

Báo cáo Bài tập lớn NLP 2025: Dịch máy Anh–Việt bằng Transformer From Scratch và Ứng dụng VLSP 2025

Nguyễn Ngọc Duy

MSSV: 23021504

23021504@vnu.edu.vn

Nguyễn Xuân Hiếu

MSSV: 23021552

23021552@vnu.edu.vn

Vũ Đăng Dũng

MSSV: 23021500

23021500@vnu.edu.vn

Nguyễn Đức Bảo

MSSV: 23021476

23021476@vnu.edu.vn

Tóm tắt nội dung

Chúng tôi nghiên cứu dịch máy Anh–Việt trong hai bối cảnh: xây dựng mô hình Transformer từ đầu và thích nghi miền y tế cho VLSP 2025 Shared Task. Ở bài toán chính, chúng tôi tự cài đặt kiến trúc Transformer Seq2Seq (multi-head attention, positional encoding dạng sinusoidal, encoder/decoder với masking, Add & LayerNorm và feed-forward network) và huấn luyện bằng cross-entropy với Adam/AdamW cùng lịch học warmup; chất lượng dịch được đánh giá bằng BLEU với greedy và beam search, đồng thời khảo sát các cải tiến tiền xử lý và huấn luyện. Ở bài toán phụ, chúng tôi (i) finetune mô hình Transformer đã huấn luyện để thích nghi miền y tế, giúp BLEU trên tập test tăng từ 36.64 lên 41.08 với greedy và từ 38.10 lên 42.26 với beam; và (ii) triển khai một hệ thống theo constrained track bằng cách tinh chỉnh Qwen2.5-1.5B-Instruct với LoRA, đạt BLEU 48.72 (EN→VI) và 38.76 (VI→EN). Cuối cùng, phân tích lỗi định tính cho thấy các lỗi còn lại tập trung ở thuật ngữ y tế, thực thể/viết tắt và thiếu–thừa thông tin, gợi ý hướng cải thiện bằng chuẩn hoá thuật ngữ và tối ưu giải mã.

1 Giới thiệu

1.1 Bối cảnh

Dịch máy (Machine Translation – MT) là bài toán chuyển đổi văn bản từ ngôn ngữ nguồn sang ngôn ngữ đích. Transformer đã trở thành kiến trúc tiêu chuẩn cho MT nhờ cơ chế attention và khả năng song song hóa (Vaswani et al., 2017).

1.2 Mục tiêu và phạm vi bài tập lớn

Bài tập lớn gồm hai bài toán:

Bài toán chính (70%): xây dựng và huấn luyện mô hình dịch máy Seq2Seq dựa trên Transformer *from scratch*.

Bài toán phụ (30%): áp dụng/tối ưu mô hình cho VLSP 2025 Shared Task Machine Translation.

1.3 Đóng góp của nhóm

Xây dựng các module Transformer cốt lõi: Attention, PE, Encoder/Decoder layer, masking.

Triển khai pipeline huấn luyện: loss, optimizer, scheduler warmup, theo dõi loss/perplexity.

Triển khai giải mã: greedy và/hoặc beam search; đánh giá BLEU; so sánh các cải tiến.

Transfer learning/finetune cho VLSP; tối ưu hyperparameters; đánh giá và phân tích lỗi.

2 Bài toán chính (70%): Transformer From Scratch cho Dịch máy Seq2Seq

2.1 Dữ liệu và tiền xử lý

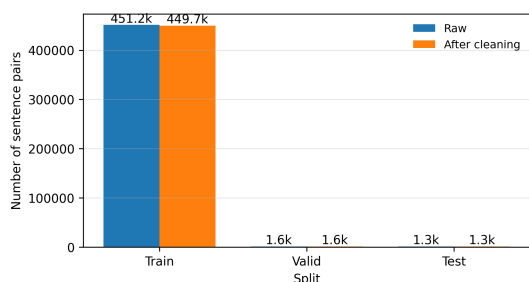
2.1.1 Nguồn dữ liệu và chia tập

Chúng tôi sử dụng dữ liệu song ngữ Anh–Việt từ IWSLT15 làm bộ chính. Tập train ban đầu của IWSLT15 có khoảng 133k cặp câu, tuy nhiên để tăng độ đa dạng và kích thước huấn luyện, chúng tôi ghép thêm TED2020 vào tập train. Do đó, dữ liệu thô sau khi ghép có kích thước: train: 451,243, valid: 1,553, test: 1,268 cặp câu (mỗi cặp gồm 1 câu EN và 1 câu VI).

Bảng 1: Thống kê số lượng cặp câu trước/sau làm sạch.

Split	Raw pairs	Kept pairs	Dropped
Train	451,243	449,692	1,551
Valid	1,553	1,550	3
Test	1,268	1,262	6

Ngoài ra, thống kê độ dài theo số từ (tách bằng khoảng trắng) trên tập train thô cho thấy: EN có trung bình **18.24** từ/câu (p95=42, p99=63, max=628) và VI có trung bình **22.97** từ/câu (p95=53, p99=80, max=850). Các câu cực dài là ngoại lệ và có thể gây chậm/khó ổn định khi huấn luyện.



Hình 1: Kích thước bộ dữ liệu trước và sau khi làm sạch theo từng tập (train/valid/test).

2.1.2 Làm sạch và chuẩn hóa

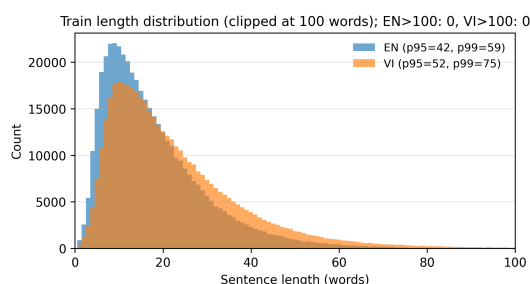
Chúng tôi áp dụng pipeline tiền xử lý nhẹ, tập trung vào tính ổn định và giữ đồng bộ cặp câu:

Loại dòng rỗng: bỏ các câu trống ở cả hai phía.

Chuẩn hóa cơ bản: `strip()` đầu/cuối dòng và đưa về chữ thường (`lower()`).

Lọc theo độ dài: loại các cặp câu có độ dài (tính theo *whitespace tokens*) vượt quá 100 từ ở một trong hai phía (EN hoặc VI).

Sau khi lọc, tập train còn **449,692** cặp câu; valid còn **1,550**; test còn **1,262** (Bảng trên).



Hình 2: Phân phối độ dài câu (theo số từ) trên tập huấn luyện sau tiền xử lý.

2.1.3 Tokenization và từ điển

Chúng tôi dùng SentencePiece với mô hình Unigram để giảm OOV và xử lý tốt từ mới/thuật ngữ. (Kudo and Richardson, 2018) Tokenizer được huấn luyện theo kiểu shared vocabulary bằng cách nối toàn bộ câu ở `train.en` và `train.vi` (sau làm sạch) vào một file chung.

Cấu hình chính:

Model: SentencePiece Unigram

Vocab size: 8,000

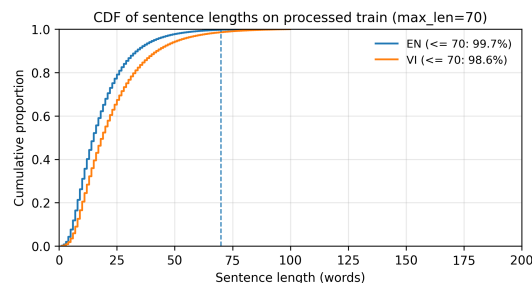
character_coverage: 0.9995

Special tokens (và id): `<pad>` (0), `<unk>` (1), `<bos>` (2), `<eos>` (3).

Với dữ liệu đích (VI), chúng tôi áp dụng cơ chế teacher forcing bằng cách tạo hai chuỗi: `tgt_in = <bos> + tokens`, và `tgt_out = tokens + <eos>`.

2.1.4 Padding, truncation, và DataLoader

Chúng tôi giới hạn độ dài tối đa để huấn luyện ổn định và tiết kiệm bộ nhớ: `max_src_len = 70` và `max_tgt_len = 70`. Các câu dài hơn sẽ bị truncation theo ngưỡng tương ứng.



Hình 3: Hàm phân phối tích lũy (CDF) độ dài câu trên tập huấn luyện sau tiền xử lý (theo số từ). Đường nét đứt biểu thị ngưỡng độ dài tối đa đã chọn.

Trong quá trình tạo batch, chúng tôi dùng dynamic padding theo batch thông qua `collate_fn`:

Pad các chuỗi nguồn/đích đến độ dài lớn nhất trong batch bằng `pad_id=0`.

Tạo các tensor đầu ra: `src_ids`, `tgt_in_ids`, `tgt_out_ids`.

Tạo **padding mask** dạng boolean: `src_padding_mask` và `tgt_padding_mask` (đánh dấu vị trí PAD để mô hình không attend vào padding và để *ignore* khi tính loss nếu cần).

