

# GIÁO TRÌNH NLP TOÀN TẬP

Đinh Công Thái

Ngày 25 tháng 8 năm 2025



# Mục lục

Lời nói đầu	1
<b>I BÁCH KHOA TOÀN THƯ VỀ LÝ THUYẾT NLP</b>	<b>3</b>
<b>1 NHẬP MÔN &amp; CÁC NGUYÊN LÝ NỀN TẢNG</b>	<b>7</b>
1.1 Định nghĩa và Phạm vi . . . . .	7
1.1.1 Xử lý Ngôn ngữ Tự nhiên (NLP) là gì? . . . . .	7
1.1.2 Mỗi quan hệ với Trí tuệ Nhân tạo, Học máy và Ngôn ngữ học . . . . .	8
1.1.3 Tổng quan các Kỷ nguyên Phát triển (Luật lệ → Thống kê → Học sâu → Transformer) . . . . .	9
1.2 Nền tảng Ngôn ngữ học cho NLP . . . . .	10
1.2.1 Các cấp độ phân tích ngôn ngữ: Hình thái học, Cú pháp, Ngữ nghĩa, Ngữ dụng . . . . .	11
1.2.2 Ngữ pháp hình thức (Formal Grammars): Ngữ pháp phi ngữ cảnh (CFG) . . . . .	12
1.2.3 Các lý thuyết cú pháp chính: Cú pháp thành phần (Constituency) và Cú pháp phụ thuộc (Dependency) . . . . .	13
1.3 Các Bài toán Cốt lõi trong NLP . . . . .	14
1.3.1 Phân tích cấp độ từ . . . . .	14
1.3.2 Phân tích cấp độ câu . . . . .	14
1.3.3 Phân tích ngữ nghĩa . . . . .	15
1.3.4 Các bài toán Ứng dụng . . . . .	16
1.4 Các Kiến thức Toán học Cần trang bị . . . . .	16
1.4.1 Đại số Tuyến tính: Vector, Ma trận, Tensor, SVD . . . . .	16
1.4.2 Giải tích: Đạo hàm, Gradient, Chain Rule . . . . .	17
1.4.3 Xác suất và Thống kê: Định lý Bayes, MLE . . . . .	18
1.4.4 Lý thuyết Thông tin: Entropy, Cross-Entropy, KL Divergence . . . . .	18
1.5 Đạo đức và Trách nhiệm trong NLP (Ethics & Responsible AI) . . . . .	19
1.5.1 Các loại Thiên kiến (Bias) trong Dữ liệu và Mô hình . . . . .	19
1.5.2 Tính Công bằng, Minh bạch và Diễn giải được (Fairness, Transparency, Interpretability) . . . . .	20
1.5.3 Quyền riêng tư và An toàn Dữ liệu (Federated Learning, Differential Privacy) . . . . .	20
1.5.4 Mỗi nguy về Thông tin Sai lệch và Lạm dụng . . . . .	20
1.5.5 Tính Bền vững và Tấn công Giả mạo (Robustness & Adversarial Attacks) . . . . .	21
<b>2 BIỂU DIỄN VĂN BẢN VÀ CÁC MÔ HÌNH THỐNG KÊ</b>	<b>23</b>
2.1 Biểu diễn Dựa trên Tần suất (Bag-of-Words, TF-IDF) . . . . .	23
2.1.1 Túi từ (Bag-of-Words - BoW) . . . . .	23
2.1.2 TF-IDF: Trọng số hóa tầm quan trọng của từ . . . . .	25
2.2 Mô hình Ngôn ngữ Thống kê (N-gram, Smoothing) . . . . .	26
2.2.1 Mô hình N-gram: Học từ Lịch sử Gần nhất . . . . .	27
2.2.2 Làm mịn (Smoothing): Giải quyết vấn đề Xác suất Zero . . . . .	28
2.2.3 Tổng kết và Hạn chế của N-gram . . . . .	30
2.3 Các thuật toán Học máy Thống kê . . . . .	30
2.3.1 Các mô hình Phân loại (Classification Models) . . . . .	30

---

2.3.2	Các mô hình Học chuỗi (Sequence Labeling Models) . . . . .	34
2.4	Mô hình hóa Chủ đề (Topic Modeling - LDA) . . . . .	36
2.4.1	Tư duy cốt lõi và bài toán . . . . .	36
2.4.2	Câu chuyện sinh dữ liệu của LDA (The Generative Story) . . . . .	36
2.4.3	Huấn luyện và Suy luận trong LDA . . . . .	37
2.4.4	Kết quả và Ứng dụng . . . . .	38
2.4.5	Ưu và Nhuược điểm . . . . .	39
<b>3</b>	<b>CÁC KIẾN TRÚC MẠNG NƠ-RON KINH ĐIỂN</b>	<b>41</b>
3.1	Word Embeddings: Cuộc cách mạng Biểu diễn Đầu tiên . . . . .	41
3.1.1	Lý thuyết Word2Vec: CBOW & Skip-gram . . . . .	41
3.1.2	Lý thuyết GloVe: Ma trận Đồng xuất hiện . . . . .	47
3.1.3	Lý thuyết FastText: Biểu diễn Subword . . . . .	50
3.2	Autoencoders: Học Biểu diễn Phi giám sát . . . . .	52
3.2.1	Trực giác và Kiến trúc . . . . .	52
3.2.2	Sức mạnh của "Nút cổ chai" và việc học biểu diễn . . . . .	53
3.2.3	Ứng dụng và Các biến thể trong NLP . . . . .	54
3.3	Mạng Nơ-ron Hồi tiếp (RNNs, LSTMs, GRUs) . . . . .	55
3.3.1	Mạng Nơ-ron Hồi tiếp Đơn giản (Vanilla RNN) . . . . .	55
3.3.2	Long Short-Term Memory (LSTM) . . . . .	57
3.3.3	Gated Recurrent Unit (GRU) . . . . .	59
3.4	Mô hình Chuỗi-sang-Chuỗi (Seq2Seq) và Cơ chế Chú ý (Attention) . . . . .	60
3.4.1	Kiến trúc Encoder-Decoder của Seq2Seq . . . . .	60
3.4.2	Cơ chế Chú ý (Attention Mechanism) . . . . .	63
3.5	Mạng Nơ-ron Tích chập (CNNs) cho NLP . . . . .	66
3.5.1	Từ Ánh 2D đến Văn bản 1D: Một sự tương đồng . . . . .	66
3.5.2	Kiến trúc và Dòng chảy Dữ liệu . . . . .	67
3.5.3	Ưu và Nhuược điểm của CNN cho NLP . . . . .	70
3.5.4	Các Kiến trúc CNN Nâng cao và Ứng dụng . . . . .	70
3.5.5	So sánh CNN và RNN cho NLP . . . . .	70
3.6	Contextualized Word Embeddings: Biểu diễn từ theo ngữ cảnh . . . . .	71
3.6.1	ELMo: Bước đệm tới Embeddings theo ngữ cảnh . . . . .	71
3.6.2	ULMFiT: Công thức Vàng cho Học Chuyển giao trong NLP . . . . .	74
3.7	Các Kiến trúc Nơ-ron Khác . . . . .	76
3.7.1	Mạng Siamese (Siamese Networks) cho việc học sự tương đồng . . . . .	76
3.7.2	Mạng Nơ-ron Đồ thị (Graph Neural Networks - GNNs) trong NLP . . . . .	79
3.8	Các Mô hình Sinh Tiền-Transformer (Pre-Transformer Generative Models) . . . . .	81
3.8.1	Variational Autoencoder (VAE) cho Văn bản . . . . .	82
3.8.2	Generative Adversarial Networks (GANs) cho Văn bản . . . . .	82
3.8.3	So sánh VAE và GAN trong Sinh Văn bản . . . . .	83
3.8.4	Chuyển tiếp sang Kỷ nguyên Transformer . . . . .	83
<b>4</b>	<b>KỶ NGUYÊN TRANSFORMER VÀ CÁC MÔ HÌNH NGÔN NGỮ LỚN (LLMs)</b>	<b>85</b>
4.1	Kiến trúc Transformer Gốc ("Attention Is All You Need") . . . . .	85
4.1.1	Chi tiết cơ chế Self-Attention: Scaled Dot-Product và Multi-Head Attention . . . . .	85
4.1.2	Kiến trúc Encoder-Decoder và các thành phần phụ . . . . .	88
4.1.3	Positional Encoding: Thêm thông tin về vị trí . . . . .	90
4.2	Phân loại Kiến trúc LLMs . . . . .	91
4.2.1	Họ Mô hình chỉ Encoder (Encoder-Only): Bậc thầy của Sự Hiểu Ngôn ngữ . . . . .	91
4.2.2	Họ Mô hình chỉ Decoder (Decoder-Only): Bậc thầy của Sự Sáng tạo . . . . .	94
4.2.3	Họ Mô hình Encoder-Decoder: Sự kết hợp của hai thế giới . . . . .	97
4.3	Các Kỹ thuật Tokenization Hiện đại . . . . .	99

4.3.1	Byte-Pair Encoding (BPE) . . . . .	100
4.3.2	WordPiece . . . . .	101
4.3.3	SentencePiece . . . . .	102
4.4	Các Biến thể Transformer cho Ngữ cảnh dài (Long-Context Transformers) . . . . .	103
4.4.1	Vấn đề của Sự chú ý Bậc hai (The Quadratic Bottleneck) . . . . .	103
4.4.2	Lý thuyết về Chú ý Thưa thớt (Sparse Attention) . . . . .	104
4.4.3	Các phương pháp khác: Thoát khỏi Self-Attention Bậc hai . . . . .	105
4.5	Lý thuyết về Mixture-of-Experts (MoE) . . . . .	107
4.5.1	Kiến trúc và Nguyên lý hoạt động . . . . .	107
4.5.2	Thách thức trong việc Huấn luyện MoE . . . . .	109
4.5.3	Tổng kết về MoE . . . . .	110
<b>5</b>	<b>LÝ THUYẾT VỀ TINH CHỈNH VÀ CĂN CHỈNH MÔ HÌNH</b>	<b>111</b>
5.1	Các Phương pháp Tinh chỉnh (Fine-tuning) . . . . .	111
5.1.1	Học Chuyển giao (Transfer Learning): Nền tảng Tư duy . . . . .	111
5.1.2	Tinh chỉnh Toàn bộ (Full Fine-tuning) . . . . .	112
5.2	Kỹ thuật Tinh chỉnh Hiệu quả Tham số (PEFT) . . . . .	113
5.2.1	LoRA: Thích ứng Thứ hạng Thấp (Low-Rank Adaptation) . . . . .	114
5.2.2	QLoRA: Lượng tử hóa LoRA (Quantized LoRA) . . . . .	115
5.2.3	Adapter-tuning . . . . .	116
5.2.4	Prompt-based Tuning: Điều chỉnh trên Lớp Đầu vào . . . . .	116
5.3	Kỹ thuật Prompt Engineering và Học trong Ngữ cảnh (In-context Learning) . . . . .	116
5.3.1	Các cấp độ của Học trong Ngữ cảnh: Zero-shot, One-shot, Few-shot . . . . .	117
5.3.2	Chuỗi Suy luận (Chain-of-Thought - CoT) Prompting . . . . .	118
5.3.3	Các kỹ thuật nâng cao dựa trên CoT . . . . .	119
5.4	Tinh chỉnh theo chỉ dẫn (Instruction Tuning) . . . . .	120
5.4.1	Vấn đề của các LLM cơ sở (Base LLMs) . . . . .	120
5.4.2	Tư duy cốt lõi và Cơ chế hoạt động . . . . .	120
5.4.3	Xây dựng Bộ dữ liệu Chỉ dẫn: Chìa khóa thành công . . . . .	120
5.4.4	Vai trò và Tác động của Instruction Tuning . . . . .	121
5.5	Các phương pháp Căn chỉnh (Alignment) với Con người . . . . .	122
5.5.1	Mục tiêu Căn chỉnh: Bộ ba H (The 3 H's) . . . . .	122
5.5.2	Học Tăng cường từ Phản hồi Con người (RLHF) . . . . .	123
5.5.3	Các phương pháp thay thế: Hướng tới sự Đơn giản và Ổn định . . . . .	124
<b>6</b>	<b>CÁC HỆ THỐNG VÀ KIẾN TRÚC NÂNG CAO</b>	<b>127</b>
6.1	Tìm kiếm và Sinh Tăng cường (Retrieval-Augmented Generation - RAG) . . . . .	127
6.1.1	Tại sao và Khi nào chúng ta cần RAG? . . . . .	127
6.1.2	Kiến trúc RAG Cơ bản (The Naive RAG Pipeline) . . . . .	128
6.1.3	Bài học từ các Công cụ Tìm kiếm: Kiến trúc Phân tầng của Google . . . . .	129
6.1.4	Tối ưu hóa Giai đoạn 1: Lập chỉ mục Thông minh (Smarter Indexing) . . . . .	130
6.1.5	Tối ưu hóa Giai đoạn 2: Tìm kiếm và Xếp hạng (Advanced Retrieval & Ranking) . . . . .	131
6.1.6	Tối ưu hóa Giai đoạn 2: Tìm kiếm và Xếp hạng (Advanced Retrieval & Ranking) . . . . .	131
6.1.7	Tối ưu hóa Giai đoạn 3: Tăng cường và Sinh (Smarter Augmentation & Generation) . . . . .	132
6.1.8	Các Mô hình RAG Nâng cao (Advanced RAG Patterns) . . . . .	132
6.1.9	Dánh giá và Triển khai Hệ thống RAG . . . . .	137
6.2	Lý thuyết về Tối ưu hóa Mô hình . . . . .	138
6.2.1	Chứng cất Tri thức (Knowledge Distillation) . . . . .	138
6.2.2	Lượng tử hóa (Quantization) . . . . .	139
6.2.3	Cắt tỉa (Pruning) . . . . .	140
6.3	Mô hình Đa phương thức (Multimodal Models) . . . . .	141

6.3.1	Nền tảng: Xây dựng cầu nối giữa Ânh và Chữ . . . . .	142
6.3.2	Ví dụ Kiến trúc: Kết hợp LLM với Thị giác . . . . .	144
6.3.3	Sinh Đa phương thức (Multimodal Generation) . . . . .	145
6.3.4	Kiểm soát Quá trình Sinh (Controlled Generation) . . . . .	147
6.3.5	Sinh Nhận biết Cấu trúc (Structure-aware Generation) . . . . .	148
6.4	Hệ thống Tác tử (Agentic Systems) với LLMs . . . . .	149
6.4.1	Vòng lặp Suy luận-Hành động: Khung sườn ReAct . . . . .	149
6.4.2	Lập kế hoạch và Sử dụng Công cụ (Planning & Tool Use) . . . . .	150
6.4.3	Bộ nhớ cho Tác tử (Agent Memory) . . . . .	150
6.4.4	Tự sửa lỗi (Self-correction) . . . . .	151
6.5	Mô hình Thế giới (World Models) và Mô phỏng . . . . .	151
6.5.1	Khái niệm về khả năng LLM xây dựng mô hình nội tại . . . . .	152
6.5.2	Hướng nghiên cứu và Tiềm năng . . . . .	153
6.6	Các Kỹ thuật Biểu diễn Nâng cao cho Truy xuất . . . . .	154
6.6.1	Biểu diễn Đa ngôn ngữ (Cross-lingual Embeddings) . . . . .	154
6.6.2	Biểu diễn Thưa và Tối ưu cho Truy xuất (Sparse & Retriever-specific Embeddings)	155
<b>7</b>	<b>LÝ THUYẾT ĐÁNH GIÁ VÀ CÁC BENCHMARK</b>	<b>157</b>
7.1	Các Metric kinh điển . . . . .	157
7.1.1	Perplexity: Đo lường chất lượng của Mô hình Ngôn ngữ . . . . .	157
7.1.2	Precision, Recall, và F1-Score: Nền tảng của các bài toán Phân loại . . . . .	157
7.1.3	BLEU: Đánh giá Dịch máy . . . . .	159
7.1.4	ROUGE: Đánh giá Tóm tắt Văn bản . . . . .	159
7.2	Các Benchmark Kinh điển và Hiện đại . . . . .	160
7.2.1	Kỷ nguyên Pre-LLM: GLUE và SuperGLUE . . . . .	160
7.2.2	Kỷ nguyên LLM: Đánh giá các Khả năng Nổi trội (Emergent Abilities) . . . . .	160
7.3	Thách thức trong Đánh giá LLM . . . . .	162
7.3.1	Nhiễm bẩn Dữ liệu (Data Contamination) . . . . .	162
7.3.2	Đánh giá bởi Con người vs. Đánh giá Tự động (Human vs. Auto Evaluation) . . . . .	163
7.3.3	Định luật Goodhart và "Dạy cho Bài kiểm tra" (Goodhart's Law & "Teaching to the Test") . . . . .	163
7.4	Đánh giá dựa trên LLM (LLM-as-a-Judge) . . . . .	164
7.4.1	Nguyên lý và Cách tiếp cận . . . . .	164
7.4.2	Ứng dụng và Tiềm năng . . . . .	165
7.4.3	Thách thức và Những điều cần Lưu ý . . . . .	166
<b>II</b>	<b>CẨM NANG KỸ THUẬT NLP THỰC CHIẾN</b>	<b>169</b>
<b>1</b>	<b>QUY TRÌNH LÀM VIỆC VÀ CÔNG CỤ XỬ LÝ DỮ LIỆU</b>	<b>173</b>
1.1	Phác thảo Quy trình Dự án NLP Tinh gọn . . . . .	173
1.1.1	Vòng lặp 0: Xác định Bài toán và Thiết lập Đường cơ sở (Baseline) . . . . .	174
1.1.2	Vòng lặp 1: Phát triển Mô hình (Model Development) . . . . .	175
1.1.3	Vòng lặp 2: Triển khai và Giám sát (Deployment & Monitoring) . . . . .	175
1.2	Kỹ thuật Thu thập Dữ liệu . . . . .	176
1.2.1	Sử dụng APIs: Cách tiếp cận "Lịch sự" và Đáng tin cậy . . . . .	176
1.2.2	Web Scraping: Trích xuất Dữ liệu từ HTML . . . . .	177
1.3	Làm sạch, Tiền xử lý và Gán nhãn Dữ liệu . . . . .	179
1.3.1	Thao tác Dữ liệu quy mô lớn: Pandas, NumPy, và Polars . . . . .	179
1.3.2	Gán nhãn Dữ liệu: Từ Thủ công đến Tự động . . . . .	179
1.3.3	Xử lý Dữ liệu Mất cân bằng (Handling Imbalanced Data) . . . . .	181
1.4	Kỹ thuật Tăng cường Dữ liệu (Data Augmentation) . . . . .	181

1.4.1	EDA: Các Phép toán Thay thế Đơn giản . . . . .	182
1.4.2	Back-Translation: Tận dụng Sức mạnh của Dịch máy . . . . .	182
1.4.3	Tăng cường Dữ liệu dựa trên LLM . . . . .	183
1.4.4	Lựa chọn Kỹ thuật Tăng cường . . . . .	183
1.5	Quản lý Dữ liệu và Phiên bản (Data Version Control - DVC) . . . . .	184
1.5.1	DVC là gì và Triết lý hoạt động . . . . .	184
1.5.2	DVC Pipelines: Tái lập toàn bộ Thủ nghiệm . . . . .	185
1.5.3	Lợi ích của việc sử dụng DVC . . . . .	186
2	<b>HUẤN LUYỆN VÀ ĐÁNH GIÁ MÔ HÌNH</b>	<b>187</b>
2.1	Thiết lập Môi trường với Hệ sinh thái Hugging Face . . . . .	187
2.1.1	Transformers: Trái tim của Hệ sinh thái . . . . .	188
2.1.2	Datasets: Quản lý Dữ liệu Hiệu quả . . . . .	188
2.1.3	Accelerate: Đơn giản hóa Huấn luyện Phân tán . . . . .	189
2.1.4	Evaluate: Một Thư viện Metric Toàn diện . . . . .	189
2.2	Kỹ thuật Đánh giá Thực tế . . . . .	190
2.2.1	Hàm <code>compute_metrics</code> : Cầu nối giữa Mô hình và Metric . . . . .	190
2.2.2	Đánh giá các Tác vụ Phức tạp hơn . . . . .	191
2.3	Theo dõi và Quản lý Thí nghiệm . . . . .	194
2.3.1	Weights & Biases (W&B): Một Nền tảng Toàn diện và Trực quan . . . . .	194
2.3.2	MLflow: Một Lựa chọn Mã nguồn mở và Linh hoạt . . . . .	196
2.4	Huấn luyện Phân tán (Distributed Training) . . . . .	197
2.4.1	Các Mô hình Song song (Parallelism Paradigms) . . . . .	197
2.4.2	Sử dụng ‘accelerate’ và ‘DeepSpeed’ để Huấn luyện Mô hình lớn . . . . .	197
3	<b>TỐI ƯU HÓA, TRIỂN KHAI VÀ VẬN HÀNH (MLOps)</b>	<b>201</b>
3.1	Cơ sở dữ liệu Vector (Vector Databases) . . . . .	201
3.1.1	Khái niệm và vai trò trong NLP hiện đại . . . . .	201
3.1.2	Các công cụ phổ biến . . . . .	202
3.2	Tối ưu hóa Mô hình với các Thư viện Sẵn có . . . . .	202
3.3	Đóng gói và Phục vụ Mô hình . . . . .	204
3.3.1	Xây dựng API với FastAPI . . . . .	204
3.3.2	Container hóa với Docker . . . . .	205
3.4	Các Framework Phục vụ Chuyên dụng cho Production . . . . .	206
3.5	Vòng đời MLOps cho LLM . . . . .	207
3.5.1	Giám sát mô hình (Model Monitoring) . . . . .	207
3.5.2	Quy trình Tái huấn luyện và Cập nhật Mô hình . . . . .	207
4	<b>CÔNG THỨC XÂY DỰNG CÁC ỨNG DỤNG PHỔ BIẾN (RECIPES)</b>	<b>209</b>
4.1	Xây dựng Hệ thống Tìm kiếm Ngữ nghĩa . . . . .	209
4.2	Xây dựng Hệ thống Hỏi-Đáp trên Tài liệu (RAG) . . . . .	210
4.3	Recipe 3: Fine-tuning một LLM cho Tác vụ Phân loại . . . . .	212
4.4	Recipe 4: Fine-tuning cho Tác vụ Nhận dạng Thực thể (NER) . . . . .	214
4.5	Recipe 5: Xây dựng một Agent đơn giản với khả năng sử dụng công cụ . . . . .	216
4.6	Recipe 6: Trích xuất Thông tin có Cấu trúc từ Văn bản . . . . .	218
	<b>Lời kết</b>	<b>229</b>

## MỤC LỤC

---

# Lời nói đầu

Xin chào, mình là **Đinh Công Thái**, sinh viên lớp IT1-K67, ngành Khoa học Máy tính, Trường Đại học Bách khoa Hà Nội. Nói thật thì mình là một sinh viên học dốt, thậm chí từng có giai đoạn chẳng biết mình học IT để làm gì. Trước đó vào cấp 3 mình có giải HSG quốc gia, nên mình chọn vào IT1 cho oai chứ cũng chẳng biết gì về lập trình :))))

Mọi thứ bắt đầu thay đổi khi mình may mắn được PGS.TS Nguyễn Phi Lê nhận vào lab IoT, viện AI4LIFE và được các anh chị, đặc biệt là anh Dũng mentor của mình dẫn dắt. Từ chỗ lạc trôi giữa mớ kiến thức IT, mình bắt đầu tìm được một niềm đam mê thực sự với lập trình, đặc biệt là trí tuệ nhân tạo. Nói đơn giản, trước đó mình như Malzahar không có nước mắt, còn bây giờ ít nhất được rùng nhợng Blue.

## Vậy cuốn giáo trình này ra đời để làm gì?

Mình (và ChatGPT :>) viết nó cho những bạn đang ở vị trí giống mình trước đây: mơ hồ, không định hướng, không biết AI bắt đầu từ đâu, nhưng có chút tò mò và muốn thử. Nội dung tập trung vào **Xử lý ngôn ngữ tự nhiên (NLP)** – vì theo mình đây là mảng vừa gần gũi (ngôn ngữ mà), vừa thực tiễn (từ chatbot cho đến dịch máy), lại vừa thú vị (dạy LLMs nói tục).

Sách này không hứa biến bạn thành chuyên gia sau một đêm (vì mình cũng đêch phải chuyên gia). Nhưng nó sẽ là một bản đồ để bạn không bị “lạc trôi” quá lâu, và nếu thấy hứng thú thì có thể đi xa hơn, sớm hơn, không bị muộn như mình.

Thực ra, để nói đúng thì cuốn giáo trình này không phải kiểu “tác phẩm hàn lâm” mà tác giả ngồi gõ từng chữ từ A đến Z. Cách làm của mình khá *sinh viên kỹ thuật*, và có thể tóm tắt như sau:

- **Soạn mục lục** → dựng cái khung nhà cho con chatbot.
- **Tìm & gom paper/tài liệu**, chứ không bọn LLM nó lại bịa tung tóe lên, đồng thời cũng để cho kiến thức được sâu sắc và bám sát.
- **Chia thư mục theo từng phần/tiểu mục** → giữ cấu trúc rõ ràng.
- **Soạn rule** → chính là *style guide*: văn phong, yêu cầu ví dụ, công thức, hình minh họa... để LLM không viết thành tiểu thuyết ngôn tình.
- **Tool 1 (lưu context)** → giúp LLM nhớ liền mạch, không quên mình vừa viết gì ở trang trước.
- **Tool 2 (API xuất bản)** → gửi query theo từng mục và nhận lại nội dung thành file tương ứng.
- **Tool 3 (tự động review & bổ sung)** → LLM đọc lại chính nội dung đã sinh, so với rule + nguồn, rồi nhận xét/gợi ý sửa.

Kết quả cuối cùng: phần kiến thức cốt lõi thì mình đã kiểm tra lại cẩn thận. Còn nếu bạn bắt gặp vài chỗ đọc hơi “lạ lạ”, thậm chí ngớ ngẩn thì có thể đó là *hallucination*. Hãy coi nó như chút gia vị AI trong cuốn sách vở vẫn này. Còn bug thì... sinh viên kỹ thuật tự mình vẫn sống chung với bug cả thôi :))). Mình chia sẻ để nếu như có bạn nào cũng muốn tự viết một giáo trình hoặc một bản ghi tóm tắt, khái quát lại kiến thức mình đã học được.

