà còn ở việc xây dựng một **chiến lược dữ liệu và huấn luyện** đủ mạnh để khai thác ưu thế của Transformer cho tiếng Việt, đồng thời kiểm chứng tính khả thi bằng các benchmark về hiệu năng.

1.1.3 Khoảng trống giữa nghiên cứu và triển khai thực tế

Trong thực tế, một hệ thống OCR có giá trị ứng dụng không chỉ cần độ chính xác cao trên ảnh sạch, mà còn cần: (i) hoạt động tốt trên ảnh tài liệu nhiều dòng, (ii) có pipeline end-to-end ổn định, (iii) có thời gian xử lý phù hợp để triển khai. Nhiều nghiên cứu học thuật tập trung vào mô hình nhận dạng thuần trên ảnh đã crop, trong khi bài toán triển khai cần xử lý cả detection, hậu xử lý hình học, sắp xếp thứ tự đọc, và đánh đổi tốc độ-chất lượng khi giải mã. Do đó, đồ án chọn hướng tiếp cận hệ thống: **tích hợp mô-đun phát hiện mạnh (DBNet)** và **mô hình ngôn ngữ thống kê (KenLM)** để bổ trợ cho recognition, đồng thời tập trung tối ưu mô-đun lõi Transformer bằng dữ liệu quy mô lớn và các kỹ thuật fine-tune phù hợp.

1.2 Tính ứng dụng thực tiễn

1.2.1 Bài toán thực tế mà hệ thống hướng tới

Hệ thống OCR chữ viết tay tiếng Việt trong đồ án hướng tới nhiệm vụ trích xuất văn bản từ:

- **Ảnh tài liệu (A4/đoạn văn)**: ảnh chụp hoặc scan, chứa nhiều dòng chữ viết tay.
- **Ảnh 1 dòng** (đã crop): phục vụ đánh giá recognition và tích hợp vào các hệ thống khác như nhập liệu nhanh, tìm kiếm, hoặc phân loại.

Mục tiêu thực tiễn là hỗ trợ chuyển đổi dữ liệu chữ viết tay sang văn bản số, giúp:

- **Giảm chi phí nhập liệu**: tự động hoá một phần lớn thao tác gõ lại văn bản.
- **Tăng tốc xử lý hồ sơ**: rút ngắn thời gian xử lý trong các quy trình hành chính/doanh nghiệp.
- **Tăng khả năng tra cứu và phân tích**: văn bản số hoá có thể tìm kiếm, thống kê, hoặc đưa vào các hệ thống NLP.

1.2.2 Kịch bản ứng dụng tiêu biểu

Bảng 1 minh hoạ một số kịch bản ứng dụng cụ thể của hệ thống.

Lĩnh vực	Nguồn dữ liệu chữ viết tay	Giá trị mang lại
Giáo dục	Bài làm, phiếu khảo sát, ghi chú lớp học	Chấm/soát bài, số hoá nội dung, phân tích học tập
Hành chính	Đơn từ, biểu mẫu điền tay, sổ sách	Tự động nhập liệu, tăng tốc xử lý hồ sơ, giảm sai sót
Y tế	Ghi chú bệnh án, đơn thuốc viết tay (tuỳ phạm vi)	Số hoá nội dung, hỗ trợ tra cứu và lưu trữ
Doanh nghiệp	Biên bản, ghi chú họp, nhật ký vận hành	Chuẩn hoá dữ liệu, tìm kiếm nhanh, phân tích nội dung

Bảng 1: Một số kịch bản ứng dụng của OCR chữ viết tay tiếng Việt.

1.2.3 Tính khả thi triển khai và yêu cầu hiệu năng

Tính ứng dụng của hệ thống phụ thuộc vào hai yếu tố: **chất lượng** và **hiệu năng**. Trong môi trường triển khai, việc nâng beam search hoặc tích hợp language model có thể cải thiện chất lượng nhưng làm tăng độ trễ. Vì vậy, đồ án không chỉ đánh giá CER/WER/SER mà còn thực hiện benchmark latency ở các chế độ suy luận khác nhau, bao gồm recognition-only và pipeline A4 end-to-end. Cách tiếp cận này giúp đưa ra cấu hình vận hành phù hợp theo yêu cầu thực tế (ưu tiên tốc độ hay ưu tiên chất lượng).

1.3 Mục tiêu đề tài (Problem Statement)

1.3.1 Mục tiêu tổng quát

Mục tiêu tổng quát của đồ án là **xây dựng và đánh giá một hệ thống OCR chữ viết tay tiếng Việt dựa trên Transformer**, có khả năng:

- Nhận dạng chính xác văn bản từ ảnh 1 dòng và ảnh tài liệu nhiều dòng.

- Tích hợp pipeline end-to-end gồm phát hiện vùng chữ, hậu xử lý và giải mã.
- Chứng minh tính khả thi thông qua đánh giá định lượng về **chất lượng** (CER/WER/SER) và **hiệu năng** (latency theo ảnh/dòng).

1.3.2 Mục tiêu cụ thể

Để đạt được mục tiêu tổng quát, đồ án được triển khai theo các mục tiêu cụ thể sau:

1. **Nghiên cứu cơ sở lý thuyết:**
   - Tìm hiểu OCR hiện đại và các kiến trúc recognition (CRNN/CTC và Transformer/seq2seq).
   - Nắm vững cơ chế self-attention, encoder-decoder, và các kỹ thuật giải mã (greedy, beam search, tích hợp language model).
2. **Thiết kế hệ thống và pipeline:**
   - Xây dựng pipeline xử lý ảnh tài liệu: DBNet detection → hậu xử lý hình học → crop dòng → recognition.
   - Thiết kế cơ chế sắp xếp thứ tự đọc và kiểm soát đầu vào recognition (chuẩn hoá kích thước, padding theo chiều rộng).
3. **Xây dựng mô hình nhận dạng Transformer cho tiếng Việt:**
   - Định nghĩa charset/alphabet tiếng Việt phù hợp, đảm bảo hỗ trợ dấu và các ký tự đặc biệt.
   - Huấn luyện mô hình theo chiến lược dữ liệu lớn: **pretrain** trên tập dữ liệu quy mô lớn và **fine-tune** để thích nghi với chữ viết tay.
4. **Tối ưu hoá dữ liệu và chiến lược huấn luyện:**
   - Thử nghiệm và so sánh mô hình mốc giai đoạn 1 (dữ liệu nhỏ) với mô hình sau tối ưu (dữ liệu lớn).
   - Bổ sung dữ liệu chữ viết tay do tôi tạo ra và áp dụng kỹ thuật ghép nối (stitching) để tăng đa dạng độ dài chuỗi và mẫu tự.
5. **Tích hợp giải mã nâng cao và mô hình ngôn ngữ:**
   - Xây dựng chế độ suy luận greedy (no-LM) và chế độ suy luận nâng cao beam+KenLM.
   - Khảo sát trade-off theo beam width và tham số language model để chọn cấu hình vận hành hợp lý.

6. **Đánh giá định lượng và chứng minh tính khả thi:**
    - Đánh giá chất lượng trên tập ảnh 1 dòng bằng CER/WER/SER.
    - Benchmark hiệu năng recognition-only và pipeline A4 end-to-end, phân rã theo module để xác định nút thắt.
    - So sánh tốc độ giữa các mô hình đại diện (CRNN vs Transformer giai đoạn 1 vs Transformer FINAL).

1.3.3 Phạm vi và giả định

Đồ án tập trung vào bài toán OCR chữ viết tay tiếng Việt trong bối cảnh ảnh tài liệu phổ biến. Do giới hạn về thời gian và nguồn lực gán nhãn, đồ án lựa chọn hướng triển khai theo nguyên tắc *reuse-integrate-optimize*: tái sử dụng các mô-đun mạnh có sẵn cho các thành phần phụ trợ và tập trung tối ưu mô-đun nhận dạng (recognition) là thành phần cốt lõi. Các giả định và phạm vi triển khai được xác định như sau:

- **Detection (DBNet) sử dụng mô hình có sẵn**: Pipeline detection sử dụng DBNet tải từ nguồn công khai và **không fine-tune** lại trên dữ liệu tiếng Việt/chữ viết tay trong phạm vi đồ án. Vì vậy, chất lượng cắt dòng phụ thuộc vào khả năng tổng quát hoá của mô hình pretrained; hướng fine-tune DBNet theo miền được đề xuất ở phần hướng phát triển.
- **Language Model (KenLM) sử dụng mô hình có sẵn**: KenLM dùng trong giai đoạn giải mã được lấy từ nguồn công khai và **không huấn luyện/fine-tune** lại trong phạm vi đồ án. Đồ án chỉ thực hiện **tinh chỉnh tham số suy luận** (ví dụ α, β, beam width) để phù hợp với dữ liệu và mục tiêu chất lượng-tốc độ.
- **Dữ liệu huấn luyện chủ yếu từ nguồn công khai, bổ sung một phần tự tạo**: Phần lớn dữ liệu dùng cho pre-training và fine-tuning được tổng hợp từ các nguồn dữ liệu công khai/thu thập trên mạng. Tôi chỉ tự tạo một phần nhỏ dữ liệu chữ viết tay (vài chục nghìn ảnh) nhằm tăng tính phù hợp miền dữ liệu mục tiêu và kiểm soát một số kiểu chữ/chuỗi ký tự trong quá trình fine-tune.
- **Trọng tâm tối ưu là recognition**: Thành phần nhận dạng (Transformer/TrOCR) là trọng tâm tối ưu chính của đồ án thông qua chiến lược dữ liệu lớn (pretrain/fine-tune) và fine-tune thích nghi chữ viết tay, nhằm cải thiện CER/WER/SER so với baseline.
- **Đánh giá theo hướng triển khai thực tế**: Hệ thống hướng tới khả năng vận hành trong môi trường thực tế, do đó đánh giá bao gồm cả **chất lượng** (CER/WER/SER) và **hiệu năng** (latency, phân rã thời gian theo module) thay vì chỉ báo cáo một chỉ số accuracy tổng quát.

1.4 Cấu trúc báo cáo

Báo cáo được tổ chức theo cấu trúc sau nhằm trình bày mạch lạc từ nền tảng lý thuyết đến hiện thực hệ thống và đánh giá:

- **Chương 1 - Giới thiệu:** Trình bày bối cảnh, động lực lựa chọn đề tài OCR chữ viết tay tiếng Việt, tính ứng dụng thực tiễn, mục tiêu đề tài và phạm vi triển khai, đồng thời giới thiệu cấu trúc báo cáo.
- **Chương 2 - Cơ sở lý thuyết và nền tảng kỹ thuật:** Trình bày các khái niệm và phương pháp cốt lõi liên quan đến OCR; kiến trúc CRNN/CTC và Transformer/TrOCR; cơ chế attention, encoder-decoder; kỹ thuật giải mã (greedy/beam) và mô hình ngôn ngữ (KenLM); cùng các khía cạnh xử lý dữ liệu/charset cho tiếng Việt.
- **Chương 3 - Thiết kế hệ thống và pipeline:** Mô tả kiến trúc tổng thể của hệ thống OCR, cách tổ chức các module DBNet detection, hậu xử lý (lọc, sắp xếp, crop), recognition Transformer, và các bước tích hợp để xử lý ảnh tài liệu nhiều dòng. Chương này cũng trình bày luồng dữ liệu, các quyết định thiết kế quan trọng và cách hệ thống vận hành trong thực tế.
- **Chương 4 - Huấn luyện và tối ưu mô hình:** Trình bày quy trình xây dựng dữ liệu, chiến lược pretrain/fine-tune quy mô lớn, bổ sung dữ liệu chữ viết tay do tôi tạo ra và kỹ thuật stitching, cũng như cách lựa chọn mô hình theo val loss/phiên bản checkpoint. Chương này tập trung giải thích các hành động tối ưu hoá đã thực hiện ở giai đoạn sau nhằm đạt được mô hình FINAL.
- **Chương 5 - Thực nghiệm và đánh giá:** Trình bày thiết lập thí nghiệm, tiêu chí đánh giá (CER/WER/SER) và benchmark hiệu năng. Bao gồm recognition-only trên tập ảnh 1 dòng ở nhiều chế độ (greedy và beam+KenLM), benchmark pipeline A4 end-to-end có phân rã thời gian theo module, so sánh latency giữa các mô hình chính, khảo sát trade-off theo beam width, và benchmark khi triển khai Web/API.
- **Chương 6 - Kết luận và hướng phát triển:** Tổng kết các kết quả đạt được, nêu hạn chế còn tồn tại và đề xuất các hướng phát triển như fine-tune DBNet theo tiếng Việt, tối ưu giải mã để giảm độ trễ, và nâng cấp thành phần ngôn ngữ (KenLM/LLM nhỏ) để tăng tính tự nhiên và giảm lỗi ngữ nghĩa.

CHƯƠNG 2: KIẾN THỨC NỀN TẢNG

2.1 Tổng quan bài toán OCR và phạm vi đồ án

OCR (Optical Character Recognition) là bài toán chuyển nội dung chữ trong ảnh thành chuỗi ký tự. Trong nhiều hệ thống thực tế, quá trình này thường gồm hai bước chính: *phát hiện vùng chữ* (xác định vị trí chữ trong ảnh) và *nhận dạng* (đọc nội dung chữ từ vùng đã cắt).

Trong đồ án này, phần trọng tâm là **nhận dạng chữ viết tay tiếng Việt ở mức 1 dòng** (line-level recognition), tức đầu vào ưu tiên là ảnh đã được cắt đúng một dòng chữ. Để có thể xử lý ảnh tài liệu (nhiều dòng), hệ thống vẫn tích hợp mô-đun phát hiện/cắt dòng để tạo đầu vào cho recognition. Tuy nhiên, đóng góp chính của đồ án tập vẫn trung vào mô hình nhận dạng dựa trên **Transformer (TrOCR)** đối với mức 1 dòng, cùng với các kĩ thuật huấn luyện và giải mã nhằm cải thiện chất lượng nhận diện chữ viết tay trong điều kiện ảnh đa dạng.

2.2 Bộ ký tự, chuẩn hóa nhãn và mã hóa chuỗi

2.2.1 Chuẩn hóa chuỗi nhãn (Unicode Normalization)

Trong tiếng Việt, một ký tự có dấu có thể được biểu diễn theo **nhiều cách Unicode tương đương về mặt hiển thị**, phổ biến nhất là:

- **Dạng dựng sẵn (precomposed)**: ví dụ "á" là một ký tự duy nhất (U+00E1).
- **Dạng tổ hợp (combining)**: ví dụ "a" (U+0061) đi kèm dấu sắc (U+0301), tức chuỗi hai mã `U+0061 U+0301`.

Nếu không chuẩn hoá, hai nhãn nhìn giống nhau có thể bị coi là **hai chuỗi khác nhau** ở mức byte/ký tự, dẫn đến các vấn đề:

- **Phân mảnh nhãn**: cùng một chữ nhưng bị thống kê như hai kiểu token/ký tự khác nhau.
- **Sai lệch khi tạo charset**: bộ ký tự thu được có thể dư thừa hoặc thiếu nhất quán.
- **Nhiễu khi đánh giá**: chuỗi dự đoán và ground-truth có thể khác ở biểu diễn Unicode dù giống về mặt hiển thị.

Vì vậy, toàn bộ nhãn văn bản được chuẩn hoá về **NFC (Normalization Form C)** trước khi mã hoá. NFC đưa chuỗi về dạng "chuẩn chính tắc" bằng cách ưu tiên **ghép (composition)** (nghĩa là kí tự và dấu thanh sẽ được xem như là 1 kí tự) khi có thể [13].

2.2.2 Bộ ký tự (charset) và các token đặc biệt

Bộ ký tự của mô hình được xây dựng từ file `alphabet` (danh sách ký tự hợp lệ). Sau đó, đồ án bổ sung các **token đặc biệt** để phục vụ mô hình sinh chuỗi kiểu encoder-decoder:

`[SOS]`, `[EOS]`, `[PAD]`.

Ý nghĩa và vai trò của từng token:

- `[SOS]` (*Start Of Sequence*): ký hiệu bắt đầu chuỗi, là token đầu tiên đưa vào decoder khi suy luận và khi huấn luyện.
- `[EOS]` (*End Of Sequence*): ký hiệu kết thúc chuỗi; khi decoder sinh ra `[EOS]` thì dừng giải mã.
- `[PAD]` (*Padding*): dùng để đệm các chuỗi trong cùng một mini-batch về cùng chiều dài; `[PAD]` sẽ **không** được tính vào loss (thường thông qua mask hoặc `ignore_index`).

Thiết kế này phù hợp với các mô hình OCR Transformer/TrOCR theo dạng sinh chuỗi token, trong đó chuỗi ground-truth được kết thúc bằng `[EOS]` và decoder được huấn luyện theo cơ chế dự đoán token kế tiếp [2].

2.2.3 Mã hóa và giải mã chuỗi (encoding/decoding)

Nội dung này trình bày cơ chế chuyển đổi giữa **chuỗi ký tự** (dạng văn bản) và **chuỗi chỉ số** (dạng dữ liệu số) để phục vụ huấn luyện và suy luận của mô hình OCR Transformer. Đồng thời, quy trình tạo nhãn theo *teacher forcing* và quy trình giải mã khi suy luận cũng được mô tả theo hướng trực quan và nhất quán với triển khai trong đề tài.

Khái niệm: ánh xạ ký tự sang chỉ số. Mô hình học sâu không thao tác trực tiếp trên các ký tự Unicode, mà yêu cầu mỗi ký tự được biểu diễn bằng một **chỉ số nguyên**. Do đó, một tập từ vựng ký tự $\mathcal{V}$ được xây dựng từ danh sách ký tự hợp lệ (file `alphabet`) và được mở rộng bằng các token đặc biệt `[SOS]`, `[EOS]`, `[PAD]`. Trên cơ sở đó, định nghĩa hai ánh xạ:

$$\phi : \mathcal{V} \to \{0, 1, \ldots, |\mathcal{V}| - 1\}, \qquad \phi^{-1} : \{0, 1, \ldots, |\mathcal{V}| - 1\} \to \mathcal{V},$$

trong đó $\phi(\cdot)$ gán **ID** cho mỗi ký tự/token, còn $\phi^{-1}(\cdot)$ dùng để phục hồi ký tự từ ID trong giai đoạn giải mã.

(1) Mã hoá nhãn (text → ids). Với một nhãn văn bản (ground-truth) của ảnh dòng:

$$y = (y_1, \ldots, y_T),$$

sau khi chuẩn hoá Unicode (NFC), nhãn được mã hoá bằng cách thay thế từng ký tự bởi ID tương ứng:

$$\text{ids}(y) = (\phi(y_1), \ldots, \phi(y_T)).$$

Như vậy, nhãn đầu ra của bài toán nhận dạng được biểu diễn dưới dạng một dãy số nguyên, là định dạng phù hợp để tính hàm mất mát theo phân phối xác suất (ví dụ cross-entropy).