Trong thực nghiệm, với batch size 32, các tensor sau collate có dạng: `src_ids`: (B, L_s), `tgt_in/out`: (B, L_t), và mask tương ứng (B, L).

2.2 Xây dựng Kiến trúc Transformer From Scratch

Chúng tôi tiến hành xây dựng toàn bộ hệ thống Transformer từ các phép toán tensor cơ bản bằng thư viện PyTorch. Mã nguồn được thiết kế linh hoạt để hỗ trợ cài đặt và so sánh hai biến thể kiến trúc phổ biến:

Post-Norm Architecture (Kiến trúc Gốc): Theo Vaswani et al. (2017), lớp chuẩn hóa (*Layer Normalization*) được thực hiện **sau** khi cộng kết nối tắt (*Residual Connection*). Quy trình: $\text{Norm}(x + \text{Sublayer}(x))$.

Pre-Norm Architecture (Kiến trúc Hiện đại): Chuẩn hóa được thực hiện **trước** khi dữ liệu đi vào lớp con (*Sub-layer*), giúp ổn định luồng gradient trong các mô hình sâu. Quy trình: $x + \text{Sublayer}(\text{Norm}(x))$.

2.2.1 Scaled Dot-Product Attention & Multi Head Attention

Thay vì chỉ sử dụng một bộ trọng số chú ý duy nhất, mô hình sử dụng cơ chế Multi-Head Attention để cho phép hệ thống tập trung vào các vị trí khác nhau trong câu từ nhiều không gian biểu diễn khác nhau.

Tính toán Q, K, V: Các ma trận Query (Q), Key (K), và Value (V) được tạo ra thông qua các phép chiếu tuyến tính từ vector đầu vào.

Tính điểm chú ý: Áp dụng cơ chế Scaled Dot-Product Attention (Vaswani et al., 2017):

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} + M \right) V \quad (1)$$

Trong đó, M là ma trận *mask* dùng để chặn các vị trí không hợp lệ và hệ số $\sqrt{d_k}$ giúp ổn định gradient khi tích vô hướng QK^T có giá trị quá lớn.

2.2.2 Positional Encoding (Sinusoidal)

Do kiến trúc Transformer xử lý dữ liệu song song và không sử dụng cơ chế hồi quy (*Recurrence*) hay tích chập (*Convolution*), mô hình không tự học được thông tin về thứ tự của các từ trong câu. Để giải quyết vấn đề này, chúng tôi cài đặt cơ chế **Sinusoidal Positional Encoding** (Vaswani et al., 2017) để "tiêm" thông tin vị trí vào vector đặc trưng (*Embedding*).

Thông tin vị trí được mã hóa dưới dạng các hàm lượng giác với tần số khác nhau theo công thức:

$$\begin{cases} PE_{(pos, 2i)} = \sin \left(\frac{pos}{10000^{2i/d_{model}}} \right) \\ PE_{(pos, 2i+1)} = \cos \left(\frac{pos}{10000^{2i/d_{model}}} \right) \end{cases} \quad (2)$$

Trong đó:

pos : Vị trí tuyệt đối của từ trong chuỗi văn bản.

i : Chỉ số chiều trong không gian đặc trưng ($0 \leq i < d_{model}/2$).

d_{model} : Tổng số chiều của vector Embedding.

$2i, 2i+1$: Chỉ số vị trí chẵn (áp dụng hàm sin) và lẻ (áp dụng hàm cos) trong vector vị trí.

Việc sử dụng các tần số khác nhau giúp mô hình nhận diện được vị trí tương đối giữa các từ, vì đối với một khoảng cách cố định k , PE_{pos+k} có thể được biểu diễn như một hàm tuyến tính của PE_{pos} .

2.2.3 Transformer Encoder Layer

Mỗi lớp Encoder đóng vai trò là một khối trích xuất đặc trưng ngữ nghĩa, xử lý song song toàn bộ chuỗi đầu vào thông qua ba thành phần chính được kết nối chặt chẽ:

Multi-Head Self-Attention: Thành phần này cho phép mô hình tính toán sự phụ thuộc giữa các từ tại mọi vị trí trong chuỗi. Thông qua việc tổng hợp thông tin từ toàn bộ ngữ cảnh, mô hình xây dựng biểu diễn đặc trưng giàu ý nghĩa cho từng từ dựa trên mối tương quan với các từ xung quanh.

Add & Layer Normalization: Thành phần điều phối luồng dữ liệu và ổn định gradient. Chúng tôi triển khai linh hoạt để phân hóa hai biến thể kiến trúc:

Post-Norm (Kiến trúc Gốc): Thực hiện chuẩn hóa lớp *sau* khi cộng kết nối tắt (Residual Connection). Cách tiếp cận này ưu tiên bảo toàn biên độ tín hiệu sau mỗi khối con.

Pre-Norm (Kiến trúc Hiện đại): Thực hiện chuẩn hóa ngay *trước* khi đi vào các lớp chức năng (Self-Attention hoặc FFN). Logic này giúp ổn định luồng gradient ở các tầng sâu, cho phép mô hình hội tụ nhanh và chịu tải được tốc độ học cao hơn.

Feed-Forward Network (FFN): Thực hiện các phép biến đổi phi tuyến tính độc lập trên từng vị trí. Chúng tôi nâng cấp lớp này với hai tùy chọn hàm kích hoạt: **ReLU** (tối ưu tốc độ) và **GeLU** (Gaussian Error Linear Unit). Việc sử dụng GeLU mang lại độ mượt gradient tốt hơn tại vùng giá trị âm, giúp mô hình nắm bắt được các đặc trưng ngôn ngữ tinh tế và phức tạp hơn.

2.2.4 Transformer Decoder Layer

Tầng Decoder thực hiện sinh chuỗi tự hồi quy (*auto-regressive*) thông qua cấu trúc ba khối con tích hợp thông tin từ chuỗi nguồn và đích:

Masked Self-Attention: Sử dụng mặt nạ *Causal Mask* để đảm bảo dự đoán tại vị trí t chỉ dựa trên các ngữ cảnh đã xuất hiện trước đó ($< t$), duy trì nguyên tắc không nhìn thấy thông tin tương lai.

Cross-Attention: Cầu nối tương tác giúp Decoder sử dụng truy vấn (Q) từ chuỗi đích để trích xuất đặc trưng ngữ nghĩa từ mã hóa của Encoder (K, V).

Add & Layer Normalization & FFN: Tương tự Encoder, chúng tôi triển khai hai cấu hình điều phối:

Post-Norm: Chuẩn hóa thực hiện *sau* mỗi khối con nhằm bảo toàn biên độ tín hiệu.

Pre-Norm: Sử dụng ba lớp chuẩn hóa riêng biệt (norm1, norm2, norm3) đặt *trước* mỗi khối chức năng. Logic này đảm bảo ổn định toán học tối đa cho luồng dữ liệu và hỗ trợ hội tụ tốt hơn ở các tầng sâu.

Mạng truyền thẳng **FFN** tích hợp hàm kích hoạt **GeLU** thay cho ReLU truyền thống để tối ưu độ mượt gradient tại vùng giá trị âm.

2.2.5 Cơ chế Masking

Cơ chế Masking đảm bảo tính đúng đắn logic bằng cách loại bỏ thông tin không hợp lệ:

Padding Mask: Triển khai trên cả hai chuỗi để "che" các ký tự đệm (*PAD*). Kỹ thuật này gán giá trị âm vô cùng ($-1e9$) tại các vị trí *PAD* trước hàm Softmax, triệt tiêu trọng số chú ý tại đó.

Causal Mask: Sử dụng ma trận tam giác dưới áp dụng tại Decoder. Chúng tôi thực hiện kết hợp đồng thời Padding Mask và Causal Mask thành một mặt nạ tổng hợp (*Target Mask*) để duy trì tính tự hồi quy và loại bỏ nhiễu từ ký tự đệm trong cùng một bước tính toán.

2.3 Huấn luyện và đánh giá

Để đảm bảo tính khách quan và khả năng hội tụ ổn định cho kiến trúc Transformer, chúng tôi thiết lập quy trình huấn luyện và đánh giá theo các thông số kỹ thuật nền tảng như sau:

2.3.1 Huấn luyện

Quá trình huấn luyện được thiết lập nhằm tối ưu hóa khả năng dự đoán chuỗi của mô hình thông qua sự kết hợp giữa các thuật toán tối ưu hiện đại và chiến lược điều phối tốc độ học linh hoạt.

Hàm mất mát (Loss function). Chúng tôi sử dụng hàm **Cross-Entropy Loss** để tối thiểu hóa sai số giữa phân phối xác suất dự đoán và nhãn thực tế, đồng thời bỏ qua các token đệm (`<pad>`). Để giải quyết vấn đề mô hình quá tự tin (*over-confidence*), kỹ thuật **Label Smoothing** được áp dụng với hệ số $\epsilon = 0.1$ (Vaswani et al., 2017). Thay vì sử dụng nhãn *one-hot* tuyệt đối, mô hình phân phối một lượng xác suất nhỏ cho các từ sai để cải thiện khả năng tổng quát hóa theo công thức:

$$y_{ls} = (1 - \epsilon) \cdot y_{target} + \frac{\epsilon}{V} \quad (3)$$

Trong đó V là kích thước bộ từ điển.