## Cấu trúc cuốn sách

- **Phần 1: Lý thuyết NLP** – kể câu chuyện từ những nền tảng cơ bản cho tới các mô hình hiện đại như Transformer. Đọc phần này để biết “tại sao lại như vậy”.
- **Phần 2: Thực chiến NLP** – biến lý thuyết thành code, dự án, ứng dụng. Đọc phần này để biết “làm thế nào”.

Hai phần này bổ sung cho nhau. Biết lý thuyết mà không thực hành thì dễ “nói cho vui”. Còn chỉ thực hành mà không hiểu gốc thì cũng dễ trở thành “copy-paste developer”.

**Tóm cái máy lại**

Học NLP hay bất kỳ thứ gì cũng giống như học thổi sáo (mình biết thổi sáo rồi): lúc đầu sẽ thấy chán, thấy khó, thấy mình thổi xịt nốt suốt. Nhưng kiên nhẫn một chút, bạn sẽ bắt đầu nghe ra giai điệu. Và biết đâu, chính bạn sẽ sáng tác được một bản nhạc mới cho riêng mình.

Chúc bạn đọc sách vui vẻ và tìm thấy đam mê thật sự của mình.

*Trân trọng,*  
Đinh Công Thái

## **Phần I**

# **BÁCH KHOA TOÀN THƯ VỀ LÝ THUYẾT NLP**



---

*Chào mừng bạn đến với nền tảng lý thuyết của Xử lý Ngôn ngữ Tự nhiên. Mục tiêu của Phần 1 là cung cấp một cái nhìn toàn diện, sâu sắc và có hệ thống về TẤT CẢ các khái niệm, kiến trúc và thuật toán lý thuyết đã định hình nên ngành NLP. Chúng ta sẽ bắt đầu từ những nguyên lý ngôn ngữ học cơ bản, đi qua kỷ nguyên thống kê, khám phá các kiến trúc mạng nơ-ron kinh điển, và cuối cùng là làm chủ kiến trúc Transformer - nền tảng của các mô hình ngôn ngữ lớn hiện đại. Mỗi chương là một khối kiến thức độc lập nhưng có sự kết nối chặt chẽ, cùng nhau xây dựng nên một nền tảng vững chắc để bạn có thể hiểu "tại sao" và "như thế nào" các công nghệ NLP hoạt động. Hãy bắt đầu hành trình khám phá này.*



# Chương 1

## NHẬP MÔN & CÁC NGUYÊN LÝ NỀN TẢNG

Chào mừng bạn đến với thế giới của Xử lý Ngôn ngữ Tự nhiên (Natural Language Processing - NLP). Trong chương mở đầu này, chúng ta sẽ cùng nhau xây dựng một nền tảng vững chắc, định hình bức tranh toàn cảnh về lĩnh vực hấp dẫn và đầy tiềm năng này. Chúng ta sẽ bắt đầu từ những định nghĩa sơ khởi nhất, khám phá mối liên hệ của NLP với các ngành khoa học khác, nhìn lại chặng đường phát triển đầy biến động, và quan trọng hơn cả, nhận thức được những bài toán cốt lõi và trách nhiệm đạo đức đi kèm khi làm việc với ngôn ngữ – công cụ giao tiếp tinh vi nhất của con người.

### 1.1. Định nghĩa và Phạm vi

Trước khi đi sâu vào các thuật toán phức tạp hay những mô hình khổng lồ, việc đầu tiên và quan trọng nhất là phải trả lời được câu hỏi: "Chính xác thì chúng ta đang nghiên cứu về cái gì?". Mục này sẽ vạch ra ranh giới, định nghĩa và phạm vi của ngành Xử lý Ngôn ngữ Tự nhiên.

#### 1.1.1. Xử lý Ngôn ngữ Tự nhiên (NLP) là gì?

Một cách tổng quát nhất, Xử lý Ngôn ngữ Tự nhiên là một lĩnh vực liên ngành của Trí tuệ Nhân tạo (AI), Khoa học Máy tính và Ngôn ngữ học, tập trung vào việc trao cho máy tính khả năng **hiểu, diễn giải, xử lý và tạo ra ngôn ngữ của con người** (dưới dạng văn bản hoặc giọng nói) theo một cách có giá trị và hữu ích.

Mục tiêu cuối cùng của NLP không chỉ dừng lại ở việc xử lý các chuỗi ký tự, mà là thu hẹp khoảng cách giao tiếp giữa con người và máy tính. Chúng ta muốn máy tính không chỉ "đọc" được văn bản, mà còn "hiểu" được ý nghĩa, ngữ cảnh, cảm xúc, và thậm chí cả những ẩn ý đằng sau con chữ.

#### Dịnh nghĩa 1: Xử lý Ngôn ngữ Tự nhiên (NLP)

NLP là một lĩnh vực của Trí tuệ Nhân tạo nhằm mục đích xây dựng các hệ thống tính toán có khả năng xử lý và hiểu được ngôn ngữ tự nhiên của con người để thực hiện các tác vụ hữu ích.

Lĩnh vực NLP bao gồm hai khía cạnh chính, thường hoạt động song hành với nhau:

- **Hiểu Ngôn ngữ Tự nhiên (Natural Language Understanding - NLU):** Đây là khía cạnh tập trung vào việc "đọc hiểu" của máy tính. NLU bao gồm các bài toán yêu cầu máy tính phân tích và rút ra ý nghĩa từ một đoạn văn bản. Các nhiệm vụ như phân tích cú pháp, nhận dạng thực thể, hay phân tích cảm xúc đều thuộc về NLU.
- **Sinh Ngôn ngữ Tự nhiên (Natural Language Generation - NLG):** Đây là khía cạnh tập trung vào việc "viết" hoặc "nói" của máy tính. NLG bao gồm các bài toán yêu cầu máy tính tự động tạo ra một đoạn văn bản mạch lạc, tự nhiên và phù hợp với ngữ cảnh. Các nhiệm vụ như dịch máy, tóm tắt văn bản, hay xây dựng các chatbot trả lời tự động đều là ví dụ điển hình của NLG.

Thử thách lớn nhất của NLP nằm ở chính bản chất phức tạp và mơ hồ của ngôn ngữ tự nhiên. Ngôn ngữ của con người không tuân theo các quy tắc logic chặt chẽ như ngôn ngữ lập trình. Nó đầy rẫy sự đa nghĩa (một từ có nhiều nghĩa), phụ thuộc vào ngữ cảnh, chứa đựng tiếng lóng, sai sót chính tả, và liên tục biến đổi. Việc mô hình hóa và xử lý được những đặc tính này chính là trọng tâm của ngành.

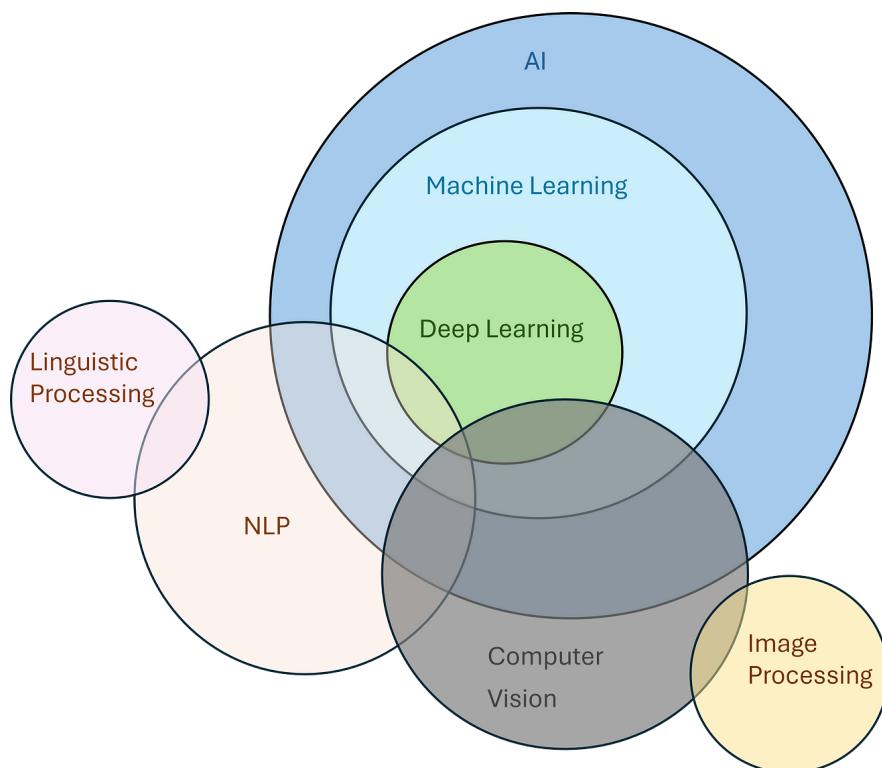
### Ví dụ 1: Một số ứng dụng thực tế của NLP

Để hình dung rõ hơn, hãy xem xét các ứng dụng của NLP mà chúng ta thường dùng hàng ngày:

- **Máy tìm kiếm (Search Engines):** Google, Bing sử dụng NLP để hiểu câu truy vấn của bạn (dù bạn gõ sai chính tả hay dùng từ ngữ thông tục) và trả về các kết quả liên quan nhất.
- **Trợ lý ảo (Virtual Assistants):** Siri, Google Assistant, và Alexa dùng NLP để hiểu mệnh lệnh bằng giọng nói và sinh ra câu trả lời phù hợp.
- **Dịch tự động (Machine Translation):** Google Translate có thể dịch tức thời giữa hàng trăm ngôn ngữ.
- **Phân tích cảm xúc (Sentiment Analysis):** Các công ty phân tích các bình luận trên mạng xã hội để biết khách hàng đang nghĩ gì về sản phẩm của họ.
- **Tự động sửa lỗi và gợi ý văn bản (Autocorrect & Predictive Text):** Tính năng bạn dùng hàng ngày trên điện thoại và trình soạn thảo văn bản.
- **Chatbots hỗ trợ khách hàng:** Tự động trả lời các câu hỏi thường gặp, giúp giảm tải cho nhân viên hỗ trợ.

### 1.1.2. Mối quan hệ với Trí tuệ Nhân tạo, Học máy và Ngôn ngữ học

NLP không tồn tại một cách độc lập. Nó là một giao điểm, một "sân chơi chung" của nhiều lĩnh vực khoa học lớn, mà nổi bật nhất là Trí tuệ Nhân tạo, Học máy, và Ngôn ngữ học. Hiểu rõ mối quan hệ này giúp chúng ta định vị đúng vai trò và phương pháp luận của NLP.



Hình 1.1: Sơ đồ minh họa mối quan hệ giữa NLP và các lĩnh vực liên quan.

- **Trí tuệ Nhân tạo (Artificial Intelligence - AI):** AI là một ngành khoa học máy tính rộng lớn với mục tiêu tạo ra các cỗ máy thông minh có khả năng suy nghĩ, học hỏi và hành động như con người. Trong bức tranh này, NLP là **một nhánh con thiết yếu của AI**. Một trí tuệ thực sự toàn diện không thể thiếu khả năng giao tiếp bằng ngôn ngữ tự nhiên. Vì vậy, NLP được xem là một trong những bài toán "AI-hoàn chỉnh"(AI-complete), tức là việc giải quyết hoàn hảo bài toán NLP có thể đồng nghĩa với việc tạo ra một trí tuệ nhân tạo ở cấp độ con người.
- **Học máy (Machine Learning - ML):** Học máy là một tập hợp các phương pháp và thuật toán cho phép máy tính "học" từ dữ liệu mà không cần được lập trình một cách tường minh. Trong lịch sử NLP, **Học máy đã tạo ra một cuộc cách mạng**. Thay vì phải viết tay hàng ngàn quy tắc ngôn ngữ phức tạp và dễ sai sót, các phương pháp học máy cho phép chúng ta xây dựng các mô hình tự động nhận diện các quy luật và mẫu hình (patterns) từ một lượng lớn văn bản. Ngày nay, hầu hết các hệ thống NLP tiên tiến đều được xây dựng trên nền tảng của học máy, và đặc biệt là học sâu (Deep Learning).
- **Ngôn ngữ học (Linguistics):** Ngôn ngữ học là ngành khoa học nghiên cứu về chính ngôn ngữ, bao gồm cấu trúc (ngữ pháp), ý nghĩa (ngữ nghĩa), và cách sử dụng trong ngữ cảnh (ngữ dụng). Ngôn ngữ học cung cấp nền tảng lý thuyết và kiến thức chuyên môn cho NLP. Các khái niệm như từ loại (Part-of-Speech), cây cú pháp (syntax tree), cấu trúc hình thái (morphology) đều bắt nguồn từ ngôn ngữ học. Mặc dù các mô hình học sâu hiện đại có thể học được nhiều quy luật ngầm mà không cần kiến thức ngôn ngữ học tường minh, việc hiểu các nguyên lý cơ bản của ngôn ngữ học vẫn cực kỳ quan trọng để thiết kế mô hình, diễn giải kết quả và chẩn đoán lỗi.

Nói một cách ví von: AI đặt ra mục tiêu (tạo ra máy móc thông minh), Ngôn ngữ học cung cấp bản đồ và quy luật của "vùng đất" ngôn ngữ, còn Học máy cung cấp những công cụ (xe ủi, máy xúc) để khai phá và xây dựng trên vùng đất đó. NLP chính là công trình được tạo ra từ sự kết hợp của cả ba yếu tố trên.

### 1.1.3. Tổng quan các Ký nguyên Phát triển (Luật lệ → Thống kê → Học sâu → Transformer)

Lịch sử phát triển của NLP là một câu chuyện hấp dẫn về sự thay đổi trong tư duy và phương pháp luận, được đánh dấu bởi bốn ký nguyên chính. Mỗi ký nguyên đại diện cho một bước nhảy vọt về khả năng và hiệu suất.

**Ký nguyên 1: Dựa trên Luật lệ (Symbolic NLP, những năm 1950 – 1980)** Đây là giai đoạn sơ khai, chịu ảnh hưởng lớn từ ngôn ngữ học hình thức và logic.

- **Tư duy cốt lõi:** Ngôn ngữ có thể được mô tả hoàn toàn bằng một bộ các quy tắc (rules) và từ điển được tạo ra thủ công bởi các chuyên gia ngôn ngữ.
- **Phương pháp:** Các hệ thống dựa trên ngữ pháp hình thức (formal grammars), phân tích cú pháp bằng tay, và các bộ quy tắc logic phức tạp. Ví dụ kinh điển là hệ thống SHRDLU [101] có thể đối thoại về một thế giới các khối hộp bằng cách áp dụng các quy tắc logic chặt chẽ.
- **Ưu điểm:** Độ chính xác có thể rất cao trong một lĩnh vực (domain) hẹp và được định nghĩa rõ ràng. Các quy tắc có thể diễn giải được, dễ hiểu tại sao hệ thống đưa ra một kết quả cụ thể.
- **Nhược điểm:** Cực kỳ giòn (brittle) – hệ thống dễ dàng thất bại khi gặp phải một câu nói không tuân theo quy tắc đã định sẵn. Tốn rất nhiều công sức của chuyên gia để xây dựng và bảo trì. Khó mở rộng và thích ứng với các lĩnh vực mới.

**Ký nguyên 2: Dựa trên Thống kê (Statistical NLP, những năm 1990 – 2010)** Sự bùng nổ của dữ liệu số và sức mạnh tính toán đã dẫn đến một sự thay đổi mô hình (paradigm shift).

- **Tư duy cốt lõi:** Thay vì cố gắng dạy cho máy tính các quy tắc, hãy để nó tự học các quy luật từ dữ liệu thực tế. Các vấn đề ngôn ngữ được định hình lại thành các bài toán dự đoán dựa trên xác suất. Câu nói nổi tiếng của Frederick Jelinek đã tóm gọn tinh thần của thời kỳ này: "Mỗi lần tôi sa thải một nhà ngôn ngữ học, hiệu suất của hệ thống nhận dạng giọng nói lại tăng lên."

- Phương pháp:** Mô hình N-gram, Mô hình Markov ẩn (HMM) [68], Hồi quy Logistic, Máy Vector Hỗ trợ (SVM) [16], và sau này là Các trường Ngẫu nhiên Có điều kiện (CRF) [44]. Các khái niệm như TF-IDF trở thành tiêu chuẩn để biểu diễn văn bản.
- Ưu điểm:** Mạnh mẽ (robust) và linh hoạt hơn nhiều so với hệ thống luật. Có khả năng xử lý ngôn ngữ tự nhiên "trong đời thực" với tất cả sự lộn xộn của nó.
- Nhược điểm:** Gặp vấn đề với *sự thừa thoát của dữ liệu* (data sparsity), đặc biệt là với các N-gram dài. Khó nắm bắt được ý nghĩa ngữ nghĩa sâu sắc và các mối quan hệ phụ thuộc tầm xa trong câu.

**Kỷ nguyên 3: Dựa trên Học sâu (Deep Learning, khoảng 2013 – 2017)** Cuộc cách mạng thực sự bắt đầu khi các mạng nơ-ron sâu được áp dụng vào NLP.

- Tự duy cốt lõi:** Tự động học các biểu diễn (representations) hoặc các đặc trưng (features) của từ và câu dưới dạng các vector dày đặc (dense vectors), hay còn gọi là *embeddings*. Các biểu diễn này nắm bắt được các mối quan hệ ngữ nghĩa và cú pháp một cách hiệu quả.
- Phương pháp:** Word2Vec [56], GloVe [64] tiên phong trong việc học biểu diễn từ. Các kiến trúc như Mạng Nơ-ron Hồi tiếp (RNN), LSTM [31], GRU [11] trở nên thống trị trong các bài toán xử lý chuỗi. Mô hình Chuỗi-sang-Chuỗi (Seq2Seq) [89] kết hợp với cơ chế Chú ý (Attention) [2] đã tạo ra bước đột phá trong dịch máy.
- Ưu điểm:** Đạt hiệu suất vượt trội trên hầu hết các benchmark. Có khả năng nắm bắt ngữ cảnh và ngữ nghĩa tốt hơn nhiều so với các mô hình thống kê.
- Nhược điểm:** RNN vẫn gặp khó khăn trong việc xử lý các phụ thuộc tầm rất xa. Việc huấn luyện tuần tự của RNN làm hạn chế khả năng song song hóa, gây tốn thời gian.

**Kỷ nguyên 4: Kỷ nguyên Transformer (The Transformer Era, 2017 – Hiện tại)** Bài báo "Attention Is All You Need" [94] vào năm 2017 đã khai sinh ra một kiến trúc mới và định hình lại toàn bộ lĩnh vực.

- Tự duy cốt lõi:** Hoàn toàn từ bỏ kiến trúc hồi tiếp (recurrence) và tích chập (convolution), thay vào đó chỉ dựa vào cơ chế *tự chú ý* (*self-attention*) để nắm bắt mối quan hệ giữa các từ trong một chuỗi.
- Phương pháp:** Kiến trúc Transformer gốc, và sau đó là sự bùng nổ của các Mô hình Ngôn ngữ Lớn (Large Language Models - LLMs) được huấn luyện trước (pre-trained) trên một lượng dữ liệu khổng lồ, ví dụ như BERT [20], GPT [69, 70, 9], T5 [72], Llama [93].
- Ưu điểm:** Khả năng song song hóa cao, cho phép huấn luyện các mô hình cực lớn. Hiệu quả vượt trội trong việc xử lý các phụ thuộc tầm xa. Mô hình "nền tảng" (foundation model) được huấn luyện trước có thể được tinh chỉnh (fine-tune) cho nhiều tác vụ khác nhau với hiệu quả đáng kinh ngạc, dẫn đến sự phát triển của học ít mẫu (few-shot learning) và học không cần mẫu (zero-shot learning).
- Nhược điểm:** Đòi hỏi tài nguyên tính toán và dữ liệu cực lớn. Các vấn đề về diễn giải, thiên kiến, và đạo đức trở nên nổi bật hơn bao giờ hết.

## 1.2. Nền tảng Ngôn ngữ học cho NLP

Trong kỷ nguyên của Học sâu, khi các mô hình Transformer khổng lồ dường như có thể "học" mọi thứ từ dữ liệu thô, một câu hỏi chính đáng được đặt ra: "Liệu kiến thức về Ngôn ngữ học có còn cần thiết?". Câu trả lời là một tiếng "Có" dứt khoát.

Mặc dù chúng ta không còn phải viết tay các bộ quy tắc ngữ pháp phức tạp, việc hiểu các cấu trúc và nguyên lý cơ bản của ngôn ngữ mang lại những lợi ích to lớn:

- Thiết kế tác vụ (Task Design):** Giúp chúng ta định hình các bài toán NLP một cách có ý nghĩa. Ví dụ, tại sao "Nhận dạng Thực thể Tên" (NER) lại là một bài toán quan trọng? Vì "thực thể" là một khái niệm ngữ nghĩa cơ bản.

- **Kỹ thuật đặc trưng (Feature Engineering):** Cung cấp ý tưởng cho các đặc trưng hữu ích, đặc biệt khi dữ liệu bị hạn chế.
- **Phân tích lỗi (Error Analysis):** Giúp chúng ta hiểu tại sao một mô hình thất bại. Liệu nó thất bại vì không hiểu được cấu trúc cú pháp phức tạp, hay vì không giải quyết được sự đa nghĩa của từ?
- **Đánh giá (Evaluation):** Giúp xây dựng các bộ dữ liệu đánh giá có thể kiểm tra các năng lực ngôn ngữ cụ thể của mô hình.

Mục này sẽ giới thiệu những khái niệm Ngôn ngữ học cốt lõi nhất, đóng vai trò là lăng kính để chúng ta nhìn vào các bài toán NLP trong suốt giáo trình này.

## 1.2.1. Các cấp độ phân tích ngôn ngữ: Hình thái học, Cú pháp, Ngữ nghĩa, Ngữ dụng

Ngôn ngữ học thường phân tích ngôn ngữ theo một hệ thống các cấp độ, đi từ đơn vị nhỏ nhất đến ý nghĩa trong bối cảnh rộng lớn nhất. Đây là một khung khái niệm vô cùng hữu ích để phân loại các bài toán NLP.

**1. Hình thái học (Morphology)** Là nghiên cứu về cấu trúc bên trong của từ. Hình thái học xem xét cách các từ được tạo thành từ những đơn vị có nghĩa nhỏ hơn gọi là **hình vị (morphemes)**.

- **Ví dụ (tiếng Anh):** Từ *unhappiness* được cấu tạo từ 3 hình vị: tiền tố *un-* (phủ định), gốc từ *happi* (hạnh phúc), và hậu tố *-ness* (danh từ hóa).
- **Ví dụ (tiếng Việt):** Tiếng Việt là một ngôn ngữ đơn lập (isolating language), nên cấu trúc hình thái không phức tạp như tiếng Anh. Tuy nhiên, khái niệm này vẫn có thể áp dụng cho các từ ghép như *học sinh* (gồm hình vị *học* và *sinh*) hay *nha khoa học* (*nha-* + *khoa học*).
- **Ứng dụng trong NLP:** Lemmatization (đưa từ về dạng gốc, ví dụ: "running" → "run"), Stemming (cắt bỏ hậu tố), Phân tích hình thái.

**2. Cú pháp (Syntax)** Là nghiên cứu về cách các từ kết hợp với nhau để tạo thành các cụm từ (phrases) và câu (sentences) đúng ngữ pháp. Cú pháp tập trung vào **quy tắc cấu trúc** của câu.

- **Ví dụ:** Câu "Con mèo đuổi con chuột" là đúng cú pháp tiếng Việt. Câu \*\*"Mèo con chuột con đuổi"\*\* thì không, mặc dù các từ vẫn giữ nguyên. Cú pháp quy định trật tự từ và các mối quan hệ ngữ pháp.
- **Ứng dụng trong NLP:** Phân tích cú pháp (Parsing), Gán nhãn Từ loại (Part-of-Speech Tagging), Kiểm tra ngữ pháp (Grammar Checking).

**3. Ngữ nghĩa (Semantics)** Là nghiên cứu về ý nghĩa của từ, cụm từ và câu, độc lập với ngữ cảnh. Ngữ nghĩa trả lời câu hỏi "Câu này có nghĩa là gì?".

- **Ví dụ:** Câu "Colorless green ideas sleep furiously" (Những ý tưởng xanh không màu ngủ một cách giận dữ) của Noam Chomsky là một câu đúng về mặt cú pháp nhưng vô nghĩa về mặt ngữ nghĩa.
- Thủ thách lớn của ngữ nghĩa là **sự mơ hồ (ambiguity)**. Ví dụ, từ "đường" có thể là "con đường" hoặc "đường ăn". Câu "Tôi thấy người đàn ông trên ngọn đồi với chiếc kính thiên văn" có thể có nghĩa là tôi dùng kính để thấy anh ta, hoặc anh ta là người có chiếc kính.
- **Ứng dụng trong NLP:** Phân biệt ý của từ (Word Sense Disambiguation - WSD), Gán nhãn vai nghĩa (Semantic Role Labeling - SRL), Biểu diễn ý nghĩa câu.

**4. Ngữ dụng (Pragmatics)** Là nghiên cứu về cách ngữ cảnh ảnh hưởng đến việc diễn giải ý nghĩa. Ngữ dụng xem xét ý định của người nói và cách người nghe hiểu được ý định đó. Nó trả lời câu hỏi "Người nói muốn nói gì khi nói câu này?".

- **Ví dụ:** Nếu ai đó nói "Ồ đây nóng quá", về mặt ngữ nghĩa, đó là một câu trào thuận về nhiệt độ. Nhưng về mặt ngữ dụng, đó có thể là một lời yêu cầu "Làm ơn bật quạt/điều hòa lên".

- **Ứng dụng trong NLP:** Phân tích hội thoại (Discourse Analysis), Phân giải đồng tham chiếu (Coreference Resolution - xác định "anh ấy" trong một câu là đang chỉ đến ai), Nhận dạng hành động nói (Speech Act Recognition), các hệ thống đối thoại (Dialogue Systems).