(2) Tạo cặp chuỗi vào/ra cho decoder (teacher forcing). Decoder trong mô hình Transformer OCR hoạt động theo cơ chế sinh chuỗi tự hồi quy, tức tại bước t sẽ dự đoán token kế tiếp dựa trên các token trước đó và đặc trưng ảnh:

$$P(y_t \mid y_{<t},\ \text{ảnh}).$$

Trong huấn luyện, để ổn định tối ưu hoá và giảm lan truyền sai số do dự đoán sớm, đề tài sử dụng *teacher forcing* [14], theo đó token đúng ở bước trước được cung cấp cho decoder thay vì token dự đoán. Do vậy, từ chuỗi nhãn $y = (y_1, \ldots, y_T)$, ta tạo được hai chuỗi như sau:

$$y_{\text{in}} = [\texttt{SOS}, y_1, \ldots, y_T], \qquad y_{\text{out}} = [y_1, \ldots, y_T, \texttt{EOS}].$$

Cách xây dựng này có thể được hiểu như là một phép "dịch 1 bước". Trong đó y_{in} là chuỗi decoder **nhìn thấy** tại đầu vào còn y_{out} là chuỗi decoder **phải dự đoán** tại đầu ra. Khi đó, tại mỗi vị trí, mô hình học quan hệ:

$$\text{prefix đầu vào} \Rightarrow \text{token kế tiếp}.$$

(3) Padding và mask trong mini-batch. Trong huấn luyện theo mini-batch kích thước B, độ dài nhãn của từng mẫu có thể khác nhau:

$$T^{(1)}, T^{(2)}, \ldots, T^{(B)}.$$

Để tạo tensor đồng kích thước, các chuỗi được đệm về độ dài lớn nhất trong batch:

$$L_{\max} = \max_{i=1..B} (T^{(i)} + 1),$$

trong đó $+1$ phản ánh việc bổ sung `[SOS]` hoặc `[EOS]`. Khi đó:

$$\tilde{y}_{\text{in}}, \tilde{y}_{\text{out}} \in \{0, \ldots, |\mathcal{V}| - 1\}^{B \times L_{\max}}.$$

Token `[PAD]` được dùng cho các vị trí đệm. Đồng thời, mask được sử dụng để đảm bảo:

- Decoder không khai thác thông tin từ các vị trí `[PAD]` trong attention.

- Các vị trí [PAD] không được tính vào hàm mất mát (ví dụ thông qua ignore_index hoặc masked loss).

Nếu không áp dụng mask, mô hình có thể bị nhiễu do học dự đoán [PAD] ở các phần đệm, làm suy giảm chất lượng tổng thể.

(4) Giải mã khi suy luận (ids → text). Trong suy luận, nhãn đúng không tồn tại nên *teacher forcing* không được áp dụng. Decoder phải tự sinh chuỗi theo từng bước, bắt đầu từ token [SOS]:

1. Khởi tạo: $s_0 = [\texttt{SOS}]$.
2. Ở bước t, từ đặc trưng ảnh (encoder output) và prefix hiện tại s_{t-1}, mô hình tạo phân phối xác suất:
$$p_t = P(\cdot \mid s_{t-1},\ \text{ảnh}).$$
3. Token tiếp theo được chọn theo chiến lược giải mã:
   - **Greedy (beam=1)**: chọn token có xác suất lớn nhất.
   - **Beam search (beam=k)**: duy trì k giả thuyết tốt nhất, cộng dồn điểm theo log-xác suất; trong chế độ nâng cao có thể cộng thêm điểm từ mô hình ngôn ngữ (LM).

   Token được chọn tại bước t ký hiệu $\hat{y}_t$, sau đó cập nhật $s_t = [s_{t-1}, \hat{y}_t]$.
4. Quá trình dừng khi $\hat{y}_t = [\texttt{EOS}]$ hoặc khi đạt giới hạn max_len.

Cuối cùng, chuỗi ID dự đoán $(\hat{y}_1, \ldots, \hat{y}_{\hat{T}})$ được loại bỏ các token đặc biệt ([SOS], [EOS], [PAD]) và ánh xạ ngược:
$$\hat{y} = \big(\phi^{-1}(\hat{y}_1), \ldots, \phi^{-1}(\hat{y}_{\hat{T}})\big),$$
để thu được văn bản đầu ra.

Lưu ý khi đánh giá OCR tiếng Việt. Để đảm bảo so sánh công bằng khi tính CER/WER, văn bản dự đoán và nhãn tham chiếu thường được chuẩn hoá theo cùng quy ước: loại bỏ token đặc biệt, chuẩn hoá khoảng trắng, và (tuỳ chọn) chuẩn hoá Unicode về NFC nhằm tránh sai khác do các biểu diễn Unicode tương đương.

2.3 Tiền xử lý ảnh và biểu diễn đầu vào dạng chuỗi

Phần này mô tả quy trình chuẩn hóa ảnh đầu vào theo định dạng mà mô hình nhận dạng (Transformer OCR) yêu cầu. Mục tiêu của tiền xử lý là

- Đưa ảnh về cùng chuẩn kích thước để huấn luyện theo mini-batch.
- Giảm nhiễu và biến thiên ánh sáng/nền giấy.
- Tạo cơ chế *mask* để mô hình không bị ảnh hưởng bởi phần padding theo chiều rộng.

2.3.1 Chuẩn hóa theo chiều cao và đệm theo chiều rộng

Ảnh đầu vào trước hết được chuyển về **thang xám** (gray scale) nhằm giảm số kênh và tập trung vào cấu trúc nét chữ. Sau đó, ảnh được **resize giữ nguyên tỉ lệ** sao cho chiều cao đạt một giá trị cố định H (ở đồ án này ta cố định $H = 118$). Việc cố định chiều cao giúp:

- đảm bảo kích thước nét chữ (stroke scale) tương đối ổn định giữa các mẫu;
- tạo điều kiện để backbone (CNN/encoder) học đặc trưng theo cấu trúc dòng chữ nhất quán;
- giảm biến thiên không cần thiết về mặt hình học khi huấn luyện.

Sau bước resize, chiều rộng ảnh W vẫn thay đổi theo độ dài dòng chữ. Để ghép nhiều mẫu vào cùng một batch, ảnh được **đệm theo chiều rộng** đến một ngưỡng W_{pad} (pad width) để tạo tensor đồng kích thước. Phần đệm thường được điền bằng giá trị nền (ví dụ 0 hoặc 255 tuỳ chuẩn hoá).

2.3.2 Chuẩn hóa cường độ ảnh và (tuỳ chọn) tăng tương phản

Sau khi chuẩn hoá kích thước, ảnh được chuẩn hoá cường độ về miền chuẩn để mô hình học ổn định hơn. Cách chuẩn hoá phổ biến là đưa ảnh về miền $[0, 1]$:

$$I' = \frac{I}{255}.$$

Bước chuẩn hoá này làm cho phân phối đầu vào ổn định hơn giữa các ảnh có độ sáng khác nhau và giúp quá trình tối ưu hoá (gradient-based) hội tụ tốt hơn.

2.3.3 Làm mờ Gaussian và nhị phân hoá thích nghi (Adaptive Threshold)

Bên cạnh chuẩn hoá kích thước và chuẩn hoá cường độ, một nhóm kỹ thuật tiền xử lý thường được sử dụng trong OCR là **tăng tương phản chữ/nền** nhằm giảm ảnh hưởng của nền giấy và điều kiện chiếu sáng. Trong đề tài, hai thao tác cơ bản thuộc nhóm này gồm làm mờ Gaussian và nhị phân hoá thích nghi.

(1) Làm mờ Gaussian (Gaussian Blur). Ảnh xám có thể chứa nhiễu cao tần (hạt nhiễu camera, vết giấy, rung nhẹ), làm cho biên chữ bị "răng cưa" hoặc xuất hiện nhiều điểm nhiễu nhỏ. Do đó, ảnh thường được làm mờ nhẹ bằng Gaussian để làm trơn nhiễu trước khi nhị phân hoá:

$$I_{\text{blur}} = G_\sigma * I$$

trong đó G_σ là kernel Gaussian với độ lệch chuẩn σ, và $*$ là phép chập. Việc làm mờ vừa phải giúp giảm nhiễu nhưng vẫn giữ được cấu trúc nét chữ, từ đó làm cho bước nhị phân hoá ổn định hơn.

(2) Nhị phân hoá thích nghi (Adaptive Threshold). Khác với nhị phân hoá toàn cục (dùng một ngưỡng cố định cho toàn ảnh), nhị phân hoá thích nghi tính ngưỡng theo **từng vùng lân cận**, phù hợp với ảnh có nền không đồng đều hoặc bị bóng đổ. Với mỗi điểm ảnh (x, y), một ngưỡng cục bộ $\tau(x, y)$ được ước lượng từ thống kê vùng lân cận (ví dụ trung bình hoặc Gaussian-weighted mean), sau đó ảnh nhị phân được tạo bởi:

$$I_{\text{bin}}(x, y) = \begin{cases} 1, & I_{\text{blur}}(x, y) \geq \tau(x, y) - C, \\ 0, & \text{ngược lại}, \end{cases}$$

trong đó C là hằng số bù nhằm điều chỉnh mức phân tách chữ/nền. Trong thực hành OCR, thường sử dụng thêm chế độ **đảo màu** (inversion) để chữ nằm ở mức "nổi bật" (ví dụ chữ trắng trên nền đen hoặc ngược lại) nhằm thuận lợi cho các bước chuẩn hoá và trích đặc trưng.

(3) Chuẩn hoá đầu vào sau nhị phân hoá. Sau khi nhị phân hoá, ảnh vẫn được đưa về miền chuẩn (ví dụ $[0, 1]$):

$$I' = \frac{I_{\text{bin}}}{255},$$

hoặc giữ ở dạng nhị phân $\{0, 1\}$ tuỳ theo quy ước của pipeline. Mục tiêu là đảm bảo mô hình nhận đầu vào có thang giá trị nhất quán.

Làm mờ Gaussian và nhị phân hoá thích nghi thường giúp tăng độ tương phản chữ/nền và giảm tác động của nền giấy, đặc biệt trong ảnh chụp tài liệu. Tuy nhiên, hiệu quả của bước này phụ thuộc mạnh vào dữ liệu (độ đậm nét, loại giấy, mức nhiễu) và cấu hình tham số (kích thước vùng lân cận, C, mức blur). Vì vậy, trong triển khai thực tế, bước này nên được xem như một lựa chọn tiền xử lý có thể bật/tắt hoặc tinh chỉnh theo miền dữ liệu.

2.3.4 Biểu diễn đầu vào dạng chuỗi và mặt nạ padding theo chiều rộng

Trong bài toán nhận dạng theo mức *một dòng* (line-level), ảnh sau khi được resize/pad sẽ được biến đổi thành biểu diễn phù hợp để Transformer xử lý. Cụ thể, ảnh dòng chữ được xem như một tín hiệu trải dài theo trục ngang, do đó sau khi qua encoder (CNN/backbone), đặc trưng được tổ chức thành một **chuỗi theo chiều rộng**.

Giả sử ảnh đầu vào có chiều rộng sau pad là W_{pad}. Sau backbone CNN, chiều rộng được giảm theo một hệ số tổng hợp s (phụ thuộc cấu hình stride/pooling của backbone). Khi đó, độ dài chuỗi đặc trưng xấp xỉ:

$$T \approx \left\lfloor \frac{W_{\text{pad}}}{s} \right\rfloor.$$

Trong đó, mỗi phần tử của chuỗi tương ứng với một "cột đặc trưng" (feature column) biểu diễn một vùng không gian hẹp theo trục ngang của ảnh.

Do các mẫu trong batch có chiều rộng thực (không tính padding) khác nhau, phần padding theo chiều rộng sẽ sinh ra các phần tử chuỗi không mang thông tin (chuỗi thuộc vùng padding

nên nó rỗng) trong chuỗi đặc trưng. Để tránh việc Transformer encoder học từ các vùng padding này, **source key padding mask** được xây dựng như sau:

- xác định chiều rộng thực W_{orig} sau bước resize (trước khi pad);
- quy đổi sang độ dài chuỗi thực $T_{\text{orig}} \approx \left\lfloor \frac{W_{\text{orig}}}{s} \right\rfloor$;
- đánh dấu các vị trí $t > T_{\text{orig}}$ là padding để encoder bỏ qua trong attention.

Cơ chế mask theo chiều rộng là cần thiết để đảm bảo mô hình chỉ khai thác vùng có thông tin chữ thật, đồng thời giúp kết quả huấn luyện và suy luận ổn định hơn khi độ dài dòng chữ biến thiên mạnh giữa các mẫu.

2.4 Mô hình baseline CRNN và CTC

Phần này trình bày mô hình baseline CRNN và hàm mất mát CTC, là một hướng tiếp cận kinh điển cho OCR chuỗi. Trong đồ án, CRNN được sử dụng như mốc tham chiếu nhằm so sánh mức cải thiện về chất lượng và hiệu năng khi chuyển sang kiến trúc Transformer.

2.4.1 Kiến trúc CRNN (Convolutional Recurrent Neural Network)

CRNN là kiến trúc kết hợp giữa mạng tích chập và mạng hồi quy, được thiết kế để nhận dạng chuỗi ký tự từ ảnh có độ dài biến thiên (thường là ảnh một dòng chữ) [4]. Về tổng thể, CRNN gồm ba khối chính:

1. **Khối trích xuất đặc trưng không gian (CNN backbone).** Ảnh đầu vào (thường đã được chuẩn hoá về chiều cao) được đưa qua các lớp tích chập và pooling để trích xuất đặc trưng. Mục tiêu của khối CNN là biến đổi ảnh từ không gian pixel thành **bản đồ đặc trưng** (feature map) giàu thông tin và ít nhạy hơn với nhiễu/biến dạng nhỏ.

2. **Biến đổi feature map thành chuỗi đặc trưng theo trục ngang.** Do văn bản trên ảnh một dòng chủ yếu thay đổi theo chiều ngang, feature map sau CNN thường được **nén theo chiều cao** để thu được một dãy vector đặc trưng theo chiều rộng:

$$\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_T\},$$

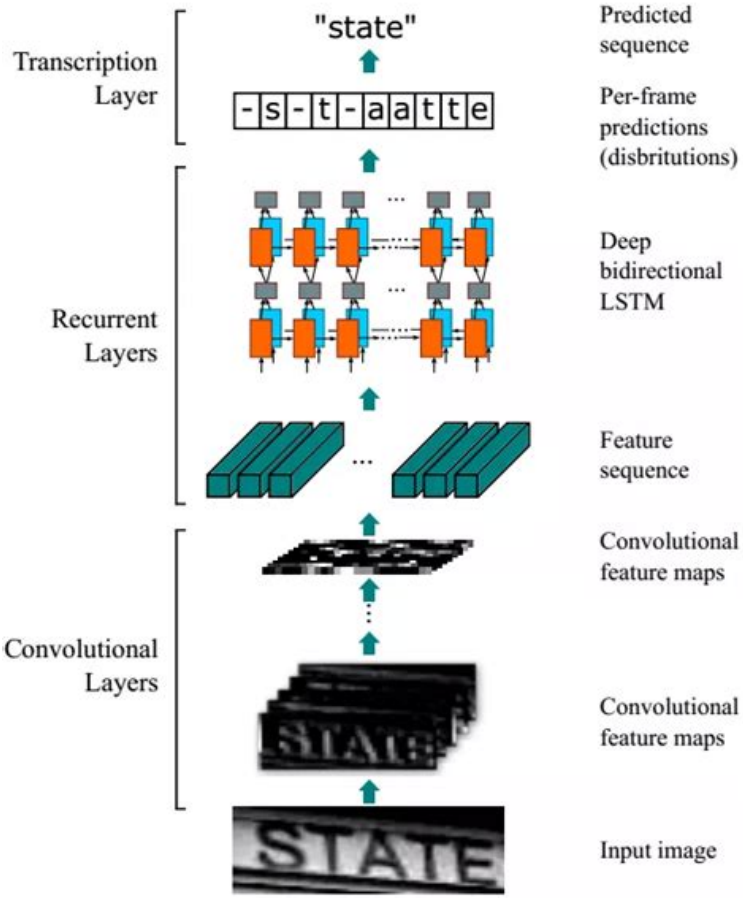
   trong đó T xấp xỉ tỉ lệ với chiều rộng ảnh sau khi qua CNN, và mỗi $\mathbf{f}_t$ biểu diễn thông tin tại một "cột đặc trưng" theo vị trí trái→phải [4].

3. **Khối mô hình hoá ngữ cảnh theo chuỗi (BiRNN: BiLSTM/GRU).** Chuỗi đặc trưng $\mathbf{F}$ được đưa qua một hoặc nhiều lớp RNN hai chiều (BiLSTM hoặc BiGRU) để khai thác phụ thuộc theo trục ngang. RNN hai chiều cho phép mỗi vị trí sử dụng đồng thời thông tin từ cả hai hướng (trái và phải), giúp tăng khả năng phân biệt ký tự trong điều kiện chữ dính nét hoặc nhiễu [4].

4. **Tầng phân loại theo từng bước (frame-wise classifier).** Tại mỗi bước t, một lớp tuyến tính ánh xạ trạng thái RNN thành phân phối xác suất trên tập lớp ký tự, bao gồm cả nhãn `blank` của CTC:

$$\mathbf{p}_t = \text{softmax}(\mathbf{W}\mathbf{h}_t + \mathbf{b}).$$

Đầu ra của CRNN là dãy phân phối $\{\mathbf{p}_1, \ldots, \mathbf{p}_T\}$ để phục vụ CTC transcription [4, 5].



Hình 1: Minh hoạ kiến trúc CRNN: CNN trích xuất đặc trưng → BiRNN mô hình hoá chuỗi → CTC transcription.

2.4.2 Hàm mất mát CTC (Connectionist Temporal Classification)

Một khó khăn cơ bản trong OCR là **không có căn chỉnh tường minh** giữa từng ký tự trong nhãn và từng vị trí trên chuỗi đặc trưng. CTC giải quyết vấn đề này bằng cách xét mọi chuỗi căn chỉnh $\pi = (\pi_1, \ldots, \pi_T)$ có thể, sau đó dùng một phép ánh xạ $\mathcal{B}$ để thu được chuỗi nhãn cuối bằng cách:

- Gộp các ký tự lặp liên tiếp
- Loại bỏ ký tự rỗng `blank` [5, 11].

Với nhãn mục tiêu y, tập các căn chỉnh hợp lệ là $\mathcal{B}^{-1}(y)$.

Xác suất của nhãn y cho ảnh x được tính bằng cách cộng xác suất của tất cả căn chỉnh hợp lệ:

$$p(y|x) = \sum_{\pi \in \mathcal{B}^{-1}(y)} p(\pi|x) = \sum_{\pi \in \mathcal{B}^{-1}(y)} \prod_{t=1}^{T} p(\pi_t|x),$$

trong đó $p(\pi_t|x)$ được lấy từ phân phối $\mathbf{p}_t$ tại bước t [5]. Trên thực tế, CTC loss được tính hiệu quả bằng quy hoạch động (forward–backward), cho phép huấn luyện end-to-end mà không cần gán nhãn vị trí từng ký tự [5, 11].