Thuật toán tối ưu (Optimizer). Chúng tôi tiếp cận bài toán tối ưu hóa thông qua thực nghiệm trên hai biến thể chính:

Adam: Sử dụng các tham số $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$ theo cấu hình gốc của kiến trúc Transformer.

AdamW (Loshchilov and Hutter, 2019): Được ưu tiên sử dụng nhằm tách biệt quá trình *Weight Decay* khỏi việc cập nhật gradient, giúp kiểm soát hiện tượng *overfitting* hiệu quả hơn.

Điều phối tốc độ học (Learning Rate Scheduler). Chúng tôi áp dụng **Noam Scheduler** (Inverse

Square Root) (Vaswani et al., 2017) để điều chỉnh tốc độ học linh hoạt qua hai giai đoạn:

Giai đoạn Warmup: Trong 4000 bước đầu, tốc độ học tăng tuyến tính để ổn định gradient khi các tham số mô hình còn sơ khởi.

Giai đoạn Decay: Sau giai đoạn khởi động, tốc độ học giảm dần theo hàm mũ nghịch đảo để mô hình hội tụ sâu.

Tốc độ học (lr) tại mỗi bước *step* được xác định bởi:

$$lr = d_{model}^{-0.5} \cdot \min \left(step^{-0.5}, step \cdot warmup_steps^{-1.5} \right)$$

Vòng lặp huấn luyện (Training Loop). Quy trình thực thi bao gồm các bước lan truyền tiến (*forward*), tính toán lỗi, lan truyền ngược (*backward*) và cập nhật trọng số. Kỹ thuật **Gradient Clipping** (ngưỡng 1.0) được sử dụng để ngăn chặn bùng nổ gradient. Hiệu suất mô hình được theo dõi liên tục thông qua chỉ số **Loss** và **Perplexity (PPL)** trên tập Validation.

2.3.2 Đánh giá

Sau khi huấn luyện, mô hình được đánh giá khả năng chuyển ngữ thông qua các giải thuật giải mã và độ đo chuẩn quốc tế:

Kỹ thuật giải mã (Decoding Strategies). Chúng tôi triển khai và so sánh hai phương pháp:

Greedy Search: Chọn token có xác suất cao nhất tại mỗi bước ($\arg \max$).

Beam Search: Duy trì $beam_size = 4$ ứng viên tiềm năng. Chúng tôi áp dụng **Length Normalization** ($\alpha = 0.6$) để tối ưu hóa việc chọn chuỗi có độ dài hợp lý:

$$score = \frac{\sum \log P(y_i | y_{<i}, \mathbf{x})}{\left(\frac{5 + len}{6}\right)^\alpha} \quad (4)$$

Tối ưu hóa Inference. Chúng tôi xây dựng cơ chế **Causal Cache** (`DECODE_MAX_LEN = 120`) để tái sử dụng mặt nạ chú ý, giúp đẩy nhanh tốc độ dịch tự hồi quy mà không cần khởi tạo lại tensor mặt nạ tại mỗi bước.

Hệ thống độ đo (Metrics). Hiệu năng được định lượng qua:

Perplexity (PPL): Tính bằng hàm mũ của Loss trung bình (e^{loss}), phản ánh độ hội tụ xác suất.

BLEU Score: Sử dụng **BLEU** (Papineni et al., 2002) và thư viện **sacrebleu** để chuẩn hoá quy trình tính điểm (Post, 2018), giúp tính toán sự

tương đồng giữa bản dịch và câu tham chiếu, thực hiện đánh giá độc lập cho cả hai phương pháp giải mã.

Quy trình lưu trữ. Trong mỗi chu kỳ (*Epoch*), nếu *Dev Loss* đạt giá trị thấp nhất mới, trọng số mô hình sẽ được lưu tại `best_finetune.pt` để đảm bảo thu được phiên bản có khả năng tổng quát hóa tốt nhất.

2.4 Tối ưu và báo cáo kết quả

Trước tiên, chúng tôi tập trung thực nghiệm so sánh để lựa chọn cấu hình kiến trúc nền tảng tối ưu nhất trước khi áp dụng các kỹ thuật tinh chỉnh chuyên sâu.

2.4.1 So sánh và lựa chọn kiến trúc nền tảng

Chúng tôi tiến hành huấn luyện song song hai biến thể kiến trúc Transformer để đánh giá hiệu năng: **Kiến trúc Baseline (Post-Norm + ReLU):** Cấu hình nguyên bản với chuẩn hóa lớp thực hiện sau kết nối tắt và sử dụng hàm kích hoạt ReLU.

Kiến trúc Modern (Pre-Norm + GeLU): Cấu hình cải tiến với chuẩn hóa lớp đặt trước các khối con (Pre-Norm) và tích hợp hàm kích hoạt GeLU.

Config: Activation=relu Pre-Norm=False			
=====			
Epoch 01	Time: 4.34m	TrLoss: 5.6961	ValLoss: 4.5286
Epoch 02	Time: 4.33m	TrLoss: 4.1689	ValLoss: 3.7797
Epoch 03	Time: 4.32m	TrLoss: 3.6125	ValLoss: 3.4649
Epoch 04	Time: 4.32m	TrLoss: 3.3320	ValLoss: 3.3219
Epoch 05	Time: 4.34m	TrLoss: 3.1718	ValLoss: 3.2434
Epoch 06	Time: 4.34m	TrLoss: 3.0626	ValLoss: 3.1858
Epoch 07	Time: 4.34m	TrLoss: 2.9826	ValLoss: 3.1381
Epoch 08	Time: 4.34m	TrLoss: 2.9174	ValLoss: 3.1098
Epoch 09	Time: 4.34m	TrLoss: 2.8638	ValLoss: 3.0855
Epoch 10	Time: 4.34m	TrLoss: 2.8195	ValLoss: 3.0802
Epoch 11	Time: 4.34m	TrLoss: 2.7790	ValLoss: 3.0690
Epoch 12	Time: 4.34m	TrLoss: 2.7440	ValLoss: 3.0638
Epoch 13	Time: 4.34m	TrLoss: 2.7121	ValLoss: 3.0482
Epoch 14	Time: 4.35m	TrLoss: 2.6844	ValLoss: 3.0397
Epoch 15	Time: 4.35m	TrLoss: 2.6576	ValLoss: 3.0509

Hình 4: Đồ thị biểu diễn sự biến thiên của *Train Loss* và *Valid Loss* theo Epoch của kiến trúc Post-Layer Normalization.

Dựa trên số liệu thực nghiệm thu được, kiến trúc **Pre-Layer Normalization** cho thấy sự ưu việt rõ rệt:

Tốc độ hội tụ nhanh. Ngay tại Epoch 01, cấu hình Modern đạt mức tổn thất trên tập kiểm chứng (*ValLoss*) là **4.1488**, thấp hơn đáng kể so với mức **4.5286** của Baseline. Điều này chứng tỏ kiến trúc

Config: Activation=gelu Pre-Norm=True			
=====			
Epoch 01	Time: 4.42m	TrLoss: 5.4975	ValLoss: 4.1488
Epoch 02	Time: 4.42m	TrLoss: 3.8113	ValLoss: 3.5721
Epoch 03	Time: 4.44m	TrLoss: 3.3453	ValLoss: 3.2927
Epoch 04	Time: 4.44m	TrLoss: 3.0919	ValLoss: 3.1794
Epoch 05	Time: 4.43m	TrLoss: 2.9449	ValLoss: 3.1111
Epoch 06	Time: 4.44m	TrLoss: 2.8421	ValLoss: 3.0680
Epoch 07	Time: 4.44m	TrLoss: 2.7627	ValLoss: 3.0548
Epoch 08	Time: 4.44m	TrLoss: 2.6996	ValLoss: 3.0323
Epoch 09	Time: 4.44m	TrLoss: 2.6454	ValLoss: 3.0255
Epoch 10	Time: 4.44m	TrLoss: 2.5984	ValLoss: 3.0197
Epoch 11	Time: 4.44m	TrLoss: 2.5586	ValLoss: 3.0188
Epoch 12	Time: 4.44m	TrLoss: 2.5216	ValLoss: 3.0162
Epoch 13	Time: 4.44m	TrLoss: 2.4891	ValLoss: 3.0199
Epoch 14	Time: 4.43m	TrLoss: 2.4591	ValLoss: 3.0216
Epoch 15	Time: 4.45m	TrLoss: 2.4318	ValLoss: 3.0275