Bốn cấp độ này tạo thành một chuỗi xử lý tự nhiên: từ nhận diện các đơn vị nhỏ nhất của từ (hình thái), đến cách chúng kết hợp thành câu (cú pháp), rồi ý nghĩa của câu đó (ngữ nghĩa), và cuối cùng là ý nghĩa trong một bối cảnh cụ thể (ngữ dụng).

### 1.2.2. Ngữ pháp hình thức (Formal Grammars): Ngữ pháp phi ngữ cảnh (CFG)

Để máy tính có thể "hiểu" được cú pháp, các nhà ngôn ngữ học và khoa học máy tính đã phát triển các hệ thống quy tắc toán học gọi là **Ngữ pháp hình thức**. Một trong những loại ngữ pháp hình thức quan trọng và phổ biến nhất trong NLP thời kỳ đầu là **Ngữ pháp phi ngữ cảnh** (Context-Free Grammar - CFG).

#### Định nghĩa 2: Ngữ pháp phi ngữ cảnh (CFG)

Một CFG là một bộ quy tắc sản sinh (production rules) được dùng để tạo ra các chuỗi trong một ngôn ngữ. Nó được gọi là "phi ngữ cảnh" vì các quy tắc có thể được áp dụng bất kể ngữ cảnh xung quanh các ký hiệu. Một CFG được định nghĩa bởi một bộ 4 thành phần  $(N, \Sigma, P, S)$ :

- $N$ : một tập hữu hạn các **ký hiệu phi-kết thúc** (non-terminal symbols), thường là các loại cụm từ như NP (Cụm danh từ), VP (Cụm động từ).
- $\Sigma$ : một tập hữu hạn các **ký hiệu kết thúc** (terminal symbols), chính là các từ trong từ vựng.
- $P$ : một tập hữu hạn các **quy tắc sản sinh** (production rules), có dạng  $A \rightarrow \alpha$ , trong đó  $A \in N$  và  $\alpha$  là một chuỗi các ký hiệu trong  $(N \cup \Sigma)^*$ .
- $S$ : **ký hiệu bắt đầu** (start symbol), thường là S (Sentence - Câu),  $S \in N$ .

#### Ví dụ 2: Một CFG đơn giản cho Tiếng Việt

Hãy xem xét một ngữ pháp rất nhỏ để tạo ra câu "con mèo đuối con chuột":

- $N = \{S, NP, VP, Det, N, V\}$
- $\Sigma = \{\text{con}, \text{mèo}, \text{đuối}, \text{chuột}\}$
- $S = S$
- $P$  là tập các quy tắc sau:
  1.  $S \rightarrow NP VP$  (Một câu được tạo thành từ một Cụm danh từ và một Cụm động từ)
  2.  $NP \rightarrow Det N$  (Một cụm danh từ được tạo thành từ một Từ hạn định và một Danh từ)
  3.  $VP \rightarrow V NP$  (Một cụm động từ được tạo thành từ một Động từ và một Cụm danh từ)
  4.  $Det \rightarrow \text{con}$
  5.  $N \rightarrow \text{mèo} \mid \text{chuột}$  (Ký hiệu ' $\mid$ ' có nghĩa là "hoặc")
  6.  $V \rightarrow \text{đuối}$

Sử dụng các quy tắc này, chúng ta có thể "sản sinh" ra câu mong muốn, và quá trình này có thể được hình dung bằng một cây cú pháp (parse tree).

CFG là nền tảng lý thuyết cho *phân tích cú pháp thành phần*, một khái niệm chúng ta sẽ tìm hiểu ngay sau đây.

### 1.2.3. Các lý thuyết cú pháp chính: Cú pháp thành phần (Constituency) và Cú pháp phụ thuộc (Dependency)

Khi phân tích cấu trúc cú pháp của một câu, có hai cách tiếp cận chính, tương ứng với hai "trường phái" lớn trong ngôn ngữ học và NLP.

**1. Cú pháp thành phần (Constituency Syntax)** Cách tiếp cận này, vốn bắt nguồn trực tiếp từ CFG của Noam Chomsky [12], cho rằng câu được cấu tạo từ các **thành phần (constituents)** hay **cụm từ (phrases)** lồng vào nhau. Mỗi cụm từ là một đơn vị ngữ pháp hoạt động như một khối thống nhất.

- **Ý tưởng cốt lõi:** Chia câu thành các cụm từ, rồi lại chia các cụm từ đó thành các cụm từ nhỏ hơn cho đến khi còn lại các từ riêng lẻ.
- **Biểu diễn:** Sử dụng cây cú pháp thành phần (constituency parse tree). Các nút trong (internal nodes) của cây là tên các cụm từ (NP, VP, PP), và các nút lá (leaf nodes) là các từ của câu.
- **Ví dụ:** Với câu "Con mèo đuổi con chuột", cây thành phần sẽ trông như sau:

$$(S (NP (Det con) (N mèo)) (VP (V đuổi) (NP (Det con) (N chuột))))$$

Cấu trúc này cho thấy rõ "con mèo" là một khối (Cụm danh từ - NP) và "đuổi con chuột" là một khối khác (Cụm động từ - VP).

**2. Cú pháp phụ thuộc (Dependency Syntax)** Cách tiếp cận này không tập trung vào các cụm từ, mà tập trung vào mối quan hệ **phụ thuộc ngữ pháp** giữa các từ riêng lẻ.

- **Ý tưởng cốt lõi:** Mỗi từ trong câu, ngoại trừ một từ gốc (thường là động từ chính), sẽ phụ thuộc vào một từ khác. Mối quan hệ này là mối quan hệ bất đối xứng giữa một **từ quản lý (head/governor)** và một **từ phụ thuộc (dependent/modifier)**.
- **Biểu diễn:** Sử dụng cây cú pháp phụ thuộc (dependency parse tree), thực chất là một đồ thị có hướng. Các nút là các từ, và các cạnh có nhãn là tên của mối quan hệ ngữ pháp (ví dụ: **nsubj** - chủ ngữ, **obj** - tân ngữ, **det** - từ hạn định).
- **Ví dụ:** Với câu "Con mèo đuổi con chuột", đồ thị phụ thuộc sẽ có các quan hệ:
  - **đuổi** là gốc (root) của câu.
  - **mèo** là chủ ngữ (**nsubj**) của **đuổi**.
  - **chuột** là tân ngữ (**obj**) của **đuổi**.
  - **con** (thứ nhất) là từ hạn định (**det**) của **mèo**.
  - **con** (thứ hai) là từ hạn định (**det**) của **chuột**.

So sánh Cú pháp Thành phần và Phụ thuộc

Cú pháp Thành phần	Cú pháp Phụ thuộc
Tập trung vào các <b>cụm từ</b> và cấu trúc lồng nhau.	Tập trung vào <b>mối quan hệ giữa các từ</b> .
Cung cấp thông tin cấu trúc rõ ràng về các khối ngữ pháp.	Cung cấp thông tin rõ ràng về quan hệ chức năng (ai làm gì ai).
Phù hợp hơn cho các ngôn ngữ có trật tự từ cố định (như tiếng Anh).	Linh hoạt hơn với các ngôn ngữ có trật tự tự do.
Phổ biến trong ngôn ngữ học lý thuyết và các hệ thống NLP thế hệ cũ.	Rất phổ biến trong các công cụ NLP hiện đại (ví dụ: spaCy, Stanza) do tính hữu ích cho các tác vụ ngữ nghĩa và tốc độ phân tích nhanh hơn.

Việc hiểu cả hai trường phái này rất quan trọng, vì chúng cung cấp những góc nhìn bổ trợ cho nhau về cấu trúc của ngôn ngữ và là nền tảng cho nhiều bài toán NLP cốt lõi mà chúng ta sẽ khám phá trong các chương tiếp theo.

### 1.3. Các Bài toán Cốt lõi trong NLP

Sau khi đã có nền tảng về các cấp độ phân tích ngôn ngữ, chúng ta sẽ chính thức ánh xạ chúng vào các bài toán (tasks) cụ thể trong NLP. Việc hiểu rõ các bài toán này là cực kỳ quan trọng, vì chúng là những "viên gạch" cơ bản để xây dựng nên các ứng dụng NLP phức tạp hơn. Các bài toán này thường được nhóm lại theo cấp độ phân tích, từ từ, đến câu, và cuối cùng là các ứng dụng hoàn chỉnh.

#### 1.3.1. Phân tích cấp độ từ

Đây là nhóm các bài toán cơ bản nhất, xử lý và gán thông tin cho từng từ (token) riêng lẻ trong văn bản. Kết quả của các bài toán này thường là đầu vào cho các tác vụ phức tạp hơn.

**Gán nhãn Từ loại (Part-of-Speech - POS Tagging)** Là quá trình gán một nhãn từ loại (danh từ, động từ, tính từ, v.v.) cho mỗi từ trong một câu, dựa vào định nghĩa và ngữ cảnh của nó.

- **Mục tiêu:** Giải quyết sự mơ hồ của từ. Ví dụ, từ "câu" trong "câu cá" là động từ, nhưng trong "câu văn" lại là danh từ.
- **Ví dụ đầu vào:** 'Tôi thích đọc sách.'
- **Ví dụ đầu ra:** 'Tôi/P thích/V đọc/V sách/N ./CH' (P: Đại từ, V: Động từ, N: Danh từ, CH: Dấu câu).
- **Tầm quan trọng:** Là một bước tiền xử lý nền tảng cho rất nhiều bài toán khác như phân tích cú pháp, nhận dạng thực thể, trích xuất thông tin.

**Đưa từ về dạng gốc (Lemmatization và Stemming)** Cả hai kỹ thuật này đều nhằm mục đích quy chuẩn hóa các dạng biến thể của một từ về một dạng chung.

- **Stemming (Rút gọn về gốc từ):** Một phương pháp đơn giản, dựa trên quy tắc để cắt bỏ các hậu tố của từ. Nó nhanh nhưng đôi khi không chính xác về mặt ngôn ngữ.
  - **Ví dụ (tiếng Anh):** "studies", "studying" → "studi".
- **Lemmatization (Đưa về dạng từ điển):** Một phương pháp phức tạp hơn, sử dụng từ điển và phân tích hình thái để đưa từ về dạng nguyên thể (lemma) của nó.
  - **Ví dụ (tiếng Anh):** "studies", "studying" → "study"; "better" → "good".
  - **Ví dụ (tiếng Việt):** "đi học", "đi làm" có thể được chuẩn hóa về động từ "đi".
- **Lựa chọn:** Lemmatization thường được ưu tiên hơn vì kết quả có ý nghĩa ngôn ngữ học, tuy nhiên nó chậm hơn và đòi hỏi nhiều tài nguyên hơn Stemming.

**Phân tích Hình thái học (Morphological Analysis)** Là bài toán phân tích một từ thành các hình vị (morphemes) cấu tạo nên nó. Bài toán này đặc biệt quan trọng đối với các ngôn ngữ chắp dính (agglutinative languages) như tiếng Thổ Nhĩ Kỳ, Phần Lan, nơi một từ có thể chứa rất nhiều thông tin ngữ pháp.

- **Ví dụ (tiếng Anh):** 'unhappiness' → 'un-' (phủ định) + 'happy' (gốc) + '-ness' (danh từ hóa).
- **Tầm quan trọng:** Giúp các mô hình hiểu được cấu trúc bên trong của từ, đặc biệt hữu ích khi gặp các từ hiếm hoặc chưa từng thấy (Out-of-Vocabulary - OOV).

#### 1.3.2. Phân tích cấp độ câu

Sau khi đã xử lý các từ riêng lẻ, bước tiếp theo là hiểu cách chúng kết hợp với nhau để tạo thành một câu có cấu trúc.

**Phân tích cú pháp (Parsing)** Đây là bài toán xác định cấu trúc ngữ pháp của một câu. Như đã đề cập ở mục 1.2.3, có hai loại phân tích cú pháp chính:

- **Phân tích cú pháp Thành phần (Constituency Parsing):**
  - **Mục tiêu:** Xây dựng một cây cú pháp thành phần, nhóm các từ thành các cụm từ có cấp bậc.

- **Đầu ra:** Một cấu trúc cây lồng nhau, ví dụ: (S (NP (N Tôi)) (VP (V thích) (NP (N sách)))).
- **Phân tích cú pháp Phụ thuộc (Dependency Parsing):**
  - **Mục tiêu:** Xác định các mối quan hệ phụ thuộc (ai làm gì ai, cái gì bổ nghĩa cho cái gì) giữa các từ.
  - **Đầu ra:** Một tập các cặp (từ quản lý, quan hệ, từ phụ thuộc), ví dụ: (thích, nsubj, Tôi), (thích, obj, sách).

Phân tích cú pháp là một trong những bài toán lâu đời và thách thức nhất trong NLP, là chìa khóa để "hiểu sâu" cấu trúc của một câu.

### 1.3.3. Phân tích ngữ nghĩa

Vượt ra ngoài cấu trúc, các bài toán ở cấp độ này cố gắng nắm bắt ý nghĩa của văn bản.

**Nhận dạng Thực thể Tên (Named Entity Recognition - NER)** Là bài toán tìm và phân loại các thực thể có tên trong văn bản thành các danh mục được định trước như Tên người (PER), Tổ chức (ORG), Địa điểm (LOC), Ngày tháng (DATE), v.v.

- **Ví dụ đầu vào:** ‘Apple được thành lập bởi Steve Jobs tại Cupertino vào năm 1976.’
- **Ví dụ đầu ra:** [Apple/ORG] được thành lập bởi [Steve Jobs/PER] tại [Cupertino/LOC] vào năm [1976/DATE].’
- **Tầm quan trọng:** Một tác vụ nền tảng cho việc trích xuất thông tin, xây dựng đồ thị tri thức và hệ thống hỏi đáp.

**Phân biệt Ý của Từ (Word Sense Disambiguation - WSD)** Là bài toán xác định xem một từ đa nghĩa đang được sử dụng với ý nghĩa nào trong một ngữ cảnh cụ thể.

- **Ví dụ:** Trong câu “Ngân hàng nhà nước vừa hạ lãi suất”, từ “ngân hàng” có nghĩa là một tổ chức tài chính, chứ không phải “bờ sông”(bờ, bank).
- **Thách thức:** Đây là một bài toán rất khó và lâu đời. Tuy nhiên, sự ra đời của các mô hình ngôn ngữ dựa trên ngữ cảnh (contextualized embeddings) như BERT đã ngầm giải quyết một phần lớn bài toán này.

**Gán nhãn Vai nghĩa (Semantic Role Labeling - SRL)** Là bài toán xác định ”ai đã làm gì, cho ai, ở đâu, khi nào, và như thế nào” xung quanh một động từ chính (vị ngữ) trong câu.

- **Mục tiêu:** Phân tích cấu trúc sự kiện (event structure) của câu.
- **Ví dụ đầu vào:** ‘Hôm qua, An đã tặng Hoa một cuốn sách ở thư viện.’
- **Vị ngữ:** ‘tặng’
- **Ví dụ đầu ra:**
  - **AGENT** (Tác nhân): ‘An’
  - **THEME** (Đối thể): ‘một cuốn sách’
  - **RECIPIENT** (Người nhận): ‘Hoa’
  - **TIME** (Thời gian): ‘Hôm qua’
  - **LOCATION** (Địa điểm): ‘ở thư viện’
- **Tầm quan trọng:** Giúp biến đổi văn bản phi cấu trúc thành thông tin có cấu trúc, rất hữu ích cho các hệ thống hỏi đáp và tóm tắt.

**Trích xuất Quan hệ (Relation Extraction)** Là bài toán xác định các mối quan hệ ngữ nghĩa giữa các thực thể đã được nhận dạng trong văn bản.

- **Mục tiêu:** Xây dựng các bộ ba (triples) có dạng ‘(Thực thể 1, Quan hệ, Thực thể 2)’.
- **Ví dụ đầu vào:** ‘Apple được thành lập bởi Steve Jobs.’
- **Thực thể đã nhận dạng:** [Apple/ORG], [Steve Jobs/PER]
- **Ví dụ đầu ra:** ‘(Steve Jobs, FounderOf, Apple)’

- **Tầm quan trọng:** Là bước cốt lõi để tự động xây dựng và bổ sung các cơ sở tri thức và đồ thị tri thức từ văn bản.

### 1.3.4. Các bài toán Ứng dụng

Đây là nhóm các bài toán ở cấp độ cao nhất, thường kết hợp kết quả từ nhiều bài toán cốt lõi ở trên để tạo ra các sản phẩm hữu ích cho người dùng cuối.

#### Ví dụ 3: Tổng quan các bài toán ứng dụng phổ biến

- **Phân loại văn bản (Text Classification):** Gán một hoặc nhiều nhãn cho một đoạn văn bản. Đây là một trong những bài toán phổ biến nhất.
  - **Ví dụ:** Phân tích cảm xúc (Tích cực/Tiêu cực/Trung tính), Phân loại chủ đề tin tức (Thể thao/Chính trị/Giải trí), Phát hiện spam.
- **Dịch máy (Machine Translation - MT):** Tự động dịch văn bản từ ngôn ngữ nguồn sang ngôn ngữ đích.
  - **Ví dụ:** Google Translate.
- **Tóm tắt văn bản (Text Summarization):** Tạo ra một phiên bản ngắn gọn, súc tích nhưng vẫn chứa đựng những thông tin quan trọng nhất của một văn bản dài.
  - **Ví dụ:** Tóm tắt một bài báo dài thành 3 gạch đầu dòng chính.
- **Hỏi đáp (Question Answering - QA):** Cung cấp một câu trả lời chính xác cho một câu hỏi do người dùng đặt ra.
  - **Ví dụ:** Tìm kiếm câu trả lời trong một đoạn văn bản cho trước (Extractive QA) hoặc tự sinh ra câu trả lời (Generative QA).
- **Hệ thống Đổi thoại (Dialogue Systems / Chatbots):** Xây dựng các tác tử (agents) có khả năng trò chuyện với con người một cách tự nhiên.
  - **Ví dụ:** Chatbot hỗ trợ khách hàng, trợ lý ảo.
- **Suy luận Ngôn ngữ Tự nhiên (Natural Language Inference - NLI):** Xác định mối quan hệ logic (kéo theo, mâu thuẫn, hoặc trung lập) giữa một cặp câu (tiền đề và giả thuyết).
  - **Ví dụ:** Tiền đề: "Một người đàn ông đang chơi guitar." Giả thuyết: "Có người đang tạo ra âm nhạc." → Mối quan hệ: Kéo theo (Entailment).

Bản đồ các bài toán này cho thấy một lộ trình rõ ràng trong việc xử lý ngôn ngữ: từ việc hiểu các thành phần nhỏ nhất, đến việc lắp ráp chúng thành cấu trúc có ý nghĩa, và cuối cùng là sử dụng sự hiểu biết đó để thực hiện các nhiệm vụ phức tạp, mang lại giá trị thực tiễn. Trong các chương sau, chúng ta sẽ lần lượt tìm hiểu các mô hình và kỹ thuật để giải quyết từng bài toán này.

## 1.4. Các Kiến thức Toán học Cần trang bị

NLP hiện đại, đặc biệt là trong kỷ nguyên Học sâu, có nền tảng vững chắc là toán học. Các mô hình phức tạp như Transformer về bản chất là những hàm số khổng lồ, và quá trình "học" của chúng chính là một bài toán tối ưu hóa.

Mục này không nhằm mục đích dạy lại toàn bộ các môn toán. Thay vào đó, nó đóng vai trò là một danh mục tham khảo, một bản đồ chỉ ra những khái niệm toán học nào là **thiết yếu** và giải thích **tại sao** chúng lại quan trọng trong NLP. Việc nắm vững những kiến thức này sẽ giúp bạn đi từ một người "sử dụng" thư viện thành một người thực sự "hiểu" và có khả năng "xây dựng" các mô hình.

### 1.4.1. Đại số Tuyến tính: Vector, Ma trận, Tensor, SVD

Đại số Tuyến tính là ngôn ngữ của dữ liệu trong Học sâu. Mọi thứ trong NLP, từ một từ đơn lẻ đến cả một kho tài liệu, đều sẽ được biểu diễn dưới dạng các đối tượng của Đại số Tuyến tính.

**Vector** Một vector là một mảng số một chiều. Trong NLP, vector không chỉ là một dãy số, mà nó là **biểu diễn ngữ nghĩa** của một đối tượng.

- **Vai trò trong NLP:** Một từ, một câu, hay một tài liệu hoàn chỉnh đều có thể được "nhúng"(embed) vào một không gian vector nhiều chiều. Vector này, hay còn gọi là **embedding**, nắm bắt các đặc tính ngữ nghĩa của đối tượng. Các từ có nghĩa tương tự nhau (ví dụ: "vua" và "nữ hoàng") sẽ có các vector biểu diễn nằm gần nhau trong không gian. Đây là nền tảng của Word2Vec, GloVe, và mọi mô hình ngôn ngữ hiện đại.
- **Ký hiệu:** Một vector  $v$  trong không gian  $d$  chiều được ký hiệu là  $v \in \mathbb{R}^d$ .

**Ma trận (Matrix)** Một ma trận là một mảng số hai chiều.

- **Vai trò trong NLP:** Ma trận có vô số ứng dụng:
  1. **Tập hợp các vector từ:** Một ma trận có thể biểu diễn toàn bộ từ vựng, trong đó mỗi hàng là vector embedding của một từ.
  2. **Trọng số của mạng nơ-ron:** Mỗi lớp (layer) trong một mạng nơ-ron thực chất là một ma trận trọng số. Quá trình học chính là điều chỉnh các giá trị trong ma trận này.
  3. **Ma trận chú ý (Attention Matrix):** Trong kiến trúc Transformer, ma trận chú ý lưu trữ điểm số thể hiện mức độ "quan tâm" của một từ đến các từ khác trong câu.

**Tensor** Tensor là sự tổng quát hóa của vector và ma trận lên nhiều chiều hơn. Một vector là tensor bậc 1, một ma trận là tensor bậc 2.

- **Vai trò trong NLP:** Các framework Học sâu như PyTorch và TensorFlow đều hoạt động dựa trên tensor. Khi xử lý dữ liệu theo lô (batch), chúng ta thường làm việc với các tensor bậc 3. Ví dụ, một lô 32 câu, mỗi câu dài 50 từ, mỗi từ được biểu diễn bằng vector 256 chiều sẽ tạo thành một tensor có kích thước '(32, 50, 256)'.

**Phân tích Giá trị suy biến (Singular Value Decomposition - SVD)** Là một kỹ thuật phân rã một ma trận thành ba ma trận khác.

- **Vai trò trong NLP:** Trong các mô hình NLP cổ điển hơn, SVD được dùng trong Phân tích Ngữ nghĩa Tiềm ẩn (Latent Semantic Analysis - LSA) để giảm chiều dữ liệu và tìm ra các "chủ đề" ẩn trong kho văn bản. Mặc dù ít được dùng trực tiếp trong các mô hình hiện đại, các ý tưởng về phân rã ma trận vẫn là nền tảng quan trọng.

#### 1.4.2. Giải tích: Đạo hàm, Gradient, Chain Rule

Nếu Đại số Tuyến tính là cách chúng ta biểu diễn dữ liệu, thì Giải tích là cách chúng ta khiến các mô hình "học" từ dữ liệu đó. Cốt lõi của việc huấn luyện mạng nơ-ron là tối ưu hóa một **hàm mất mát (loss function)**,  $\mathcal{L}$ , để đo lường mức độ "tệ" của dự đoán từ mô hình so với kết quả thực tế. Giải tích cung cấp bộ công cụ để thực hiện việc tối ưu hóa này.

**Hàm Mất mát (Loss Function)** Là một hàm số tính toán một giá trị vô hướng (scalar) biểu thị "sai số" hoặc "chi phí" của mô hình trên một tập dữ liệu. Mục tiêu của việc huấn luyện là tìm bộ tham số (trọng số)  $\theta$  của mô hình sao cho giá trị của hàm mất mát là nhỏ nhất.

- **Vai trò trong NLP:** Việc lựa chọn hàm mất mát phụ thuộc vào tác vụ. Ví dụ, hàm Cross-Entropy (xem Mục 1.4.4) thường được dùng cho các bài toán phân loại. Quá trình học chính là việc tìm kiếm  $\theta^* = \arg \min_{\theta} \mathcal{L}(\theta)$ .

**Đạo hàm (Derivative)** Đo lường tốc độ thay đổi của một hàm số.

- **Vai trò trong NLP:** Cho biết việc thay đổi một trọng số nhỏ trong mạng nơ-ron sẽ ảnh hưởng đến hàm mất mát tổng thể như thế nào.

**Gradient** Là một vector chứa tất cả các đạo hàm riêng (partial derivatives) của hàm mất mát  $\mathcal{L}$  theo từng tham số  $\theta_i$  trong mô hình, ký hiệu là  $\nabla_{\theta} \mathcal{L}(\theta)$ .

- **Vai trò trong NLP:** Gradient chỉ ra hướng mà tại đó hàm mất mát **tăng nhanh nhất**. Để tối ưu hóa mô hình (giảm thiểu mất mát), thuật toán **Gradient Descent** sẽ cập nhật các trọng số theo hướng *ngược lại* với gradient:  $\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta)$ , trong đó  $\eta$  là tốc độ học (learning rate).

**Quy tắc chuỗi (Chain Rule)** Là một công thức để tính đạo hàm của một hàm hợp.

- **Vai trò trong NLP:** Mạng nơ-ron sâu là một chuỗi các hàm số lồng vào nhau rất phức tạp. Quy tắc chuỗi là nền tảng toán học của thuật toán **làn truyền ngược (Backpropagation)** [76], cho phép chúng ta tính toán gradient của hàm mất mát theo từng tham số một cách hiệu quả, từ lớp cuối cùng ngược về lớp đầu tiên. Đây là một trong những khái niệm quan trọng nhất của Học sâu.

#### 1.4.3. Xác suất và Thống kê: Định lý Bayes, MLE

Ngôn ngữ tự nhiên vốn dĩ không chắc chắn và đầy mơ hồ. Xác suất và Thống kê cung cấp một bộ khung để mô hình hóa sự không chắc chắn này.