Giải mã CTC (CTC decoding). Trong suy luận, cách đơn giản nhất là **greedy decoding**: tại mỗi bước t chọn lớp có xác suất lớn nhất, sau đó áp dụng $\mathcal{B}$ để gộp lặp và bỏ `blank` [5, 11]. Ngoài ra, beam search CTC và kết hợp mô hình ngôn ngữ có thể được sử dụng, tuy nhiên việc tích hợp ràng buộc ngôn ngữ thường kém linh hoạt hơn so với cơ chế giải mã tự hồi quy trong mô hình Transformer OCR [2].

Nhận xét về giới hạn của CTC. Mặc dù CTC xử lý tốt chuỗi đặc trưng theo chiều ngang, phương pháp này gặp hạn chế do cơ chế dự đoán độc lập tại từng vị trí, khiến việc tận dụng ngữ cảnh dài hạn để sửa lỗi gặp khó khăn. Ngoài ra, bước căn chỉnh (alignment) của CTC kém ổn định khi xử lý chữ viết dính hoặc đứt nét. Do đó, khả năng tích hợp tri thức ngôn ngữ của CTC thường kém linh hoạt hơn so với các mô hình tự hồi quy như Transformer [12, 1].

2.4.3 Vai trò baseline trong đồ án

Trong phạm vi đồ án, CRNN được lựa chọn làm baseline chủ yếu vì đề tài được triển khai theo hướng **kế thừa và cải thiện** một mã nguồn mở OCR chữ viết tay tiếng Việt, trong đó mô-đun nhận dạng ban đầu được xây dựng dựa trên CRNN [8]. Do đó, việc giữ CRNN làm baseline giúp:

- **So sánh trước–sau một cách nhất quán**: đánh giá trực tiếp mức cải thiện khi thay thế mô-đun recognition từ CRNN sang Transformer trong cùng pipeline và cùng quy trình tiền/hậu xử lý.
- **Đảm bảo tính công bằng của thực nghiệm**: các khác biệt về dữ liệu, tiền xử lý và cách đánh giá được kiểm soát, từ đó kết luận tập trung vào tác động của kiến trúc recognition.
- **Cung cấp mốc tham chiếu thực dụng**: CRNN là cấu hình nền đã được cộng đồng sử dụng, có tốc độ suy luận tốt và phù hợp làm mốc đối chiếu về chất lượng (CER/WER/SER) và hiệu năng so với mô hình Transformer.

2.5 Kiến trúc Transformer OCR theo encoder-decoder

2.5.1 Encoder CNN: nén chiều cao, tạo chuỗi theo chiều rộng

Mục tiêu encoder CNN là chuyển ảnh $H \times W$ thành feature map có chiều cao nhỏ (thực nghiệm là $h' = 3$) và chiều rộng giảm theo hệ số 8 [4, 2]. Khi đó, feature map có thể được **trải phẳng theo cột** để tạo chuỗi:

$$\mathbf{F} \in \mathbb{R}^{B\times C\times h'\times T} \;\Rightarrow\; \mathbf{X} \in \mathbb{R}^{B\times T\times (C\cdot h')}.$$

Sau đó, một lớp tuyến tính chiếu $\mathbf{X}$ về không gian mô hình d_{model} để đưa vào Transformer encoder:

$$\mathbf{Z} = \mathbf{X}\mathbf{W}_p + \mathbf{b}_p, \qquad \mathbf{Z} \in \mathbb{R}^{B\times T\times d_{\text{model}}}.$$

2.5.2 Positional encoding

Do self-attention bất biến theo hoán vị, cần cung cấp thông tin vị trí. Với positional encoding dạng sin-cos [1], tại vị trí pos và chiều i:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right).$$

Đầu vào encoder là $\mathbf{Z} + PE$.

2.5.3 Self-attention và multi-head attention

Với truy vấn Q, khóa K và giá trị V, scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

Multi-head attention (MHA) chia không gian thành h đầu:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \quad \text{head}_j = \text{Attention}(QW_j^Q, KW_j^K, VW_j^V).$$

[1]

2.5.4 Decoder tự hồi quy và teacher forcing

Decoder Transformer sinh chuỗi ký tự theo cơ chế tự hồi quy:

$$p(y|x) = \prod_{t=1}^{T} p(y_t|y_{<t}, x).$$

Trong huấn luyện, dùng teacher forcing: tại bước t decoder nhận $y_{<t}$ từ nhãn thật [1, 2]. Hàm mất mát là cross-entropy theo từng vị trí, bỏ qua token `[PAD]`:

$$\mathcal{L} = -\sum_{t} \log p(y_t^* \mid y_{<t}^*, x).$$

Ngoài ra, decoder cần *subsequent mask* (mặt nạ tam giác) để chặn truy cập thông tin tương lai, và *padding mask* để bỏ qua vùng đệm.

2.6 Giải mã (decoding), tích hợp Language Model và tiêu chí đánh giá

2.6.1 Greedy decoding và beam search

Ở suy luận, greedy decoding chọn ký tự xác suất lớn nhất tại mỗi bước nhưng dễ mắc lỗi cục bộ. Beam search duy trì K giả thuyết tốt nhất, mở rộng theo top-K xác suất tại mỗi bước. Điểm số thường dựa trên tổng log-probability, kèm chuẩn hóa độ dài (length normalization) nhằm tránh thiên lệch chuỗi ngắn [15]:

$$\text{score}(\mathbf{y}) = \frac{\log P(\mathbf{y}|x)}{|\mathbf{y}|^{\alpha}},$$

trong đó α là hệ số length penalty. Ngoài ra, có thể dùng *repetition penalty* để hạn chế lặp ký tự bất thường trong OCR.

2.6.2 Tích hợp mô hình ngôn ngữ KenLM

Để tận dụng tri thức ngôn ngữ (đặc biệt hữu ích cho tiếng Việt và chữ viết tay), giải mã có thể kết hợp điểm từ OCR và điểm từ Language Model (LM). Với KenLM (n-gram), điểm LM được cộng khi hoàn tất một từ (khi gặp khoảng trắng hoặc kết thúc chuỗi) [6]. Một dạng kết hợp điểm điển hình:

$$\text{score}_{\text{total}} = \text{score}_{\text{ocr}} + \lambda \cdot \text{score}_{\text{lm}} + \beta \cdot |\mathbf{y}|,$$

trong đó λ điều chỉnh mức ảnh hưởng của LM, và β đóng vai trò bù/penalty theo độ dài (tùy thiết kế). Cách kết hợp này giúp giảm lỗi chính tả, cải thiện phân tách từ và tăng tính hợp ngữ nghĩa của kết quả.

2.6.3 Tiêu chí đánh giá CER, WER, SER

Chất lượng OCR được đo bằng các chỉ số chuẩn [16]:

- **CER (Character Error Rate):** khoảng cách Levenshtein trên cấp ký tự, chuẩn hóa theo độ dài nhãn thật:

$$\text{CER} = \frac{\text{Lev}(\text{chars}(\hat{y}), \text{chars}(y))}{|y|}.$$

- **WER (Word Error Rate):** khoảng cách Levenshtein trên cấp từ (tách theo khoảng trắng):

$$\mathrm{WER} = \frac{\mathrm{Lev}(\mathrm{words}(\hat{y}), \mathrm{words}(y))}{|\mathrm{words}(y)|}.$$

- **SER (Sentence Error Rate):** tỷ lệ mẫu có dự đoán không khớp hoàn toàn:

$$\mathrm{SER} = \begin{cases} 0, & \hat{y} = y, \\ 1, & \hat{y} \neq y. \end{cases}$$

Trong thực nghiệm, CER/WER thường được tính không phân biệt hoa thường (chuyển về lowercase) để phản ánh sai số nội dung thay vì kiểu chữ.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1 Mục tiêu và yêu cầu hệ thống

Mục tiêu của đồ án là xây dựng một hệ thống OCR tiếng Việt dựa trên Transformer có khả năng:

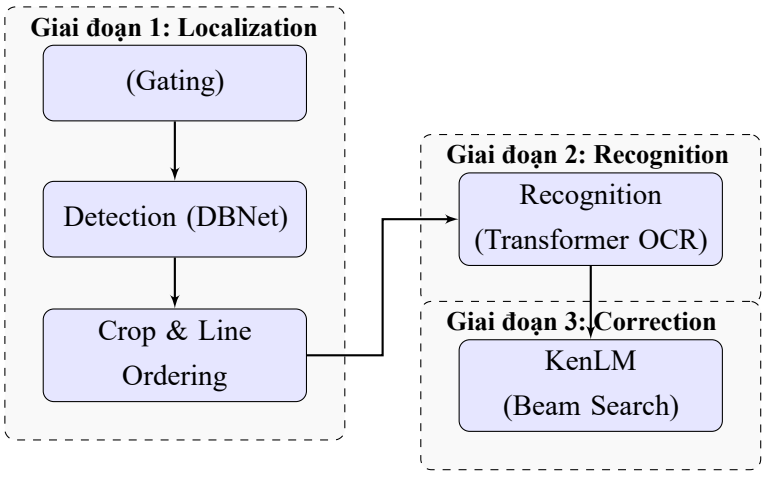
- Nhận dạng chính xác văn bản **theo cấp dòng** (line-level OCR), ưu tiên ảnh chữ viết tay.
- Xử lý được cả hai loại ảnh đầu vào:
  1. Ảnh **đã cắt sẵn** (pre-cropped) chỉ chứa một dòng văn bản.
  2. Ảnh **tài liệu/ảnh A4** chứa nhiều dòng, cần phát hiện vị trí dòng trước khi nhận dạng.
- Cho phép **hậu xử lý ngôn ngữ** nhằm giảm lỗi dấu tiếng Việt, lỗi tách từ và lỗi dấu câu.

Yêu cầu phi chức năng:

- **Tính nhất quán** giữa báo cáo và triển khai: mọi tham số/heuristic quan trọng đều mô tả đúng theo code thực thi.
- **Tính tái lập**: pipeline cung cấp CLI rõ ràng, ghi log và lưu kết quả ra file.
- **Tính thực dụng**: hỗ trợ chạy trên GPU (cuda) để suy luận nhanh; vẫn có thể chuyển CPU khi cần triển khai.

3.2 Kiến trúc tổng quan hệ thống

Hệ thống được thiết kế theo pipeline ba giai đoạn:



Về mặt triển khai, mô hình nhận dạng (Transformer OCR) được đóng gói trong một checkpoint bao gồm:

- `charset`: danh sách ký tự và token đặc biệt `[SOS]`, `[EOS]`, `[PAD]`.

- `height` và `pad_w`: tham số tiền xử lý ảnh dòng (`IMG_H`, `PAD_W`).
- `model_state`: trọng số mô hình.

Cách đóng gói này giúp pipeline suy luận tự động lấy đúng cấu hình ảnh dòng từ checkpoint, giảm rủi ro lệch cấu hình giữa train và infer.

3.3 Thiết kế và phân bố dữ liệu

Dựa trên chiến lược huấn luyện hai giai đoạn (Pretrain và Finetune), dữ liệu được tổ chức thành hai tập riêng biệt nhằm đảm bảo mô hình vừa học được đặc trưng tổng quát, vừa thích nghi tốt với miền chữ viết tay mục tiêu.

3.3.1 Nguồn dữ liệu và tiền xử lý

Dữ liệu huấn luyện được tổng hợp và chuẩn hoá từ các nguồn công khai uy tín trong cộng đồng nghiên cứu OCR tiếng Việt, kết hợp với dữ liệu tự thu thập. Cụ thể:

- **Nguồn dữ liệu gốc (Open Source):** Bao gồm các bộ dataset từ *TomHuynhSG* (Vietnamese Handwriting Recognition OCR) [8], *UIT-HWDB* (Đại học CNTT - ĐHQG TP.HCM) [7], và dự án *VietOCR* (pbcquoc) [9]. Đây là các nguồn cung cấp lượng lớn ảnh chữ viết tay và chữ in đa dạng, tạo nền tảng vững chắc cho giai đoạn Pretrain.
- **Dữ liệu tự tạo (Self-collected):** Để khắc phục sự thiếu hụt các mẫu ký tự hiếm và dấu câu trong các bộ dữ liệu mở, tôi đã tự viết tay và thu thập thêm khoảng 37.000 mẫu ảnh dòng bằng cách stitch các từ đơn, tập trung vào các trường hợp mô hình thường nhận diện sai.
- **Kỹ thuật Stitching:** Từ các ký tự/từ đơn lẻ, tôi áp dụng thuật toán ghép (stitching) để tạo ra các dòng văn bản dài ngẫu nhiên, giúp mô hình học được ngữ cảnh và sự biến thiên khoảng cách giữa các chữ.

Dữ liệu sau khi thu thập được tiền xử lý thống nhất:

- **Resize:** Chiều cao cố định $H = 118$ pixel, chiều rộng giữ tỷ lệ.
- **Padding:** Đệm về chiều rộng tối đa $W = 3500$ pixel (sử dụng màu trắng hoặc median padding).
- **Augmentation:** Áp dụng xoay ($\pm 3°$), chỉnh độ sáng/tương phản $[0.8, 1.2]$ và làm mờ Gaussian để tăng tính bền vững của mô hình.

3.3.2 Phân chia tập dữ liệu (Train/Validation)

Tỷ lệ phân chia được cấu hình khác nhau cho từng giai đoạn để tối ưu hóa lượng dữ liệu học:

Bảng 2: Thống kê phân bố dữ liệu cho các giai đoạn huấn luyện

Giai đoạn	Tỷ lệ Train:Val	Mục tiêu	File cấu hình
1. Pretrain	99.9% : 0.1%	Học biểu diễn đặc trưng tổng quát	`config_FINAL.yaml`
2. Finetune	95% : 5%	Tinh chỉnh trên miền chữ viết tay	`config_FINETUNE_FINAL.yaml`

Bảng 3: Thống kê dữ liệu theo các giai đoạn huấn luyện và đánh giá

Giai đoạn	Nguồn dữ liệu	Quy mô	Mục tiêu
GĐ1 (khởi tạo)	Tập ảnh dòng tiếng Việt ban đầu (baseline)	∼9k	Thiết lập mốc ban đầu và checkpoint khởi tạo
Pre-train (FINAL)	Dữ liệu quy mô lớn (tổng hợp + đa miền)	∼1M	Học biểu diễn thị giác tổng quát, khởi tạo trọng số tốt
Fine-tune (FINAL)	Dữ liệu cùng miền quy mô lớn (mục tiêu)	>1M	Thích nghi miền và tối ưu chất lượng recognition trên dữ liệu mục tiêu
Fine-tune HTR-1	Tập baseline 9k + 6k chữ viết tay tự tạo (stitch từ ký tự/từ)	∼15k (9k+6k)	Tăng phù hợp chữ viết tay, học nét bút và dấu tiếng Việt tốt hơn
Fine-tune HTR-2	Tập 15k + 8k ảnh stitch bổ sung	∼23k (15k+8k)	Mở rộng đa dạng mẫu chữ/độ dài dòng, giảm lỗi nhầm ký tự và dấu
Fine-tune HTR-3 (FINAL)	Tập chữ viết tay mở rộng (phiên bản cuối)	∼26k	Tinh chỉnh cuối cho HTR, tối ưu tổng thể trên chữ viết tay
Eval (line-level)	Tập ảnh dòng tách riêng để đánh giá	∼1.2k	Đo CER/WER/SER cho recognition

3.4 Cấu hình và chiến lược huấn luyện

Quá trình huấn luyện được thực hiện trên thư viện PyTorch với các tối ưu hóa phần cứng (Flash Attention, Mixed Precision) để tận dụng hiệu năng GPU.

3.4.1 Hàm mất mát và Tối ưu hóa

- **Loss Function:** Sử dụng `CrossEntropyLoss` với cơ chế `ignore_index` cho token `[PAD]`, giúp mô hình không bị nhiễu bởi phần đệm.
- **Optimizer:** Sử dụng **AdamW** với cờ `fused=True` để tăng tốc độ cập nhật trọng số.
- **Mixed Precision:** Huấn luyện dưới định dạng `BFloat16` (AMP) giúp giảm bộ nhớ VRAM và tăng tốc độ tính toán mà không làm giảm đáng kể độ chính xác hội tụ.

3.4.2 Siêu tham số huấn luyện (Hyperparameters)

Các tham số được tinh chỉnh riêng cho từng giai đoạn dựa trên thực nghiệm, cụ thể như sau:

Bảng 4: Cấu hình siêu tham số chi tiết

Tham số	**Giai đoạn Pretrain**	**Giai đoạn Finetune**
Learning Rate (LR)	3×10^{-4}	5×10^{-6}
Batch Size	17	8
Validation Batch Size	11	12
Số Epochs	120	120
LR Scheduler	ReduceLROnPlateau (Factor=0.5, Patience=3)	

Nhận xét cấu hình:

- Giai đoạn Pretrain sử dụng Learning Rate lớn ($3e^{-4}$) và Batch Size lớn hơn (17) để mô hình hội tụ nhanh trên tập dữ liệu lớn.
- Giai đoạn Finetune giảm Learning Rate xuống rất thấp ($5e^{-6}$) và Batch Size nhỏ (8) để tránh hiện tượng "catastrophic forgetting" (quên tri thức cũ) và giúp mô hình hội tụ vào đặc điểm của chữ viết tay tiếng Việt.
- Chiến lược `ReduceLROnPlateau` tự động giảm LR khi Loss trên tập Validation không cải thiện sau 3 epoch, giúp mô hình vượt qua các điểm cực trị địa phương (local minima).

3.5 Gating: phân loại ảnh đầu vào (pre-cropped vs. ảnh tài liệu)

Một điểm nhấn thiết kế của hệ thống là cơ chế **gating** nhằm tránh chạy Detection khi ảnh đã là dòng đơn (để tăng tốc), đồng thời tránh nhận nhầm ảnh A4 thành dòng (để không bỏ sót nhiều dòng chữ).

Gating sử dụng **hai tiêu chí**:

3.5.1 Tiêu chí tỷ lệ khung hình

Gọi W, H là chiều rộng và chiều cao ảnh đầu vào. Tỷ lệ:

$$r = \frac{W}{H}.$$

Hệ thống chỉ xem ảnh là pre-cropped nếu r đủ lớn (ảnh "dẹt"), với ngưỡng:

$$r \geq r_{\text{th}}, \quad r_{\text{th}} = 5.0.$$

Nếu $r < 5.0$, ảnh bị kết luận là *không phải dòng đơn* và bắt buộc chạy Detection.

3.5.2 Tiêu chí mật độ điểm ảnh chữ (pixel density)

Sau khi ảnh vượt qua tiêu chí khung hình, hệ thống tiếp tục kiểm tra mật độ pixel chữ bằng cách:

- chuyển grayscale,
- làm mờ Gaussian,
- nhị phân hóa thích nghi (adaptive threshold, kiểu `THRESH_BINARY_INV`),
- đếm số pixel foreground (pixel chữ).