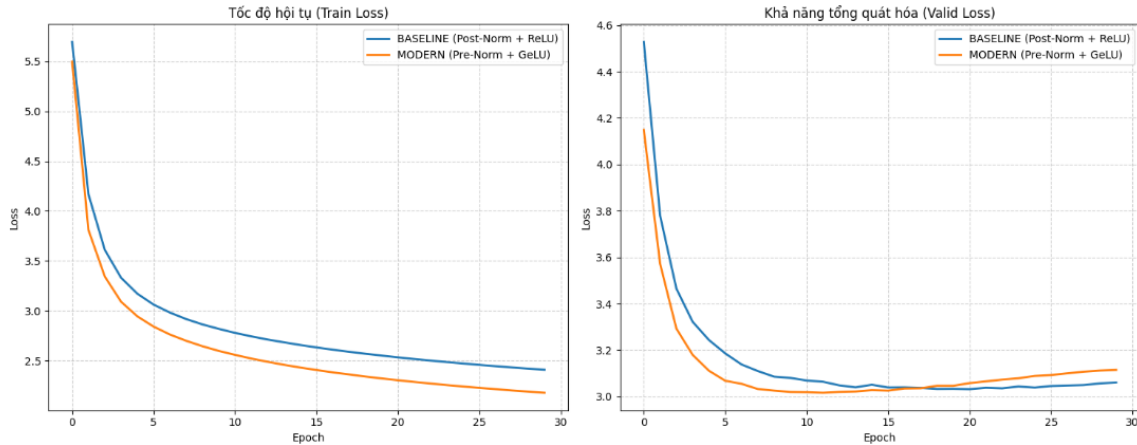
Hình 5: Đồ thị biểu diễn sự biến thiên của *Train Loss* và *Valid Loss* theo Epoch của kiến trúc Pre-Layer Normalization.

Pre-Norm giúp ổn định luồng gradient ngay từ giai đoạn khởi đầu.

Hiệu quả tối ưu hóa trên tập huấn luyện. Sau quá trình huấn luyện kéo dài 15 Epoch, cấu hình kiến trúc Modern đã chứng minh khả năng hội tụ vượt trội. Cụ thể, hàm mất mát trên tập huấn luyện (*TrLoss*) của mô hình này giảm sâu xuống mức **2.4318**, tạo ra khoảng cách hiệu năng đáng kể so với mức **2.6576** của cấu hình Baseline. Điều này cho thấy kiến trúc Modern (với Pre-Norm/GeLU) tạo điều kiện thuận lợi hơn cho dòng chảy gradient, giúp mô hình cực tiểu hóa sai số hiệu quả hơn.

Độ ổn định và Khả năng tổng quát hóa. Trên tập dữ liệu kiểm định, cấu hình Modern không chỉ đạt được sự ổn định sớm mà còn duy trì chỉ số *ValLoss* dao động quanh ngưỡng **3.01**. Hành vi hội tụ sớm này phản ánh khả năng trích xuất và học các đặc trưng ngữ nghĩa phức tạp một cách nhanh chóng và bền vững, khắc phục được hiện tượng dao động hoặc hội tụ chậm thường thấy ở cấu hình truyền thống.

Kết luận: Chúng tôi quyết định lựa chọn kiến trúc **Pre-Layer Normalization** làm khung sườn chính để tiếp tục thực hiện các bước tối ưu hóa tiếp theo.



Hình 6: So sánh tốc độ hội tụ giữa kiến trúc Post-Layer Normalization và Pre-Layer Normalization

2.4.2 Các kỹ thuật cải tiến

Sau khi xác định được kiến trúc Pre-Layer Normalization (Pre-LN) kết hợp hàm kích hoạt GeLU là cấu hình nền tảng tối ưu, chúng tôi đã triển khai một loạt các kỹ thuật cải tiến từ khâu xử lý dữ liệu đến chiến lược huấn luyện để tối đa hóa hiệu suất của mô hình.

Cải tiến cấu hình kiến trúc. Chúng tôi tinh chỉnh siêu tham số để tối ưu giữa hiệu năng và tốc độ:

Quy mô mạng: Thiết lập $d_{model} = 384$, 8 đầu chú ý ($N_{heads} = 8$) và 4 tầng mã hóa/giải mã. Mở rộng tầng Feed-forward lên 1536 đơn vị để duy trì khả năng học các cấu trúc ngôn ngữ phức tạp.

Điều tiết (Regularization): Sử dụng **Dropout** (0.1) và **Label Smoothing** (0.1) nhằm hạn chế quá khớp và tăng tính tổng quát hóa cho mô hình.

Chiến lược huấn luyện và tối ưu hóa. Quy trình huấn luyện tập trung vào sự ổn định của gradient:

Tốc độ học: Áp dụng *Noam Scheduler* với 4000 bước *warmup*, giúp mô hình hội tụ ổn định và tránh các điểm tối ưu cục bộ sớm.

Ổn định hệ thống: Duy trì *Gradient Clipping* tại ngưỡng 1.0 và sử dụng **Mixed Precision (AMP)** để đẩy nhanh tốc độ thực nghiệm và tiết kiệm tài nguyên GPU.

Tinh chỉnh chuyên sâu (Fine-tuning). Sau khi huấn luyện cơ bản, chúng tôi thực hiện tinh chỉnh trên trọng số tốt nhất:

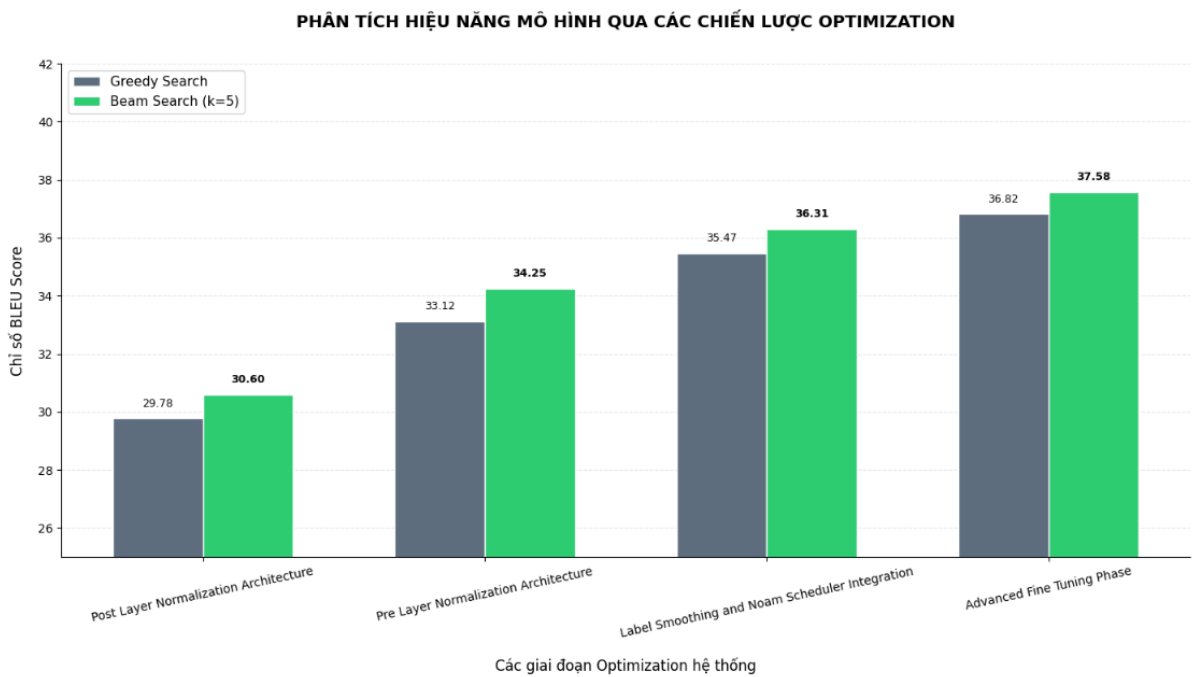
Kế thừa trạng số: Sử dụng trạng thái tối ưu từ giai đoạn trước làm điểm khởi đầu thay vì huấn luyện lại từ đầu.

Hiệu chỉnh tinh vi: Áp dụng tốc độ học cực thấp (5×10^{-5}) để mài dũa tham số, giúp tối ưu hóa chỉ

số *Loss* và *Perplexity* trước khi chốt mô hình cuối cùng.

2.4.3 Kết quả thực nghiệm và đánh giá chất lượng

Dưới đây là tổng hợp kết quả đánh giá mô hình thông qua các chỉ số định lượng và mẫu dịch thuật thực tế. Các dữ liệu này phản ánh hiệu quả của quá trình Optimization cũng như sự khác biệt giữa các chiến lược Greedy Search và Beam Search.



Hình 7: So sánh chỉ số BLEU giữa Greedy Search và Beam Search qua các giai đoạn Optimization hệ thống.

Để làm rõ hơn sự khác biệt này về mặt ngôn ngữ, chúng tôi thực hiện đánh giá định tính thông qua các mẫu dịch thuật thực tế dưới đây.