**Định lý Bayes (Bayes' Theorem)** Mô tả xác suất của một sự kiện, dựa trên kiến thức có trước về các điều kiện có thể liên quan đến sự kiện đó.

- **Vai trò trong NLP:** Là nền tảng của các mô hình sinh (generative models) và là trái tim của bộ phân loại Naive Bayes, một thuật toán phân loại văn bản kinh điển và hiệu quả.

**Uớc lượng Hợp lý Cực đại (Maximum Likelihood Estimation - MLE)** Là một phương pháp ước tính các tham số của một mô hình thống kê.

- **Vai trò trong NLP:** Ý tưởng cốt lõi của MLE là: "Hãy tìm bộ tham số (trọng số của mô hình) sao cho dữ liệu quan sát được là có khả năng xảy ra cao nhất". Việc huấn luyện hầu hết các mô hình học có giám sát bằng cách tối thiểu hóa một hàm mất mát (như Cross-Entropy) về mặt toán học tương đương với việc thực hiện MLE.

#### 1.4.4. Lý thuyết Thông tin: Entropy, Cross-Entropy, KL Divergence

Lý thuyết Thông tin [82], một nhánh của toán học ứng dụng, cung cấp các công cụ để định lượng thông tin. Trong NLP, nó đặc biệt hữu ích để định nghĩa các hàm mất mát.

**Entropy** Đo lường mức độ "bất ngờ" hay "không chắc chắn" trung bình của một biến ngẫu nhiên. Một phân phối đều (mọi kết quả có xác suất như nhau) có entropy cao nhất.

- **Vai trò trong NLP:** Là một khái niệm nền tảng để hiểu các hàm mất mát.
- **Công thức:**

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i) \quad (1.1)$$

**Cross-Entropy (Entropy chéo)** Đo lường "khoảng cách" giữa hai phân phối xác suất,  $p$  (phân phối thật) và  $q$  (phân phối dự đoán).

- **Vai trò trong NLP:** Đây là **hàm mất mát phổ biến nhất** cho các bài toán phân loại [25]. Trong một bài toán như vậy, chúng ta có phân phối xác suất "thật"  $p$  (ví dụ: nhãn đúng là 1, các nhãn khác là 0) và phân phối xác suất "dự đoán"  $q$  từ mô hình (đầu ra của lớp softmax). Mục tiêu của việc huấn luyện là tối thiểu hóa Cross-Entropy, tức là làm cho phân phối dự đoán  $q$  càng giống phân phối thật  $p$  càng tốt.

- Công thức:

$$H(p, q) = - \sum_{i=1}^n p(x_i) \log_2 q(x_i) \quad (1.2)$$

**Phân kỳ Kullback-Leibler (KL Divergence)** Cũng là một thước đo khoảng cách giữa hai phân phối xác suất  $p$  và  $q$ . Nó có mối quan hệ chặt chẽ với Cross-Entropy:  $H(p, q) = H(p) + D_{KL}(p||q)$ .

- **Vai trò trong NLP:** Được sử dụng rộng rãi trong các mô hình sinh (generative models) như Variational Autoencoders (VAEs) và trong việc so sánh sự khác biệt giữa các mô hình ngôn ngữ.

## 1.5. Đạo đức và Trách nhiệm trong NLP (Ethics & Responsible AI)

Khi các mô hình NLP ngày càng trở nên mạnh mẽ và được tích hợp sâu rộng vào xã hội – từ việc quyết định hồ sơ xin việc, duyệt đơn vay vốn, đến việc định hình dư luận – thì trách nhiệm của người tạo ra chúng cũng ngày càng lớn. Một mô hình NLP không chỉ là một công trình kỹ thuật; nó là một tác nhân có thể gây ra những ảnh hưởng thực tế, cả tích cực và tiêu cực, đến cuộc sống con người.

Do đó, việc xem xét các khía cạnh về đạo đức và xây dựng AI có trách nhiệm không phải là một lựa chọn, mà là một yêu cầu bắt buộc đối với bất kỳ ai làm việc trong lĩnh vực này. Mục này sẽ thảo luận về những thách thức và phương pháp tiếp cận chính.

### 1.5.1. Các loại Thiên kiến (Bias) trong Dữ liệu và Mô hình

Thiên kiến là một trong những vấn đề đạo đức phổ biến và nguy hiểm nhất trong NLP. Mô hình học máy không tự "suy nghĩ" ra thiên kiến; chúng học và khuếch đại những thiên kiến đã tồn tại sẵn trong dữ liệu mà chúng được huấn luyện.

#### Nguyên tắc cốt lõi: "Rác vào, Rác ra"

Mô hình NLP là tấm gương phản chiếu dữ liệu huấn luyện. Nếu dữ liệu chứa đựng các định kiến về giới tính, chủng tộc, hay văn hóa của xã hội, mô hình sẽ học và tái tạo lại những định kiến đó, thường với một quy mô lớn hơn rất nhiều. Đây còn được gọi là nguyên tắc "Garbage In, Garbage Out", nhưng trong bối cảnh đạo đức, nó trở thành "Bias In, Bias Out".

Một số loại thiên kiến phổ biến bao gồm:

- **Thiên kiến xã hội (Societal Bias):** Phản ánh các định kiến và khuôn mẫu trong xã hội.
  - **Ví dụ:** Các mô hình word embedding cổ điển khi được huấn luyện trên kho văn bản lớn thường học được các liên kết như: "đàn ông" gần với "lập trình viên", trong khi "phụ nữ" gần với "nội trợ". Một hệ thống tuyển dụng dựa trên mô hình này có thể sẽ tự động đánh giá thấp hồ sơ của ứng viên nữ cho vị trí kỹ thuật.
- **Thiên kiến chọn mẫu (Selection Bias):** Xảy ra khi dữ liệu được thu thập không đại diện cho thực tế.
  - **Ví dụ:** Xây dựng một mô hình phân tích cảm xúc chỉ dựa trên các bài đánh giá sản phẩm công nghệ có thể sẽ hoạt động kém khi áp dụng cho các bình luận về phim ảnh, vì cách dùng từ và biểu đạt cảm xúc là khác nhau.
- **Thiên kiến từ mô hình (Model Bias):** Bản thân kiến trúc hoặc quá trình tối ưu hóa của mô hình có thể vô tình tạo ra hoặc khuếch đại thiên kiến.
  - **Ví dụ:** Một mô hình được tối ưu hóa để đạt độ chính xác cao nhất trên một bộ dữ liệu mất cân bằng (99% email là không spam, 1% là spam) có thể học được một chiến lược đơn giản là "luôn dự đoán không spam" để đạt độ chính xác 99%, nhưng hoàn toàn vô dụng trong thực tế.

### 1.5.2. Tính Công bằng, Minh bạch và Diễn giải được (Fairness, Transparency, Interpretability)

Để đối phó với thiên kiến và xây dựng các hệ thống đáng tin cậy, ba khái niệm sau đây là trụ cột chính:

**Tính Công bằng (Fairness)** Là nỗ lực đảm bảo rằng các quyết định của mô hình không gây bất lợi một cách có hệ thống cho các nhóm nhân khẩu học nhất định (dựa trên giới tính, chủng tộc, tuổi tác, v.v.). Đây là một khái niệm phức tạp, không có một định nghĩa toán học duy nhất nào là hoàn hảo cho mọi trường hợp.

**Tính Minh bạch (Transparency)** Đề cập đến khả năng hiểu được cơ chế hoạt động bên trong của mô hình. Các mô hình đơn giản như Hồi quy Logistic có tính minh bạch cao (chúng ta có thể xem các trọng số). Tuy nhiên, các mô hình ngôn ngữ lớn (LLMs) là những "hộp đen" (black boxes) gần như không thể minh bạch.

**Tính Diễn giải được (Interpretability / Explainable AI - XAI)** Khi không thể đạt được sự minh bạch hoàn toàn, chúng ta hướng tới tính diễn giải được: khả năng giải thích *tại sao* mô hình lại đưa ra một dự đoán cụ thể. Hai kỹ thuật phổ biến để làm điều này là:

- **LIME (Local Interpretable Model-agnostic Explanations):** Giải thích một dự đoán riêng lẻ bằng cách xây dựng một mô hình đơn giản, có thể diễn giải được (như một mô hình tuyến tính) chỉ hoạt động tốt "tại vùng lân cận" của dự đoán đó. Nó trả lời câu hỏi: "Những từ nào trong câu này đã đóng góp nhiều nhất vào quyết định của mô hình?".
- **SHAP (SHapley Additive exPlanations):** Dựa trên lý thuyết trò chơi, SHAP tính toán sự đóng góp của mỗi đặc trưng (mỗi từ) vào đầu ra của mô hình một cách công bằng và nhất quán. Nó cung cấp một nền tảng lý thuyết vững chắc hơn cho việc diễn giải.

### 1.5.3. Quyền riêng tư và An toàn Dữ liệu (Federated Learning, Differential Privacy)

Các mô hình NLP được huấn luyện trên lượng dữ liệu khổng lồ, thường được thu thập từ người dùng. Điều này làm dấy lên những lo ngại nghiêm trọng về quyền riêng tư.

**Vấn đề:** Các mô hình, đặc biệt là LLMs, có thể "ghi nhớ" và vô tình tiết lộ thông tin nhận dạng cá nhân (Personally Identifiable Information - PII) có trong dữ liệu huấn luyện, chẳng hạn như số điện thoại, địa chỉ email, hoặc thông tin y tế nhạy cảm.

Hai kỹ thuật tiên tiến để giải quyết vấn đề này là:

- **Học Liên kết (Federated Learning - FL):** Thay vì thu thập tất cả dữ liệu về một máy chủ trung tâm để huấn luyện, phương pháp này "gửi" mô hình đến các thiết bị của người dùng (ví dụ: điện thoại di động). Mô hình được huấn luyện cục bộ trên dữ liệu của người dùng, sau đó chỉ có các cập nhật (gradients) của mô hình được gửi về máy chủ để tổng hợp. Dữ liệu không bao giờ rời khỏi thiết bị của người dùng.
- **Bảo toàn riêng tư vi phân (Differential Privacy - DP):** Cung cấp một sự đảm bảo toán học mạnh mẽ về quyền riêng tư. Ý tưởng cốt lõi là thêm một lượng nhiễu (noise) được kiểm soát cẩn thận vào dữ liệu hoặc kết quả của thuật toán, sao cho việc thêm hay bớt dữ liệu của một cá nhân bất kỳ trong tập dữ liệu gần như không làm thay đổi kết quả cuối cùng. Điều này khiến cho việc xác định thông tin của một cá nhân cụ thể trở nên bất khả thi.

### 1.5.4. Mọi nguy về Thông tin Sai lệch và Lạm dụng

Sự ra đời của các mô hình sinh ngôn ngữ (Generative Models) có khả năng tạo ra văn bản chất lượng cao, mạch lạc và thuyết phục đã mở ra một "chiếc hộp Pandora" về các nguy cơ lạm dụng.

- **Thông tin sai lệch (Misinformation & Disinformation):** Các mô hình có thể được sử dụng để tạo ra hàng loạt tin tức giả, các bài đăng trên mạng xã hội, các bài đánh giá sản phẩm giả mạo với quy mô và tốc độ chưa từng có, gây ảnh hưởng đến dư luận, chính trị và xã hội.
- **Lừa đảo và Tấn công mạng (Scams & Phishing):** Tự động tạo ra các email lừa đảo tinh vi, được cá nhân hóa cao, khiến người dùng khó phân biệt hơn nhiều.
- **Sáng tạo nội dung độc hại (Generation of Harmful Content):** Các mô hình có thể bị lợi dụng để tạo ra ngôn từ thù địch, tuyên truyền cực đoan, hoặc nội dung quấy rối.

Việc giải quyết những vấn đề này không chỉ nằm ở khía cạnh kỹ thuật (ví dụ: xây dựng các mô hình phát hiện văn bản do AI tạo ra) mà còn đòi hỏi sự chung tay của toàn xã hội, bao gồm việc xây dựng các chính sách quản lý, nâng cao nhận thức và giáo dục kỹ năng số cho cộng đồng.

Kết thúc chương đầu tiên này, hy vọng bạn không chỉ nắm được "NLP là gì?" mà còn nhận thức được trách nhiệm to lớn đi kèm với sức mạnh của nó.

#### 1.5.5. Tính Bền vững và Tấn công Giả mạo (Robustness & Adversarial Attacks)

Ngoài các vấn đề về thiên kiến và quyền riêng tư, một khía cạnh quan trọng khác của AI có trách nhiệm là đảm bảo mô hình hoạt động một cách đáng tin cậy trong thế giới thực. Một mô hình có thể đạt độ chính xác 99% trên bộ dữ liệu kiểm thử sạch (clean test set), nhưng lại có thể thất bại thảm hại khi đối mặt với dữ liệu thực tế vốn luôn nhiễu và không hoàn hảo. Đây là bài toán về **tính bền vững (robustness)**.

Đáng lo ngại hơn, các mô hình NLP, đặc biệt là các mô hình học sâu, lại rất dễ bị tổn thương trước các **tấn công giả mạo (adversarial attacks)** – những thay đổi nhỏ, thường không thể nhận biết bởi con người, được thiết kế một cách có chủ đích để đánh lừa mô hình.

##### Sự Mong manh của các Mô hình "Thông minh"

Một tấn công giả mạo có thể thay đổi dự đoán của mô hình một cách hoàn toàn. Ví dụ, một mô hình phân tích cảm xúc có thể phân loại câu "Bộ phim này thật tuyệt vời" là **TÍCH CỰC**. Khi tấn công có thể chỉ cần thêm một vài ký tự vô hình hoặc thay một từ bằng một từ đồng nghĩa được chọn lọc cẩn thận để biến nó thành câu "Bộ phim này thật *xuất chúng*", và mô hình đột ngột thay đổi dự đoán thành **TIÊU CỰC**. Đối với con người, ý nghĩa không thay đổi, nhưng đối với mô hình, kết quả đã bị đảo ngược.

Các kỹ thuật tấn công phổ biến bao gồm:

- **Tấn công cấp độ ký tự (Character-level):** Thay thế, chèn, hoặc xóa các ký tự. Ví dụ: thay chữ "o" bằng số "0", hoặc chèn các ký tự không thể in ra (non-printable characters). *DeepWordBug* [23] hay *HotFlip* [21] thực hiện các kỹ thuật này.
- **Tấn công cấp độ từ (Word-level):** Thay thế các từ bằng từ đồng nghĩa của chúng. Thách thức ở đây là tìm ra từ đồng nghĩa nào sẽ đánh lừa mô hình mà không làm thay đổi ý nghĩa của câu đối với con người. *TextFooler* [37] là một ví dụ kinh điển, nó xác định các từ quan trọng nhất trong câu và thay thế chúng một cách chiến lược.
- **Tấn công cấp độ câu (Sentence-level):** Diễn giải (paraphrase) lại toàn bộ câu để giữ nguyên ý nghĩa nhưng sử dụng cấu trúc và từ vựng khác, nhằm mục đích tìm ra một phiên bản mà mô hình không thể hiểu đúng.

Việc một mô hình dễ bị tấn công cho thấy nó không thực sự "hiểu" ngôn ngữ, mà chỉ đang dựa vào các quy luật và mẫu thống kê bề mặt. Để phòng thủ, các nhà nghiên cứu đã phát triển nhiều kỹ thuật, trong đó phổ biến nhất là:

- **Huấn luyện Giả mạo (Adversarial Training):** Đây là phương pháp phòng thủ trực tiếp và hiệu quả nhất. Trong quá trình huấn luyện, chúng ta chủ động tạo ra các mẫu dữ liệu giả mạo và đưa chúng vào tập huấn luyện cùng với nhãn đúng. Điều này giống như việc "tiêm vắc-xin" cho mô hình, giúp nó học cách miễn nhiễm với các loại tấn công tương tự trong tương lai.

- **Phát hiện Dữ liệu Giả mạo (Adversarial Detection):** Xây dựng một mô hình phụ để xác định xem một đầu vào có phải là một mẫu giả mạo hay không trước khi đưa nó vào mô hình chính. Việc xây dựng các mô hình bền vững không chỉ là một thách thức kỹ thuật mà còn là một yêu cầu về mặt đạo đức, đặc biệt khi NLP được triển khai trong các hệ thống có tính rủi ro cao như lọc nội dung độc hại, phát hiện tin giả, hay các ứng dụng an ninh mạng.
- 

## KẾT THÚC CHƯƠNG 1

*Kết thúc chương mở đầu này, bạn đã có trong tay một tấm bản đồ toàn diện về thế giới NLP. Chúng ta đã cùng nhau định nghĩa lĩnh vực, khám phá các nền tảng ngôn ngữ học và toán học, vạch ra các bài toán cốt lõi, và quan trọng nhất, thiết lập một la bàn đạo đức cho hành trình phía trước. Giờ đây, khi đã hiểu rõ "cái gì" và "tại sao", đã đến lúc chúng ta đi sâu vào kỹ thuật đầu tiên: làm thế nào để biến văn bản thành các con số có thể tính toán được. Chương tiếp theo sẽ giới thiệu các phương pháp biểu diễn văn bản kinh điển, khởi đầu cho kỷ nguyên thống kê của NLP.*

## Chương 2

# BIỂU DIỄN VĂN BẢN VÀ CÁC MÔ HÌNH THỐNG KẾ

Chào mừng bạn đến với kỷ nguyên thống kê của NLP. Tại đây, chúng ta sẽ tạm rời xa các bộ quy tắc ngôn ngữ học cứng nhắc để bước vào một thế giới nơi dữ liệu là vua. Nền tảng của kỷ nguyên này nằm ở một câu hỏi vô cùng cơ bản nhưng quan trọng: **Làm thế nào để biến văn bản – một chuỗi các ký tự – thành một dạng thức mà máy tính có thể hiểu và tính toán được, tức là các con số?**

Chương này sẽ giới thiệu các phương pháp kinh điển để "số hóa" ngôn ngữ, từ những kỹ thuật đơn giản đến các mô hình xác suất phức tạp hơn. Đây là những viên gạch nền móng, mà trên đó các thuật toán học máy thống kê đã xây dựng nên những ứng dụng NLP thành công đầu tiên.

### 2.1. Biểu diễn Dựa trên Tần suất (Bag-of-Words, TF-IDF)

Phương pháp tiếp cận đầu tiên và trực quan nhất để biểu diễn văn bản là dựa trên một giả định đơn giản: *tần suất xuất hiện của các từ trong một tài liệu phản ánh nội dung chính của tài liệu đó*. Hai kỹ thuật tiêu biểu nhất cho trường phái này là Bag-of-Words và TF-IDF.

#### 2.1.1. Túi từ (Bag-of-Words - BoW)

##### Túi duy cốt lõi

Mô hình Bag-of-Words (BoW) là một trong những phương pháp biểu diễn văn bản đơn giản nhưng mạnh mẽ nhất. Túi duy cốt lõi của nó là **hoàn toàn bỏ qua trật tự từ và cấu trúc ngữ pháp**, và chỉ **quan tâm đến việc từ nào xuất hiện và xuất hiện bao nhiêu lần** trong một tài liệu.

Hãy tưởng tượng bạn lấy tất cả các từ trong một câu, cho chúng vào một chiếc túi, xáo trộn lên và rồi thống kê số lượng của từng từ. Chiếc túi đó chính là "Bag-of-Words". Kết quả của quá trình này là một vector số, trong đó mỗi chiều của vector tương ứng với một từ trong từ vựng và giá trị của chiều đó là tần suất xuất hiện của từ.

##### Quy trình xây dựng mô hình BoW

Để xây dựng biểu diễn BoW cho một tập hợp các tài liệu (corpus), chúng ta thực hiện ba bước chính:

**Bước 1: Tokenization (Tách từ)** Tách các câu trong tài liệu thành các từ riêng lẻ (tokens). Đây là bước cơ bản để có được đơn vị cần đếm. Ví dụ, câu "NLP rất thú vị" được tách thành ['NLP', 'rất', 'thú vị'].

**Bước 2: Xây dựng Từ vựng (Vocabulary Building)** Tập hợp tất cả các từ độc nhất từ toàn bộ kho tài liệu để tạo ra một bộ từ vựng. Thứ tự của các từ trong từ vựng này sẽ quyết định thứ tự các chiều trong vector biểu diễn cuối cùng. Ví dụ, nếu từ vựng là ['NLP', 'rất', 'thú vị', 'học', 'thích'], thì từ 'NLP' sẽ tương ứng với chiều thứ nhất, 'rất' với chiều thứ hai, v.v.

**Bước 3: Vector hóa (Vectorization)** Đối với mỗi tài liệu, tạo ra một vector có số chiều bằng kích thước của từ vựng. Với mỗi từ trong từ vựng, ta đếm số lần nó xuất hiện trong tài liệu và điền con số đó vào chiều tương ứng của vector. Nếu một từ trong từ vựng không xuất hiện trong tài liệu, giá trị ở chiều đó sẽ là 0.

#### Ví dụ 4: Xây dựng biểu diễn BoW

Giả sử chúng ta có một kho tài liệu nhỏ (corpus) gồm 2 câu:

- Câu 1: "Tôi thích học NLP."
- Câu 2: "Học NLP rất rất thú vị."

#### Bước 1 & 2: Tokenization và Xây dựng Từ vựng

Sau khi tách từ và loại bỏ dấu câu, chúng ta có các từ: 'tôi', 'thích', 'học', 'nlp', 'học', 'nlp', 'rất', 'thú vị'. Từ vựng độc nhất (sắp xếp theo alphabet) sẽ là: [ 'học', 'NLP', 'rất', 'thích', 'thú vị', 'tôi' ] Kích thước từ vựng là 6.

#### Bước 3: Vector hóa

Bây giờ, chúng ta biểu diễn mỗi câu bằng một vector 6 chiều:

- Câu 1: "tôi thích học nlp"
  - 'học': 1 lần
  - 'NLP': 1 lần
  - 'rất': 0 lần
  - 'thích': 1 lần
  - 'thú vị': 0 lần
  - 'tôi': 1 lần
 → Vector biểu diễn: [1, 1, 0, 1, 0, 1]
- Câu 2: "học nlp rất rất thú vị"
  - 'học': 1 lần
  - 'NLP': 1 lần
  - 'rất': 2 lần
  - 'thích': 0 lần
  - 'thú vị': 1 lần
  - 'tôi': 0 lần
 → Vector biểu diễn: [1, 1, 2, 0, 1, 0]

#### Ưu điểm và Nhược điểm của BoW

Mô hình BoW, mặc dù đơn giản, lại là nền tảng cho nhiều hệ thống phân loại văn bản kinh điển.

#### Dánh giá mô hình Bag-of-Words

##### Ưu điểm:

- **Đơn giản và trực quan:** Rất dễ hiểu và dễ triển khai.
- **Hiệu quả tính toán:** Việc tạo và sử dụng các vector BoW tương đối nhanh.
- **Hoạt động tốt cho các tác vụ phân loại chủ đề:** Nếu hai tài liệu có các bộ từ vựng tương tự nhau, chúng có khả năng cùng một chủ đề. BoW nắm bắt rất tốt điều này.

##### Nhược điểm:

- **Mất thông tin về trật tự từ:** Câu "chó cắn người" và "người cắn chó" có cùng biểu diễn BoW, nhưng ý nghĩa hoàn toàn trái ngược. Đây là nhược điểm lớn nhất.
- **Không nắm bắt được ngữ nghĩa:** BoW không hiểu rằng "tốt" và "tuyệt vời" là những từ gần nghĩa. Đối với nó, đây là hai chiều hoàn toàn khác biệt trong không gian vector.
- **Vấn đề về kích thước từ vựng và độ thừa thót (Sparsity):** Với một kho dữ liệu lớn, kích thước từ vựng có thể lên tới hàng trăm nghìn từ. Điều này tạo ra các vector có số chiều rất lớn, và hầu hết các giá trị trong vector đều là 0 (vì một tài liệu chỉ chứa một phần

nhỏ của từ vựng). Các vector thưa thớt này gây khó khăn cho nhiều thuật toán học máy.

### 2.1.2. TF-IDF: Trọng số hóa tầm quan trọng của từ

Một trong những vấn đề của BoW là nó coi mọi từ có vai trò như nhau. Một từ xuất hiện nhiều lần (như "là", "thì", "của", và gọi là các stop words) có thể lấn át các từ mang ý nghĩa quan trọng nhưng xuất hiện ít hơn. TF-IDF (Term Frequency - Inverse Document Frequency) [85] ra đời để giải quyết vấn đề này.

#### Tư duy cốt lõi

Tư duy của TF-IDF là: **Một từ càng quan trọng đối với một tài liệu nếu nó xuất hiện nhiều lần trong tài liệu đó (tính quan trọng cục bộ), nhưng lại xuất hiện ít trong các tài liệu khác của toàn bộ kho dữ liệu (tính độc nhất).**

TF-IDF tính toán một trọng số cho mỗi từ trong mỗi tài liệu, thay vì chỉ đếm tần suất. Trọng số này được cấu thành từ hai thành phần: TF và IDF.

#### Các thành phần của TF-IDF

**Term Frequency (TF) - Tần suất của Từ** TF đo lường tần suất xuất hiện của một từ  $t$  trong một tài liệu  $d$ . Có nhiều cách tính TF, nhưng cách đơn giản nhất là đếm số lần xuất hiện.

$$\text{TF}(t, d) = \text{số lần từ } t \text{ xuất hiện trong tài liệu } d$$

Để tránh việc các tài liệu dài có lợi thế, người ta thường chuẩn hóa TF, ví dụ bằng cách chia cho tổng số từ trong tài liệu.

**Inverse Document Frequency (IDF) - Tần suất Nghịch của Tài liệu** IDF đo lường mức độ "hiếm" hay "độc nhất" của một từ trên toàn bộ kho tài liệu. Nếu một từ xuất hiện trong rất nhiều tài liệu, IDF của nó sẽ thấp và ngược lại.

$$\text{IDF}(t, D) = \log \left( \frac{N}{|\{d \in D : t \in d\}|} \right)$$

Trong đó:

- $N$  là tổng số tài liệu trong kho dữ liệu  $D$ .
- $|\{d \in D : t \in d\}|$  là số tài liệu có chứa từ  $t$ .
- Logarit được dùng để "làm mịn" giá trị, tránh việc các từ cực hiếm có trọng số quá lớn. Để tránh chia cho 0, trong thực tế mẫu số thường được cộng thêm 1.