Gọi ρ là phần trăm pixel chữ:

$$\rho = \frac{N_{\text{fg}}}{W \cdot H} \times 100.$$

Ngưỡng sử dụng:

$$\rho > \rho_{\text{th}}, \quad \rho_{\text{th}} = 1.93.$$

Nếu ρ đủ lớn, ảnh được kết luận là pre-cropped (dòng đơn); ngược lại chạy Detection.

Algorithm 1 Gating xác định ảnh pre-cropped (dòng đơn)

Require: Ảnh đầu vào I, ngưỡng $r_{\text{th}} = 5.0$, $\rho_{\text{th}} = 1.93$
Ensure: Quyết định `is_pre_cropped` (True/False)

$W, H \leftarrow$ kích thước ảnh I
$r \leftarrow W/H$
if $r < r_{\text{th}}$ **then** ▷ Ảnh không đủ "dẹt"
    return False
end if
$I_g \leftarrow \text{grayscale}(I)$
$I_b \leftarrow \text{GaussianBlur}(I_g)$
$I_t \leftarrow \text{AdaptiveThreshold}(I_b, \texttt{BINARY\_INV})$
$\rho \leftarrow \frac{\#\{I_t \neq 0\}}{W \cdot H} \times 100$
return $(\rho > \rho_{\text{th}})$

Ý nghĩa thực nghiệm: tiêu chí $r_{\text{th}} = 5.0$ giúp tránh việc ảnh A4 hoặc ảnh đoạn văn nhiều dòng nhưng có mật độ chữ cao bị nhầm là ảnh dòng. Tiêu chí $\rho_{\text{th}} = 1.93$ giúp phân biệt ảnh dòng (mật độ chữ thường cao) với ảnh tài liệu (nhiều khoảng trắng, lề, nền).

3.6 Giai đoạn 1: Detection bằng DBNet và thiết kế cắt dòng

3.6.1 Mô-đun Detection

Hệ thống sử dụng DBNet ở dạng ONNX thông qua OpenCV DNN [17, 3]. Ảnh đầu vào được chuẩn hóa theo cấu hình:

$$\texttt{scale} = 1/255, \quad \texttt{input\_size} = (736, 736), \quad \texttt{mean} = (122.6789, 116.6687, 104.0069).$$

DBNet xuất ra một danh sách các rotated rectangles (mỗi box gồm tâm, kích thước và góc quay).

3.6.2 Quy đổi tọa độ box từ ảnh DBNet về ảnh gốc

Do DBNet chạy trên ảnh đã resize/pad, hệ thống cần đưa box về hệ tọa độ ảnh gốc. Gọi:

$$s_w = \frac{W_{\text{orig}}}{W_{\text{db}}}, \quad s_h = \frac{H_{\text{orig}}}{H_{\text{db}}}.$$

Với box ở hệ DBNet $((c_x, c_y), (w, h), \theta)$, box tương ứng trên ảnh gốc:

$$c'_x = c_x \cdot s_w, \quad c'_y = c_y \cdot s_h, \quad w' = w \cdot s_w, \quad h' = h \cdot s_h, \quad \theta' = \theta.$$

3.6.3 Lọc box bằng Shapely (V4: area + center + overlap)

Detection thực tế thường trả về box dư (rác) hoặc nhiều box chồng lên nhau. Để tăng ổn định, hệ thống áp dụng lọc theo ba bước:

(1) Lọc theo diện tích tương đối. Gọi A_i là diện tích box thứ i, $A_{\max} = \max_i A_i$. Nếu:

$$A_i < 0.05 \cdot A_{\max}$$

thì box bị loại (rất nhỏ so với box lớn nhất, thường là nhiễu).

(2) Lọc theo điều kiện "tâm nằm trong box đã giữ". Với mỗi box ứng viên, nếu tâm (c_x, c_y) của nó nằm trong một box đã giữ, box ứng viên bị loại (hàm ý box này là box con hoặc box rác nằm trong vùng chữ đã được bao phủ).

(3) Lọc theo overlap ratio (Shapely intersection). Dựng đa giác cho mỗi rotated rectangle. Gọi $A(i \cap j)$ là diện tích giao nhau. Hệ thống dùng tỉ lệ:

$$\text{overlap\_ratio}(i, j) = \frac{A(i \cap j)}{A(i)}.$$

Nếu $\text{overlap\_ratio}(i, j) > 0.4$ với một box đã giữ j, box i bị loại vì bị che phủ quá lớn.

3.6.4 Sắp xếp box theo thứ tự dòng

Sau lọc, các box được sắp xếp theo y của tâm box tăng dần để khôi phục thứ tự đọc từ trên xuống dưới. Bước này giúp ghép lại đoạn văn đúng thứ tự khi đầu vào là ảnh A4 nhiều dòng.

3.6.5 Mở rộng rotated rectangle để tránh cắt dấu tiếng Việt

Một lỗi thường gặp của OCR tiếng Việt là dấu bị cắt (dấu mũ, dấu sắc/huyền/hỏi/ngã/nặng) khi box quá sát. Để giảm lỗi này, hệ thống mở rộng box theo hai hệ số:

$$\gamma_w = 1.15, \qquad \gamma_h = 1.10.$$

Với box sau quy đổi về ảnh gốc có kích thước (w', h'), kích thước mới:

$$w'' = \gamma_w \cdot w', \qquad h'' = \gamma_h \cdot h'.$$

Lưu ý quan trọng: hệ thống **giữ nguyên tâm** (c'_x, c'_y) và chỉ tăng kích thước, nhằm tránh tình trạng dịch lệch box lên/xuống làm mất nội dung.

3.6.6 Cắt dòng bằng biến đổi phối cảnh

Box quay được chuyển về 4 đỉnh thông qua `cv2.boxPoints`. Sau đó, dùng `warpPerspective` để biến đổi phối cảnh, thu về ảnh dòng đã "làm thẳng". Kết quả crop được đưa sang Recognition.

Algorithm 2 Detection → lọc box → crop dòng

Require: Ảnh I, DBNet, $\gamma_w = 1.15, \gamma_h = 1.10$, ngưỡng lọc $0.05, 0.4$
Ensure: Danh sách ảnh dòng $\{I_k\}$ theo thứ tự đọc

$\mathcal{B} \leftarrow \text{DBNet}(I)$
$\mathcal{B}' \leftarrow$ lọc theo area (5% $A_{\max}$)
$\mathcal{B}'' \leftarrow$ lọc theo "tâm nằm trong box đã giữ"
$\mathcal{B}''' \leftarrow$ lọc overlap ratio > 0.4
Sắp xếp $\mathcal{B}'''$ theo y tâm tăng dần
for all box $b = ((c_x, c_y), (w, h), \theta)$ trong $\mathcal{B}'''$ **do**
    Quy đổi box về ảnh gốc bằng s_w, s_h
    $w \leftarrow 1.15 \cdot w, \quad h \leftarrow 1.10 \cdot h$
    Dựng 4 đỉnh và crop bằng `warpPerspective` $\Rightarrow I_k$
end for
return $\{I_k\}$

3.7 Giai đoạn 2: Recognition bằng Transformer OCR (CNN + Encoder-Decoder)

3.7.1 Đầu vào, charset và token đặc biệt

Mô hình recognition hoạt động theo cấp **ký tự** dựa trên `charset` trong checkpoint. Charset luôn bao gồm ba token đặc biệt:

$$\texttt{[SOS]},\ \texttt{[EOS]},\ \texttt{[PAD]}.$$

Trong suy luận, pipeline đọc `charset` từ checkpoint để dựng ánh xạ:

$$\text{char\_to\_idx} : c \mapsto i, \qquad \text{idx\_to\_char} : i \mapsto c.$$

Qua đó, pipeline không phụ thuộc vào file charset bên ngoài và tránh lệch cấu hình giữa các phiên bản mô hình.

3.7.2 Tiền xử lý ảnh dòng theo đúng checkpoint (`IMG_H`, `PAD_W`)

Mỗi ảnh dòng I_k (sau crop) được tiền xử lý theo tham số lấy trực tiếp từ checkpoint:

$$\texttt{IMG\_H} = \texttt{ckpt['height']}, \qquad \texttt{PAD\_W} = \texttt{ckpt['pad\_w']}.$$

Quy trình:

1. **Grayscale**: nếu ảnh có 3 kênh thì chuyển sang ảnh xám.
2. **Resize giữ tỉ lệ theo chiều cao** về `IMG_H`. Gọi W_{resized} là chiều rộng sau resize.

3. **Pad/crop về `PAD_W`**:
    - Nếu $W_{\text{resized}} < \texttt{PAD\_W}$: pad bằng thống kê median (giảm nhiễu nền so với pad 0).
    - Nếu $W_{\text{resized}} > \texttt{PAD\_W}$: resize về đúng $(\texttt{PAD\_W}, \texttt{IMG\_H})$ để tránh tensor vượt kích thước.
4. **Gaussian + Adaptive Threshold** (binary inverse) nhằm làm nổi nét chữ.
5. **Chuẩn hóa** về $[0, 1]$ bằng chia 255.0.
6. Đưa về tensor kích thước $[1, 1, \texttt{IMG\_H}, \texttt{PAD\_W}]$.

3.7.3 CNN backbone và hệ số giảm chiều rộng $s = 8$

CNN backbone gồm chuỗi khối Conv-BN-ReLU kết hợp MaxPool. Theo thiết kế pooling, chiều rộng bị giảm theo hệ số:

$$s = 2 \times 2 \times 1 \times 1 \times 2 = 8.$$

Do đó, nếu ảnh đầu vào có bề rộng `PAD_W`, thì độ dài chuỗi đặc trưng sau CNN xấp xỉ:

$$T = \left\lfloor \frac{\texttt{PAD\_W}}{8} \right\rfloor.$$

Hệ thống định nghĩa:

$$\texttt{CNN\_OUTPUT\_W} = \texttt{PAD\_W}//8.$$

3.7.4 Mask theo độ rộng hợp lệ (src_key_padding_mask)

Do ảnh được pad theo chiều rộng, phần padding không chứa thông tin chữ. Vì vậy, hệ thống dựng `src_key_padding_mask` để Transformer encoder bỏ qua phần padding.

Gọi W_{resized} là bề rộng ảnh sau resize (trước khi pad/crop). Khi đó:

$$T_{\text{valid}} = \min\left(\left\lfloor \frac{W_{\text{resized}}}{8} \right\rfloor, \texttt{CNN\_OUTPUT\_W} \right).$$

Mask kích thước $[1, \texttt{CNN\_OUTPUT\_W}]$:

$$\texttt{mask}[0, 0 : T_{\text{valid}}] = \texttt{False}, \quad \texttt{mask}[0, T_{\text{valid}} :] = \texttt{True}.$$

Cách dựng mask này là điểm mấu chốt để mô hình không bị nhiễu bởi vùng pad trong tự-attention.

3.7.5 Cấu hình Transformer encoder-decoder (theo triển khai)

Kiến trúc recognition dùng Transformer encoder-decoder với cấu hình:

$$d_{\text{model}} = 512, \quad n_{\text{head}} = 8, \quad N_{\text{enc}} = 3, \quad N_{\text{dec}} = 3, \quad d_{\text{ff}} = 2048,$$

dropout $= 0.2$ và kích hoạt GELU. Decoder sinh chuỗi tự hồi quy, và trong suy luận dùng beam search (mục sau).

3.7.6 API encode/decode trong pipeline

Trong suy luận từng crop (batch size = 1), pipeline thực hiện:

- `memory = model.encode(tensor, mask)`: tạo biểu diễn encoder đã xét mask.
- `seq = beam_search_decode_with_lm(...)`: giải mã chuỗi ký tự với KenLM.

Giới hạn độ dài giải mã của pipeline đặt `max_len = 150` token cho mỗi dòng, nhằm tránh vòng lặp sinh quá dài trong trường hợp ảnh nhiễu.

3.8 Giai đoạn 3: Giải mã Beam Search tích hợp KenLM

3.8.1 Động cơ tích hợp KenLM

Mặc dù Transformer OCR học được ngữ cảnh trong ảnh dòng, chữ viết tay tiếng Việt vẫn thường gây lỗi:

- nhầm ký tự có hình dạng gần nhau,
- thiếu/nhầm dấu,
- tách từ sai do khoảng trắng mờ,
- dấu câu không ổn định.

Vì vậy, hệ thống tích hợp KenLM (mô hình n-gram) để tăng tính hợp lệ ngôn ngữ khi chọn giả thuyết cuối trong beam search.

3.8.2 Hàm điểm kết hợp đúng theo triển khai

Mỗi giả thuyết chuỗi $\hat{\mathbf{y}} = (y_1, \ldots, y_L)$ được chấm điểm:

$$S(\hat{\mathbf{y}}) = S_{\text{ocr}}(\hat{\mathbf{y}}) + \alpha \cdot S_{\text{lm}}(\hat{\mathbf{y}}) + \beta \cdot L, \tag{1}$$

trong đó:

- $S_{\text{ocr}} = \sum_{t=1}^{L} \log p(y_t \mid y_{<t}, I)$ (log-softmax cộng dồn).

- S_{lm} là điểm KenLM cộng dồn.
- $L = |\hat{\mathbf{y}}|$ là **độ dài token** của chuỗi hiện tại (pipeline dùng trực tiếp `len(seq)`).
- $\alpha =$ `lm_alpha`, $\beta =$ `lm_beta` được truyền qua CLI.

3.8.3 Cập nhật KenLM theo biên từ (space/EOS)

KenLM hoạt động theo **từ**, do đó pipeline duy trì một "từ đang xây" (current_word) dưới dạng danh sách chỉ số ký tự. Khi decoder sinh ký tự:

- Nếu ký tự không phải khoảng trắng và không phải EOS: append vào current_word, *không cộng điểm LM ngay*.
- Nếu gặp **khoảng trắng** hoặc **EOS**: convert current_word thành string, gọi `lm_model.BaseScore(prev_state, word, new_state)` để lấy điểm và cập nhật trạng thái.
- Nếu EOS: cộng thêm điểm kết thúc câu `"</s>"` để hoàn thiện ngữ cảnh LM.

Cơ chế này làm LM đóng vai trò *rescoring theo từ*, phù hợp tiếng Việt (không phải ngôn ngữ chắp dính theo ký tự) và giảm chi phí tính LM so với chấm theo ký tự ở mọi bước.

Algorithm 3 Beam Search với KenLM (tóm tắt đúng logic triển khai)

Require: Encoder memory, mask, α, β, beam width B, SOS/EOS/PAD, KenLM model
Ensure: Chuỗi chỉ số token tốt nhất

Khởi tạo $\texttt{state}_0$ bằng `BeginSentenceWrite`
$\mathcal{H} \leftarrow \{([SOS],\ s_{\text{ocr}} = 0,\ \texttt{state}_0,\ s_{\text{lm}} = 0,\ \texttt{current\_word} = \emptyset)\}$
for $t = 1$ **to** `max_len` **do**
  $\mathcal{C} \leftarrow \emptyset$
  for all hypothesis $h \in \mathcal{H}$ **do**
    if last token của h là EOS **then**
      Đưa h vào $\mathcal{C}$ với điểm $S(h) = s_{\text{ocr}} + \alpha s_{\text{lm}} + \beta|seq|$
      continue
    end if
    Tính log-prob của decoder tại bước t, lấy top-B token
    for all token ứng viên u trong top-B **do**
      Bỏ qua nếu u là PAD
      Cập nhật $s_{\text{ocr}} \leftarrow s_{\text{ocr}} + \log p(u)$
      if u là space hoặc EOS **then** ▷ chạm biên từ
        word $\leftarrow$ giải mã current_word
        $s_{\text{lm}} \leftarrow s_{\text{lm}} + \texttt{BaseScore}(\texttt{state}, word)$
        reset current_word $\leftarrow \emptyset$
        if u là EOS **then**
          $s_{\text{lm}} \leftarrow s_{\text{lm}} + \texttt{BaseScore}(\texttt{state}, \text{"}</s>\text{"})$
        end if
      else
        append u vào current_word
      end if
      Tính $S = s_{\text{ocr}} + \alpha s_{\text{lm}} + \beta|seq|$
      Đưa hypothesis mới vào $\mathcal{C}$
    end for
  end for
  Chọn B hypothesis có S lớn nhất làm $\mathcal{H}$
end for
return hypothesis tốt nhất trong $\mathcal{H}$

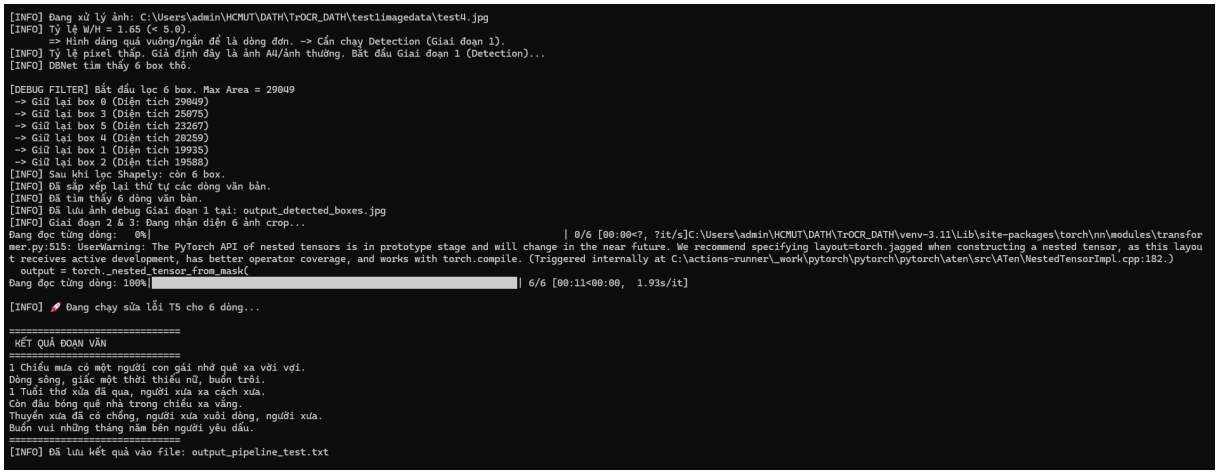
3.9 Giao diện CLI và kịch bản chạy

Pipeline cung cấp CLI để chạy end-to-end trên một ảnh bất kỳ. Ví dụ:

```
python pipeline.py ^
  --image "C:\Users\admin\HCMUT\DATH\TrOCR_DATH\test1imagedata\
```

```
    test4.jpg" ^
  --detection_model "C:\Users\admin\HCMUT\DATH\TrOCR_DATH\
    DB_TD500_resnet50.onnx" ^
  --recognition_model "C:\Users\admin\HCMUT\DATH\TrOCR_DATH\
    outputs_finetune_v2\best_transformer_26k_0.0298.pt" ^
  --lm_model "C:\Users\admin\HCMUT\DATH\TrOCR_DATH\5-gram-lm.binary
    " ^
  --beam_width 20 ^
  --lm_alpha 0.5 ^
  --lm_beta 0.2
```

Listing 1: Chạy pipeline end-to-end (Detection + Recognition + KenLM)



Hình 2: Kết quả thu được

Trong log chạy thực tế, pipeline:

- kết luận ảnh không phải dòng đơn do $W/H < 5.0$ và mật độ chữ thấp,
- chạy DBNet phát hiện 6 box,
- lọc Shapely và giữ lại 6 box,
- crop được 6 dòng và nhận dạng từng dòng (Recognition),

3.10 Thiết kế thực nghiệm và tổ chức so sánh mô hình

Để đánh giá một cách có hệ thống đóng góp của từng thành phần trong pipeline, đề tài tổ chức thực nghiệm theo ba trục so sánh chính:

- **So sánh theo kiến trúc recognition** (Subsection 3.10.1): đánh giá tác động của việc thay thế mô-đun nhận dạng CRNN gốc bằng Transformer OCR.