Bảng 2: Ví dụ về các bản dịch có điểm chrF thấp do Modern Transformer tạo ra (Popović, 2015).

Source	Reference	Hypothesis	chrF ↑
they are hardly monogamous.	họ hầu như không thể chế độ một vợ-một chồng.	chúng cực kỳ độc lập.	6.26
ninety-nine. well, that's an improvement.	chín mươi chín. vâng, một tiến bộ tón.	90. đó là một sự cải thiện.	8.72
if kids grow kale, kids eat kale.	nếu bọn trẻ trồng cải xoăn, chúng sẽ ăn cải xoăn.	nếu trẻ em lớn lên, lũ trẻ ăn kale.	10.62
this is on the street for a reason.	mảnh vườn này nằm trên đường là có lý do của nó.	đây là trên phố vì một lý do.	11.46
and now you're of course curious if it also worked.	bây giờ chắc các bạn đang thắc mắc liệu nó có hiệu quả hay không.	và giờ bạn cũng tò mò nếu nó làm việc.	11.94

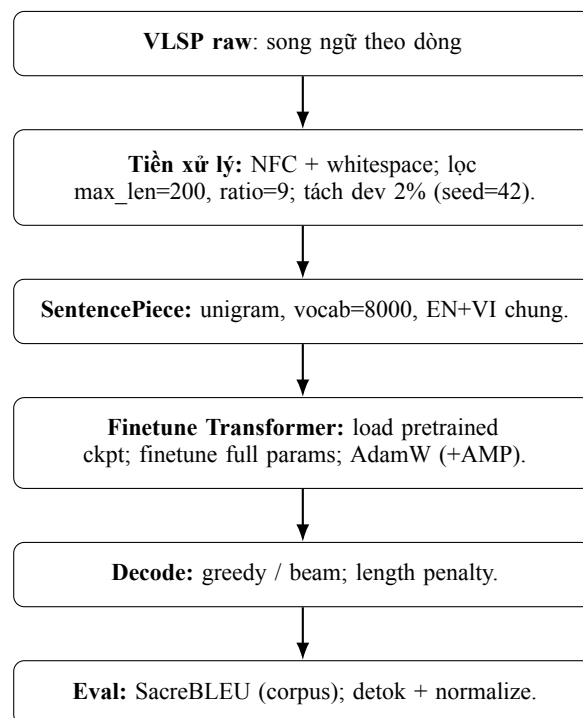
Phân tích lỗi định tính. Bảng 2 minh họa các trường hợp mô hình Modern tạo bản dịch có điểm sentence chrF thấp trên tập kiểm tra. Các ví dụ được chọn bằng cách xếp hạng theo chrF ở mức câu (và loại bỏ các câu quá ngắn để tránh thiên lệch của độ đo trên câu 1–3 từ). Lưu ý rằng chrF (Popović, 2015) được dùng để *phát hiện mẫu lỗi* ở mức câu; đánh giá định lượng tổng thể vẫn dựa trên BLEU như đã báo cáo ở Hình 7.

Nhóm lỗi quan sát được. (i) **Sai nghĩa trọng tâm (mistranslation):** ví dụ “hardly monogamous” bị dịch sai thành “cực kỳ độc lập” (hàng 1). (ii) **Lỗi số và chuẩn hóa biểu thức số:** “ninety-nine” bị rút gọn/chuẩn hóa sai thành “90” (hàng 2). (iii) **Sai nghĩa theo ngữ cảnh và giữ nguyên từ ngoại lai:** “grow kale” bị hiểu thành “lớn lên” và từ “kale” không được Việt hóa (hàng 3). (iv) **Dịch sát chữ làm giảm độ tự nhiên:** cấu trúc “on the street for a reason” và “worked” được dịch theo nghĩa đen, làm giảm tính trôi chảy/đúng collocation tiếng Việt (hàng 4–5). Ngoài ra, tập tham chiếu có thể chứa nhiều/typo (ví dụ “tón”), điều này có thể làm giảm điểm ở mức câu dù bản dịch hợp lý về mặt ngữ nghĩa.

3 Bài toán phụ (30%): VLSP 2025 Shared Task Machine Translation

3.1 Finetune từ mô hình Transformer tự huấn luyện

3.1.1 Dữ liệu VLSP và quy trình tiền xử lý



Hình 8: Quy trình finetune Transformer trên VLSP 2025.

Chúng tôi sử dụng bộ dữ liệu **VLSP 2025 Shared Task MT** (miền y tế) dưới dạng các tệp song ngữ theo dòng, mỗi dòng tương ứng một cặp câu Anh–Việt đã căn chỉnh. Mục tiêu của tiền xử lý là (i) chuẩn hoá dữ liệu về định dạng train/dev/test phục vụ huấn luyện, (ii) giảm nhiễu bằng lọc chất lượng, và (iii) đảm bảo tính tái lập (reproducibility) bằng tham số/seed cố định.

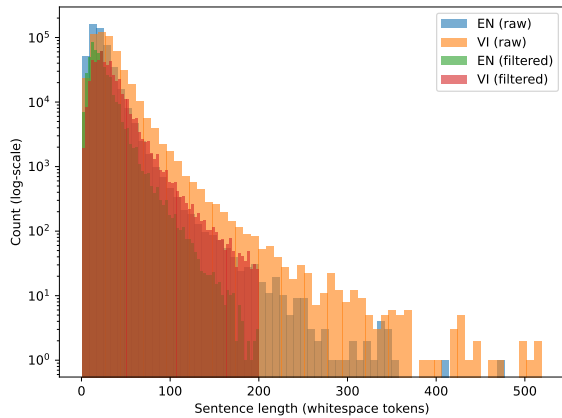
Chuẩn hoá văn bản. Mỗi câu được chuẩn hoá Unicode theo NFC và chuẩn hoá khoảng trắng (gộp nhiều khoảng trắng thành một), nhằm giảm biến thể bề mặt gây nhiễu cho mô hình dịch.

Lọc chất lượng. Chúng tôi loại bỏ các cặp câu thuộc một trong các trường hợp: (i) có phía nguồn hoặc phía đích rỗng; (ii) quá dài theo ngưỡng $\text{max_len}=200$ token (đếm thô theo khoảng trắng); (iii) lệch độ dài nghiêm trọng giữa hai phía khi tỷ lệ độ dài vượt $\text{max_ratio}=9$. Các tiêu chí (ii)–(iii) giúp hạn chế mẫu nhiễu do lỗi căn chỉnh hoặc câu bất thường, đồng thời ổn định huấn luyện và suy diễn (đặc biệt với beam search).

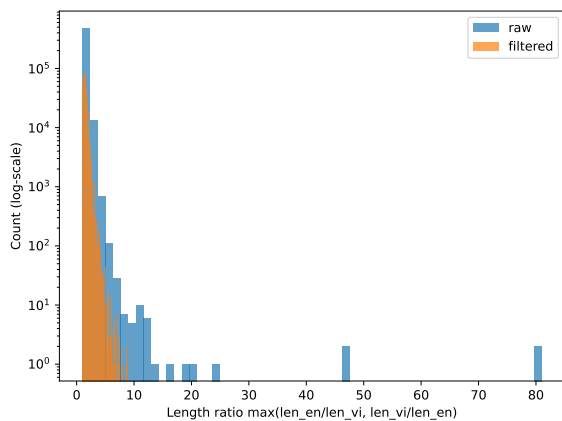
Tổ chức dữ liệu và chia tập. Sau khi lọc, chúng tôi tạo tập phát triển dev bằng cách lấy ngẫu nhiên $dev_frac=2\%$ từ tập huấn luyện đã lọc, với seed cố định ($seed=42$) để đảm bảo có thể tái lập kết quả. Tập test được giữ nguyên phục vụ đánh giá cuối cùng.

Bảng 3: Thống kê dữ liệu VLSP sau tiền xử lý và chia tập.

Tập	Số cặp	Ghi chú
Train (gốc)	500,000	–
Train (sau lọc)	499,575	-397 dài; -28 lệch tỷ lệ
Dev	9,992	2% ($seed=42$)
Train (cuối)	489,583	sau tách dev
Test	3,000	giữ nguyên



Hình 9: Phân phối độ dài câu (đếm theo khoảng trắng) trước và sau lọc chất lượng trên tập train VLSP. Trục tung dùng thang log để quan sát phần đuôi dài.



Hình 10: Phân phối tỷ lệ độ dài giữa hai phía (max ratio) trước và sau lọc. Ngưỡng lọc $max_ratio=9$ loại bỏ các cặp lệch mạnh do sai căn chỉnh.

3.1.2 Tokenizer miền y tế (SentencePiece)

Do dữ liệu y tế chứa nhiều thuật ngữ hiếm, tên thuốc, viết tắt và dạng ghép từ, chúng tôi sử dụng **SentencePiece** để phân mảnh subword (Kudo and Richardson, 2018), giúp: (i) giảm OOV, (ii) tăng khả năng tổng quát hoá với thuật ngữ mới/hiếm, và (iii) ổn định suy diễn.