**Trọng số TF-IDF** Trọng số cuối cùng của từ  $t$  trong tài liệu  $d$  là tích của TF và IDF.

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \quad (2.1)$$

#### Ví dụ 5: Tính toán trọng số TF-IDF

Chúng ta tiếp tục với kho tài liệu ở ví dụ ??:

- Câu 1 (d1): "Tôi thích học NLP."
- Câu 2 (d2): "Học NLP rất rất thú vị."
- **Từ vựng:** ['học', 'NLP', 'rất', 'thích', 'thú vị', 'tôi']'
- **Tổng số tài liệu N = 2**

1. Tính TF (dùng số đếm thô):

Từ (t)	TF(t, d1)	TF(t, d2)
'học'	1	1
'NLP'	1	1
'rất'	0	2
'thích'	1	0
'thú vị'	0	1
'tôi'	1	0

**2. Tính IDF:**

- 'học': xuất hiện trong 2 tài liệu  $\rightarrow \text{IDF} = \log(2/2) = 0$
- 'NLP': xuất hiện trong 2 tài liệu  $\rightarrow \text{IDF} = \log(2/2) = 0$
- 'rất': xuất hiện trong 1 tài liệu  $\rightarrow \text{IDF} = \log(2/1) \approx 0.693$
- 'thích': xuất hiện trong 1 tài liệu  $\rightarrow \text{IDF} = \log(2/1) \approx 0.693$
- 'thú vị': xuất hiện trong 1 tài liệu  $\rightarrow \text{IDF} = \log(2/1) \approx 0.693$
- 'tôi': xuất hiện trong 1 tài liệu  $\rightarrow \text{IDF} = \log(2/1) \approx 0.693$

**3. Tính TF-IDF = TF \* IDF:**

Từ (t)	TF-IDF(t, d1)	TF-IDF(t, d2)
'học'	$1 * 0 = 0$	$1 * 0 = 0$
'NLP'	$1 * 0 = 0$	$1 * 0 = 0$
'rất'	$0 * 0.693 = 0$	$2 * 0.693 = 1.386$
'thích'	$1 * 0.693 = 0.693$	$0 * 0.693 = 0$
'thú vị'	$0 * 0.693 = 0$	$1 * 0.693 = 0.693$
'tôi'	$1 * 0.693 = 0.693$	$0 * 0.693 = 0$

**Kết quả Vector TF-IDF:**

- Câu 1 (d1): '[0, 0, 0, 0.693, 0, 0.693]'
- Câu 2 (d2): '[0, 0, 1.386, 0, 0.693, 0]'

**Nhận xét quan trọng:** Các từ 'học' và 'NLP' xuất hiện trong cả hai câu, bị coi là "thông tin chung" và có trọng số TF-IDF bằng 0. Ngược lại, các từ 'thích', 'tôi' trở thành đặc trưng của Câu 1, trong khi 'rất', 'thú vị' là đặc trưng của Câu 2. Đây chính là sức mạnh của TF-IDF.

**Tổng kết về TF-IDF**

TF-IDF là một bước tiến lớn so với BoW. Nó vẫn giữ được sự đơn giản trong tính toán nhưng thông minh hơn trong việc trọng số hóa các từ. Nó đã và đang được sử dụng rộng rãi trong các hệ thống tìm kiếm thông tin (information retrieval) và phân loại văn bản.

Tuy nhiên, TF-IDF vẫn kế thừa các nhược điểm cố hữu của họ phương pháp dựa trên túi từ: nó vẫn bỏ qua trật tự từ và không hiểu được ngữ nghĩa. Những hạn chế này chính là động lực để cộng đồng NLP phát triển các phương pháp biểu diễn phức tạp và mạnh mẽ hơn, mà chúng ta sẽ khám phá trong phần tiếp theo: Word Embeddings.

**2.2. Mô hình Ngôn ngữ Thống kê (N-gram, Smoothing)**

Trong mục trước, chúng ta đã học cách biểu diễn văn bản như một "túi từ" không có trật tự. Bây giờ, chúng ta sẽ đưa trật tự từ trở lại và giới thiệu một trong những ý tưởng mạnh mẽ nhất của NLP thống kê: **Mô hình Ngôn ngữ (Language Model - LM)**.

**Định nghĩa 3: Mô hình Ngôn ngữ (LM)**

Một Mô hình Ngôn ngữ là một mô hình xác suất có khả năng gán một giá trị xác suất cho một chuỗi các từ bất kỳ. Nói cách khác, một LM có thể trả lời câu hỏi: "Xác suất để một chuỗi từ  $W = w_1, w_2, \dots, w_n$  xuất hiện trong một ngôn ngữ là bao nhiêu?", ký hiệu là  $P(W)$ .

Một LM tốt sẽ gán xác suất cao cho những câu "tự nhiên", đúng ngữ pháp (ví dụ: "tôi thích học NLP") và gán xác suất rất thấp cho những câu vô nghĩa, sai ngữ pháp (ví dụ: "thích tôi học NLP NLP").

**Ứng dụng của Mô hình Ngôn ngữ:** LM là trái tim của rất nhiều ứng dụng NLP, đặc biệt là các ứng dụng sinh ngôn ngữ:

- **Dịch máy:** Chọn câu dịch có xác suất cao nhất trong ngôn ngữ đích.
- **Nhận dạng giọng nói:** Phân biệt giữa hai câu phát âm gần giống nhau (ví dụ: "I saw a van" và "ice Havana") bằng cách chọn câu có xác suất ngôn ngữ cao hơn.
- **Kiểm tra chính tả và ngữ pháp:** Gọi ý sửa lỗi "I is a student" thành "I am a student" vì câu sau có xác suất cao hơn nhiều.
- **Gợi ý từ tiếp theo (Predictive Text):** Dự đoán từ có khả năng xuất hiện tiếp theo nhất dựa trên các từ đã gõ.

**2.2.1. Mô hình N-gram: Học từ Lịch sử Gần nhất****Thách thức của việc tính toán xác suất chuỗi**

Để tính xác suất của một chuỗi từ  $P(W) = P(w_1, w_2, \dots, w_n)$ , chúng ta có thể sử dụng quy tắc chuỗi (chain rule) trong xác suất:

$$P(w_1, w_2, \dots, w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \times \dots \times P(w_n|w_1, \dots, w_{n-1})$$

$$P(W) = \prod_{i=1}^n P(w_i|w_1, \dots, w_{i-1})$$

Tuy nhiên, việc tính toán trực tiếp công thức này là **bất khả thi** trong thực tế. Để tính  $P(w_n|w_1, \dots, w_{n-1})$ , chúng ta cần phải thống kê được tần suất của chuỗi  $w_1, \dots, w_{n-1}, w_n$  đã xuất hiện trong kho dữ liệu huấn luyện. Với các câu dài, chuỗi này gần như chắc chắn chưa từng xuất hiện bao giờ, dẫn đến vấn đề **dữ liệu thừa thớt (sparsity)** nghiêm trọng.

**Giả định Markov và N-gram**

Để giải quyết vấn đề này, các mô hình N-gram đưa ra một giả định đơn giản hóa, gọi là **Giả định Markov (Markov Assumption)**:

**Giả định Markov**

Xác suất của từ tiếp theo không phụ thuộc vào toàn bộ lịch sử các từ trước đó, mà chỉ phụ thuộc vào một vài từ đứng ngay trước nó. Cụ thể, nó chỉ phụ thuộc vào  $N - 1$  từ gần nhất.

Dựa trên giả định này, xác suất của từ  $w_i$  được xấp xỉ như sau:

$$P(w_i|w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-N+1}, \dots, w_{i-1})$$

Một chuỗi gồm  $N$  từ được gọi là **N-gram**. Dựa trên giá trị của  $N$ , chúng ta có các mô hình cụ thể:

- **Unigram ( $N=1$ ):** Xác suất của một từ không phụ thuộc vào bất kỳ từ nào trước đó.  $P(w_i|w_1, \dots, w_{i-1}) \approx P(w_i)$ . Mô hình này tương đương với Bag-of-Words.

- Bigram (N=2):** Xác suất của một từ chỉ phụ thuộc vào từ đứng ngay trước nó.  $P(w_i|w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-1})$ .
- Trigram (N=3):** Xác suất của một từ phụ thuộc vào hai từ đứng ngay trước nó.  $P(w_i|w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$ .

### Ước lượng xác suất N-gram

Xác suất của một N-gram có thể được ước lượng dễ dàng từ một kho văn bản huấn luyện bằng phương pháp Ước lượng Hợp lý Cực đại (MLE):

$$P(w_i|w_{i-N+1}, \dots, w_{i-1}) = \frac{\text{Count}(w_{i-N+1}, \dots, w_{i-1}, w_i)}{\text{Count}(w_{i-N+1}, \dots, w_{i-1})}$$

Nói cách khác, xác suất của một bigram  $P(w_i|w_{i-1})$  là số lần chuỗi " $w_{i-1}w_i$ " xuất hiện, chia cho số lần tiền tố " $w_{i-1}$ " xuất hiện.

#### Ví dụ 6: Xây dựng và sử dụng mô hình Bigram

Giả sử chúng ta có một kho văn bản rất nhỏ sau khi đã thêm các ký hiệu bắt đầu ('<s>') và kết thúc ('</s>') câu:

- '<s> tôi thích học NLP </s>'
- '<s> tôi thích AI </s>'
- '<s> học NLP rất vui </s>'

##### 1. Đếm tần suất Bigram:

Bigram	Count	Bigram	Count
'(<s>, tôi)'	2	'(học, NLP)'	2
'(tôi, thích)'	2	'(NLP, </s>)'	1
'(thích, học)'	1	'(NLP, rất)'	1
'(thích, AI)'	1	'(rất, vui)'	1
'(AI, </s>)'	1	'(vui, </s>)'	1
'(<s>, học)'	1		

2. Đếm tần suất Unigram (tiền tố): 'Count(<s>) = 3', 'Count(tôi) = 2', 'Count(thích) = 2', 'Count(học) = 2', 'Count(NLP) = 2', ...

##### 3. Tính xác suất Bigram:

- $P(\text{tôi}|<\text{s}>) = \frac{\text{Count}(<\text{s}>, \text{tôi})}{\text{Count}(<\text{s}>)} = \frac{2}{3}$
- $P(\text{học}|\text{thích}) = \frac{\text{Count}(\text{thích}, \text{học})}{\text{Count}(\text{thích})} = \frac{1}{2}$
- $P(\text{AI}|\text{thích}) = \frac{\text{Count}(\text{thích}, \text{AI})}{\text{Count}(\text{thích})} = \frac{1}{2}$
- $P(\text{rất}|\text{NLP}) = \frac{\text{Count}(\text{NLP}, \text{rất})}{\text{Count}(\text{NLP})} = \frac{1}{2}$

4. Tính xác suất cho một câu mới: Hãy tính xác suất của câu "tôi thích NLP":

$$P(<\text{s}> \text{tôi thích NLP } </\text{s}>) = P(\text{tôi}|<\text{s}>) \times P(\text{thích}|\text{tôi}) \times P(\text{NLP}|\text{thích}) \times P(</\text{s}>|\text{NLP})$$

Tuy nhiên, hãy nhìn vào bảng tần suất. Bigram '(thích, NLP)' chưa từng xuất hiện! 'Count(thích, NLP) = 0'. Điều này dẫn đến  $P(\text{NLP}|\text{thích}) = 0$ . Và kết quả là  $P(<\text{s}> \text{tôi thích NLP } </\text{s}>) = 0$ .

Mô hình của chúng ta đã gán xác suất bằng 0 cho một câu hoàn toàn hợp lý. Đây chính là vấn đề xác suất zero.

#### 2.2.2. Làm mịn (Smoothing): Giải quyết vấn đề Xác suất Zero

Vấn đề xác suất zero xảy ra vì kho văn bản huấn luyện của chúng ta luôn có giới hạn và không thể chứa tất cả các N-gram có thể có trong một ngôn ngữ. Việc gán xác suất bằng 0 cho một N-gram chưa từng thấy là rất nguy hiểm, vì nó sẽ khiến xác suất của cả câu bằng 0.

Làm mịn (Smoothing) là một tập hợp các kỹ thuật được thiết kế để "vay mượn" một phần khối lượng xác suất từ các N-gram đã thấy và phân phối lại nó cho các N-gram chưa từng thấy.

### Cộng Laplace (Laplace Add-One Smoothing)

Đây là kỹ thuật làm mịn đơn giản nhất và trực quan nhất. Ý tưởng là: Hãy giả vờ rằng chúng ta đã thấy tất cả các N-gram có thể có ít nhất một lần.

#### Dịnh nghĩa 4: Làm mịn cộng Laplace

Khi tính toán xác suất, chúng ta cộng 1 vào tử số (số đếm N-gram) và cộng  $V$  (kích thước từ vựng) vào mẫu số (số đếm tiền tố).

$$P_{\text{Laplace}}(w_i|w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i) + 1}{\text{Count}(w_{i-1}) + V}$$

#### Ví dụ 7: Áp dụng làm mịn Laplace

Quay trở lại ví dụ Bigram. Giả sử từ vựng của chúng ta gồm 6 từ: 'tôi, thích, học, NLP, AI, rất, vui' cộng với '<s>, </s>', vậy  $V = 9$ . Chúng ta muốn tính lại  $P(\text{NLP}|\text{thích})$ .

- 'Count(thích, NLP) = 0'
- 'Count(thích) = 2'

$$P_{\text{Laplace}}(\text{NLP}|\text{thích}) = \frac{0 + 1}{2 + 9} = \frac{1}{11}$$

Xác suất bây giờ đã khác 0! Tuy nhiên, hãy xem điều gì xảy ra với các xác suất khác:

$$P_{\text{Laplace}}(\text{học}|\text{thích}) = \frac{1 + 1}{2 + 9} = \frac{2}{11}$$

**Vấn đề của Laplace Smoothing:** Nó hoạt động, nhưng nó thường "cho đi" quá nhiều khối lượng xác suất. Trong các từ vựng lớn, nó làm thay đổi đáng kể xác suất của các N-gram đã thấy, khiến mô hình kém chính xác hơn.

### Các kỹ thuật làm mịn tiên tiến hơn

Do nhược điểm của Laplace, nhiều kỹ thuật phức tạp hơn đã được phát triển. Dưới đây là ý tưởng chính của một số phương pháp phổ biến:

**Làm mịn Add-k (Add-k Smoothing)** Một sự tổng quát hóa của Laplace, thay vì cộng 1, chúng ta cộng một hằng số nhỏ  $k$  (ví dụ:  $k = 0.1$ ).

$$P_{\text{Add-k}}(w_i|w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i) + k}{\text{Count}(w_{i-1}) + kV}$$

Điều này giúp giảm bớt việc "cho đi" quá nhiều xác suất, nhưng việc chọn  $k$  tối ưu cũng là một thách thức.

**Good-Turing Smoothing** Ý tưởng cốt lõi là sử dụng số lượng các N-gram chỉ xuất hiện một lần ('Count=1') để ước tính tổng xác suất của các N-gram chưa từng thấy ('Count=0'). Nó dựa trên một quan sát thông minh: tần suất của những thứ bạn chưa thấy có thể được ước tính bằng tần suất của những thứ bạn chỉ thấy một lần.

**Kneser-Ney Smoothing** Đây được coi là kỹ thuật làm mịn **hiện đại và hiệu quả nhất** cho các mô hình N-gram. Nó rất phức tạp nhưng ý tưởng chính là: xác suất của một N-gram chưa thấy không nên được tính dựa trên tần suất của tiền tố, mà nên dựa trên *số lượng các ngữ cảnh khác nhau mà từ cuối cùng của N-gram đó đã xuất hiện*. Nó trả lời câu hỏi "Từ 'San Francisco' có khả năng đi sau từ 'I live in' như thế nào?". Kneser-Ney sẽ ưu tiên các từ như 'Francisco' (vốn thường đi sau các từ khác như 'San') hơn là các từ phổ biến nhưng ít đa dạng về ngữ cảnh.

### 2.2.3. Tổng kết và Hạn chế của N-gram

Mô hình N-gram là một công cụ mạnh mẽ, tương đối đơn giản và đã thống trị các ứng dụng NLP trong nhiều năm. Tuy nhiên, chúng có những hạn chế cố hữu:

- **Vấn đề thưa thớt (Sparsity):** Mặc dù có các kỹ thuật làm mịn, vấn đề này vẫn tồn tại, đặc biệt với  $N > 3$ . Việc lưu trữ tất cả các N-gram cũng rất tốn bộ nhớ.
- **Không nắm bắt được phụ thuộc tầm xa:** Một mô hình trigram không thể nắm bắt được mối liên hệ giữa chủ ngữ và động từ trong câu "The man who I saw yesterday in the park is my friend." Nó chỉ nhìn thấy "in the park is" và sẽ gặp khó khăn.
- **Không có khái niệm về ngữ nghĩa:** Giống như BoW, N-gram không hiểu rằng "mèo" và "hổ" đều là động vật.

Những hạn chế này chính là động lực thúc đẩy sự ra đời của các mô hình dựa trên mạng nơ-ron và word embeddings, những chủ đề chúng ta sẽ khám phá trong phần tiếp theo của chương này và các chương sau.

## 2.3. Các thuật toán Học máy Thống kê

Sau khi đã biến đổi văn bản thành các vector số (sử dụng BoW hoặc TF-IDF), chúng ta đã sẵn sàng để "dạy" cho máy tính. Các thuật toán học máy thống kê là những công cụ cho phép chúng ta xây dựng các mô hình có khả năng học các mẫu (patterns) từ dữ liệu đã được gán nhãn, và sau đó sử dụng các mẫu đó để đưa ra dự đoán trên dữ liệu mới.

Mục này sẽ giới thiệu một số thuật toán kinh điển đã tạo nên xương sống của NLP trong kỷ nguyên thống kê. Chúng được chia thành hai nhóm chính: các mô hình cho bài toán phân loại và các mô hình cho bài toán học chuỗi.

### 2.3.1. Các mô hình Phân loại (Classification Models)

Đây là nhóm các thuật toán dùng để giải quyết các bài toán phân loại văn bản, ví dụ như phân tích cảm xúc (tích cực/tiêu cực), phân loại chủ đề tin tức, phát hiện thư rác.

#### Naive Bayes

Naive Bayes là một trong những thuật toán phân loại đơn giản, nhanh và đáng ngạc nhiên là rất hiệu quả cho các tác vụ văn bản [54].

**Trực giác cốt lõi** Thuật toán này dựa trên **Định lý Bayes**. Để phân loại một tài liệu  $d$  vào một lớp  $c$  (ví dụ: 'spam' hoặc 'not-spam'), Naive Bayes tính toán xác suất của mỗi lớp khi biết tài liệu đó,  $P(c|d)$ , và chọn lớp có xác suất cao nhất.

Sử dụng định lý Bayes, ta có:

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}$$

Trong đó:

- $P(c|d)$ : Xác suất hậu nghiệm (posterior) - xác suất của lớp  $c$  khi biết tài liệu  $d$ . Đây là thứ chúng ta muốn tìm.

- $P(d|c)$ : Xác suất khả năng (likelihood) - xác suất của tài liệu  $d$  khi biết nó thuộc lớp  $c$ .
- $P(c)$ : Xác suất tiên nghiệm (prior) - xác suất của lớp  $c$  trong toàn bộ dữ liệu.
- $P(d)$ : Xác suất của tài liệu  $d$ . Vì  $P(d)$  là hằng số đối với tất cả các lớp, chúng ta có thể bỏ qua nó khi so sánh các lớp với nhau.

Do đó, việc tối đa hóa  $P(c|d)$  tương đương với việc tối đa hóa  $P(d|c) \cdot P(c)$ .

**Giả định "Ngây thơ"(The "Naive" Assumption)** Để tính  $P(d|c) = P(w_1, w_2, \dots, w_n|c)$ , Naive Bayes đưa ra một giả định rất mạnh và "ngây thơ"(naive): **các từ (đặc trưng) trong tài liệu là độc lập có điều kiện với nhau khi biết lớp  $c$** .

$$P(d|c) = P(w_1, w_2, \dots, w_n|c) \approx \prod_{i=1}^n P(w_i|c)$$

Mặc dù giả định này rõ ràng là sai trong thực tế (trật tự từ và sự phụ thuộc giữa các từ là rất quan trọng), nó giúp việc tính toán trở nên cực kỳ đơn giản và hiệu quả.

**Công thức cuối cùng** Lớp  $\hat{c}$  được dự đoán cho tài liệu  $d$  là:

$$\hat{c} = \arg \max_{c \in C} \left( P(c) \cdot \prod_{i=1}^n P(w_i|c) \right) \quad (2.2)$$

Trong thực tế, do tích của nhiều xác suất nhỏ có thể dẫn đến sai số làm tròn, người ta thường tính toán trên logarit:

$$\hat{c} = \arg \max_{c \in C} \left( \log P(c) + \sum_{i=1}^n \log P(w_i|c) \right)$$

Các xác suất  $P(c)$  và  $P(w_i|c)$  được học từ tập dữ liệu huấn luyện bằng cách đếm tần suất và sử dụng kỹ thuật làm mịn (ví dụ: Laplace) để tránh xác suất zero.

### Dánh giá Naive Bayes

#### Ưu điểm:

- **Cực kỳ nhanh:** Cả quá trình huấn luyện và dự đoán đều chỉ là các phép đếm và nhân/cộng, rất hiệu quả.
- **Cần ít dữ liệu huấn luyện:** Hoạt động tương đối tốt ngay cả với dữ liệu nhỏ.
- **Mô hình diễn giải được:** Có thể kiểm tra xác suất  $P(w|c)$  để xem từ nào đóng góp nhiều nhất vào việc phân loại một lớp.

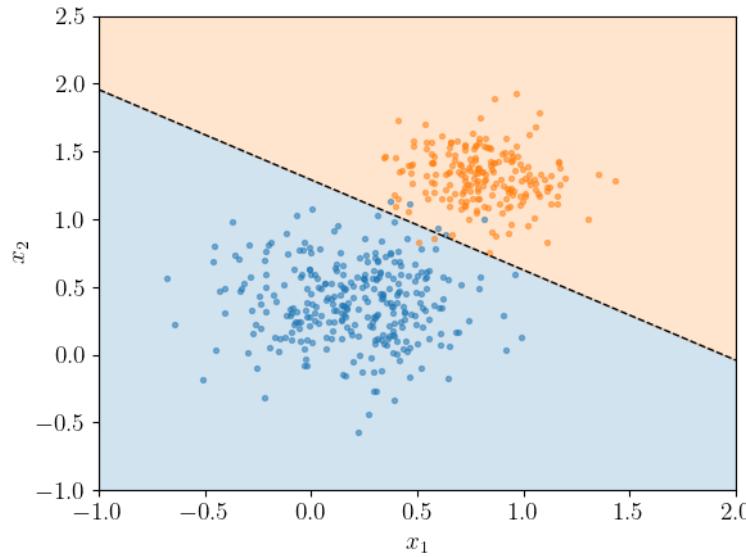
#### Nhược điểm:

- **Giả định độc lập "ngây thơ":** Không thể nắm bắt mối quan hệ và trật tự từ, làm hạn chế hiệu năng ở các bài toán phức tạp.

### Hồi quy Logistic (Logistic Regression)

Hồi quy Logistic [32], mặc dù tên gọi có chữ "hồi quy", lại là một thuật toán phân loại mạnh mẽ và được sử dụng cực kỳ rộng rãi. Nó thuộc nhóm các mô hình phân biệt (discriminative models), trái ngược với Naive Bayes là một mô hình sinh (generative model).

**Trực giác cốt lõi** Thay vì mô hình hóa  $P(d|c)$  như Naive Bayes, Hồi quy Logistic mô hình hóa trực tiếp xác suất hậu nghiệm  $P(c|d)$ . Nó học một ranh giới quyết định (decision boundary) tuyến tính để phân tách các lớp trong không gian đặc trưng.



Hình 2.1: Hồi quy Logistic học một đường thẳng (hoặc siêu phẳng trong không gian nhiều chiều) để phân chia dữ liệu thành các lớp.

**Cơ chế hoạt động** Cho một vector đầu vào  $x$  (ví dụ, vector TF-IDF của tài liệu), mô hình thực hiện 2 bước:

1. **Tính toán điểm số tuyến tính (Linear Score):** Mô hình tính một tổng có trọng số của các đặc trưng đầu vào. Mỗi đặc trưng  $x_j$  (tương ứng với một từ trong từ vựng) có một trọng số  $w_j$ .

$$z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m = w^T x$$

Trọng số  $w_j$  có thể được diễn giải là: một giá trị dương lớn cho thấy sự hiện diện của từ  $j$  làm tăng khả năng tài liệu thuộc lớp tích cực, và ngược lại.

2. **Ánh xạ qua hàm Sigmoid:** Điểm số  $z$  có thể nhận bất kỳ giá trị thực nào. Để biến nó thành một xác suất (nằm trong khoảng  $[0, 1]$ ), chúng ta đưa nó qua hàm Sigmoid (còn gọi là hàm logistic).

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Giá trị đầu ra  $\sigma(z)$  chính là xác suất dự đoán  $P(c = 1|d)$ .

**Huấn luyện bằng Gradient Descent** Quá trình huấn luyện của Hồi quy Logistic là tìm ra bộ trọng số  $w$  tốt nhất để tối thiểu hóa hàm mất mát **Cross-Entropy** trên toàn bộ tập dữ liệu huấn luyện. Hàm mất mát cho một điểm dữ liệu  $(x, y)$  (với  $y \in \{0, 1\}$  là nhãn thật) là:

$$\mathcal{L}(w) = -[y \log(\sigma(w^T x)) + (1 - y) \log(1 - \sigma(w^T x))]$$

Gradient của hàm mất mát này theo trọng số  $w$  được tính toán, và sau đó thuật toán **Gradient Descent** (hoặc các biến thể của nó như SGD, Adam) được sử dụng để cập nhật các trọng số  $w$  một cách lặp đi lặp lại cho đến khi hội tụ.