- **So sánh theo chiến lược giải mã** (Subsection 3.10.2): phân tích ảnh hưởng của việc tích hợp KenLM trong beam search so với giải mã thuần.
- **So sánh theo pipeline tổng thể** (Subsection 3.10.3): đánh giá hệ thống theo các cấu hình pipeline khác nhau, bao gồm recognition-only và end-to-end (detection + recognition + KenLM).

3.10.1 So sánh theo kiến trúc Recognition

Thực nghiệm so sánh recognition được thiết kế để đối chiếu trực tiếp mức cải thiện khi thay thế mô-đun nhận dạng trong mã nguồn nền bằng mô hình Transformer OCR. Ba nhóm cấu hình đại diện được sử dụng:

- **CRNN (baseline):** cấu hình recognition gốc của mã nguồn nền, được đánh giá bằng script riêng và báo cáo các chỉ số CER/WER/SER trên tập ảnh 1 dòng.
- **Transformer OCR giai đoạn 1 (9k data):** cấu hình Transformer được huấn luyện ở giai đoạn 1; checkpoint đại diện tốt nhất được chọn là `best_transformer_171.pt`.
- **Transformer OCR giai đoạn sau (FINAL):** cấu hình Transformer sau khi được pretrain và finetune quy mô lớn; khi suy luận có thể kết hợp KenLM nhằm tối ưu chuỗi đầu ra theo ràng buộc ngôn ngữ.

Việc sử dụng ba cấu hình trên cho phép đặt mô hình giai đoạn 1 như một *mốc trung gian*, từ đó tách bạch tác động của (i) thay đổi kiến trúc (CRNN → Transformer) và (ii) mở rộng dữ liệu/chiến lược huấn luyện ở giai đoạn sau.

3.10.2 So sánh theo chiến lược giải mã

Để phân tích ảnh hưởng của bước suy luận, các chiến lược giải mã được khảo sát theo hai nhóm chính:

- **Beam search không Language Model (no-LM):** cấu hình này phản ánh chất lượng và tốc độ của mô hình recognition thuần, phù hợp để kiểm chứng năng lực mô hình hóa thị giác–chuỗi mà không bị chi phối bởi tri thức ngôn ngữ bên ngoài. Trong trường hợp cần thiết, cơ chế chuẩn hóa độ dài (length normalization) và/hoặc phạt lặp (repetition penalty) có thể được áp dụng như các heuristic nhằm ổn định chuỗi sinh ra.
- **Beam search có Language Model (KenLM):** điểm số được kết hợp giữa xác suất từ mô hình OCR và điểm ngôn ngữ từ KenLM theo hàm điểm (1); điểm LM được cập nhật theo biên từ (ví dụ khi gặp khoảng trắng hoặc kết thúc chuỗi), phù hợp với thiết kế pipeline của đề tài.

Thiết kế này nhằm làm rõ vai trò của LM trong việc cải thiện tính hợp lệ ngôn ngữ, đồng thời định lượng chi phí độ trễ phát sinh do beam search và truy vấn LM.

3.10.3 So sánh theo pipeline tổng thể

Bên cạnh recognition-only, hệ thống được đánh giá theo các cấu hình pipeline nhằm phản ánh đúng các kịch bản triển khai thực tế:

- **Recognition-only trên ảnh 1 dòng:** đầu vào là ảnh đã được cắt đúng một dòng; bỏ qua detection để tập trung đánh giá mô-đun nhận dạng và chiến lược giải mã.
- **Detection + Recognition (ảnh tài liệu/A4):** đầu vào là ảnh tài liệu; DBNet được sử dụng để phát hiện và cắt dòng, sau đó recognition được chạy trên từng crop.
- **Detection + Recognition + KenLM (pipeline cuối):** bổ sung bước giải mã có LM trên từng dòng nhằm cải thiện tính hợp lệ ngôn ngữ ở mức câu/đoạn văn.

Cách phân rã trên giúp tách riêng tác động của detection/crop và đánh giá năng lực end-to-end theo đúng cấu trúc pipeline của đề tài.

3.11 Thiết kế dữ liệu và các cải tiến giai đoạn 2 ở mức hệ thống

Ngoài các lựa chọn kiến trúc và suy luận, giai đoạn 2 tập trung vào chiến lược *data-centric* nhằm mở rộng dữ liệu và tăng tính phù hợp miền cho OCR chữ viết tay tiếng Việt. Các cải tiến được tổ chức theo quy trình nhiều bước để tăng dần mức độ tương đồng với dữ liệu mục tiêu:

- **Pretrain trên hơn** 1 **triệu mẫu:** mô hình được tiền huấn luyện trên tập dữ liệu quy mô lớn để học biểu diễn thị giác tổng quát và phân phối ký tự phổ biến, tạo nền tảng ổn định cho giai đoạn thích nghi miền.
- **Finetune trên hơn** 1 **triệu mẫu dữ liệu mục tiêu:** mô hình tiếp tục được tinh chỉnh trên dữ liệu cùng miền với bài toán, nhằm tăng chất lượng nhận dạng trong điều kiện gần với kiểm thử.
- **Bổ sung vài chục nghìn ảnh chữ viết tay tự tạo và kỹ thuật stitching:** một tập ảnh chữ viết tay do tôi tạo ra được đưa vào quy trình huấn luyện; các ảnh được ghép (stitching) để tạo ảnh dòng có độ dài đa dạng, qua đó tăng khả năng học dấu tiếng Việt, nét bút thực và các biến thiên thường gặp trong thu thập thực tế.
- **Finetune tập trung trên bộ chữ viết tay tự tạo:** giai đoạn tinh chỉnh cuối tập trung vào dữ liệu chữ viết tay nhằm tối ưu hóa bài toán HTR (Handwritten Text Recognition) và tăng độ phù hợp miền mục tiêu.

Các bước trên được thiết kế để xử lý hai điểm nghẽn phổ biến của OCR chữ viết tay: (i) thiếu dữ liệu đúng phân phối nét bút của miền mục tiêu, và (ii) thiếu ngữ cảnh ở mức dòng khi dữ liệu chỉ ở mức từ/ký tự hoặc các chuỗi ngắn.

3.12 Phân tích độ phức tạp và điểm nghẽn hiệu năng

Phần này phân tích các thành phần có chi phí tính toán lớn trong pipeline, qua đó chỉ ra các yếu tố quyết định độ trễ end-to-end.

Detection (DBNet). Chi phí suy luận của DBNet phụ thuộc trực tiếp vào kích thước ảnh đầu vào của mô hình (xấp xỉ 736×736). Trong kịch bản ảnh tài liệu/A4, đây thường là một trong các bước tốn thời gian nhất do phải xử lý toàn ảnh ở độ phân giải tương đối lớn.

Recognition (Transformer). Với ảnh đã được đệm về `PAD_W`, độ dài chuỗi đặc trưng có thể xấp xỉ $T \approx \texttt{PAD\_W}/8$ (theo thiết kế giảm mẫu). Khi đó, cơ chế self-attention trong encoder có xu hướng tốn chi phí theo $O(T^2)$. Do vậy, việc giảm mẫu theo hệ số $s = 8$ và áp dụng mask cho phần padding theo chiều rộng là cần thiết nhằm tránh tính attention trên các vùng không mang thông tin.

Beam search và KenLM. Beam search làm tăng chi phí giải mã theo `beam_width` do phải duy trì và mở rộng nhiều giả thuyết song song. Đối với KenLM, thiết kế cập nhật điểm theo biên từ giúp chi phí LM tăng gần tuyến tính theo số từ (thay vì theo từng ký tự ở mọi bước), từ đó giảm phần nào chi phí truy vấn LM trong pipeline.

3.13 Kết luận chương

Chương này đã mô tả cách tổ chức hệ thống và thực nghiệm bám sát triển khai FINAL. Cụ thể, các trục so sánh được thiết kế để đánh giá riêng rẽ (i) kiến trúc recognition (CRNN, Transformer giai đoạn 1, Transformer FINAL), (ii) chiến lược giải mã (no-LM và có KenLM), và (iii) cấu hình pipeline (recognition-only và end-to-end trên ảnh tài liệu). Đồng thời, giai đoạn 2 được trình bày như một chuỗi cải tiến ở mức dữ liệu và huấn luyện (pretrain quy mô lớn, finetune miền mục tiêu, bổ sung dữ liệu chữ viết tay tự tạo, stitching và finetune tập trung cho HTR). Sau cùng, các điểm nghẽn hiệu năng được phân tích theo từng mô-đun để làm cơ sở cho các benchmark ở chương đánh giá, bao gồm chi phí detection theo kích thước ảnh, chi phí self-attention theo độ dài chuỗi đặc trưng, và chi phí beam search/LM theo cấu hình suy luận.

CHƯƠNG 4: THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1 Mục tiêu thực nghiệm và phạm vi đánh giá

Chương này trình bày:

- Môi trường hiện thực và thiết lập chạy thực nghiệm bám sát mã nguồn triển khai.
- Các tiêu chí đánh giá được sử dụng cho từng thành phần của hệ thống.
- Kết quả định lượng trên tập kiểm thử thống nhất, nhằm làm rõ mức cải thiện của các phiên bản mô hình theo tiến trình huấn luyện (pretrain/finetune) và theo cấu hình giải mã (decoding).

Phạm vi đánh giá tập trung vào:

- **Recognition cấp dòng (line-level OCR):** đánh giá định lượng trên tập kiểm thử cố định gồm khoảng ~ 1201 ảnh 1 dòng (tập thống nhất để so sánh giữa các phiên bản mô hình).
- **Pipeline end-to-end:** (Detection $\rightarrow$ Recognition $\rightarrow$ LM) cho kịch bản ảnh tài liệu/đoạn văn; phần này chủ yếu phục vụ minh hoạ khả năng vận hành theo luồng xử lý và phân tích định tính (không đặt trọng tâm vào bộ chỉ số detection do thiếu ground-truth box chuẩn hoá).

4.2 Môi trường hiện thực

4.2.1 Nền tảng phần mềm và thư viện

Hệ thống được hiện thực bằng Python và tổ chức theo các mô-đun tương ứng với pipeline. Các thư viện chính được sử dụng gồm:

- **PyTorch**: hiện thực mô hình CNN–Transformer OCR; huấn luyện và suy luận; hỗ trợ CUDA và AMP (mixed precision) trong các pha huấn luyện quy mô lớn.
- **OpenCV**: nạp và suy luận mô hình DBNet (định dạng ONNX) thông qua OpenCV DNN; đồng thời thực hiện các thao tác tiền xử lý ảnh và biến đổi phối cảnh khi cắt dòng.
- **Shapely**: xử lý hình học phẳng để lọc box phát hiện (intersection/overlap) trong bước Detection.
- **KenLM**: mô hình ngôn ngữ n-gram, tích hợp vào beam search để chấm điểm giả thuyết theo ràng buộc ngôn ngữ.
- **Thư viện hỗ trợ**: PyYAML (quản lý cấu hình), NumPy (xử lý mảng), tqdm (theo dõi tiến trình).

4.2.2 Nền tảng phần cứng và chế độ chạy

Thiết lập thực nghiệm được xây dựng để có thể vận hành ở hai chế độ:

- **Huấn luyện/finetune**: ưu tiên GPU NVIDIA hỗ trợ CUDA để tăng tốc; cơ chế AMP (autocast) được sử dụng nhằm giảm tiêu thụ bộ nhớ và tăng thông lượng.
- **Suy luận/đánh giá**: recognition được đo ở chế độ **batch=1** nhằm mô phỏng đúng kịch bản triển khai theo ảnh dòng; pipeline end-to-end xử lý theo từng ảnh đầu vào và số dòng phát hiện được.

4.2.3 Công bố mã nguồn và Demo trực tuyến.

Để đảm bảo tính minh bạch và khả năng tái lập của nghiên cứu, toàn bộ mã nguồn huấn luyện, dữ liệu cấu hình và pipeline dự án đã được công khai. Đồng thời, hệ thống được đóng gói và triển khai thử nghiệm (deploy) trên nền tảng HuggingFace Spaces để minh hoạ khả năng hoạt động thực tế.

- **Mã nguồn dự án (GitHub):** https://github.com/wuannhoangg/CRNN-To-TrOCR-Project
- **Demo trải nghiệm (Web App):** https://huggingface.co/spaces/wuann/TrOCR_VN_Handwritting

4.3 Các tiêu chí đánh giá (Evaluation Metrics)

4.3.1 Nhóm chỉ số chính cho OCR (CER, WER, SER)

Do đặc thù bài toán nhận dạng chuỗi, báo cáo sử dụng các chỉ số dựa trên khoảng cách Levenshtein:

- **CER (Character Error Rate):** tỷ lệ lỗi theo ký tự.

$$\mathrm{CER} = \frac{d_{\mathrm{lev}}(\hat{y}_{\mathrm{char}}, y_{\mathrm{char}})}{|y_{\mathrm{char}}|}.$$

- **WER (Word Error Rate):** tỷ lệ lỗi theo từ (tách theo khoảng trắng).

$$\mathrm{WER} = \frac{d_{\mathrm{lev}}(\hat{y}_{\mathrm{word}}, y_{\mathrm{word}})}{|y_{\mathrm{word}}|}.$$

- **SER (Sentence Error Rate):** tỷ lệ mẫu sai hoàn toàn (chuỗi dự đoán không khớp tuyệt đối).

$$\mathrm{SER} = \mathbb{I}[\hat{y} \neq y].$$

4.3.2 Quy đổi sang độ chính xác (Accuracy) trong OCR chuỗi

Trong ngữ cảnh OCR chuỗi, "Accuracy" thường được diễn giải tương ứng với các chỉ số lỗi:

- **Character Accuracy:** $\text{Acc}_{\text{char}} = 1 - \text{CER}$.
- **Word Accuracy:** $\text{Acc}_{\text{word}} = 1 - \text{WER}$.
- **Exact-match Accuracy:** $\text{Acc}_{\text{sent}} = 1 - \text{SER}$.

Các chỉ số Accuracy được báo cáo như dạng quy đổi để trực quan hoá mức đúng tương ứng với CER/WER/SER.

4.3.3 Precision và Recall (áp dụng cho Detection khi có ground-truth box)

Đối với mô-đun phát hiện vùng văn bản (Detection), Precision/Recall được định nghĩa như sau:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}.$$

Trong phạm vi báo cáo này, Detection (DBNet) là một mô hình pretrain có sẵn được sử dụng như bước tiền xử lý để cắt dòng khi tích hợp vào pipeline, kết quả định lượng trọng tâm được đặt ở chất lượng recognition (CER/WER/SER) của mô hình Transformer OCR. Precision/Recall cho detection có thể được bổ sung khi có tập nhãn hộp chuẩn và quy ước IoU/ngưỡng ghép cặp rõ ràng.

4.4 Thiết lập thực nghiệm và nguyên tắc tái lập

4.4.1 Dữ liệu kiểm thử và quy ước tiền xử lý

Đánh giá định lượng recognition được thực hiện trên tập kiểm thử cố định gồm $\sim$ 1201 ảnh dòng. Để đảm bảo tính nhất quán giữa huấn luyện và suy luận:

- Các tham số tiền xử lý quan trọng (`img_height`, `pad_w`) và bộ ký tự (`charset`) được đọc trực tiếp từ checkpoint thay vì cấu hình cứng trong mã suy luận.
- Nhãn và chuỗi dự đoán được chuẩn hoá trước khi tính chỉ số (ví dụ chuẩn hoá Unicode và chuẩn hoá khoảng trắng) nhằm giảm sai khác do định dạng văn bản.

4.4.2 Thiết lập chia tập train/validation trong huấn luyện

Trong các pha huấn luyện/finetune, tập dữ liệu được chia train/validation theo tham số `test_size` trong YAML (thiết lập mặc định `test_size` $= 0.1$). Validation loss được sử dụng làm tín hiệu chính để:

- điều khiển lịch học (learning rate schedule),
- lựa chọn checkpoint tốt nhất theo tiêu chí hội tụ.

4.4.3 Cấu hình huấn luyện (Hyperparameters) bám sát mã nguồn

Các tham số huấn luyện trọng yếu của các pha finetune (pha quyết định chất lượng) được thiết lập theo cấu hình YAML và logic tối ưu hoá trong code:

- **Input:** ảnh xám được chuẩn hoá chiều cao `img_height` $= 118$; chiều rộng biến thiên và được pad/crop về `pad_w` $= 3500$.
- **Batch size:** `batch_size` $= 8$ cho train; `val_batch_size` $= 16$ cho validation.
- **Epochs (finetune):** `epochs` $= 120$.
- **Optimizer:** AdamW (triển khai dùng fused AdamW khi chạy trên CUDA); `weight_decay` theo mặc định của AdamW (0.01 nếu không ghi đè).
- **Learning rate:** pha finetune sử dụng `learning_rate` $= 10^{-5}$ theo YAML.
- **Scheduler:** ReduceLROnPlateau theo validation loss (giảm LR khi val loss chững lại với `factor`=0.5, `patience`=3 theo cấu hình trong code).
- **Ổn định huấn luyện:** AMP (autocast, dtype BF16 theo code) và gradient clipping với ngưỡng $\|\nabla\|_2 \leq 0.5$.

4.5 Kết quả thực nghiệm (Recognition line-level)

4.5.1 Các cấu hình so sánh

Các cấu hình mô hình được đưa vào so sánh trên cùng tập kiểm thử line-level gồm:

- **CRNN (baseline):** mô hình recognition gốc của mã nguồn nền, dùng làm mốc tham chiếu.
- **Transformer giai đoạn đầu (Model 171):** phiên bản Transformer huấn luyện trên tập dữ liệu nhỏ (baseline Transformer), phản ánh giới hạn khi dữ liệu chưa đủ bao phủ.
- **Transformer FINAL:** các phiên bản sau pretrain/finetune quy mô lớn và finetune chuyên biệt (nhóm `1m`, `23k`, `26k`).

4.5.2 So sánh các mốc baseline và mô hình tốt nhất

Bảng 5 tổng hợp ba mốc đại diện: CRNN baseline, Transformer giai đoạn đầu (171), và phiên bản tốt nhất theo bảng xếp hạng (26k). Chỉ số Char Accuracy được quy đổi trực tiếp từ CER.

Bảng 5: So sánh hiệu năng các mô hình trên tập kiểm thử line-level ($\sim$ 1201 ảnh).