Chúng tôi huấn luyện một mô hình SentencePiece chung cho cả hai phía (EN+VI) bằng cách gộp `train.en` và `train.vi`, với `vocab_size=8000` và `model_type=unigram`. Để tương thích với pipeline masking và loss (bỏ qua padding), các token đặc biệt được cố định ID: `pad=0`, `unk=1`, `bos=2`, `eos=3`.

3.1.3 Thiết lập mô hình và chiến lược transfer learning

Mô hình nền (pretrained Transformer). Chúng tôi xây dựng mô hình dịch máy dựa trên kiến trúc **Transformer encoder-decoder**, được huấn luyện trước (pretrain) trên bài toán chính nhằm học năng lực dịch tổng quát. Khi chuyển sang VLSP (miền y tế), chúng tôi áp dụng **transfer learning** bằng cách khởi tạo trọng số từ checkpoint pretrained, sau đó **finetune toàn bộ tham số** (encoder + decoder) trên dữ liệu VLSP.

Hàm mục tiêu. Mô hình được huấn luyện theo negative log-likelihood chuẩn cho dịch máy:

$$\mathcal{L}(\theta) = - \sum_{t=1}^T \log p_{\theta}(y_t | y_{<t}, x), \quad (5)$$

với teacher forcing trong huấn luyện; token pad được bỏ qua khi tính loss.

Cấu hình huấn luyện. Bảng 4 tóm tắt cấu hình được dùng trong thí nghiệm finetune. Các giá trị này được cố định để dễ tái lập và thuận tiện so sánh với mô hình trước finetune.

Bảng 4: Tóm tắt cấu hình huấn luyện/finetune trên VLSP.

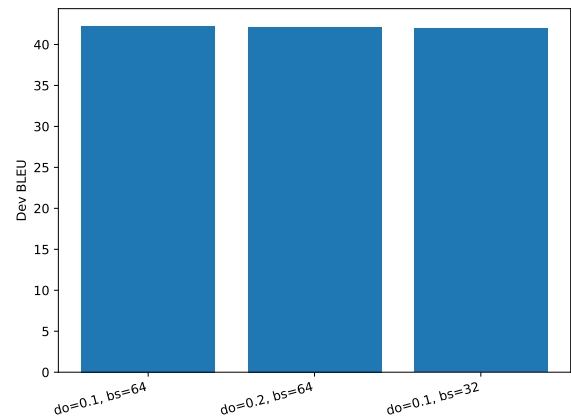
Thành phần	Thiết lập
Tokenizer	SentencePiece unigram, vocab 8000; special IDs: <code>pad/unk/bos/eos = 0/1/2/3</code>
Giới hạn độ dài	Cắt/pad tối đa 80 subword tokens cho nguồn và đích trong huấn luyện
Tối ưu	AdamW, learning rate 3×10^{-4} , weight decay 0.01, $\beta = (0.9, 0.98)$
Batch size	64
Dropout	0.1 (thiết lập mặc định; có kiểm tra độ nhạy ở mục tiếp theo)
Epoch	3; lưu checkpoint tốt nhất theo BLEU trên dev

3.1.4 Tối ưu siêu tham số và kiểm tra độ nhạy

Do VLSP là miền chuyên biệt, nguy cơ overfitting cao hơn so với dữ liệu tổng quát. Chúng tôi tối ưu siêu tham số dựa trên BLEU của dev, tập trung vào: (i) **dropout** (regularization), (ii) **batch size** (ổn định gradient và giới hạn bộ nhớ), và (iii) các lựa chọn suy diễn như **beam size** và **length normalization**.

Bảng 5: Ablation cho tối ưu hyperparameters trên VLSP

Cấu hình	BLEU (Dev)	Ghi chú
dropout=0.1, bs=64	42.26	cấu hình chọn báo cáo
dropout=0.2, bs=64	42.13	tăng dropout
dropout=0.1, bs=32	41.97	batch nhỏ hơn



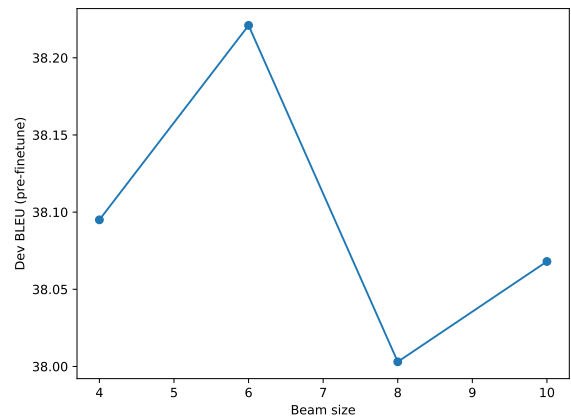
Hình 11: So sánh BLEU trên dev giữa các cấu hình dropout và batch size (ablation).

3.1.5 Suy diễn và đánh giá

Thiết lập suy diễn. Chúng tôi báo cáo hai chế độ: (i) **greedy decoding** (chọn token xác suất cao nhất tại mỗi bước), và (ii) **beam search** nhằm tăng chất lượng bằng cách giữ k giả thuyết tốt nhất. Trong thực nghiệm trước finetune, chúng tôi kiểm tra độ nhạy theo beam size và nhận thấy chất lượng bão hoà khi tăng beam (ví dụ beam=6 có thể nhỉnh hơn beam=4 nhưng không ổn định khi tiếp tục tăng beam).

Chỉ số đánh giá. Chúng tôi sử dụng **BLEU** ở mức corpus (Papineni et al., 2002), tính bằng SacreBLEU để chuẩn hoá quy trình đánh giá (Post, 2018) và áp dụng hậu xử lý nhất quán (giải mã subword, chuẩn hoá khoảng trắng) trước khi tính điểm để đảm bảo so sánh công bằng giữa các cấu

hình.



Hình 12: So sánh BLEU trên dev giữa các cấu hình beam.

Bảng 6: Kết quả Transformer trên VLSP 2025 (BLEU). Finetune cải thiện mạnh nhờ thích nghi thuật ngữ và phong cách miền y tế.

Hệ thống	Greedy	Beam
Trước finetune	36.6413	38.0952
Sau finetune	41.0817	42.2641

3.1.6 Phân tích lỗi (Error Analysis)

Chúng tôi thực hiện phân tích lỗi định tính bằng cách đối chiếu dự đoán với tham chiếu trên test. Mẫu phân tích được ưu tiên từ các câu có chất lượng thấp (ví dụ sai khác rõ rệt hoặc BLEU theo câu thấp), kết hợp thêm một số mẫu ngẫu nhiên để tránh thiên lệch.

Các lỗi được chúng tôi thành bốn loại chính: (i) **thuật ngữ y tế**: dịch sai/thiếu chuẩn hoá thuật ngữ hoặc dùng từ phổ thông thay cho thuật ngữ chuyên ngành; (ii) **tên riêng và viết tắt**: xử lý chưa ổn định với tên thuốc/tên bệnh/tên xét nghiệm và các viết tắt (MRI, CT, ECG, ...); (iii) **thiếu/thừa thông tin**: bỏ sót chi tiết quan trọng (liều lượng, thời gian, đối tượng) hoặc thêm thông tin không có trong nguồn; (iv) **ngữ pháp và trật tự từ**: câu tiếng Việt chưa tự nhiên, đặc biệt ở câu dài nhiều mệnh đề.

Nhìn chung, finetune giúp mô hình phù hợp hơn với mẫu câu và thuật ngữ miền VLSP, tuy nhiên vẫn nhạy với thuật ngữ hiếm và câu dài/phức tạp (đòi hỏi beam search và/hoặc ràng buộc thuật ngữ mạnh hơn).

Bảng 7: Ví dụ minh họa các nhóm lỗi định tính.

Loại lỗi	Nguồn (EN)	Tham chiếu (VI)	Dự đoán (VI)
Thuật ngữ	Acute pneumonia.	Viêm phổi cấp.	Cảm lạnh cấp.
Tên riêng/viết tắt	MRI shows no stroke.	MRI không thấy đột quỵ.	MR không thấy nhồi máu.
Thiếu/ thừa thông tin	Take 1 tab twice daily.	Uống 1 viên ngày 2 lần.	Uống 1 viên mỗi ngày.
Ngữ pháp/trật tự từ	Consult a doctor now.	Hãy gặp bác sĩ ngay.	Hãy ngay gặp bác sĩ.