### Danh giá Hồi quy Logistic

#### Ưu điểm:

- **Hiệu quả và mạnh mẽ:** Thường cho kết quả tốt hơn Naive Bayes vì nó không có giả định độc lập ngẫu thơ.
- **Đầu ra là xác suất:** Cung cấp một thước đo độ tin cậy cho dự đoán.
- **Mô hình diễn giải được:** Giống Naive Bayes, có thể kiểm tra các trọng số  $w$  để hiểu tầm quan trọng của các từ.

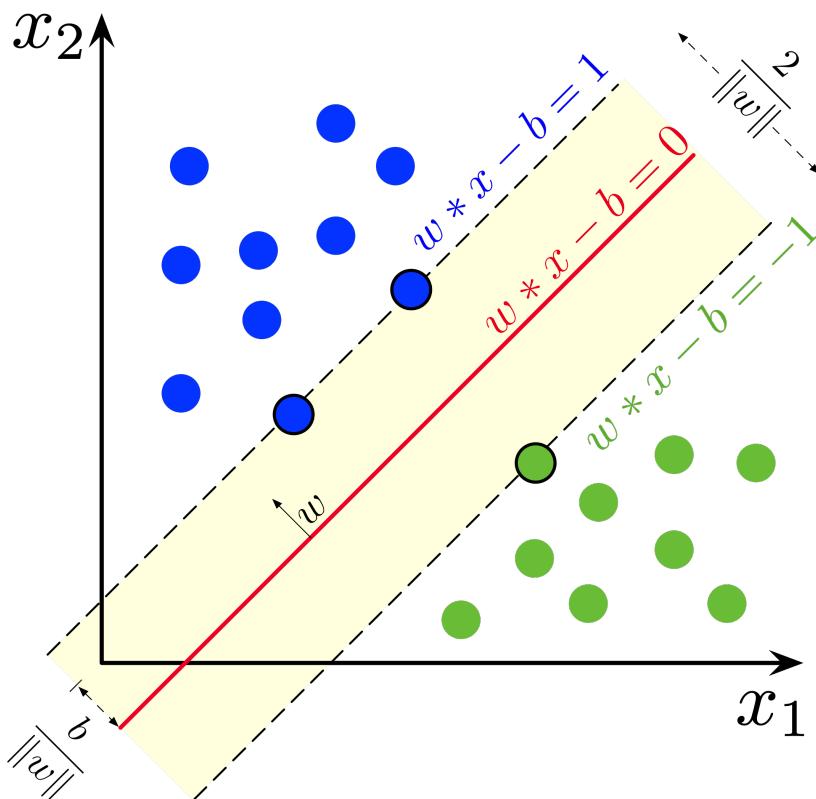
#### Nhược điểm:

- **Bản chất tuyến tính:** Không thể học được các ranh giới quyết định phi tuyến phức tạp. Tuy nhiên, điều này có thể được khắc phục phần nào bằng cách tạo ra các đặc trưng tương tác.

### Máy Vector Hỗ trợ (Support Vector Machines - SVMs)

SVM [16] là một thuật toán phân loại mạnh mẽ khác, nhưng thay vì tập trung vào xác suất, nó tập trung vào hình học.

**Trục giác cốt lõi** Mục tiêu của SVM là tìm ra một **siêu phẳng (hyperplane)** phân tách các lớp dữ liệu sao cho lề (margin) – khoảng cách từ siêu phẳng đến các điểm dữ liệu gần nhất của mỗi lớp – là lớn nhất có thể (**maximum margin**).



Hình 2.2: SVM tìm siêu phẳng (đường nét đứt) có lề (margin) rộng nhất giữa các lớp. Các điểm nằm trên lề được gọi là các vector hỗ trợ (support vectors).

Việc tối đa hóa lề này giúp mô hình có khả năng tổng quát hóa tốt hơn trên dữ liệu mới, vì ranh giới quyết định ít bị ảnh hưởng bởi nhiễu.

**The Kernel Trick** Điểm mạnh thực sự của SVM nằm ở "thủ thuật kernel"(kernel trick). Nếu dữ liệu không thể phân tách bằng một đường thẳng, SVM có thể sử dụng một hàm kernel (ví dụ: RBF kernel) để ánh xạ dữ liệu lên một không gian nhiều chiều hơn, nơi chúng có thể trở nên phân tách tuyến tính. Điều này cho phép SVM học được các ranh giới quyết định phi tuyến rất phức tạp mà không cần tính toán trực tiếp trong không gian nhiều chiều đó, rất hiệu quả.

**Huấn luyện và Hinge Loss** Quá trình huấn luyện SVM là một bài toán tối ưu hóa nhằm tìm ra siêu phẳng có lề lớn nhất. Điều này được thực hiện bằng cách tối thiểu hóa một hàm mất mát đặc biệt gọi là **Hinge Loss**:

$$\mathcal{L}(w) = \max(0, 1 - y_i(w^T x_i - b))$$

Hàm mất mát này sẽ bằng 0 nếu một điểm dữ liệu nằm đúng phía của lề, và sẽ tăng tuyến tính nếu nó vi phạm lề. Việc tối ưu hóa này thường được giải quyết bằng các phương pháp phức tạp hơn như Quy hoạch toàn phương (Quadratic Programming).

### Dánh giá SVM

#### Ưu điểm:

- **Hiệu quả cao trong không gian nhiều chiều:** Rất phù hợp với các vector TF-IDF có số chiều lớn.
- **Học được các ranh giới phi tuyến phức tạp:** Nhờ vào kernel trick.
- **Ít bị quá khớp (overfitting):** Do nguyên lý tối đa hóa lề.

#### Nhược điểm:

- **Chi phí tính toán cao:** Huấn luyện SVM có thể rất chậm trên các bộ dữ liệu lớn.
- **Mô hình hộp đen (Black box):** Việc diễn giải một mô hình SVM với kernel phi tuyến là rất khó.
- **Nhạy cảm với việc lựa chọn kernel và tham số.**

### 2.3.2. Các mô hình Học chuỗi (Sequence Labeling Models)

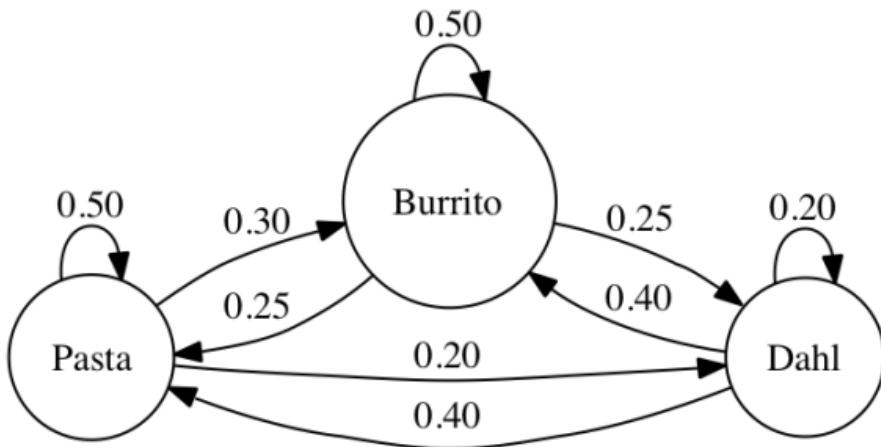
Các bài toán như Gán nhãn Từ loại (POS Tagging) hay Nhận dạng Thực thể Tên (NER) không chỉ đơn thuần là phân loại từng từ một cách độc lập. Nhãn của một từ phụ thuộc rất nhiều vào nhãn của các từ xung quanh. Các mô hình học chuỗi được thiết kế để giải quyết vấn đề này.

#### Mô hình Markov ẩn (Hidden Markov Models - HMMs)

HMM [68] là một mô hình sinh kinh điển cho các bài toán gán nhãn chuỗi.

**Trực giác cốt lõi** HMM giả định rằng một chuỗi các quan sát (observations) - tức là các từ trong câu - được "sinh ra" bởi một chuỗi các trạng thái ẩn (hidden states) - tức là các nhãn (ví dụ, các nhãn POS). Quá trình sinh này tuân theo hai loại xác suất:

1. **Xác suất chuyển trạng thái (Transition Probabilities):** Xác suất chuyển từ một trạng thái ẩn này sang một trạng thái ẩn khác. Ví dụ,  $P(\text{Verb}|\text{Noun})$  - xác suất một động từ xuất hiện sau một danh từ. Điều này tuân theo giả định Markov.
2. **Xác suất phát sinh (Emission Probabilities):** Xác suất một trạng thái ẩn "sinh ra" một từ quan sát được. Ví dụ,  $P(\text{"sách"}|\text{Noun})$  - xác suất từ "sách" được sinh ra khi trạng thái ẩn là Danh từ.



Hình 2.3: Sơ đồ một chuỗi Markov (Markov chain) với ba trạng thái: Pasta, Burrito, và Dahl. Các con số biểu thị xác suất chuyển đổi giữa các trạng thái.

Nhiệm vụ của HMM là tìm ra chuỗi trạng thái ăn (chuỗi nhãn) có khả năng nhất đã sinh ra chuỗi các từ quan sát được. Bài toán này được giải quyết hiệu quả bằng một thuật toán quy hoạch động gọi là **Thuật toán Viterbi (Viterbi Algorithm)**.

### Trường Ngẫu nhiên Có điều kiện (Conditional Random Fields - CRFs)

CRF, và đặc biệt là Linear-chain CRF, là một cải tiến vượt bậc so với HMM và đã trở thành mô hình thống kê tiêu chuẩn cho các bài toán gán nhãn chuỗi trong nhiều năm.

**Sự khác biệt chính với HMM** HMM là một mô hình sinh, nó mô hình hóa xác suất kết hợp  $P(\text{chuỗi quan sát}, \text{chuỗi nhãn})$ . CRF là một mô hình phân biệt, nó mô hình hóa trực tiếp xác suất có điều kiện  $P(\text{chuỗi nhãn} | \text{chuỗi quan sát})$ . Sự thay đổi này mang lại hai lợi thế lớn:

1. **Bỏ giả định độc lập của đặc trưng:** HMM giả định rằng một từ quan sát chỉ phụ thuộc vào trạng thái ẩn hiện tại. CRF không có giả định này. Nó có thể sử dụng một loạt các đặc trưng phức tạp của chuỗi đầu vào, ví dụ: "từ hiện tại có viết hoa không?", "hậu tố của từ hiện tại là gì?", "từ trước đó là gì?". Điều này làm cho CRF linh hoạt và mạnh mẽ hơn nhiều.
2. **Tránh tính toán  $P(\text{chuỗi quan sát})$ :** Việc mô hình hóa xác suất của chuỗi quan sát là rất khó, CRF đã loại bỏ được gánh nặng này.

**Trực giác cốt lõi của CRF** Thay vì tính toán các xác suất chuyển và phát sinh riêng lẻ, một CRF gán một điểm số (score) cho mỗi cặp (chuỗi quan sát, chuỗi nhãn) dựa trên một tổ hợp tuyến tính của các hàm đặc trưng (feature functions). Các hàm đặc trưng này có thể bắt các mối quan hệ phức tạp giữa các từ và nhãn. Sau đó, nó sử dụng một hàm chuẩn hóa (tương tự softmax) trên toàn bộ các chuỗi nhãn có thể có để biến điểm số thành xác suất. Nhiệm vụ cũng là tìm ra chuỗi nhãn có xác suất (điểm số) cao nhất, và cũng thường được giải bằng thuật toán Viterbi.

### So sánh HMM và CRF

Hidden Markov Models (HMM)	Conditional Random Fields (CRF)
Mô hình sinh (Generative)	Mô hình phân biệt (Discriminative)
Mô hình hóa $P(\text{observations}, \text{labels})$	Mô hình hóa $P(\text{labels}   \text{observations})$
Giả định các quan sát độc lập có điều kiện	Không có giả định độc lập, cho phép các đặc trưng chồng chéo, phức tạp.
Huấn luyện nhanh hơn	Huấn luyện chậm hơn (cần các thuật toán lặp)
Thường cho hiệu năng thấp hơn	Thường cho hiệu năng vượt trội, là state-of-the-art trong kỹ nguyên thống kê.

## 2.4. Mô hình hóa Chủ đề (Topic Modeling - LDA)

Cho đến nay, các mô hình chúng ta đã thảo luận chủ yếu thuộc hai loại: học có giám sát (supervised learning) để phân loại hoặc gán nhãn (Naive Bayes, SVM, CRF), hoặc học biểu diễn cho các đơn vị nhỏ như từ (Word2Vec, GloVe). Trong mục cuối cùng của chương này, chúng ta sẽ khám phá một hướng tiếp cận khác hẳn: **học phi giám sát (unsupervised learning)** để tự động khám phá các **chủ đề (topics)** tiềm ẩn trong một bộ sưu tập lớn các tài liệu.

Công cụ mạnh mẽ và phổ biến nhất cho nhiệm vụ này là **Phân bổ Dirichlet Tiềm ẩn (Latent Dirichlet Allocation - LDA)**[6].

### 2.4.1. Tự duy cốt lõi và bài toán

Hãy tưởng tượng bạn được giao một kho lưu trữ gồm 1 triệu bài báo và được yêu cầu cho biết "Các bài báo này đang nói về những chủ đề chính nào?". Việc đọc thủ công là bất khả thi. Topic Modeling, và cụ thể là LDA, được sinh ra để giải quyết chính xác bài toán này.

LDA được xây dựng dựa trên một loạt các giả định trực quan về cách các tài liệu được "sinh ra":

1. Mỗi tài liệu là một **hỗn hợp của nhiều chủ đề** với các tỷ lệ khác nhau. Ví dụ, một bài báo về việc Apple ra mắt iPhone mới có thể là 70% chủ đề "Công nghệ", 20% chủ đề "Kinh doanh" và 10% chủ đề "Thiết kế".
2. Mỗi chủ đề là một **phân phối xác suất trên các từ**. Ví dụ, chủ đề "Công nghệ" sẽ có xác suất cao cho các từ như "máy tính", "phần mềm", "dữ liệu", "AI", và xác suất rất thấp cho các từ như "chính trị", "bầu cử".

Nhiệm vụ của LDA là, khi chỉ được cho xem các tài liệu (tức là các từ), nó phải **suy ngược (infer)** ra các cấu trúc ẩn đã sinh ra chúng: (1) Hỗn hợp chủ đề cho mỗi tài liệu là gì? và (2) Phân phối từ cho mỗi chủ đề là gì?

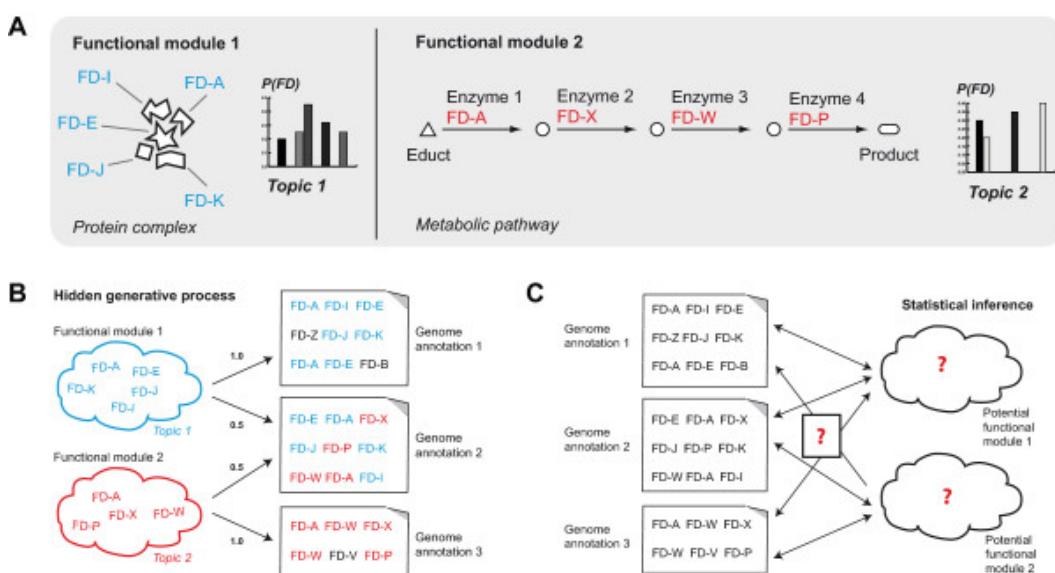
### 2.4.2. Câu chuyện sinh dữ liệu của LDA (The Generative Story)

Cách dễ nhất để hiểu LDA là thông qua "câu chuyện" về cách một tài liệu được tạo ra theo quy tắc của nó. Đây là một thí nghiệm tưởng tượng:

### Câu chuyện sinh dữ liệu

Giả sử bạn muốn viết một tài liệu mới có  $N$  từ, và đã có sẵn một bộ  $K$  chủ đề (mỗi chủ đề là một túi từ có xác suất).

1. **Chọn hỗn hợp chủ đề cho tài liệu:** Đầu tiên, bạn quyết định tỷ lệ các chủ đề cho tài liệu của mình. Bạn lấy ngẫu nhiên một vector phân phối chủ đề  $\theta_d$  từ một phân phối Dirichlet. Vector này có thể trông như: '[Topic1: 0.7, Topic2: 0.2, Topic3: 0.1, ...]'.
2. **Viết từng từ một:** Để viết từ thứ  $i$  trong tài liệu (với  $i$  từ 1 đến  $N$ ):
  - (a) **Chọn một chủ đề:** Dựa trên phân phối chủ đề  $\theta_d$  của tài liệu, bạn "tung xúc xác" để chọn ra một chủ đề  $z_i$ . Ví dụ, với xác suất 70% bạn sẽ chọn Topic1.
  - (b) **Chọn một từ từ chủ đề đó:** Bây giờ bạn đã có chủ đề  $z_i$ . Mỗi chủ đề lại có một phân phối từ riêng của nó (ký hiệu là  $\phi_{z_i}$ ). Bạn lại "tung xúc xác" một lần nữa, dựa trên phân phối từ này, để chọn ra một từ  $w_i$ .
3. Lặp lại bước 2 cho đến khi bạn viết đủ  $N$  từ.



Hình 2.4: Mô hình sinh của LDA. Để tạo ra từ  $w_{d,n}$  trong tài liệu  $d$ , mô hình trước hết chọn một chủ đề  $z_{d,n}$  từ phân phối chủ đề  $\theta_d$  của tài liệu, sau đó chọn từ  $w_{d,n}$  từ phân phối từ  $\phi_{z_{d,n}}$  của chủ đề đó.

Tất nhiên, trong thực tế chúng ta không làm vậy. Chúng ta có sẵn các từ ( $w$ ) và nhiệm vụ của thuật toán LDA là tìm ra các phân phối ẩn  $\theta$  (phân phối chủ đề cho mỗi tài liệu) và  $\phi$  (phân phối từ cho mỗi chủ đề). Đây là một bài toán suy luận Bayes (Bayesian inference).

#### 2.4.3. Huấn luyện và Suy luận trong LDA

Việc tính toán chính xác các phân phối ẩn trong LDA là bất khả thi về mặt toán học. Do đó, người ta sử dụng các thuật toán xấp xỉ, trong đó phổ biến nhất là **Lấy mẫu Gibbs (Gibbs Sampling)**.

Trực giác của Lấy mẫu Gibbs như sau:

1. **Khởi tạo:** Đi qua tất cả các từ trong tất cả các tài liệu, gán ngẫu nhiên một trong  $K$  chủ đề cho mỗi từ.
2. **Lặp lại nhiều lần:** Với mỗi từ  $w$  trong một tài liệu  $d$ :
  - Tạm thời "bỏ gán nhãn" chủ đề cho từ  $w$ .
  - Tính toán lại xác suất từ  $w$  thuộc về mỗi chủ đề  $k$ , dựa trên hai yếu tố:
    - (a) Phân phối các chủ đề của các từ khác trong tài liệu  $d$  (Tài liệu này đang nói nhiều về chủ đề nào?).

- (b) Phân phối của từ  $w$  trên tất cả các tài liệu khác (Từ  $w$  này thường xuất hiện trong chủ đề nào?).
- Gán lại một chủ đề mới cho từ  $w$  dựa trên xác suất vừa tính được.

Sau khi lặp lại quá trình này rất nhiều lần, việc gán nhãn chủ đề cho các từ sẽ hội tụ về một trạng thái ổn định, phản ánh cấu trúc chủ đề thực sự của kho văn bản. Từ các phân công chủ đề cuối cùng này, chúng ta có thể dễ dàng ước tính  $\theta$  và  $\phi$ .

#### 2.4.4. Kết quả và Ứng dụng

Sau khi huấn luyện một mô hình LDA, chúng ta thu được hai kết quả vô cùng hữu ích:

**1. Các chủ đề của kho văn bản** Mỗi chủ đề được biểu diễn bằng một danh sách các từ có xác suất cao nhất thuộc về nó. Điều này cho phép con người diễn giải ý nghĩa của chủ đề.

##### Ví dụ 8: Đầu ra chủ đề của LDA

Huấn luyện LDA trên một kho tin tức có thể cho ra các chủ đề như:

- **Chủ đề 1 (Thể thao):** ‘bóng đá, đội tuyển, trận đấu, cầu thủ, vô địch, bàn thắng, ...’
- **Chủ đề 2 (Kinh tế):** ‘cổ phiếu, thị trường, công ty, kinh doanh, tăng trưởng, lạm phát, ...’
- **Chủ đề 3 (Chính trị):** ‘chính phủ, luật, bầu cử, quốc hội, đảng, quyết định, ...’

**2. Biểu diễn tài liệu bằng chủ đề** Mỗi tài liệu giờ đây có thể được biểu diễn bằng một vector dày đặc, có số chiều bằng số chủ đề  $K$ . Mỗi chiều của vector là tỷ lệ của chủ đề đó trong tài liệu.

##### Ví dụ 9: Biểu diễn tài liệu bằng LDA

Một bài báo có nội dung ”Đội tuyển bóng đá quốc gia giành chức vô địch sau một trận đấu kịch tính” có thể được biểu diễn bằng vector: [Thể thao: 0.9, Kinh tế: 0.05, Chính trị: 0.05]

Ứng dụng thực tế của LDA:

- **Khám phá dữ liệu:** Nhanh chóng nắm bắt các chủ đề chính trong một kho văn bản lớn.
- **Hệ thống gợi ý (Recommender Systems):** Gợi ý các bài báo tương tự cho người dùng dựa trên sự tương đồng về phân phối chủ đề.
- **Kỹ thuật đặc trưng (Feature Engineering):** Sử dụng các vector chủ đề làm đặc trưng đầu vào cho các mô hình phân loại có giám sát. Thay vì dùng một vector BoW hàng chục ngàn chiều, chúng ta có thể dùng một vector chủ đề chỉ vài trăm chiều, vừa giảm chiều dữ liệu vừa mang tính ngữ nghĩa cao.

### 2.4.5. Ưu và Nhược điểm

#### Đánh giá mô hình LDA

##### Ưu điểm:

- **Học phi giám sát:** Không cần dữ liệu gán nhãn, cực kỳ hữu ích cho việc khám phá.
- **Cung cấp kết quả diễn giải được:** Con người có thể đọc và hiểu ý nghĩa của các chủ đề.
- **Là một kỹ thuật giảm chiều dữ liệu mạnh mẽ.**

##### Nhược điểm:

- **Phải chọn trước số chủ đề ( $K$ ):** Việc chọn giá trị  $K$  tối ưu là một thách thức và thường đòi hỏi thử nghiệm.
- **Giả định Bag-of-Words:** LDA bỏ qua trật tự từ, giống như BoW.
- **Các chủ đề có thể không mạch lạc:** Đôi khi mô hình tạo ra các chủ đề là một mớ hỗn độn các từ có tần suất cao nhưng không có ý nghĩa rõ ràng.

## KẾT THÚC CHƯƠNG 2

Chương 2 đã đưa chúng ta vào một hành trình qua kỷ nguyên thống kê, từ việc biểu diễn văn bản như những "túi từ" đơn giản đến các vector ngữ nghĩa phức tạp. Cuộc cách mạng Word Embeddings đã cho chúng ta thấy rằng ý nghĩa của từ có thể được nắm bắt trong các không gian vector dày đặc, một ý tưởng nền tảng cho toàn bộ NLP hiện đại. Tuy nhiên, các embedding tĩnh này vẫn chưa nắm bắt được sự thay đổi của ý nghĩa theo ngữ cảnh. Đây chính là động lực để chúng ta bước sang chương tiếp theo, khám phá các kiến trúc mạng nơ-ron được thiết kế để "đọc" và "ghi nhớ" các chuỗi văn bản, mở ra kỷ nguyên của sự hiểu biết ngữ cảnh sâu sắc.

## CHƯƠNG 2. BIỂU ĐIỂN VĂN BẢN VÀ CÁC MÔ HÌNH THỐNG KÊ

## Chương 3

# CÁC KIẾN TRÚC MẠNG NƠ-RON KINH ĐIỂN

Chào mừng bạn đến với kỹ nguyên học sâu của NLP. Trong chương 2, chúng ta đã thấy các mô hình thống kê như N-gram có thể nắm bắt các phụ thuộc cục bộ, nhưng lại bất lực trước các mối quan hệ tầm xa. Cuộc cách mạng word embedding đã cho chúng ta các vector từ mang đầy ngữ nghĩa, nhưng làm thế nào để kết hợp chúng lại nhằm hiểu được ý nghĩa của cả một câu hay một đoạn văn?

Chương này sẽ giới thiệu các kiến trúc mạng nơ-ron nền tảng được thiết kế đặc biệt để xử lý dữ liệu dạng chuỗi như ngôn ngữ tự nhiên. Những kiến trúc này không chỉ đơn thuần "nhìn" vào các từ một cách riêng lẻ, mà còn có khả năng "ghi nhớ" và "lý luận" dựa trên thông tin đã xử lý trước đó, cho phép chúng nắm bắt được ngữ cảnh và các phụ thuộc phức tạp trong câu.