Mô hình	WER	CER	SER	Char Accuracy
CRNN Baseline	0.5221	0.2154	0.9717	78.46%
Transformer (GĐ1 - Model 171)	0.6894	0.4011	0.9509	59.89%
Transformer Final (26k)	**0.2690**	**0.1088**	**0.7602**	**89.12%**

Nhận xét. Kết quả cho thấy nhóm mô hình Transformer FINAL đạt cải thiện đáng kể khi so sánh với các mốc baseline. Đặc biệt, CER của Transformer FINAL (26k) giảm mạnh so với Transformer giai đoạn đầu (171), phản ánh hiệu quả của chiến lược dữ liệu lớn (pretrain/finetune) và tinh chỉnh theo miền chữ viết tay. So với CRNN baseline, mô hình FINAL cải thiện rõ rệt ở WER và CER, cho thấy khả năng xử lý chuỗi dài và dấu tiếng Việt ổn định hơn trong bối cảnh dữ liệu mục tiêu.

4.6 Bảng xếp hạng và phân tích quá trình cải thiện

Bảng 6 trình bày kết quả của các phiên bản checkpoint trong quá trình thực nghiệm. Bảng được rút gọn để giữ các mốc có ý nghĩa đánh giá (các mô hình không phục vụ phân tích được lược bỏ).

Bảng 6: Bảng xếp hạng các phiên bản mô hình (sắp xếp theo độ chính xác).

Model Checkpoint	WER	CER	SER
best_transformer_26k_0.0298.pt	**0.2690**	**0.1088**	**0.7602**
best_transformer_23k_00729.pt	0.2716	0.1174	0.7577
best_transformer_23k_00702_best.pt	0.2737	0.1086	0.7594
best_transformer_26k_0.0289.pt	0.2748	0.1137	0.7677
best_transformer_1m_01784.pt	0.3018	0.1213	0.7635
best_transformer_1m_01530.pt	0.3123	0.1197	0.7619
best_transformer_pretrain.pt	0.3856	0.1623	0.8326
best_transformer_0713.pt	0.4015	0.1846	0.8743
best_transformer16.pt	0.4678	0.2424	0.9101
best_transformer_171.pt	0.6894	0.4011	0.9509
best_transformer.pt	0.9996	0.7463	0.9925

4.6.1 Phân tích diễn tiến thực nghiệm (Story of Improvement)

Dựa trên Bảng 6, tiến trình thực nghiệm phản ánh một chuỗi quyết định chiến lược về dữ liệu và không gian nhãn (charset):

1. **Giới hạn của mô hình khởi tạo (Model 171):** Mô hình giai đoạn đầu được huấn luyện trên tập dữ liệu nhỏ ($\sim 9k$ ảnh) với bộ ký tự giới hạn. Trong điều kiện dữ liệu chưa đủ bao phủ biến thiên chữ viết tay và dấu tiếng Việt, mô hình đạt mức CER cao ($\sim 40\%$), phản ánh giới hạn về khả năng khái quát.

2. **Thất bại khi mở rộng Charset (các mô hình trung gian):** Khi bộ ký tự được mở rộng (bao gồm đầy đủ tổ hợp dấu tiếng Việt) nhưng dữ liệu huấn luyện vẫn ở quy mô nhỏ, một số checkpoint trung gian cho kết quả kém (ví dụ WER rất cao). Hiện tượng này phù hợp với underfitting theo nghĩa dữ liệu không đủ để mô hình học ánh xạ ổn định trong không gian nhãn lớn hơn.

3. **Bước ngoặt Pretrain quy mô lớn (nhóm 1M):** Chiến lược pretrain trên dữ liệu quy mô lớn tạo nền biểu diễn thị giác tổng quát cho nhiều lớp ký tự, giúp CER giảm về vùng $\sim 12\%$ trên các checkpoint nhóm `1m`. Đây là điều kiện cần để chuyển sang các pha finetune theo miền mục tiêu.

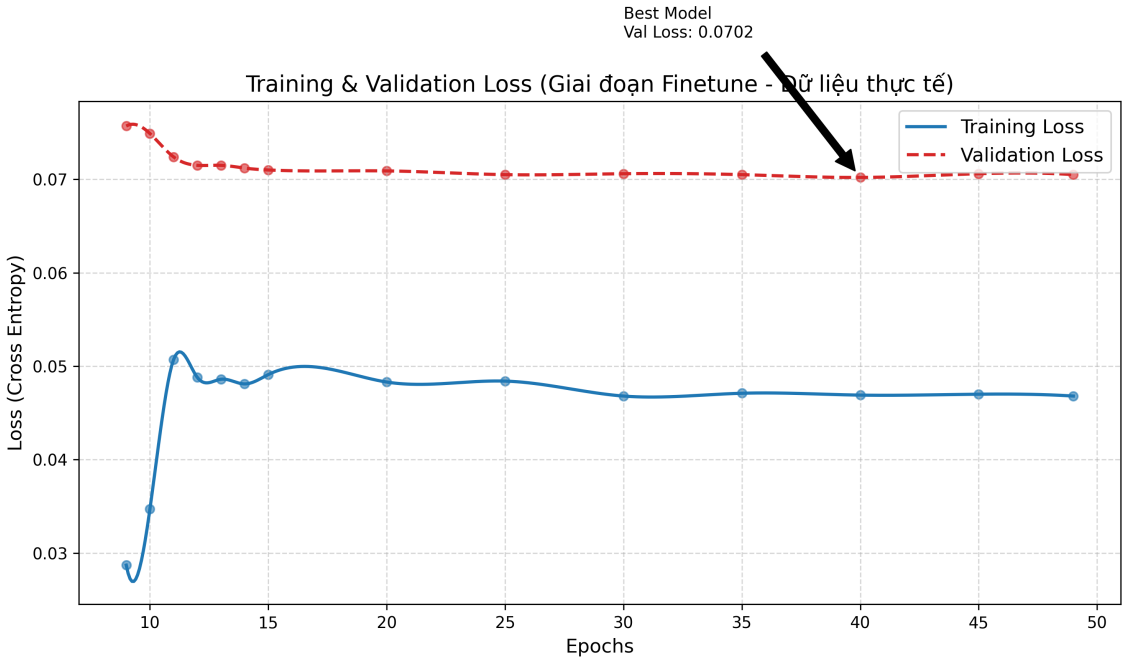
4. **Tối ưu hóa bằng finetune chuyên biệt (nhóm 23k/26k):** Sau khi có nền pretrain mạnh, finetune trên dữ liệu mục tiêu chữ viết tay giúp mô hình thích nghi với nét bút và nhiễu thực tế, đưa CER xuống mức tối ưu $\sim 10.88\%$ ở checkpoint tốt nhất.

4.7 Biểu đồ quá trình học và phân tích hội tụ

4.7.1 Biểu đồ Loss đại diện (Representative Loss Curve)

Hình 3 trình bày đường cong *training loss* và *validation loss* trong một pha tinh chỉnh (finetune) trên dữ liệu thực tế. Dữ liệu được trích xuất trực tiếp từ nhật ký huấn luyện của một checkpoint thuộc nhóm `23k`, trong đó giá trị tốt nhất ghi nhận được ở pha này là `val_loss` $= 0.0702$ (điểm được đánh dấu trên hình). Biểu đồ được sử dụng nhằm:

- Kiểm tra xu hướng hội tụ của quá trình huấn luyện.
- Đánh giá độ ổn định của tối ưu hoá.
- Nhận diện các hiện tượng overfitting/underfitting thông qua so sánh khoảng cách train/validation.



Hình 3: Đồ thị training/validation loss trong một pha finetune trên dữ liệu thực tế (minh hoạ checkpoint có `val_loss` $= 0.0702$).

4.7.2 Phân tích hội tụ và đánh giá Overfitting/Underfitting

Từ Hình 3, có thể rút ra các nhận xét sau:

- **Xu hướng hội tụ:** Validation loss giảm rõ rệt ở giai đoạn đầu của pha finetune và sau đó đi vào vùng ổn định quanh mức xấp xỉ 0.070. Điểm tốt nhất trong pha minh hoạ đạt `val_loss` $= 0.0702$, cho thấy quá trình tối ưu hoá đã tiệm cận ngưỡng bão hoà của pha tinh chỉnh này trên tập validation.

- **Dao động của training loss và nguyên nhân hợp lý:** Training loss không giảm đơn điệu mà dao động quanh vùng hội tụ. Hiện tượng dao động cục bộ là phù hợp trong bối cảnh huấn luyện có áp dụng *data augmentation online* (xoay nhẹ, làm mờ, nhiễu, thay đổi tương phản), khiến phân phối độ khó của từng batch thay đổi theo thời gian và làm loss tức thời tăng/giảm quanh giá trị trung bình.

- **Đánh giá overfitting:** Overfitting thường biểu hiện bởi việc training loss tiếp tục giảm trong khi validation loss tăng trở lại (hoặc khoảng cách train–val mở rộng dần theo epoch). Trong pha minh hoạ, validation loss không xuất hiện xu hướng tăng kéo dài mà duy trì ổn định sau giai đoạn giảm nhanh. Vì vậy, không quan sát thấy dấu hiệu overfitting chi phối trong pha huấn luyện được minh hoạ.

- **Đánh giá underfitting:** Underfitting thường xảy ra khi cả training loss và validation loss đều duy trì cao và giảm rất chậm. Trong hình minh hoạ, validation loss giảm rõ rệt và hội tụ, do đó underfitting không phải hiện tượng chính của pha này.

- **Liên hệ với mô hình cuối:** Checkpoint có `val_loss` $= 0.0702$ là một mốc trung gian trong chuỗi finetune. Sau mốc này, quá trình finetune tiếp tục được thực hiện và mô hình cuối cùng đạt `val_loss` $= 0.0298$ (checkpoint `best_transformer_26k_0.0298.pt`). Đồ thị loss tương ứng với mốc 0.0298 không được lưu lại ở thời điểm huấn luyện; tuy nhiên, kết quả của checkpoint cuối đã được xác nhận thông qua bước *evaluate* và thể hiện nhất quán trong bảng xếp hạng mô hình (Bảng 6).

4.7.3 Biểu đồ xu hướng cải thiện hiệu năng theo giai đoạn (Performance Trend)

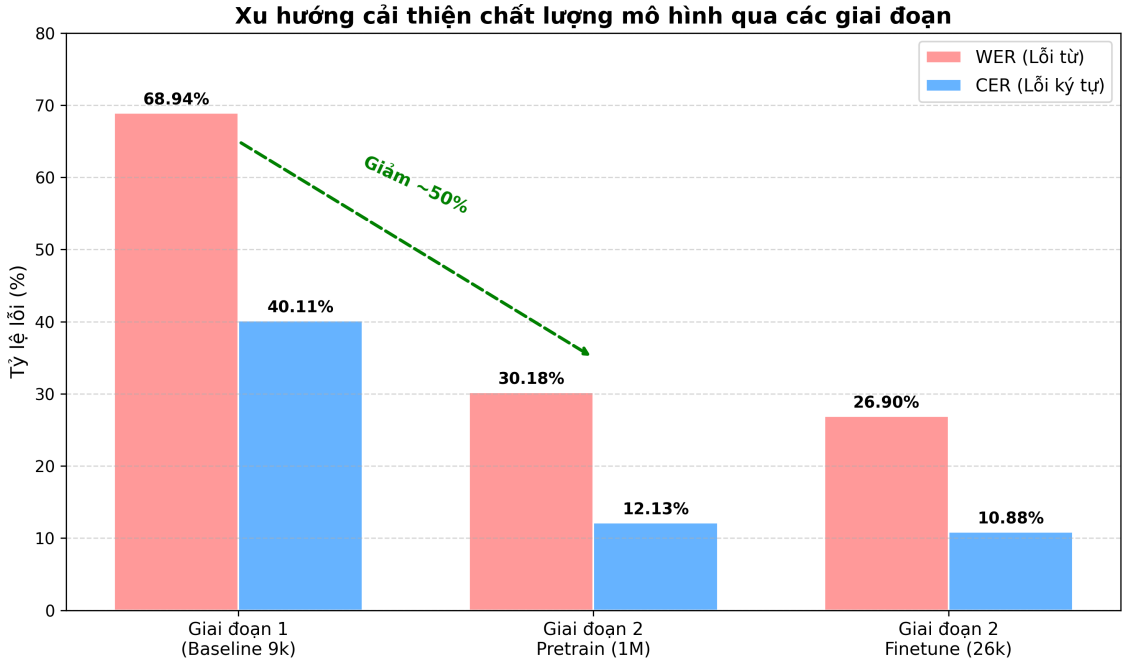
```
:\Users\admin\HCMUT\DATH\TrOCR_DATH\venv-3.11\Lib\site-packages\torch\nn\modules\transformer.py:515: UserWarning: The PyTorch API of nes
recommend specifying layout=torch.jagged when constructing a nested tensor, as this layout receives active development, has better opera
ctions-runner\_work\pytorch\pytorch\pytorch\aten\src\ATen\NestedTensorImpl.cpp:182.)
  output = torch._nested_tensor_from_mask(
Ep 1 | 241m15s | LR: 1.0e-05 | TLoss: 0.2165 | VLoss: 0.2790 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.2790)
Ep 2 | 240m2s | LR: 1.0e-05 | TLoss: 0.2083 | VLoss: 0.2505 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.2505)
Ep 3 | 239m49s | LR: 1.0e-05 | TLoss: 0.2025 | VLoss: 0.2548 | CER: 1.0000 | WER: 1.0000
Ep 4 | 239m48s | LR: 1.0e-05 | TLoss: 0.1981 | VLoss: 0.2839 | CER: 1.0000 | WER: 1.0000
Ep 5 | 239m59s | LR: 1.0e-05 | TLoss: 0.1941 | VLoss: 0.2678 | CER: 1.0000 | WER: 1.0000
Ep 6 | 239m42s | LR: 5.0e-06 | TLoss: 0.1913 | VLoss: 0.2622 | CER: 1.0000 | WER: 1.0000
Ep 7 | 239m5s | LR: 5.0e-06 | TLoss: 0.1818 | VLoss: 0.2558 | CER: 1.0000 | WER: 1.0000
Ep 8 | 239m20s | LR: 5.0e-06 | TLoss: 0.1794 | VLoss: 0.2622 | CER: 1.0000 | WER: 1.0000
Ep 9 | 238m19s | LR: 5.0e-06 | TLoss: 0.1778 | VLoss: 0.2428 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.2428)
Ep 10 | 238m29s | LR: 5.0e-06 | TLoss: 0.1765 | VLoss: 0.2689 | CER: 1.0000 | WER: 1.0000
Ep 11 | 240m47s | LR: 5.0e-06 | TLoss: 0.1756 | VLoss: 0.2422 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.2422)
Ep 12 | 239m30s | LR: 5.0e-06 | TLoss: 0.1742 | VLoss: 0.2499 | CER: 1.0000 | WER: 1.0000
Ep 13 | 243m1s | LR: 5.0e-06 | TLoss: 0.1733 | VLoss: 0.2566 | CER: 1.0000 | WER: 1.0000
Ep 14 | 260m14s | LR: 5.0e-06 | TLoss: 0.1722 | VLoss: 0.2419 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.2419)
Ep 15 | 239m11s | LR: 5.0e-06 | TLoss: 0.1713 | VLoss: 0.2387 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.2387)
Ep 16 | 239m20s | LR: 5.0e-06 | TLoss: 0.1706 | VLoss: 0.2428 | CER: 1.0000 | WER: 1.0000
Ep 17 | 239m24s | LR: 5.0e-06 | TLoss: 0.1695 | VLoss: 0.2547 | CER: 1.0000 | WER: 1.0000
Ep 18 | 238m52s | LR: 5.0e-06 | TLoss: 0.1684 | VLoss: 0.1946 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1946)
Ep 19 | 238m10s | LR: 5.0e-06 | TLoss: 0.1671 | VLoss: 0.1784 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1784)
Ep 20 | 238m58s | LR: 5.0e-06 | TLoss: 0.1657 | VLoss: 0.1725 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1725)
Ep 21 | 238m27s | LR: 5.0e-06 | TLoss: 0.1650 | VLoss: 0.1663 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1663)
Ep 22 | 238m7s | LR: 5.0e-06 | TLoss: 0.1638 | VLoss: 0.1693 | CER: 1.0000 | WER: 1.0000
Ep 23 | 239m12s | LR: 5.0e-06 | TLoss: 0.1636 | VLoss: 0.1604 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1604)
Ep 24 | 239m22s | LR: 5.0e-06 | TLoss: 0.1625 | VLoss: 0.1627 | CER: 1.0000 | WER: 1.0000
Ep 25 | 237m58s | LR: 5.0e-06 | TLoss: 0.1618 | VLoss: 0.1619 | CER: 1.0000 | WER: 1.0000
Ep 26 | 238m49s | LR: 5.0e-06 | TLoss: 0.1612 | VLoss: 0.1634 | CER: 1.0000 | WER: 1.0000
Ep 27 | 237m59s | LR: 2.5e-06 | TLoss: 0.1606 | VLoss: 0.1625 | CER: 1.0000 | WER: 1.0000
Ep 28 | 238m5s | LR: 2.5e-06 | TLoss: 0.1574 | VLoss: 0.1596 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1596)
Ep 29 | 238m16s | LR: 2.5e-06 | TLoss: 0.1559 | VLoss: 0.1585 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1585)
Ep 30 | 238m24s | LR: 2.5e-06 | TLoss: 0.1552 | VLoss: 0.1564 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1564)
Ep 31 | 238m0s | LR: 2.5e-06 | TLoss: 0.1549 | VLoss: 0.1556 | CER: 1.0000 | WER: 1.0000
 -> Saved best checkpoint (Loss: 0.1556)
Ep 32/120 [Train]:  77%|
```

Hình 4: Bất thường trong nhật ký huấn luyện: CER/WER train/val không phản ánh đúng chất lượng thực tế.

Trong quy trình huấn luyện, các chỉ số CER/WER theo epoch được thiết lập để theo dõi nội bộ. Tuy nhiên, nhật ký huấn luyện ghi nhận hiện tượng bất thường: CER/WER trong pha train/validation có thời điểm nhận giá trị không phản ánh đúng chất lượng (ví dụ luôn bằng 1), trong khi khi trích checkpoint ra đánh giá độc lập thì CER/WER vẫn cho kết quả hợp lệ. Để tránh đưa vào báo cáo các đường cong theo epoch không đáng tin cậy, phần đánh giá xu hướng được xây dựng dựa trên các mốc **đánh giá độc lập** từ các checkpoint đại diện.
Hình 5 tổng hợp diễn tiến WER/CER theo ba giai đoạn chính:

- Giai đoạn huấn luyện ban đầu trên tập dữ liệu nhỏ (9k).
- Giai đoạn pretrain quy mô lớn (1M).
- Giai đoạn finetune chuyên biệt trên dữ liệu chữ viết tay (26k).

Các giá trị tại từng mốc được đối chiếu trực tiếp với Bảng 5 và Bảng 6.