3.2 Finetune từ mô hình Qwen trên VLSP 2025 (Medical MT)

3.2.1 Bài toán và ràng buộc thực nghiệm

VLSP 2025 Shared Task Machine Translation đặt trong miền dữ liệu Y tế (medical domain) với yêu cầu chất lượng dịch cao trong bối cảnh xuất hiện nhiều thuật ngữ chuyên ngành, cấu trúc câu đặc thù và sắc thái ngữ nghĩa chuyên môn. Bài toán gồm hai chiều dịch Anh→Việt và Việt→Anh. Theo *constrained track*, hệ thống chỉ được huấn luyện trên dữ liệu do ban tổ chức cung cấp, và mô hình nền (Base LLM) được khuyến nghị thuộc họ Qwen. Chất lượng hệ thống được đánh giá chính bằng BLEU (Papineni et al., 2002) (và Gemini theo quy định của Shared Task).

3.2.2 Chuẩn bị dữ liệu và kiểm tra chất lượng

Nguồn dữ liệu và chia tập. Dữ liệu gốc gồm hai cặp tệp song song: train.en.txt/train.vi.txt và public_test.en.txt/public_test.vi.txt. Chúng tôi kiểm tra căn chỉnh theo số dòng và xác nhận mỗi cặp song ngữ có số dòng tương ứng bằng nhau. Tập train (500,000 cặp câu) được xáo trộn chỉ số với seed=42, sau đó tách valid 10,000 mẫu và giữ lại train 490,000 mẫu. Tập test lấy trực tiếp từ public_test với 3,000 mẫu.

Quét song song (parallel scan). Chúng tôi quét dữ liệu theo từng split để phát hiện các lỗi thường gặp: (i) lệch căn chỉnh nguồn–đích, (ii) dòng trống, (iii) ký tự thay thế do lỗi mã hoá (), và (iv) thống kê độ dài câu. Kết quả cho thấy không có lệch căn chỉnh và không có dòng trống; tuy nhiên phát hiện một số ít dòng chứa ký tự phía tiếng Anh. Trong huấn luyện, các cặp chứa được loại bỏ khỏi train/valid để tránh nhiễu, trong khi tập test

vẫn được giữ nguyên để đảm bảo đánh giá đúng theo dữ liệu công bố.

Bảng 8: Kích thước dữ liệu sau bước chia tập.

Tập dữ liệu	#cặp câu	Ghi chú	Seed
Train	490,000	tách từ train.*	42
Valid	10,000	tách từ train.*	42
Test	3,000	public_test.*	–

Bảng 9: Kiểm tra căn chỉnh và lỗi ký tự.

Split	Lệch dòng	Dòng rỗng	(EN)	(VI)
Train	0	0	1	0
Valid	0	0	0	0
Test	0	0	1	0

Độ dài theo token và lựa chọn max_seq_length. Sử dụng tokenizer của Qwen2.5-1.5B-Instruct, chúng tôi ước lượng độ dài mẫu SFT (prompt + đáp án) trên tập con ngẫu nhiên. Kết quả cho thấy median khoảng 88 tokens, p95 khoảng 193 tokens và p99 khoảng 304 tokens, gợi ý đặt max_seq_length quanh 320–512 để bao phủ phần lớn dữ liệu mà vẫn phù hợp giới hạn bộ nhớ GPU.

Bảng 10: Thống kê độ dài token của mẫu huấn luyện theo tokenizer Qwen (ước lượng trên mẫu ngẫu nhiên).

Chiều dịch	P50	P90	P95	P99	Max
EN→VI	88	158	193	304	1332
VI→EN	88	158	193	304	1332

3.2.3 Định dạng instruction–response cho Qwen

Qwen là mô hình Causal Language Model dạng instruction-tuned, do đó mỗi cặp câu song ngữ được chuyển thành mẫu huấn luyện theo cấu trúc *prompt–completion*. Trong huấn luyện SFT, hàm loss chỉ áp dụng trên phần *completion* (câu đích), còn phần *prompt* được mask nhãn bằng -100 để không tính loss.

Listing 1: Template prompt cho hai chiều dịch.

```
# EN -> VI
"You are a professional medical translator.\n"
"### Task: Translate English to Vietnamese (medical domain)\n"
f"### English: {src_en}\n"
```

```

"### Vietnamese:"

# VI -> EN
"You are a professional medical
  translator.\n"
"### Task: Translate Vietnamese to
  English (medical domain)\n"
f"### Vietnamese: {src_vi}\n"
"### English:"

```

3.2.4 Thiết lập finetune (PEFT/LoRA)

Mô hình nền. Chúng tôi sử dụng Qwen/Qwen2.5-1.5B-Instruct làm mô hình khởi tạo, nhằm tận dụng tri thức tiền huấn luyện và khả năng tuân thủ instruction.

Kỹ thuật tinh chỉnh. Để giảm chi phí và phù hợp tài nguyên GPU, mô hình được tinh chỉnh theo hướng Parameter-Efficient Fine-Tuning (PEFT) bằng LoRA (Hu et al., 2021). Các lớp mục tiêu gồm các projection của attention và MLP: q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj. Trong thực nghiệm FP16, các tham số LoRA trainable được giữ ở FP32 để ổn định gradient (và tránh lỗi unscale FP16 gradients khi dùng GradScaler).

Tài nguyên và song song dữ liệu. Huấn luyện được thực hiện trên 2 GPU (DDP) thông qua torchrun --nproc_per_node=2.

Bảng 11: Cấu hình huấn luyện LoRA cho Qwen trên VLSP.

Thành phần	Thiết lập
Mô hình nền	Qwen/ Qwen2.5-1.5B-Instruct
Chiều dài chuỗi Batch	max_seq_length=320 per_device=4, grad_acc=8, 2 GPU \Rightarrow effective ≈ 64
Tối ưu	AdamW (adamw_torch), $lr=2 \cdot 10^{-4}$, warmup_ratio=0.03
Số bước	max_steps=8000 (≈ 1 epoch)
LoRA Precision	$r=16$, $\alpha=32$, dropout=0.05 FP16 (tham số LoRA trainable ở FP32)
Đánh giá trong train	theo eval_loss mỗi 800 steps

3.2.5 Tối ưu hyperparameters

Việc lựa chọn siêu tham số được dẫn dắt bởi (i) thống kê độ dài token ở Bảng 10, và (ii) giới hạn bộ nhớ khi huấn luyện/giải mã. Cụ

thể: (i) max_seq_length=320 được chọn để bao phủ xấp xỉ p99 (304 tokens) và giảm chi phí; (ii) batch hiệu dụng ≈ 64 giúp huấn luyện ổn định; (iii) max_new_tokens=128 dùng khi đánh giá để hạn chế hiện tượng sinh lan man. Trong chiều VI \rightarrow EN, dữ liệu đầu vào đôi khi chứa viết tắt/bảng biểu, dễ gây lặp (repetition), do đó trong các thử nghiệm tiếp theo có thể bổ sung các ràng buộc giải mã như no_repeat_ngram_size và repetition_penalty.

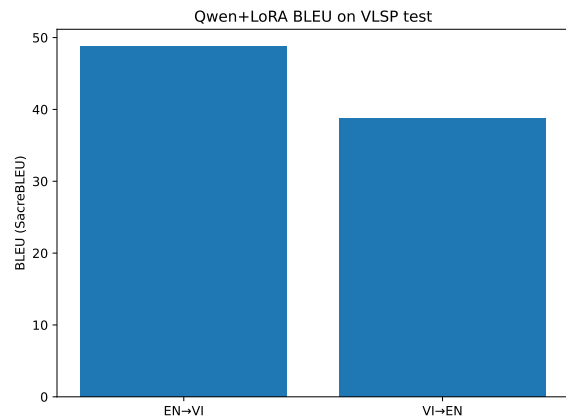
3.2.6 Đánh giá và kết quả

Thiết lập giải mã. Mô hình được đánh giá trên tập test bằng SacreBLEU (Post, 2018), tokenize 13a. Giải mã sử dụng beam search với num_beams=4, do_sample=False, và max_new_tokens=128. Để đảm bảo trích xuất đúng phần dự đoán, chuỗi sinh ra được cắt (prompt-cut) theo đúng chiều dài input đã padding trong batch.

Kết quả. Bảng 12 báo cáo BLEU cho hai chiều dịch. Kết quả cho thấy EN \rightarrow VI đạt BLEU cao hơn VI \rightarrow EN. Một nguyên nhân quan trọng là ở chiều VI \rightarrow EN, một số tham chiếu tiếng Anh có xu hướng rút gọn mạnh (telegraphic), trong khi mô hình sinh bản dịch đầy đủ theo câu nguồn, dẫn đến giảm BLEU do mismatch theo tham chiếu.

Bảng 12: Kết quả mô hình Qwen (LoRA) trên VLSP 2025 test.