### 3.1. Word Embeddings: Cuộc cách mạng Biểu diễn Đầu tiên

Các phương pháp biểu diễn dựa trên tần suất như BoW hay TF-IDF, mặc dù hữu ích, lại mắc phải hai nhược điểm chí mạng:

1. **Độ thừa thớt (Sparsity):** Chúng tạo ra các vector có số chiều khổng lồ và hầu hết là số 0.
2. **Thiếu vắng ngữ nghĩa (Lack of Semantics):** Các vector này không hề nắm bắt được mối quan hệ ngữ nghĩa giữa các từ. Đối với BoW, hai từ "tốt" và "tuyệt vời" là hoàn toàn khác biệt, không có mối liên hệ nào.

**Word Embeddings** (Nhúng từ) ra đời để giải quyết triệt để hai vấn đề trên. Đây là một tập hợp các kỹ thuật học máy có mục tiêu biểu diễn mỗi từ trong từ vựng bằng một **vector dày đặc (dense vector)** có số chiều tương đối thấp (thường từ 50 đến 300 chiều). Quan trọng hơn, các vector này được học sao cho chúng có thể nắm bắt được các mối quan hệ ngữ nghĩa và cú pháp giữa các từ.

Ý tưởng nền tảng của Word Embeddings dựa trên **Giả thuyết Phân bố (Distributional Hypothesis)** của Firth (1957), được tóm gọn trong câu nói nổi tiếng:

#### Giả thuyết Phân bố

"You shall know a word by the company it keeps." (Bạn có thể hiểu một từ qua những từ thường đi cùng với nó.)

Nói cách khác, những từ xuất hiện trong các ngữ cảnh tương tự nhau thì có khả năng mang ý nghĩa tương tự nhau. Word Embeddings là hiện thực hóa của giả thuyết này bằng các mô hình mạng nơ-ron.

#### 3.1.1. Lý thuyết Word2Vec: CBOW & Skip-gram

Được giới thiệu bởi Tomas Mikolov và các cộng sự tại Google vào năm 2013 [56, 57], Word2Vec không phải là một thuật toán duy nhất mà là một bộ công cụ bao gồm hai kiến trúc mô hình chính: **CBOW (Continuous Bag-of-Words)** và **Skip-gram**. Word2Vec đã tạo ra một cuộc cách mạng trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên (NLP), dân chủ hóa việc sử dụng *word embeddings* nhờ vào hiệu quả tính toán và chất lượng của vector biểu diễn mà nó tạo ra.

### Kiến trúc chung: Mạng Nô-ron Nông

Cốt lõi của Word2Vec là một ý tưởng đơn giản đến bất ngờ: thay vì xây dựng một mô hình ngôn ngữ phức tạp, chúng ta thiết kế một nhiệm vụ "giả"(fake task) và trong quá trình huấn luyện mô hình để giải quyết nhiệm vụ đó, chúng ta sẽ học được các vector từ chất lượng cao.

**Nhiệm vụ "giả" so với Mô hình Ngôn ngữ "thật"** Gọi là "giả" vì mục tiêu cuối cùng của chúng ta không phải là giải quyết tốt nhiệm vụ dự đoán từ, mà là học được ma trận trọng số  $W$ . Một mô hình ngôn ngữ "thật"(true language model), chẳng hạn như các mô hình dựa trên RNN, sẽ có mục tiêu phức tạp hơn nhiều: dự đoán từ tiếp theo dựa trên toàn bộ lịch sử các từ trước đó ( $P(w_t|w_1, \dots, w_{t-1})$ ). Việc này đòi hỏi kiến trúc sâu và phức tạp hơn.

Ngược lại, Word2Vec đơn giản hóa triệt để vấn đề: nó chỉ dự đoán từ dựa trên một cửa sổ ngữ cảnh cục bộ nhỏ. Sự đơn giản hóa này chính là chìa khóa thành công: nó cho phép mô hình xử lý hiệu quả một lượng dữ liệu văn bản khổng lồ. Gradient từ nhiệm vụ đơn giản này, tuy không hoàn hảo cho việc dự đoán ngôn ngữ, lại cực kỳ hiệu quả trong việc định hình một khung gian vector mà ở đó các từ có ngữ cảnh tương tự được đặt gần nhau.

Cả CBOW và Skip-gram đều sử dụng kiến trúc mạng nô-ron "nông"(shallow) chỉ với một lớp ẩn duy nhất. Giả sử từ vựng của chúng ta có  $V$  từ độc nhất và chúng ta muốn học các vector biểu diễn có  $N$  chiều (embedding dimension).

- **Lớp vào (Input Layer):** Đầu vào là một vector *one-hot* có kích thước  $1 \times V$ , đại diện cho một từ.
- **Lớp ẩn (Hidden Layer):** Không có hàm kích hoạt. Điểm mấu chốt nằm ở ma trận trọng số  $W$  kết nối lớp vào và lớp ẩn. Ma trận này có kích thước  $V \times N$ .
- **Lớp ra (Output Layer):** Ma trận trọng số  $W'$  kết nối lớp ẩn và lớp ra, có kích thước  $N \times V$ .

#### Ghi chú về Thiết kế Kiến trúc

##### 1. Tại sao không có hàm kích hoạt ở lớp ẩn?

Lớp ẩn của Word2Vec thực chất là một lớp tuyến tính (linear projection). Việc loại bỏ hàm phi tuyến (như ReLU hay Tanh) là một lựa chọn có chủ ý. Mục đích là để các mối quan hệ ngữ nghĩa (ví dụ: các phép loại suy – analogies) có thể được biểu diễn dưới dạng các phép toán vector tuyến tính đơn giản (ví dụ:  $\vec{v}_{king} - \vec{v}_{man} + \vec{v}_{woman} \approx \vec{v}_{queen}$ ). Nếu thêm hàm phi tuyến, cấu trúc tuyến tính này sẽ bị phá vỡ, làm cho các vector học được khó diễn giải và sử dụng hơn theo cách này. Lớp ẩn chỉ đóng vai trò như một bảng tra cứu (lookup table) để lấy ra vector embedding.

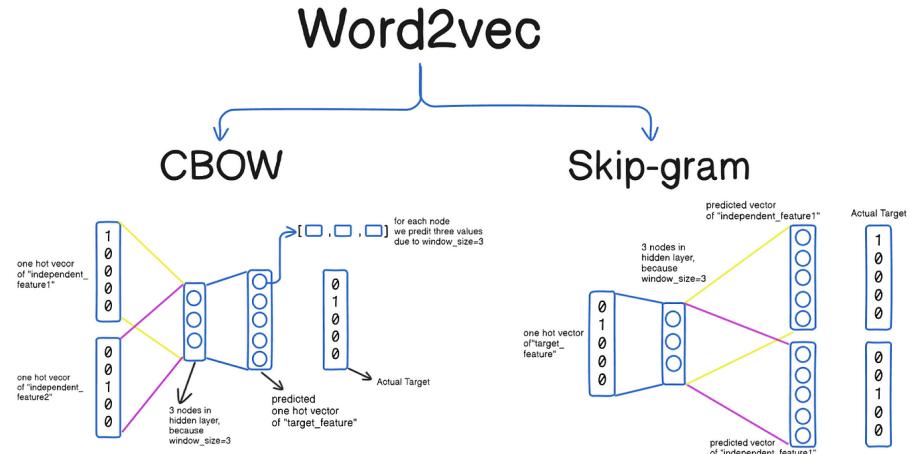
##### 2. Tại sao có hai ma trận trọng số $W$ và $W'$ ?

Mô hình có hai bộ vector cho mỗi từ:

- **Input vectors (trong  $W$ ):** Khi từ được dùng làm đầu vào (từ trung tâm trong Skip-gram, từ ngữ cảnh trong CBOW).
- **Output vectors (trong  $W'$ ):** Khi từ được dùng làm mục tiêu dự đoán (từ ngữ cảnh trong Skip-gram, từ trung tâm trong CBOW).

Về mặt lý thuyết, việc có hai vai trò riêng biệt cho phép mô hình linh hoạt hơn. Sau khi huấn luyện xong, chúng ta thường chỉ sử dụng ma trận đầu vào  $W$  làm embedding cuối cùng. Đây là một quy ước phổ biến và cho kết quả tốt trong thực tế. Một số nghiên cứu cũng đề xuất các cách kết hợp khác (ví dụ, lấy trung bình cộng của hai vector tương ứng từ  $W$  và  $W'$ ), nhưng chỉ dùng  $W$  vẫn là phương pháp tiêu chuẩn.

Điều quan trọng là, chúng ta không thực sự quan tâm đến kết quả dự đoán của mạng. Mục tiêu cuối cùng của chúng ta là học được ma trận trọng số  $W$ . Sau khi huấn luyện, mỗi hàng của ma trận  $W$  chính là vector embedding  $N$ -chiều cho một từ tương ứng trong từ vựng. Do đó,  $W$  còn được gọi là **ma trận embedding (embedding matrix)**.



Hình 3.1: Kiến trúc chung của Word2Vec. Khi nhân vector one-hot của một từ với ma trận  $W$ , thực chất chúng ta đang "chọn" ra hàng tương ứng với từ đó, chính là vector embedding của nó. Ma trận  $W$  là mục tiêu chúng ta cần học.

### Mô hình CBOW (Continuous Bag-of-Words)

**Tư duy cốt lõi** Mô hình CBOW hoạt động dựa trên giả định: "Một từ có thể được dự đoán bởi các từ xung quanh nó". Do đó, CBOW cố gắng dự đoán một từ trung tâm (center word) dựa vào các từ ngữ cảnh (context words).

Ví dụ, trong câu "chú mèo [ngồi] trên chiếc chiếu" và cửa sổ ngữ cảnh là 2, CBOW sẽ lấy các từ ngữ cảnh {"chú", "mèo", "trên", "chiếc"} để dự đoán từ trung tâm là {"ngồi"}.

#### Cơ chế hoạt động

- Đầu vào:** Lấy các vector one-hot của các từ ngữ cảnh (ví dụ: **chú**, **mèo**, **trên**, **chiếc**).
- Lấy vector embedding:** Từ ma trận embedding  $W$ , ta lấy ra các vector embedding tương ứng cho từng từ ngữ cảnh.
- Tổng hợp ngữ cảnh:** Các vector embedding này được lấy **trung bình cộng** để tạo thành một vector ngữ cảnh duy nhất  $v_{\text{context}}$ . Việc lấy trung bình này giải thích cho tên gọi "Bag-of-Words", vì nó làm mất đi thông tin về thứ tự các từ trong ngữ cảnh.
- Dự đoán:** Vector  $v_{\text{context}}$  được nhân với ma trận  $W'$  và đưa qua hàm softmax để tạo ra một phân phối xác suất trên toàn bộ từ vựng. Vector đầu ra ý có kích thước  $1 \times V$ , trong đó phần tử thứ  $j$  biểu diễn xác suất từ trung tâm là từ thứ  $j$  trong từ vựng.
- Huấn luyện:** So sánh vector xác suất dự đoán ý với vector one-hot y của từ trung tâm thực tế (từ **ngồi**). Hàm mất mát thường được sử dụng là **Cross-Entropy**, và lỗi được lan truyền ngược (backpropagation) để cập nhật các trọng số trong cả hai ma trận  $W$  và  $W'$ .

$$\mathcal{L} = - \sum_{j=1}^V y_j \log(\hat{y}_j)$$

**Phân tích lựa chọn thiết kế của CBOW** Việc lấy trung bình các vector ngữ cảnh là một lựa chọn thiết kế vì sự đơn giản và hiệu quả tính toán. Nó không cần tham số nào để học và có thể được tính rất nhanh. Tuy nhiên, lựa chọn này có hai hệ quả quan trọng:

- Mất thông tin thứ tự:** CBOW không phân biệt được hai câu có cùng các từ ngữ cảnh nhưng khác trật tự. Trong nhiều tác vụ, việc mất thông tin cú pháp này là một hạn chế đáng kể.
- "Lu mờ" từ hiếm:** Khi lấy trung bình, đóng góp của một vector từ hiếm có thể bị "lấn át" bởi các vector của các từ ngữ cảnh phổ biến hơn. Đây là lý do CBOW thường hoạt động kém hơn Skip-gram đối với các từ hiếm.

Các phương pháp phức tạp hơn như lấy trung bình có trọng số (dựa trên khoảng cách tới từ trung tâm) hoặc thậm chí dùng một cơ chế attention đơn giản có thể được xem xét, nhưng chúng sẽ làm tăng độ phức tạp và đi ngược lại triết lý "đơn giản, nhanh" của Word2Vec.

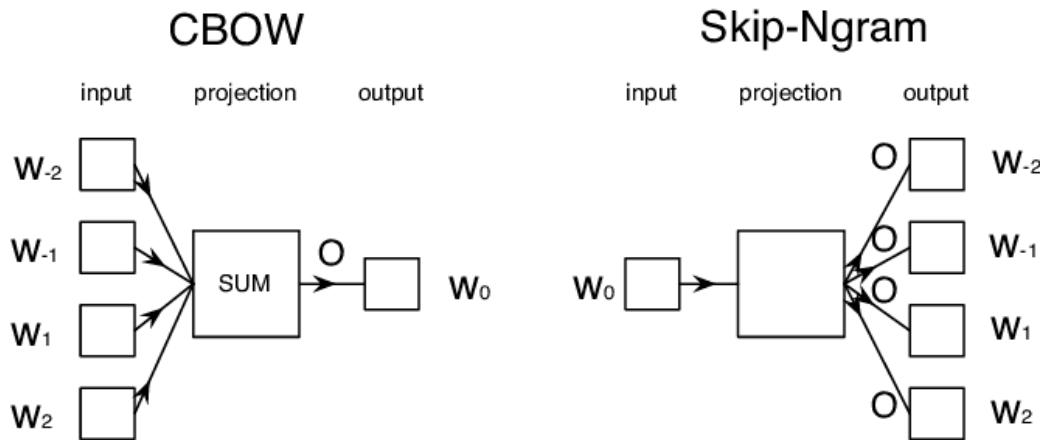
### Mô hình Skip-gram

**Tư duy cốt lõi** Skip-gram đảo ngược lại nhiệm vụ của CBOW. Nó hoạt động dựa trên giả định: "Một từ có thể cho biết những từ nào sẽ xuất hiện xung quanh nó". Do đó, Skip-gram cố gắng dự đoán các từ ngữ cảnh dựa vào một từ trung tâm.

Ví dụ, với cùng câu trên, Skip-gram sẽ lấy từ trung tâm {"ngồi"} để cố gắng dự đoán từng từ trong tập ngữ cảnh {"chú", "mèo", "trên", "chiếc"}.

**Cơ chế hoạt động** Khác với CBOW, Skip-gram không gộp ngữ cảnh lại. Thay vào đó, nó tạo ra nhiều mẫu huấn luyện hơn từ một cửa sổ ngữ cảnh.

1. **Đầu vào:** Lấy vector one-hot của từ trung tâm (ví dụ: **ngồi**).
2. **Lấy vector embedding:** Lấy ra vector embedding  $v_{center}$  của từ trung tâm từ ma trận  $W$ .
3. **Dự đoán:** Vector  $v_{center}$  được đưa thẳng đến lớp ra (nhân với  $W'$  và qua hàm softmax) để tạo ra một vector xác suất  $\hat{y}$ .
4. **Huấn luyện:** Quá trình này được lặp lại cho từng từ trong ngữ cảnh. Với mỗi cặp (từ trung tâm, từ ngữ cảnh), một hàm mất mát Cross-Entropy được tính toán giữa dự đoán  $\hat{y}$  và vector one-hot của từ ngữ cảnh thực tế. Tổng các lỗi từ tất cả các cặp được sử dụng để cập nhật trọng số qua backpropagation. Điều này giúp Skip-gram học được các biểu diễn tinh vi hơn, đặc biệt cho các từ hiếm, nhưng cũng tốn nhiều thời gian huấn luyện hơn.



Hình 3.2: So sánh trực quan giữa CBOW (tổng hợp ngữ cảnh để dự đoán 1 từ) và Skip-gram (dùng 1 từ để dự đoán nhiều từ trong ngữ cảnh, tạo thành các cặp huấn luyện riêng biệt).

### Vì sao Word2Vec học được ngữ nghĩa và gần-xa theo cosine?

**Nguyên lý phân bố (Distributional Hypothesis).** Từ ngữ xuất hiện trong *ngữ cảnh* giống nhau thường có nghĩa giống nhau. Khi huấn luyện Word2Vec để dự đoán từ-ngữ cảnh, các vector bị tối ưu sao cho những từ chia sẻ nhiều ngữ cảnh sẽ có *hướng* gần nhau trong không gian.

**CBOW: gom ngữ cảnh để dự đoán từ.** Gọi  $C$  là tập ngữ cảnh,  $|C| = m$ ,  $v_c \in \mathbb{R}^N$  là embedding của từ  $c \in C$ , và  $u_w \in \mathbb{R}^N$  là vector "ra" cho từ trung tâm  $w$ . CBOW tính

$$h = \frac{1}{m} \sum_{c \in C} v_c, \quad p(w | C) = \text{softmax}(u_w^\top h).$$

Tối đa hoá  $\log p(w | C)$  khiến  $u_w^\top h$  tăng. Do  $h$  là trung bình các  $v_c$ , mọi  $v_c$  đều nhận gradient *đẩy đồng hướng* với  $u_w$ . Khi hai từ  $w_1, w_2$  thường cùng chia sẻ ngữ cảnh, các cập nhật lặp đi lặp lại làm  $v_{w_1}, v_{w_2}$  trở nên *đồng hướng*.

**Skip-gram với Negative Sampling (SGNS): kéo–đẩy cục bộ.** Với cặp dương  $(w, c)$  và các âm  $n \in \mathcal{N}$ , hàm mục tiêu một bước là

$$\mathcal{L}_{\text{SGNS}} = -\log \sigma(u_c^\top v_w) - \sum_{n \in \mathcal{N}} \log \sigma(-u_n^\top v_w),$$

trong đó  $\sigma(x) = \frac{1}{1+e^{-x}}$ . Gradient theo  $v_w$  là

$$\frac{\partial \mathcal{L}}{\partial v_w} = (\sigma(u_c^\top v_w) - 1) u_c + \sum_{n \in \mathcal{N}} \sigma(u_n^\top v_w) u_n.$$

*Trực giác:* với cặp dương, hệ số  $\sigma(u_c^\top v_w) - 1 < 0$  kéo  $v_w$  *tiến* về hướng  $u_c$  (tăng tích vô hướng); với cặp âm, các hạng  $+\sigma(u_n^\top v_w) u_n$  *đẩy*  $v_w$  *ra xa* các  $u_n$ . Qua thời gian, các từ chia sẻ ngữ cảnh dương sẽ bị kéo về *cùng vùng* (gần nhau), còn các từ hiếm khi cùng ngữ cảnh sẽ bị *đẩy xa*.

**Vì sao cosine similarity?** Huấn luyện tối ưu *tích vô hướng*  $u^\top v$  (do xuất hiện trong softmax/SGNS). Khi chuẩn hoá độ dài, tối đa hoá tích vô hướng tương đương tối đa hoá *cosine*:

$$u^\top v = \|u\| \|v\| \cos \theta.$$

Trong thực tế, độ dài có thể dao động, nhưng *hướng* mới mã hoá “vị trí ngữ nghĩa”. Do đó, dùng cosine để đo gần-xa phản ánh đúng mục tiêu tối ưu hoá hướng.

### Gần nghĩa, khác nghĩa, và trái nghĩa.

- **Gần nghĩa  $\Rightarrow$  gần nhau:** nếu hai từ thường xuất hiện với *cùng loại ngữ cảnh*, các cập nhật kéo chúng về *đồng hướng*  $\Rightarrow$  cosine cao.
- **Khác nghĩa  $\Rightarrow$  xa nhau:** nếu hầu như không chia sẻ ngữ cảnh, vector bị các cập nhật khác hướng  $\Rightarrow$  cosine thấp.
- **Trái nghĩa vẫn có thể gần nhau:** các cặp như *nóng-lạnh* xuất hiện trong *ngữ cảnh hình thức* tương tự (“thời tiết nóng/lạnh”) nên vẫn bị kéo về gần nhau; Word2Vec không mô hình hoá *độ phân cực nghĩa*. Muốn phân biệt polarity thường cần thêm tín hiệu (sentiment, lexicon, hoặc mô hình ngôn ngữ ngữ cảnh hoá).

**Liên hệ lý thuyết: SGNS và Phân rã Ma trận PMI** Một khám phá lý thuyết quan trọng của Levy và Goldberg [48] đã chỉ ra rằng mục tiêu tối ưu của Skip-gram với Negative Sampling (SGNS) thực chất tương đương với việc **phân rã ngầm định (implicit factorization)** một ma trận đặc biệt. Cụ thể, hàm mục tiêu của SGNS hội tụ về việc tối ưu hóa sao cho:

$$\mathbf{w}_i^\top \mathbf{c}_j \approx \text{PMI}(w_i, c_j) - \log k$$

trong đó  $\mathbf{w}_i$  và  $\mathbf{c}_j$  là các vector đầu vào và đầu ra,  $k$  là số mẫu âm, và PMI (Pointwise Mutual Information) là một thước đo mức độ liên quan giữa hai từ:

$$\text{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)}$$

Phát hiện này kết nối Word2Vec (một mô hình dự đoán, dựa trên mạng nơ-ron) với các phương pháp dựa trên đếm truyền thống (như LSA, GloVe) vốn trực tiếp phân rã ma trận đồng xuất hiện hoặc PMI. Nó cho thấy trực giác ”kéo–đẩy” của SGNS không chỉ là một heuristic, mà là một cách hiệu quả để xấp xỉ một cấu trúc thống kê toàn cục của ngôn ngữ, làm sáng tỏ lý do tại sao các mối quan hệ ngữ nghĩa lại có thể được học một cách hiệu quả như vậy.

### Hệ quả và giới hạn.

- **Tổng quát hoá theo ngữ cảnh:** “nghĩa” trong Word2Vec chủ yếu là *tương đồng ngữ cảnh* (contextual co-occurrence).
- **Đa nghĩa:** một vector cho nhiều nghĩa sẽ trộn các ngữ cảnh (polysemy). Các mô hình *ngữ cảnh hoá* (ELMo, BERT) khắc phục bằng vector phụ thuộc câu.
- **Cosine là thước đo mặc định:** vì hướng là tín hiệu chính của “ngữ nghĩa” đã học qua  $u^\top v$ .

#### Trực giác “kéo–đẩy” trong SGNS

**Kéo:** cặp dương  $(w, c)$  làm tăng  $u_c^\top v_w \Rightarrow v_w$  tiến về hướng  $u_c$ .    **Đẩy:** cặp âm  $(w, n)$  làm giảm  $u_n^\top v_w \Rightarrow v_w$  rời xa hướng  $u_n$ .

Lặp lại trên toàn corpora tạo ra cấu trúc không gian nơi *các trường ngữ cảnh* được biểu diễn bằng *hướng vector*; cosine đo độ tương đồng các trường ngữ cảnh đó.

### Thách thức tính toán và các kỹ thuật tối ưu hóa

Cả hai mô hình đều đối mặt với một nút thắt cổ chai lớn: hàm **softmax** ở lớp ra. Việc tính toán mẫu số của hàm softmax đòi hỏi phải duyệt qua toàn bộ V từ trong từ vựng, cực kỳ tốn kém khi V có thể lên tới hàng triệu. Để giải quyết vấn đề này, hai kỹ thuật chính được đề xuất:

**Subsampling và Cửa sổ ngữ cảnh động** Trước khi đi vào các kỹ thuật tối ưu hóa chính, có hai thủ thuật quan trọng khác giúp cải thiện chất lượng embedding và tăng tốc độ huấn luyện:

- **Subsampling các từ thường xuyên:** Các từ xuất hiện rất thường xuyên (như “là”, “và”, “trong” trong tiếng Việt, hay “the”, “a” trong tiếng Anh) mang ít thông tin ngữ nghĩa đặc trưng. Chúng xuất hiện trong gần như mọi ngữ cảnh, do đó các cặp huấn luyện tạo ra từ chúng không giúp ích nhiều mà còn làm chậm quá trình huấn luyện. Subsampling là một kỹ thuật loại bỏ ngẫu nhiên các từ này trong quá trình tạo mẫu, với xác suất loại bỏ càng cao khi tần suất của từ càng lớn. Điều này giúp mô hình tập trung hơn vào các từ hiếm và quan trọng hơn.
- **Cửa sổ ngữ cảnh động (Dynamic Window):** Thay vì sử dụng một cửa sổ ngữ cảnh cố định (ví dụ, luôn là 5), nhiều triển khai của Word2Vec sử dụng một kích thước cửa sổ động. Với mỗi từ trung tâm, một kích thước cửa sổ  $C$  sẽ được chọn ngẫu nhiên trong khoảng từ 1 đến kích thước tối đa cho phép. Kỹ thuật này có tác dụng ngầm định là gán trọng số cao hơn cho các từ ngữ cảnh ở gần từ trung tâm, vì chúng có xác suất được lấy mẫu cao hơn các từ ở xa.

**Negative Sampling** Đây là kỹ thuật phổ biến hơn. Thay vì cập nhật trọng số cho toàn bộ từ vựng, ý tưởng là biến bài toán phân loại đa lớp (multi-class) thành một loạt các bài toán phân loại nhị phân (binary classification). Với mỗi cặp đầu vào (ví dụ, trong Skip-gram là (**từ trung tâm, từ ngữ cảnh**):

1. Chúng ta coi cặp này là một mẫu **dương tính (positive sample)**.
2. Chúng ta lấy ngẫu nhiên  $k$  từ khác từ từ vựng (dựa trên một phân phối xác suất nhất định) và tạo ra  $k$  mẫu **âm tính (negative samples)**.
3. Mô hình sẽ được huấn luyện để dự đoán đúng “nhận”(1 cho mẫu dương tính, 0 cho các mẫu âm tính).

Bằng cách này, trong mỗi bước cập nhật, chúng ta chỉ cần tính toán và cập nhật trọng số cho  $k + 1$  từ thay vì toàn bộ V từ, giúp tăng tốc độ huấn luyện lên rất nhiều.

### Lựa chọn Thiết kế trong Negative Sampling

#### 1. Phân phối lấy mẫu âm (The Sampling Distribution):

Việc lấy mẫu các từ "âm tính" không phải là ngẫu nhiên đồng nhất trên toàn bộ từ vựng. Thay vào đó, Mikolov đề xuất sử dụng một phân phối Unigram đã được điều chỉnh, trong đó xác suất chọn một từ  $w$  làm mẫu âm là:

$$P(w) = \frac{f(w)^{3/4}}{\sum_{j=1}^V f(w_j)^{3/4}}$$

trong đó  $f(w)$  là tần suất của từ  $w$ . Lũy thừa  $3/4$  là một heuristic được chọn qua thực nghiệm.