Hình 5: Xu hướng cải thiện WER/CER theo các giai đoạn huấn luyện (tổng hợp từ kết quả evaluate trên tập kiểm thử thống nhất).

Nhận xét. Biểu đồ cho thấy xu hướng cải thiện nhất quán khi chuyển từ dữ liệu nhỏ sang chiến lược huấn luyện dữ liệu lớn:

- Từ giai đoạn 1 (9k) sang giai đoạn pretrain (1M), WER và CER giảm mạnh, phản ánh tác động của dữ liệu lớn trong việc giúp mô hình học biểu diễn ký tự và mẫu hình thị giác tổng quát, đặc biệt khi không gian nhãn tiếng Việt có dấu rộng hơn.
- Giai đoạn finetune chuyên biệt (26k) tiếp tục cải thiện so với mốc pretrain, cho thấy việc tinh chỉnh theo miền chữ viết tay và dữ liệu mục tiêu giúp mô hình thích nghi tốt hơn với phân phối thực tế (nét bút, nhiễu nền, khoảng trắng và dấu tiếng Việt).

4.8 Benchmark hiệu năng

4.8.1 Benchmark 1: Recognition-only trên ảnh 1 dòng (Transformer-FINAL, R1 vs R2)

Mục tiêu benchmark này là đo latency của mô hình Transformer-FINAL trên bài toán **nhận diện ảnh 1 dòng** (đầu vào đã crop đúng một dòng), với hai chế độ suy luận:

- **R1: Greedy (no LM)** - triển khai dưới dạng beam search không LM với **beam_width=1**.
- **R2: Beam + KenLM** - cấu hình vận hành benchmark: **beam_width=20**, **lm_alpha=0.5**, **lm_beta=0.2**.

Ghi chú: cấu hình R2 ở đây được dùng cho mục tiêu đo độ trễ chế độ "FINAL runtime"; trong khi cấu hình tối ưu chất lượng (Bảng **??**) có thể khác bộ tham số (α, β).

Thiết lập đo: **1201 ảnh**, **warm-up 20 ảnh** $\Rightarrow$ **1181 ảnh đo**; **GPU RTX 5070**, **batch=1**. Kết quả latency được thống kê theo mean/p50/p90/min/max như Bảng 7.

Chế độ	**mean**	**p50**	**p90 (ms/img)**	**min**	**max**
R1: Greedy (no LM, beam=1)	175.80	158.69	285.74	103.81	387.75
R2: Beam + KenLM (beam=20)	2188.63	1765.57	4868.11	340.32	9051.17

Bảng 7: Benchmark recognition-only của Transformer-FINAL (1181 ảnh đo, GPU RTX 5070, batch=1).

Phân tích.

- R2 chậm hơn R1 do beam search mở rộng nhiều giả thuyết và cộng điểm LM theo tiến trình giải mã, làm tăng *tail latency* (p90/max).
- Giá trị max lớn phản ánh các trường hợp dòng dài/nhiễu khiến số bước giải mã tăng và beam search trở nên đắt đỏ.

4.8.2 Benchmark 2: Full pipeline trên ảnh tài liệu/A4 (DBNet $\rightarrow$ crop $\rightarrow$ recognition)

Benchmark này đo pipeline xử lý ảnh tài liệu:

P1: DBNet detection $\rightarrow$ P2: Post-process (filter/sort/crop) $\rightarrow$ P3: Recognition $\rightarrow$ P5: End-to-end

Trong đó P3 được quy đổi thêm **ms/dòng** dựa trên số dòng crop được.

Thiết lập đo: tối đa **30 ảnh**, warm-up 5, đo thực tế **25 ảnh**; **beam=20**. Số dòng trung bình: **7.36 dòng/ảnh**. Chạy hai phiên bản: **không LM** và **có KenLM** (beam=20, alpha=0.5, beta=0.2).

Module	**mean**	**p50**	**p90 (ms/ảnh)**	**min**	**max**
P1: Detection	441.07	431.00	474.70	395.07	543.03
P2: Post-process	7.39	4.01	19.49	0.01	40.29
P3: Recognition	40835.18	22775.49	90120.57	2728.29	97289.86
P5: End-to-end	41283.73	23252.75	90541.10	3159.90	97778.72

Bảng 8: Pipeline A4 (không LM, beam=20), 25 ảnh đo.

Kết quả pipeline (không LM).

Chỉ số quy đổi	mean	p50	p90 (ms/dòng)	min	max
P3: Recognition / line	6412.63	5984.99	11597.10	2558.34	12473.43

Bảng 9: Quy đổi theo dòng (không LM), trung bình 7.36 dòng/ảnh.

Module	mean	p50	p90 (ms/ảnh)	min	max
P1: Detection	429.47	417.93	457.06	385.98	601.70
P2: Post-process	5.86	3.02	16.36	0.01	20.24
P3: Recognition+LM	50701.56	38015.88	100280.91	5407.29	122667.73
P5: End-to-end	51136.98	38441.91	100702.82	5831.03	123080.10

Bảng 10: Pipeline A4 (có KenLM, beam=20, alpha=0.5, beta=0.2), 25 ảnh đo.

Kết quả pipeline (có KenLM).

Quy đổi đơn vị (để dễ diễn giải). Do các mốc thời gian của pipeline A4 có giá trị lớn ở đơn vị mili-giây, quy đổi sang giây như sau:

- **A4 không LM (beam=20)**: 41283.73 ms/ảnh $\approx$ 41.28 s/ảnh; và 6412.63 ms/dòng $\approx$ 6.41 s/dòng.
- **A4 có KenLM (beam=20, alpha=0.5, beta=0.2)**: 51136.98 ms/ảnh $\approx$ 51.14 s/ảnh; và 7946.79 ms/dòng $\approx$ 7.95 s/dòng.

Phân tích pipeline.

- **Bottleneck** nằm ở P3 (Recognition/Decoding), chiếm gần như toàn bộ thời gian end-to-end; DBNet detection ổn định ở mức $\approx$ 0.43-0.44 s/ảnh.
- Khi bật KenLM (cùng beam=20), tổng thời gian end-to-end tăng trung bình khoảng $51.14 - 41.28 \approx 9.86$ s/ảnh và tăng khoảng $7.95 - 6.41 \approx 1.54$ s/dòng.
- Khoảng cách lớn giữa p50 và p90 của P3/P5 phản ánh độ nhạy với độ dài dòng và độ khó ảnh (tail latency).

4.8.3 Benchmark 3: So sánh latency giữa các mô hình chính (CRNN vs Transformer-9k vs Transformer-FINAL)

Benchmark này đo **recognition-only** trên tập **1201 ảnh 1 dòng**, warm-up 20 ảnh $\Rightarrow$ **1181 ảnh đo**, chạy trên **GPU RTX 5070**, **batch=1**. Với Transformer, decode cố định **không LM, beam_width=1** nhằm đảm bảo công bằng khi so sánh tốc độ suy luận thuần.

Chỉ số quy đổi	mean	p50	p90 (ms/dòng)	min	max
P3: (Recognition+LM) / line	7946.79	8023.13	11775.11	2900.41	13153.63

Bảng 11: Quy đổi theo dòng (có KenLM), trung bình 7.36 dòng/ảnh.

Mô hình	mean	p50	p90 (ms/img)	min	max
CRNN (CTC greedy)	37.33	37.25	38.99	33.40	46.52
Transformer-9k (model 171, no LM, beam=1)	144.17	141.17	173.00	96.66	279.98
Transformer-FINAL (no LM, beam=1)	175.05	158.59	277.79	107.11	375.40

Bảng 12: So sánh latency recognition-only giữa ba mô hình (1181 ảnh đo, GPU RTX 5070, batch=1).

Nhận xét.

- CRNN nhanh nhất do pipeline CNN/CTC không phải giải mã autoregressive như Transformer.
- Transformer-FINAL có tail latency (p90/max) lớn hơn, phản ánh độ nhạy với độ dài chuỗi giải mã và đặc tính từng ảnh.
- Kết hợp với Bảng **??**, có thể thấy Transformer-FINAL đạt chất lượng tốt hơn, đánh đổi bằng độ trễ suy luận cao hơn.

4.8.4 Benchmark 4: Ảnh hưởng của beam width (trade-off tốc độ-chất lượng)

Mục tiêu của benchmark này là định lượng quan hệ đánh đổi giữa **độ trễ suy luận** và **độ chính xác** khi thay đổi tham số *beam width* trong quá trình giải mã. Thí nghiệm được thực hiện trên tập **1201 ảnh 1 dòng** (recognition-only), chạy với **batch=1** trên GPU và có **warm-up 20 ảnh**. Do đó, số mẫu dùng để thống kê **latency** là **1181 ảnh** (`timing_count`), trong khi CER/WER/SER được tính trên **toàn bộ 1201 ảnh** (`metric_count`). Tham số Language Model được cố định ở `lm_alpha=0.5` và `lm_beta=0.2`; các giá trị beam được khảo sát gồm $\{1, 5, 10, 20\}$.

Beam	mean	p50 (ms/img)	p90	CER (%)	WER (%) (tập 1 dòng)	SER (%)
1	180.75	162.96	288.01	11.79	28.21	77.27
5	590.29	495.37	1158.55	11.42	27.47	75.94
10	1043.74	832.66	2387.18	11.66	27.63	76.02
20	2185.61	1763.12	4962.52	11.69	27.67	76.02

Bảng 13: Trade-off theo beam width khi cố định KenLM ở `lm_alpha` $= 0.5$, `lm_beta` $= 0.2$ (timing_count=1181; metric_count=1201).

Phân tích và thảo luận.

- **Cải thiện chất lượng rõ nhất ở beam nhỏ.** Khi tăng beam từ 1 lên 5, cả CER và WER đều giảm đáng kể: CER từ 11.79% xuống 11.42% và WER từ 28.21% xuống 27.47%,

đồng thời SER giảm từ 77.27% xuống 75.94%. Điều này cho thấy việc duy trì nhiều giả thuyết giúp beam search tìm được chuỗi ký tự hợp lý hơn trong các trường hợp mơ hồ.

- **Độ trễ tăng phi tuyến và *tail latency* tăng rất mạnh.** Latency trung bình tăng từ 180.75 ms/img (beam=1) lên 590.29 ms/img (beam=5), và đạt 2185.61 ms/img ở beam=20 (tăng khoảng 12.1× so với beam=1). Đồng thời p90 tăng từ 288.01 lên 4962.52 ms/img, phản ánh các mẫu khó sẽ kéo dài quá trình giải mã (nhiều bước và nhiều nhánh).
- **Chất lượng bão hoà sau beam=5 dưới bộ tham số LM cố định.** Khi tăng tiếp lên beam=10 và beam=20, WER chỉ dao động nhẹ quanh 27.63%-27.67% và SER giữ ở mức 76.02%, trong khi latency tăng rất mạnh. Đây là dấu hiệu cho thấy với (α, β) cố định, tăng beam không còn đem lại lợi ích chất lượng tương xứng, đồng thời chi phí suy luận tăng đáng kể.

Khuyến nghị cấu hình. Với bài toán OCR chữ viết tay theo hướng triển khai thực tế, **beam=5** là điểm cân bằng hợp lý giữa chất lượng và tốc độ trong điều kiện $(\texttt{lm\_alpha}, \texttt{lm\_beta}) = (0.5, 0.2)$. Các cấu hình beam lớn hơn nên đi kèm cơ chế kiểm soát độ trễ như giới hạn `max_len`, *early stopping*, hoặc chiến lược điều chỉnh beam theo độ dài/độ tin cậy của chuỗi dự đoán.

4.8.5 Benchmark khi triển khai Web/API (độ trễ hệ thống end-to-end)

Chuyển tiếp từ benchmark mô hình sang benchmark hệ thống. Các benchmark trước đó tập trung đo **latency của mô hình và pipeline** trong điều kiện thực thi cục bộ (local), nhằm định lượng chi phí suy luận thuần (compute + decoding) và phân rã theo từng mô-đun. Tuy nhiên, trong kịch bản triển khai thực tế dưới dạng dịch vụ, độ trễ người dùng quan sát được còn chịu tác động của **tầng hệ thống** (truyền tải ảnh, hàng đợi, I/O, tiền/hậu xử lý, và cơ chế phục vụ). Vì vậy, benchmark Web/API được bổ sung như một phép đo **end-to-end ở mức ứng dụng**, nhằm hoàn thiện bức tranh đánh đổi giữa chất lượng và trải nghiệm triển khai.

Bên cạnh benchmark local, hệ thống được triển khai dưới dạng dịch vụ Web/API và đo độ trễ phản hồi **end-to-end** từ phía client thông qua HTTP `POST` (multipart/form-data). Khác với latency local, chỉ số Web/API bao gồm tổng hợp nhiều thành phần: thời gian upload ảnh, độ trễ truyền mạng, hàng đợi và lịch trình thực thi trên máy chủ, chi phí I/O và tiền/hậu xử lý, cũng như thời gian suy luận mô hình. Vì vậy, benchmark Web/API được dùng để **chứng minh khả năng vận hành khi triển khai**, không dùng để so sánh trực tiếp với latency mô hình đo tại local.

```
PS C:\Users\admin> $url="https://wuann-trocr-vn-handwritting.hf.space/ocr/predict"
PS C:\Users\admin> $img="C:\Users\admin\HCMUT\DATH\TrOCR_DATH\test1imagedata\test3.jpg"
PS C:\Users\admin>
PS C:\Users\admin> # warm-up
PS C:\Users\admin> curl.exe -s -X POST $url -F "image=@$img" -o NUL
PS C:\Users\admin>
PS C:\Users\admin> # measure 10 runs
PS C:\Users\admin> $times = 1..10 | ForEach-Object {
>>   (Measure-Command { curl.exe -s -X POST $url -F "image=@$img" -o NUL }).TotalSeconds
>> }
PS C:\Users\admin>
PS C:\Users\admin> $times | Measure-Object -Average -Minimum -Maximum


Count    : 10
Average  : 59.38117365
Sum      :
Maximum  : 65.4652557
Minimum  : 52.5299282
Property :
```

Hình 6: Benchmark độ trễ Web/API end-to-end

Thực nghiệm thực hiện $n = 10$ lần gọi liên tiếp trên cùng một ảnh đầu vào, kết quả:

- Trung bình: 59.38 s/lần,
- Nhỏ nhất: 52.53 s/lần,
- Lớn nhất: 65.47 s/lần.

Nhận xét. Độ trễ Web/API cao hơn đáng kể so với benchmark local do phụ thuộc mạnh vào hạ tầng triển khai và điều kiện mạng (biến thiên theo tải hệ thống, *cold start*/*warm start*, giới hạn tài nguyên nền tảng). Vì vậy, các benchmark local (recognition-only và pipeline A4) vẫn là số liệu chính để đánh giá năng lực thuật toán và mô hình; benchmark Web/API đóng vai trò bổ sung để minh hoạ tính khả thi của triển khai ở mức ứng dụng.

4.9 Kết luận chương

Chương này đã trình bày đầy đủ quy trình thực nghiệm và đánh giá theo đúng luồng triển khai của đề tài, bao gồm môi trường hiện thực, tiêu chí đo lường, thiết lập tái lập, kết quả chất lượng trên tập kiểm thử thống nhất, phân tích hội tụ, và các benchmark về độ trễ.
Các kết luận chính gồm:

- **Môi trường hiện thực và khả năng vận hành:** Hệ thống được hiện thực theo cấu trúc pipeline gồm Detection–Recognition–Language Model, với Recognition dựa trên CNN–Transformer và Detection dựa trên DBNet ONNX. Thiết lập chạy được mô tả rõ theo hai chế độ (huấn luyện/finetune và suy luận/đánh giá), trong đó AMP và gradient clipping được sử dụng để tăng ổn định tối ưu hoá và giảm chi phí bộ nhớ trong các pha dữ liệu lớn.

- **Chuẩn hoá đánh giá và nguyên tắc tái lập:** Quy trình đánh giá recognition được chuẩn hoá trên tập kiểm thử line-level cố định ($\sim$ 1201 ảnh 1 dòng). Các tham số tiền xử lý và bộ ký tự (`img_height`, `pad_w`, `charset`) được đồng bộ hoá theo checkpoint, giúp giảm sai lệch giữa huấn luyện và suy luận, đồng thời tăng tính tái lập khi đánh giá lại các phiên bản mô hình.

- **Tiêu chí đánh giá phù hợp bài toán OCR chuỗi:** CER/WER/SER được sử dụng làm thước đo cốt lõi cho recognition, kèm quy đổi sang các dạng Accuracy tương ứng nhằm trực quan hoá mức đúng. Precision/Recall được nêu như thước đo chuẩn cho detection và được định vị là phần có thể bổ sung khi có ground-truth box và quy ước IoU/ghép cặp đầy đủ.

- **Cải thiện chất lượng theo tiến trình huấn luyện:** Kết quả định lượng và bảng xếp hạng checkpoint cho thấy xu hướng cải thiện nhất quán khi chuyển từ dữ liệu nhỏ sang chiến lược dữ liệu lớn. Pretrain quy mô lớn giúp giảm mạnh CER/WER so với mốc huấn luyện sớm; finetune theo miền chữ viết tay tiếp tục cải thiện, với checkpoint tốt nhất `best_transformer_26k_0.0298.pt` đạt CER $= 0.1088$ (tương ứng $\text{Acc}_{\text{char}} = 89.12\%$) và duy trì WER/SER ở mức tốt nhất trong các mốc được báo cáo.

- **Hội tụ và độ ổn định tối ưu hoá:** Đường cong training/validation loss trong pha finetune đại diện cho thấy validation loss giảm nhanh và tiến tới vùng ổn định, phản ánh quá trình hội tụ hợp lý của tối ưu hoá. Dao động cục bộ của training loss được xem là phù hợp với cơ chế data augmentation online và không hàm ý suy giảm chất lượng trên validation trong pha minh hoạ. Ngoài ra, do tồn tại bất thường CER/WER theo epoch trong log huấn luyện (không phản ánh đúng chất lượng khi đánh giá độc lập), biểu đồ xu hướng hiệu năng được xây dựng dựa trên các mốc evaluate từ checkpoint để đảm bảo tính khách quan.

- **Benchmark hiệu năng và đánh đổi chất lượng–độ trễ:** Các benchmark local cho thấy (i) decoding autoregressive của Transformer tạo độ trễ cao hơn CRNN, (ii) trong pipeline A4, bottleneck chủ yếu nằm ở bước Recognition/Decoding (P3) trong khi Detection ổn định ở mức dưới 1 giây/ảnh, và (iii) beam search kết hợp KenLM làm tăng độ trễ theo cách phi tuyến và làm tăng tail latency. Thí nghiệm trade-off theo beam width chỉ ra mức beam nhỏ (ví dụ beam=5 dưới bộ tham số LM cố định) có thể là điểm cân bằng hợp lý giữa chất lượng và tốc độ trong cấu hình khảo sát.