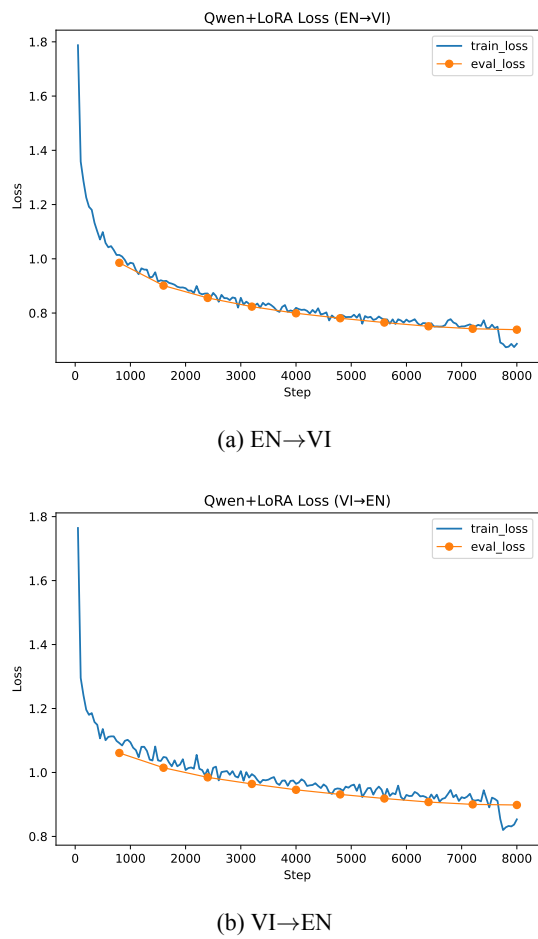
Hệ thống	BLEU (test)	Ghi chú
Qwen + LoRA (EN \rightarrow VI)	48.72	BP=0.953, ratio=0.954
Qwen + LoRA (VI \rightarrow EN)	38.76	BP=0.998, ratio=0.998



Hình 13: So sánh BLEU giữa hai chiều dịch (EN \rightarrow VI và VI \rightarrow EN).

3.2.7 Biểu đồ huấn luyện (Loss curves)

Trong quá trình huấn luyện, train_loss và eval_loss được ghi theo chu kỳ (logging_steps và eval_steps). Hình 14 minh hoạ diễn biến loss theo bước huấn luyện cho hai chiều dịch.



Hình 14: Đồ thị train/eval loss của Qwen+LoRA theo bước huấn luyện.

3.2.8 Phân tích lỗi (Error Analysis)

Chúng tôi phân tích lỗi dựa trên so sánh câu nguồn, tham chiếu và dự đoán trên tập test, tập trung vào ba nhóm lỗi điển hình:

Lỗi thực thể/tên riêng (Named Entity). Ở chiều EN→VI, mô hình đôi khi dịch sai hoặc “bịa” địa danh/tên riêng (ví dụ thay cụm địa danh gốc bằng một địa danh quen thuộc hơn).

Lỗi lặp (Repetition) trên đầu vào dạng viết tắt/bảng biểu. Ở chiều VI→EN, các chuỗi nhiều viết tắt có thể kích hoạt lặp cụm từ trong quá trình sinh, làm câu dự đoán dài bất thường.

Mismatch phong cách với tham chiếu (style/length mismatch).

Một số tham chiếu tiếng Anh rút gọn mạnh so với câu nguồn tiếng Việt; mô hình sinh bản dịch đầy đủ sẽ bị BLEU phạt dù về mặt trung thành nội dung có thể tốt.

Bảng 13: Ví dụ các nhóm lỗi điển hình quan sát được trên tập test (rút gọn hiển thị).

Loại lỗi	Nguồn	Tham chiếu	Dự đoán
Tên riêng	EN: ...	VI: ...	VI: ...
	Phone	Phone	Điện
	Hong and	Hong và	Biên Phủ ...
	Keo Oudom	Keo Oudom	
Lặp	VI: Đường	EN: ...	EN: Type
	máu BT	Normal	2 diabetes
	RLDNG...	Glycemia...	Mellitus ...
	Bảng 3.2.	Table 3.2.	(lặp)
Style mismatch	VI: ... thay	EN: ...	EN: ...
	van hai lá	ASSESSMENT	Comments
	... bệnh viện	THE	on some
	Trung ương	CLINICAL...	clinical...
	Huế		Hue
			Central
			Hospital

Hướng cải thiện. Đối với lỗi tên riêng, có thể bổ sung ràng buộc trong prompt như “giữ nguyên tên riêng/địa danh”, hoặc hậu xử lý bằng nhận dạng thực thể (NER) để hạn chế thay đổi thực thể. Đối với lỗi lặp ở VI→EN, có thể bổ sung no_repeat_ngram_size và repetition_penalty trong giải mã, đồng thời giảm max_new_tokens cho các mẫu ngắn/viết tắt.

4 Kết luận

Báo cáo trình bày hai hướng giải cho bài tập lớn dịch máy Anh–Việt: (i) tự cài đặt và huấn luyện Transformer Seq2Seq từ đầu, và (ii) thích nghi miền y tế cho VLSP 2025 bằng finetune Transformer và finetune Qwen2.5-1.5B-Instruct theo LoRA. Với bài toán chính, thực nghiệm cho thấy biến thể Pre-LN + GeLU hội tụ ổn định hơn cấu hình Post-Norm + ReLU, và việc dùng beam search (kèm length normalization) giúp cải thiện chất lượng so với greedy. Ở bài toán VLSP, finetune Transformer từ checkpoint pretrained giúp BLEU tăng từ 36.64 lên 41.08 (greedy) và từ 38.10 lên 42.26 (beam), trong khi Qwen+LoRA đạt BLEU

48.72 cho EN→VI và 38.76 cho VI→EN.

Phân tích lỗi cho thấy hạn chế chính nằm ở thuật ngữ y tế hiếm, thực thể/viết tắt và hiện tượng thiếu–thừa thông tin. Các hướng cải thiện gần nhất gồm chuẩn hoá/áp ràng buộc thuật ngữ, xử lý thực thể tốt hơn, và tối ưu giải mã để giảm lặp và kiểm soát độ dài.

References

- Edward J. Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.
- Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Maja Popović. 2015. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation (WMT)*.
- Matt Post. 2018. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation (WMT)*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

A Phụ lục

A.1 Tóm tắt cấu hình cuối cùng

Các bảng dưới đây tóm tắt các cấu hình *cuối cùng* được dùng để báo cáo kết quả trong bài.

A.2 Ghi chú đánh giá và giải mã

Decode. Greedy chọn token xác suất cao nhất mỗi bước; beam search dùng $k = 4$ và length normalization $\alpha = 0.6$. Giới hạn độ dài sinh dùng DECODE_MAX_LEN=120 (Transformer) và max_new_tokens=128 (Qwen).

Metric. BLEU corpus tính bằng SacreBLEU; phần Qwen dùng tokenize 13a.

Bảng 14: Cấu hình cuối cùng: Main Transformer (from scratch).

Thành phần	Thiết lập
Tokenizer	SentencePiece unigram; shared vocab=8000; pad/unk/bos/eos=0/1/2/3
Cắt/pad train	max_src_len=70, max_tgt_len=70 (subword); dynamic padding theo batch
Kiến trúc	Pre-LN Transformer; $d_{model} = 384$, $n_{heads} = 8$, 4 enc + 4 dec, $d_{ff} = 1536$, GeLU
Regularization	dropout=0.1; label smoothing $\epsilon = 0.1$
Tối ưu & schedule	Adam/AdamW; grad clip=1.0; AMP; Noam warmup=4000
Batch	32
Giải mã & đánh giá	greedy; beam=4, length norm $\alpha = 0.6$; max decode len=120; BLEU (SacreBLEU)

Bảng 15: Cấu hình cuối cùng: VLSP Transformer (finetune).

Thành phần	Thiết lập
Tiền xử lý	NFC + normalize whitespace; lọc max_len=200 (whitespace), max_ratio=9; dev=2% (seed=42)
Tokenizer	SentencePiece unigram; shared vocab=8000; pad/unk/bos/eos=0/1/2/3
Khởi tạo	load ckpt pretrained từ bài toán chính; finetune full params
Cắt/pad train	tối đa 80 subword tokens cho nguồn và đích
Tối ưu	AdamW lr= 3×10^{-4} ; wd=0.01; $\beta = (0.9, 0.98)$; AMP
Batch/Epoch	batch=64; epoch=3; chọn ckpt tốt nhất theo BLEU dev
Đánh giá	BLEU corpus (SacreBLEU); detok SentencePiece + normalize whitespace

Bảng 16: Cấu hình cuối cùng: Qwen2.5-1.5B-Instruct + LoRA (VLSP).

Thành phần	Thiết lập
Base model	Qwen/Qwen2.5-1.5B-Instruct
Data format	instruction–response; mask loss phần prompt bằng nhân -100
Max length	max_seq_length=320
LoRA	r=16, $\alpha=32$, dropout=0.05; target: q/k/v/o_proj + gate/up/down_proj
Train	2 GPU DDP; per_device=4; grad_acc=8 (effective ≈ 64)
Optimizer	AdamW lr= 2×10^{-4} ; warmup_ratio=0.03; max_steps=8000
Decode/Eval	num_beams=4; do_sample=False; max_new_tokens=128; SacreBLEU tok=13a