- **Benchmark Web/API ở mức hệ thống:** Benchmark Web/API được sử dụng như phép đo bổ trợ để phản ánh độ trễ end-to-end trong kịch bản triển khai dịch vụ, bao gồm cả các chi phí ngoài mô hình (truyền tải, hàng đợi, I/O, tiền/hậu xử lý). Do bản chất phụ thuộc hạ tầng và điều kiện mạng, số liệu Web/API được dùng để chứng minh khả năng vận hành ứng dụng và không thay thế cho các benchmark local khi đánh giá năng lực mô hình.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận (Conclusion)

Dựa trên mục tiêu ban đầu đặt ra là xây dựng hệ thống OCR chữ viết tay tiếng Việt hiệu năng cao, đồ án đã hoàn thành các hạng mục sau:

5.1.1 Tổng kết các kết quả đạt được

- **Xây dựng thành công Pipeline End-to-End:** Hệ thống tích hợp trọn vẹn 3 module: Detection (DBNet) → Recognition (Transformer) → Correction (KenLM), cho phép xử lý từ ảnh tài liệu thô đến văn bản hoàn chỉnh.
- **Tối ưu hóa mô hình Recognition:** Áp dụng chiến lược Pretrain trên 1 triệu dữ liệu và Finetune chuyên sâu, giúp giảm CER từ 40.11% (Baseline) xuống còn 10.88% (Final).
- **Giải quyết bài toán dữ liệu:** Tạo ra bộ dữ liệu chữ viết tay tự nhiên và dữ liệu ghép (stitching) phong phú, khắc phục tình trạng thiếu data cho các ký tự tiếng Việt hiếm.

5.1.2 Đánh giá Ưu điểm và Nhược điểm

Bảng 14: Tổng hợp ưu/nhược điểm của hệ thống hiện tại

Ưu điểm (Pros)	**Nhược điểm (Cons)**
- Độ chính xác cao ($\sim$ 89% Char Accuracy) nhờ kiến trúc Transformer hiện đại. - Khả năng cải thiện tính hợp lệ của câu (giảm lỗi từ vựng) nhờ tích hợp KenLM. - Pipeline linh hoạt, có thể chạy trên cả GPU và CPU.	- Tốc độ suy luận chậm khi bật Beam Search kết hợp KenLM (độ trễ cao trên ảnh nhiều dòng). - Module Detection (DBNet) chưa được finetune sâu cho tiếng Việt, đôi khi cắt dòng bị lẹm dấu. - Yêu cầu tài nguyên phần cứng cao (VRAM) để huấn luyện.

5.2 Hướng phát triển (Future Work)

Để khắc phục các hạn chế trên và nâng cao tính ứng dụng thực tế, tôi đề xuất các giải pháp cải thiện trong tương lai theo ba hướng chính:

5.2.1 Mở rộng và Tăng cường dữ liệu (Data-Centric AI)

- **Thu thập dữ liệu thực tế đa dạng:** Mở rộng tập Target Data với các mẫu chữ viết tay từ nhiều người viết khác nhau, trên các loại giấy và điều kiện ánh sáng phức tạp (chụp điện

thoại, giấy nhăn).

- **Advanced Augmentation:** Áp dụng các kỹ thuật tăng cường dữ liệu nâng cao như Elastic Transform (biến dạng đàn hồi) để mô phỏng chữ viết nguệch ngoạc, hoặc GAN để sinh dữ liệu chữ viết tay nhân tạo.

5.2.2 Cải tiến kiến trúc mô hình

- **Nâng cấp Encoder:** Thử nghiệm thay thế Backbone CNN hiện tại bằng các kiến trúc mới hơn như *Swin Transformer* hoặc *EfficientNet* để trích xuất đặc trưng tốt hơn.
- **Tích hợp Mô hình Ngôn ngữ lớn (LLM):** Trong tương lai, có thể thay thế KenLM (n-gram đơn giản) bằng các mô hình ngôn ngữ nơ-ron (như T5, Llama, hoặc GPT API) để thực hiện hậu xử lý ở mức ngữ nghĩa (semantic level), giúp sửa lỗi chính tả và dấu câu chính xác hơn.

5.2.3 Tối ưu hóa hiệu năng triển khai

- **Model Quantization:** Lượng tử hóa mô hình về định dạng INT8 hoặc sử dụng ONNX Runtime/TensorRT để giảm kích thước model và tăng tốc độ suy luận trên các thiết bị biên (Edge Devices).
- **Distillation (Chưng cất tri thức):** Huấn luyện một mô hình nhỏ (Student) học từ mô hình lớn hiện tại (Teacher) để đạt được sự cân bằng tốt nhất giữa tốc độ và độ chính xác.
- **Tối ưu Beam Search:** Áp dụng kỹ thuật *Adaptive Beam Search* (tự động điều chỉnh beam width dựa trên độ tin cậy của dự đoán) để giảm thiểu độ trễ không cần thiết.

5.3 Kết luận chung

Đồ án đã chứng minh tiềm năng to lớn của việc ứng dụng Deep Learning vào bài toán số hóa tài liệu tiếng Việt. Mặc dù còn những thách thức về tốc độ xử lý, nhưng với nền tảng kiến trúc vững chắc và chiến lược dữ liệu đúng đắn, hệ thống hoàn toàn có thể được phát triển tiếp để trở thành một sản phẩm thương mại hoàn chỉnh trong tương lai.

Tài liệu

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. Advances in neural information processing systems, 30.

[2] Li, M., Lv, T., Chen, J., Cui, L., Lu, Y., Florêncio, D., Zhang, C., Li, Z., & Wei, F. (2023). *TrOCR: Transformer-based optical character recognition with pre-trained models*. Proceedings of the AAAI Conference on Artificial Intelligence, 37, 13094-13102.

[3] Liao, M., Wan, Z., Yao, C., Chen, K., & Bai, X. (2020). *Real-time scene text detection with differentiable binarization*. Proceedings of the AAAI Conference on Artificial Intelligence, 34(07), 11474-11481.

[4] Shi, B., Bai, X., & Yao, C. (2016). *An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition*. arXiv preprint arXiv:1507.05717.

[5] Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). *Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks*. Proceedings of the 23rd International Conference on Machine Learning (ICML), 369-376.

[6] Heafield, K. (2011). *KenLM: faster and smaller language model queries*. In Proceedings of the Sixth Workshop on Statistical Machine Translation (WMT 2011) (pp. 187-197).

[7] Nguyen, N. H., Vo, D. T. D., & Nguyen, K. V. (2022). *UIT-HWDB: Using transferring method to construct a novel benchmark for evaluating unconstrained handwriting image recognition in Vietnamese*. In 2022 RIVF International Conference on Computing and Communication Technologies (pp. 659-664).

[8] Tom Huynh. (2021). *Vietnamese Handwriting Recognition OCR*. GitHub Repository. URL: https://github.com/TomHuynhSG/Vietnamese-Handwriting-Recognition-OCR

[9] Pham Ba Quan (pbcquoc). (2021). *VietOCR: Transformer based OCR for Vietnamese*. GitHub Repository. URL: https://github.com/pbcquoc/vietocr

[10] Tuan Nguyen Van Anh. (2023). *Vietnamese KenLM for ASR*. Kaggle Dataset. URL: https://www.kaggle.com/datasets/tuannguyenvananh/vietnamese-kenlm-for-asr

[11] Hannun, A. (2017). *Sequence Modeling with CTC*. Distill. URL: https://distill.pub/2017/ctc/

[12] Nozaki, J., & Komatsu, T. (2021). *Relaxing the Conditional Independence Assumption of CTC-based ASR by Conditioning on Intermediate Predictions*. In Proceedings of Interspeech 2021.

[13] The Unicode Consortium. (2025). *Unicode Standard Annex #15: Unicode Normalization Forms*. URL: https://unicode.org/reports/tr15/

[14] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into Deep Learning: Sequence-to-Sequence Learning for Machine Translation*. URL: https://d2l.ai/

[15] Wu, Y., et al. (2016). *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv:1609.08144.

[16] Levenshtein, V. I. (1966). *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*. Soviet Physics Doklady, 10(8), 707–710.

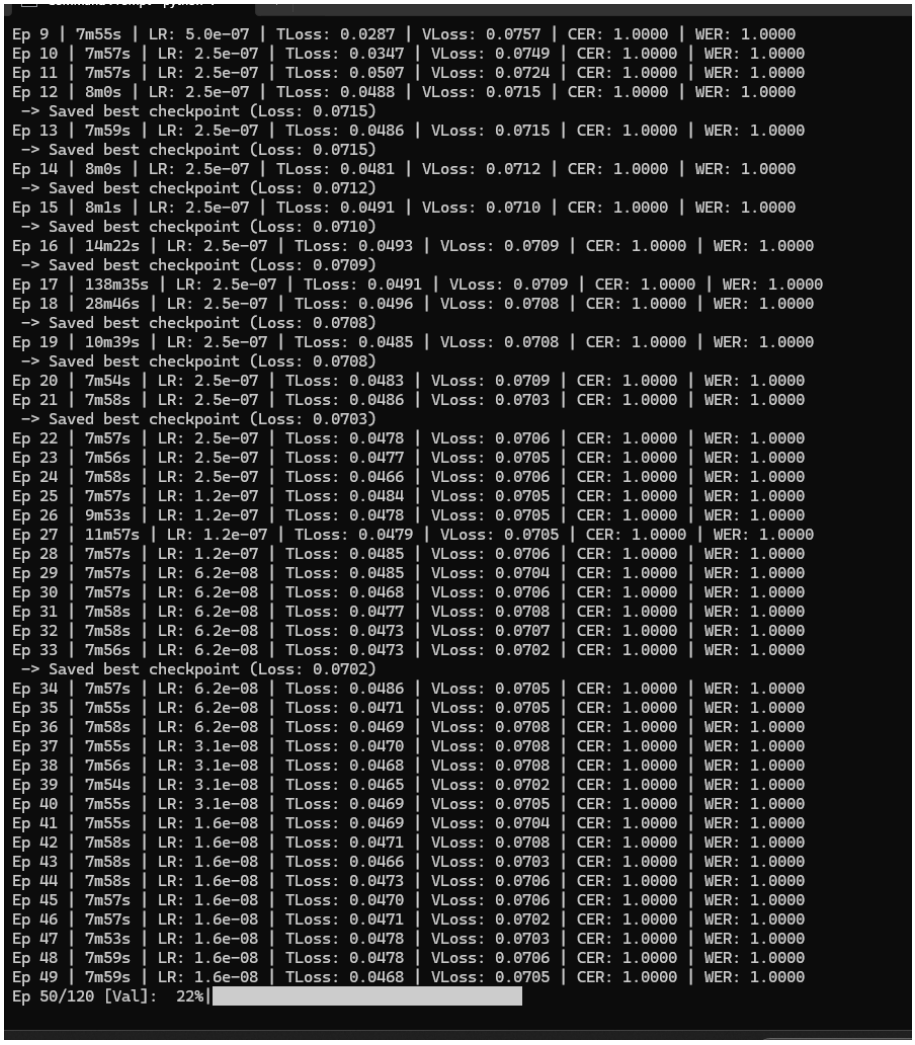
[17] OpenCV Documentation. (2023). *Text Detection Model DB_TD500_resnet50*. OpenCV DNN Module.

CHƯƠNG A: MỘT SỐ HÌNH ẢNH MINH CHỨNG THỰC NGHIỆM

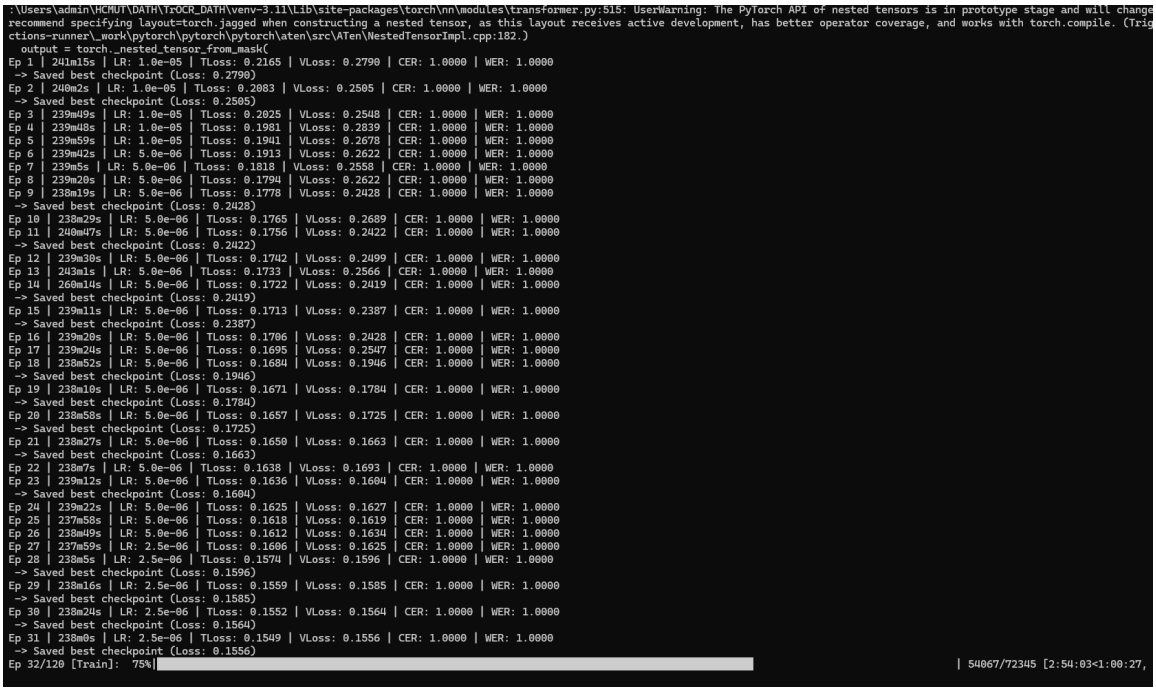
Phần này cung cấp các hình ảnh chụp màn hình thực tế (screenshot) trong quá trình huấn luyện và kiểm thử hệ thống, nhằm minh bạch hoá các số liệu đã báo cáo ở Chương 4.

A.1 Nhật ký huấn luyện (Training Logs)

Dưới đây là hình ảnh ghi nhận quá trình hội tụ của mô hình trong giai đoạn Finetune (Checkpoint 23k/26k). Log thể hiện các chỉ số Loss giảm ổn định và không xuất hiện hiện tượng Overfitting.



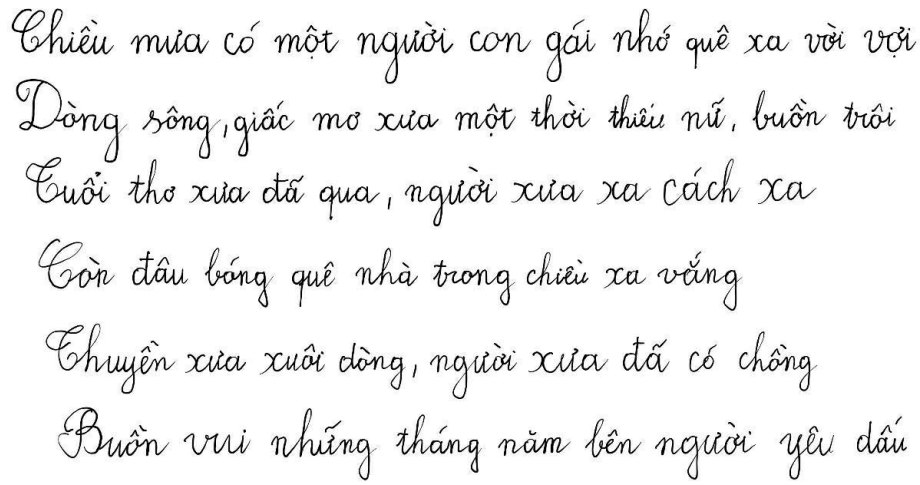
Hình 7: Ảnh chụp màn hình console quá trình Finetune: Training Loss và Validation Loss giảm ổn định.



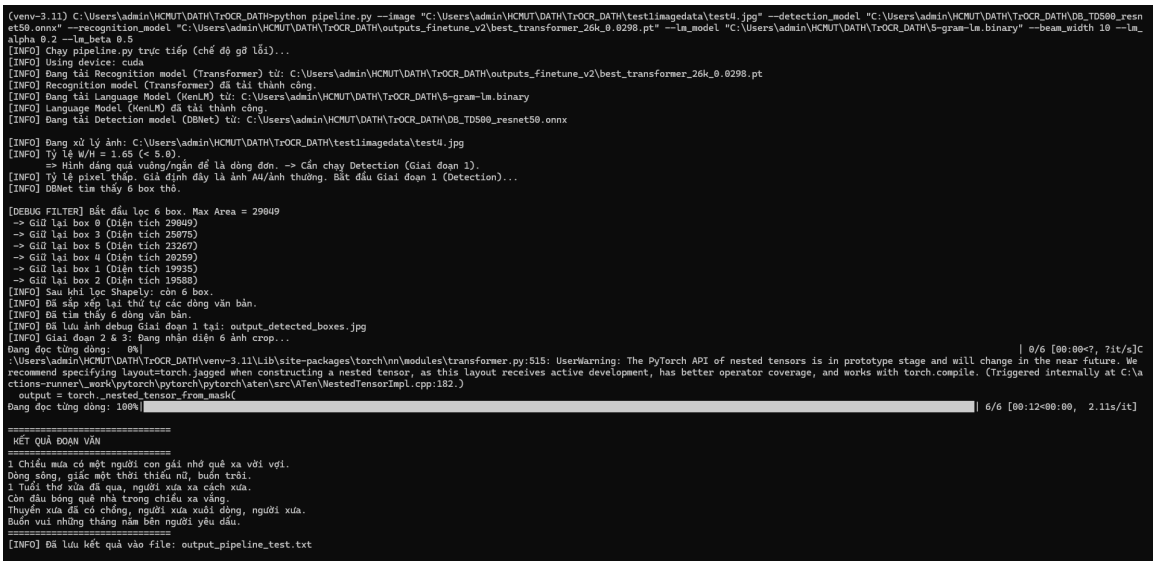
Hình 8: Ảnh chụp màn hình quá trình Pretrain trên tập dữ liệu 1 triệu mẫu.

A.2 Một số kết quả dự đoán thực tế (Demo)

Dưới đây là minh hoạ kết quả chạy pipeline end-to-end trên các mẫu chữ viết tay ngẫu nhiên, thể hiện khả năng ghép dòng và sửa lỗi của hệ thống.



Hình 9: Ảnh cần nhận dạng



Hình 10: Kết quả nhận dạng thực tế trên ảnh văn bản viết tay nhiều dòng.

CHƯƠNG B: CÁC LINK QUAN TRỌNG

- Link Github: https://github.com/wuannhoangg/CRNN-To-TrOCR-Project
- Link Data: https://drive.google.com/drive/folders/1r9xUVx5-SSoBBR7uUHyQNg_ze8bevslG?usp=sharing
- Link Models: https://drive.google.com/drive/folders/1kS9x2vLasqhu5VRu-GuxH5Wh9MxmH9-r?usp=sharing
- Link Demo Web/API: https://huggingface.co/spaces/wuann/TrOCR_VN_Handwritting