- Nếu dùng lũy thừa 1 (theo tần suất thô), các từ rất phổ biến sẽ được chọn làm mẫu âm quá thường xuyên.
- Nếu dùng lũy thừa 0 (phân phối đều), các từ hiếm sẽ được chọn thường xuyên một cách phi thực tế.
- Lũy thừa  $3/4$  là một sự cân bằng hợp lý: nó làm giảm nhẹ tần suất của các từ phổ biến và tăng nhẹ tần suất của các từ hiếm, tạo ra các mẫu âm đa dạng và thách thức hơn cho mô hình.

#### 2. Số lượng mẫu âm ( $k$ ):

Tham số  $k$  (số mẫu âm cho mỗi mẫu dương) thể hiện sự đánh đổi giữa chất lượng và tốc độ.

- Với các tập dữ liệu nhỏ,  $k$  lớn hơn (ví dụ: 5-20) thường cho kết quả tốt hơn vì mỗi mẫu dương cần nhiều ví dụ âm để mô hình học được ranh giới quyết định.
- Với các tập dữ liệu cực lớn,  $k$  có thể nhỏ hơn (ví dụ: 2-5), vì mô hình đã thấy đủ đa dạng mẫu trong dữ liệu.

Giá trị của  $k$  càng lớn, gradient càng ổn định nhưng chi phí tính toán cho mỗi bước cập nhật càng cao.

**Hierarchical Softmax** Kỹ thuật này sử dụng một cây nhị phân (cụ thể là cây Huffman) để biểu diễn tất cả các từ trong từ vựng. Các từ thường xuyên xuất hiện sẽ có đường đi ngắn hơn trên cây. Thay vì dự đoán một từ, mô hình sẽ dự đoán một chuỗi các lựa chọn trái/phải để đi từ gốc đến lá của từ đó. Điều này giảm độ phức tạp tính toán từ  $O(V)$  xuống còn  $O(\log_2 V)$ .

### So sánh CBOW và Skip-gram

CBOW	Skip-gram
Dự đoán từ trung tâm từ ngữ cảnh.	Dự đoán các từ ngữ cảnh từ từ trung tâm.
Nhanh hơn, chỉ một lần cập nhật cho mỗi cửa sổ ngữ cảnh.	Chậm hơn, phải cập nhật nhiều lần (mỗi từ trong ngữ cảnh).
Kém hiệu quả với từ hiếm do việc lấy trung bình làm "lu mờ" chúng.	Tốt hơn với từ hiếm vì mỗi từ vẫn tạo ra một cặp huấn luyện riêng.
Tốt cho các từ thường xuyên xuất hiện.	Thường tốt hơn với dữ liệu lớn và từ hiếm.
<b>Khi nào dùng?</b> Khi tốc độ là ưu tiên và dữ liệu rất lớn.	<b>Khi nào dùng?</b> Thường được ưa chuộng trong nghiên cứu vì chất lượng biểu diễn cao.

Nhìn chung, Skip-gram thường được ưa chuộng hơn trong nghiên cứu.

#### 3.1.2. Lý thuyết GloVe: Ma trận Đồng xuất hiện

GloVe [64] (*Global Vectors for Word Representation*) được giới thiệu năm 2014 bởi các nhà nghiên cứu tại Stanford như một phương pháp học **word embeddings** dựa trên thống kê toàn cục của ngôn

ngữ. Khác với Word2Vec (mô hình *predictive*, học từ các ngữ cảnh cục bộ), GloVe là một mô hình *count-based* được huấn luyện bằng cách duy tối ưu giống mô hình dự đoán, nhằm tận dụng cả thông tin toàn cục và cục bộ.

**Tần suất đồng xuất hiện** Tần suất đồng xuất hiện giữa các từ mang ý nghĩa ngữ nghĩa quan trọng. Ví dụ: từ *ice* và *steam* đều đồng xuất hiện với *water*, nhưng:

- *ice* đồng xuất hiện nhiều hơn với *solid*, ít với *gas*.
- *steam* đồng xuất hiện nhiều hơn với *gas*, ít với *solid*.

So sánh tỉ lệ xác suất đồng xuất hiện giúp phân biệt nghĩa của các từ.

#### GloVe: Sự giao thoa giữa Count-based và Predictive

Một câu hỏi thường gặp là: Nếu GloVe xây dựng trên ma trận đếm toàn cục, tại sao nó không đơn giản là phân rã ma trận đó (như LSA)? Và nếu hàm mất mát của nó trông giống một bài toán dự đoán, tại sao nó không phải là mô hình predictive như Word2Vec?

Câu trả lời nằm ở chỗ GloVe kết hợp những điểm mạnh của cả hai phương pháp:

- **Tận dụng thống kê toàn cục (Count-based):** Bằng cách tính toán ma trận đồng xuất hiện  $X$ , GloVe có cái nhìn bao quát về toàn bộ kho văn bản ngay từ đầu. Mọi thông tin về tần suất và sự đồng xuất hiện đều được mã hóa trong ma trận này.
- **Tối ưu hóa hiệu quả (Predictive-style):** Thay vì thực hiện các phép phân rã ma trận phức tạp và tốn kém (như SVD trong LSA), GloVe định nghĩa một hàm mất mát dạng "hồi quy có trọng số". Mô hình sau đó học các vector bằng cách tối ưu hàm mất mát này trên từng cặp  $(i, j)$  không rỗng trong ma trận  $X$ . Cách học lặp đi lặp lại trên các mẫu riêng lẻ này rất giống với tinh thần của các mô hình dự đoán, giúp việc huấn luyện trở nên hiệu quả và có thể song song hóa.

Do đó, GloVe có thể được xem là một mô hình "lai": nó trực tiếp tối ưu hóa để học các vector sao cho chúng có thể tái tạo lại được các thống kê toàn cục của ngôn ngữ.

**Tỉ lệ xác suất** Cho từ  $i$ ,  $j$  và từ ngữ cảnh  $k$ , xác suất đồng xuất hiện được định nghĩa:

$$P_{ik} = \frac{\text{số lần từ } k \text{ xuất hiện gần } i}{\text{tổng số lần xuất hiện ngữ cảnh của } i}$$

GloVe chỉ ra rằng **tỉ lệ**  $\frac{P_{ik}}{P_{jk}}$  chứa thông tin ngữ nghĩa: nếu  $k$  liên quan nhiều đến  $i$  hơn  $j$ , tỉ lệ này sẽ lớn.

**Từ Tỉ lệ đến Tích vô hướng: Cầu nối Toán học** Đây là bước nhảy vọt tinh tế nhất của GloVe. Các tác giả đã lập luận như sau:

1. Mỗi quan hệ giữa ba từ  $i, j, k$  được thể hiện qua tỉ lệ  $P_{ik}/P_{jk}$ . Mỗi quan hệ này nên được mô hình hóa bởi các vector của chúng:  $w_i, w_j, w_k$ . Ta có thể bắt đầu với một hàm tổng quát  $F$ :

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}}$$

2. Để mã hóa thông tin trong không gian vector tuyến tính, các tác giả đề xuất rằng  $F$  nên chỉ phụ thuộc vào hiệu của hai từ đầu tiên,  $F(w_i - w_j, w_k)$ . Điều này giúp nắm bắt các mối quan hệ loại suy (analogy).
3. Để đơn giản hóa hơn nữa, họ nhận thấy rằng  $w_i - w_j$  phải là một đại lượng vô hướng, trong khi  $w_k$  trái là một hàm của các vector. Cách tự nhiên nhất để biến các vector thành vô hướng là dùng tích vô hướng. Điều này dẫn đến:

$$F(w_i - w_j, w_k) \rightarrow (w_i - w_j)^\top w_k = w_i^\top w_k - w_j^\top w_k$$

4. Kết hợp các bước trên và một vài biến đổi toán học (sử dụng tính chất của hàm đồng cấu giữa nhóm cộng và nhân), họ đi đến kết luận rằng một dạng hàm phù hợp là:

$$w_i^\top w_k \approx \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

Trong đó  $X_{ik}$  là số lần  $k$  xuất hiện trong ngũ cảnh của  $i$ , và  $X_i$  là tổng số lần các từ xuất hiện trong ngũ cảnh của  $i$ .

5. Hạng tử  $\log(X_i)$  không phụ thuộc vào  $k$ . Nó có thể được gộp vào một số hạng bias  $b_i$ . Tương tự, để giữ tính đối xứng, một bias  $b_k$  cũng được thêm vào cho từ ngũ cảnh. Điều này dẫn đến công thức mục tiêu cuối cùng, chính là nền tảng cho hàm mất mát của GloVe.

Quá trình này cho thấy công thức mục tiêu không phải là ngẫu nhiên, mà là kết quả của một chuỗi lập luận chặt chẽ nhằm mã hóa các tỉ lệ thống kê vào một cấu trúc vector tuyến tính.

### Cơ chế hoạt động

1. **Xây dựng ma trận đồng xuất hiện  $X$ :** Duyệt toàn bộ kho văn bản, xây dựng ma trận  $X \in \mathbb{R}^{V \times V}$  với:

$$X_{ij} = \text{số lần từ } j \text{ xuất hiện trong ngũ cảnh của } i.$$

*Ngũ cảnh* thường được định nghĩa bởi một cửa sổ từ nhất định (window size).

2. **Mục tiêu tối ưu:** Ta muốn tìm vector từ  $w_i, w_j \in \mathbb{R}^d$  và bias  $b_i, b_j$  sao cho:

$$w_i^\top w_j + b_i + b_j \approx \log X_{ij}.$$

Lý do dùng log là vì tần suất đồng xuất hiện phân bố lệch (long-tailed), logarit giúp “nén” khoảng giá trị.

3. **Hàm mất mát:**

$$J = \sum_{i,j=1}^V f(X_{ij}) \left( w_i^\top w_j + b_i + b_j - \log X_{ij} \right)^2 \quad (3.1)$$

Trong đó,  $f(X_{ij})$  là hàm trọng số:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{nếu } x < x_{\max}, \\ 1 & \text{nếu } x \geq x_{\max} \end{cases}$$

với  $\alpha \approx 0.75$  và  $x_{\max}$  thường khoảng 100. Hàm này giảm ảnh hưởng của cặp từ xuất hiện quá thường xuyên và bỏ qua nhiều từ cặp quá hiếm.

4. **Kết quả:** Sau khi tối ưu, tích vô hướng  $w_i^\top w_j$  xấp xỉ log xác suất đồng xuất hiện, nghĩa là khoảng cách giữa các vector phản ánh mối quan hệ thống kê toàn cục của ngôn ngữ.

### Giải mã Hàm mất mát của GloVe

Mỗi thành phần trong hàm mất mát của GloVe đều có một vai trò quan trọng và được lựa chọn cẩn thận:

- **Vai trò của Logarit ( $\log X_{ij}$ ):** Tần suất đồng xuất hiện  $X_{ij}$  có phân bố trải dài trên nhiều bậc độ lớn (từ 1 đến hàng triệu). Việc lấy logarit giúp "nén" khoảng giá trị này lại. Điều này ngăn các cặp từ có tần suất cực lớn (ví dụ: "the" và "is") tạo ra một giá trị lỗi quá lớn và lấn át hoàn toàn đóng góp của các cặp từ hiếm nhưng mang nhiều ý nghĩa.
- **Ý nghĩa của các Bias ( $b_i, b_j$ ):** Các số hạng bias này không chỉ là một kỹ thuật thông thường. Chúng giúp hấp thụ các hiệu ứng chính (main effects) liên quan đến tần suất. Cụ thể,  $b_i$  có thể học để mã hóa sự "nổi tiếng" chung của từ  $i$  (một từ rất phổ biến sẽ có  $X_{ij}$  lớn với nhiều từ  $j$ ). Bằng cách để các bias xử lý phần thông tin này, vector chính  $w_i$  và  $w_j$  có thể tập trung hơn vào việc mã hóa mối quan hệ tương tác ngữ nghĩa giữa hai từ, làm cho các vector trở nên "trong sáng" hơn.
- **Tại sao cần hàm trọng số  $f(X_{ij})$ ?:** Đây là thành phần quan trọng nhất, giải quyết hai vấn đề lớn:
  1. **Các cặp có  $X_{ij} = 0$ :** Có vô số cặp từ không bao giờ đồng xuất hiện. Nếu không có  $f(X_{ij})$ , chúng ta sẽ phải tính tổng trên tất cả  $V \times V$  cặp, và  $\log(0)$  là không xác định. Hàm trọng số được thiết kế sao cho  $f(0) = 0$ , giúp loại bỏ hoàn toàn các cặp này khỏi hàm mất mát một cách tự nhiên.
  2. **Giảm trọng số các cặp quá phổ biến:** Các cặp từ đồng xuất hiện rất thường xuyên (ví dụ: "it is", "of the") thường mang ít thông tin ngữ nghĩa đặc biệt. Dù đã có logarit, chúng vẫn có thể thống trị quá trình huấn luyện. Hàm  $f(x)$  với  $\alpha < 1$  (thường là 0.75) và ngưỡng  $x_{\max}$  đảm bảo rằng các cặp này không có trọng số quá lớn, cho phép mô hình quan tâm hơn đến các cặp từ hiếm hơn nhưng thú vị hơn về mặt ngữ nghĩa.

### Điểm mạnh và hạn chế

- **Ưu điểm:**
  - Kết hợp thông tin toàn cục (thống kê đồng xuất hiện) và cục bộ (tối ưu vector).
  - Huấn luyện nhanh hơn Word2Vec trên dữ liệu lớn vì sử dụng ma trận đồng xuất hiện tiền tính toán.
  - Vector tạo ra ổn định hơn với nhiều lần train.
- **Hạn chế:**
  - Yêu cầu lưu trữ ma trận đồng xuất hiện rất lớn khi từ vựng lớn.
  - Không sinh embedding phụ thuộc ngữ cảnh (mỗi từ một vector cố định).

#### 3.1.3. Lý thuyết FastText: Biểu diễn Subword

FastText được phát triển bởi nhóm Facebook AI Research (FAIR) năm 2017 [7], nhằm khắc phục một hạn chế quan trọng của các mô hình như Word2Vec và GloVe: **không xử lý được các từ ngoài từ vựng** (Out-of-Vocabulary - OOV). Trong Word2Vec/GloVe, mỗi từ được coi như một "đơn vị nguyên tử" (atomic unit), và nếu từ đó không xuất hiện trong quá trình huấn luyện thì mô hình hoàn toàn không có vector cho nó.

**Tu duy cốt lõi** Thay vì gán mỗi từ một vector duy nhất, FastText biểu diễn từ như một **tập hợp các n-gram ký tự** (*bag of character n-grams*). Điều này cho phép mô hình:

- Tận dụng thông tin cấu trúc hình thái (morphology) của từ.
- Sinh vector cho từ mới dựa trên các thành phần subword quen thuộc.

### Cơ chế hoạt động

- Phân rã từ thành n-gram ký tự: Với kích thước  $n = 3$  (3-gram), từ *sách* sẽ được bao quanh ký hiệu biên giới từ và phân rã thành:

$$\langle s, \quad sa, \quad sc, \quad ch, \quad ch \rangle$$

Ngoài ra, chính từ hoàn chỉnh *sách* cũng được coi là một n-gram. Ký hiệu  $\langle \dots \rangle$  đánh dấu bắt đầu và kết thúc từ để phân biệt tiền tố/hậu tố.

- Học embedding cho subword: FastText học một vector embedding cho **mỗi** n-gram ký tự, không chỉ cho từ hoàn chỉnh.
- Tạo vector từ: Vector biểu diễn từ  $w$  là trung bình (hoặc tổng) các vector embedding của tất cả các subword của  $w$ :

$$\mathbf{v}(w) = \frac{1}{|G_w|} \sum_{g \in G_w} \mathbf{z}_g$$

trong đó:

- $G_w$  = tập các n-gram ký tự của  $w$ .
- $\mathbf{z}_g$  = embedding của n-gram  $g$ .

- Huấn luyện: Về kiến trúc, FastText vẫn sử dụng Skip-gram (hoặc CBOW) như Word2Vec, nhưng thay vì ánh xạ từ  $\rightarrow$  vector, nó ánh xạ tập hợp subword  $\rightarrow$  vector.

#### Ghi chú sâu về Biểu diễn Subword

##### 1. Tại sao vẫn giữ vector cho từ hoàn chỉnh?

Việc giữ lại vector cho cả từ gốc (ví dụ, embedding cho ‘*s>sách</s>’’) bên cạnh các n-gram của nó là một lựa chọn thiết kế quan trọng. Vector của từ hoàn chỉnh đóng vai trò như một “mỏ neo”, mang ý nghĩa tổng thể và giúp phân biệt các từ có chung nhiều n-gram nhưng nghĩa lại khác nhau. Nếu không có nó, các từ ngắn và có cấu trúc tương tự (ví dụ: ‘sun’ và ‘sin’) có thể trở nên quá gần nhau trong không gian vector. Vector của từ hoàn chỉnh đảm bảo mỗi từ có một “danh tính” riêng biệt.*

##### 2. Tại sao lấy trung bình thay vì các phép toán khác?

Lấy trung bình là một cách đơn giản và hiệu quả để kết hợp các vector subword.

- So với Tổng (Sum):** Việc lấy trung bình giúp chuẩn hóa độ dài của vector từ. Nếu dùng tổng, các từ dài (có nhiều n-gram hơn) sẽ có xu hướng tạo ra các vector có độ lớn (norm) cao hơn, điều này có thể không mong muốn.
- So với các cơ chế phức tạp hơn (Attention):** Mặc dù attention có thể học cách gán trọng số khác nhau cho các subword (ví dụ, gốc từ quan trọng hơn hậu tố), nó sẽ làm tăng đáng kể độ phức tạp tính toán, đi ngược lại triết lý “nhanh”(fast) của FastText.

##### 3. Quản lý bộ nhớ với Hashing Trick

Số lượng n-gram ký tự có thể là cực lớn. Để giữ cho mô hình có kích thước hợp lý, FastText không lưu một vector cho **mỗi** n-gram duy nhất. Thay vào đó, nó sử dụng một hàm băm (hashing function) để ánh xạ **mỗi** n-gram vào một trong số  $B$  ”bucket”(thường  $B$  khoảng 2 triệu). Tất cả các n-gram rơi vào cùng một bucket sẽ chia sẻ cùng một vector embedding. Kỹ thuật này giúp kiểm soát chật chẽ bộ nhớ, dù có thể xảy ra xung đột (hai n-gram khác nhau cùng được ánh xạ vào một bucket), nhưng trong thực tế với một không gian băm đủ lớn, ánh xạ tiêu cực là không đáng kể.

### Ưu điểm nổi bật

- Xử lý OOV:** Với từ mới chưa từng thấy, mô hình vẫn sinh được vector bằng cách cộng/trung bình embedding của các subword đã biết.
- Hiểu hình thái:** Các từ cùng gốc hoặc chia sẻ tiền tố/hậu tố sẽ có embedding gần nhau. Ví dụ: *running, runner, runs* cùng chia sẻ n-gram *run*.

## Hạn chế

- Kích thước mô hình lớn hơn do phải lưu embedding cho nhiều n-gram.
- Không xử lý được ý nghĩa phụ thuộc ngữ cảnh (mỗi từ vẫn một vector tĩnh).
- **Vấn đề từ đồng âm và sự trùng hợp ngẫu nhiên:** FastText không giải quyết được vấn đề đa nghĩa. Tệ hơn, nó có thể tạo ra các mối liên hệ sai lệch. Hai từ hoàn toàn không liên quan về nghĩa nhưng tình cờ có chung các n-gram ký tự (ví dụ, ‘pain’ và ‘Spain’ đều chứa ‘ain’) có thể bị đẩy lại gần nhau một cách không mong muốn trong không gian embedding.
- **Hiệu quả phụ thuộc vào đặc tính ngôn ngữ:** Phương pháp dựa trên n-gram ký tự hoạt động đặc biệt hiệu quả với các ngôn ngữ có hình thái học phong phú (như tiếng Đức, Thổ Nhĩ Kỳ, Phần Lan), nơi cấu trúc subword thực sự mã hóa thông tin ngữ pháp. Tuy nhiên, với các ngôn ngữ đơn lập như tiếng Việt hoặc các ngôn ngữ dùng hệ chữ tượng hình như tiếng Trung, lợi ích từ việc phân rã ký tự có thể ít rõ rệt hơn so với việc phân tích ở cấp độ từ hoặc âm tiết.

## 3.2. Autoencoders: Học Biểu diễn Phi giám sát

Trong khi Word2Vec học biểu diễn cho từng từ riêng lẻ, một câu hỏi tự nhiên được đặt ra: Làm thế nào để chúng ta có thể học được một vector biểu diễn **cho cả một câu hoặc một tài liệu?** Một trong những câu trả lời kinh điển và mạnh mẽ nhất đến từ một kiến trúc mạng nơ-ron thanh lịch có tên là **Autoencoder (AE)**.

### 3.2.1. Trực giác và Kiến trúc

Ý tưởng cốt lõi của Autoencoder rất đơn giản nhưng vô cùng hiệu quả: Thay vì cố gắng dự đoán một nhãn bên ngoài, mô hình được giao một nhiệm vụ **tự giám sát** (self-supervised): *hãy học cách tái tạo lại chính xác những gì nó nhìn thấy*.

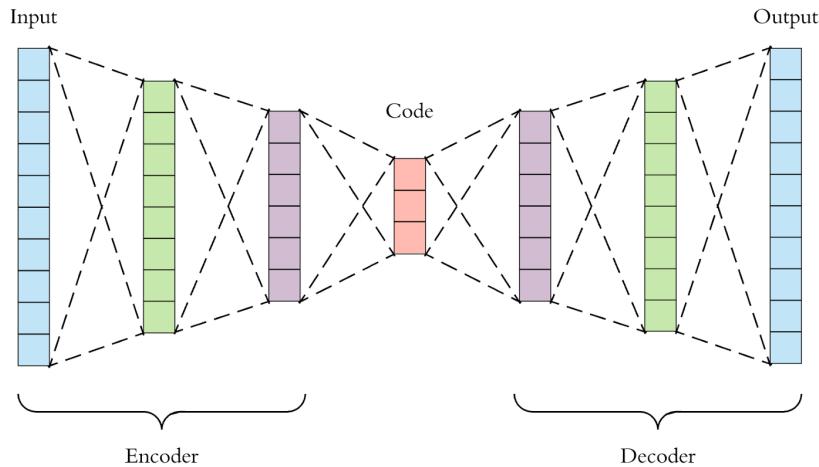
Để làm được điều này một cách có ý nghĩa, kiến trúc của Autoencoder được thiết kế có chủ đích theo dạng ”đồng hồ cát”(hourglass) gồm hai phần đối xứng:

- **Bộ mã hóa (Encoder):** Phần đầu vào của ”đồng hồ cát”, có nhiệm vụ nhận một vector đầu vào  $x$  có số chiều lớn (ví dụ, một vector Bag-of-Words cho một câu) và **nén** nó xuống một vector biểu diễn ẩn  $z$  có số chiều nhỏ hơn nhiều. Vector  $z$  này được gọi là **biểu diễn tiềm ẩn** (latent representation) hoặc **code**.
- **Bộ giải mã (Decoder):** Phần đầu ra của ”đồng hồ cát”, nhận vector nén  $z$  và cố gắng giải nén nó để tái tạo lại vector đầu vào ban đầu, ký hiệu là  $\hat{x}$ .

**Huấn luyện** Mục tiêu của mạng là tối thiểu hóa **lỗi tái tạo (reconstruction error)**, tức là làm cho vector đầu ra  $\hat{x}$  càng giống vector đầu vào  $x$  càng tốt. Điều này được thực hiện bằng cách định nghĩa một **hàm mất mát**, ví dụ như sai số bình phương trung bình (Mean Squared Error - MSE):

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Trong quá trình huấn luyện, lỗi này được lan truyền ngược (**backpropagation**) từ đầu ra trở về đầu vào, cập nhật các trọng số của cả Decoder và Encoder để chúng có thể tạo ra một biểu diễn nén  $z$  tốt hơn và tái tạo lại  $x$  chính xác hơn.



Hình 3.3: Kiến trúc tổng quát của Autoencoder. Phần hẹp nhất (*bottleneck*) chính là không gian ẩn chứa thông tin nén.

### 3.2.2. Sức mạnh của "Nút cổ chai" và việc học biểu diễn

Điểm mấu chốt làm cho Autoencoder trở nên hữu ích chính là lớp ẩn ở giữa có số chiều nhỏ, hay còn gọi là "**nút cổ chai**"(bottleneck). Lớp này buộc mạng phải học một dạng nén dữ liệu thông minh.

Nếu không có nút cổ chai (tức là số chiều của  $z$  bằng hoặc lớn hơn số chiều của  $x$ ), mạng sẽ rất dễ dàng học được một **ánh xạ đồng nhất** (identity function) tầm thường. Nó chỉ đơn giản là sao chép đầu vào ra đầu ra mà không cần phải "hiểu" bất kỳ cấu trúc tiềm ẩn nào của dữ liệu. Một mô hình như vậy sẽ tái tạo đầu vào một cách hoàn hảo nhưng biểu diễn ẩn  $z$  sẽ không có giá trị tổng quát hóa.

Ngược lại, sự tồn tại của nút cổ chai đặt ra một thách thức: làm thế nào để đi qua một không gian hẹp như vậy mà vẫn giữ đủ thông tin để tái tạo lại đầu ra một cách chính xác? Điều này buộc bộ mã hóa phải học cách:

- **Chắt lọc thông tin tinh túy:** Giữ lại những đặc trưng quan trọng nhất, có tính phân biệt cao nhất của dữ liệu.
- **Loại bỏ thông tin dư thừa và nhiễu:** Những chi tiết không cần thiết hoặc ngẫu nhiên sẽ bị loại bỏ trong quá trình nén.
- **Khai thác các mối tương quan:** Mô hình phải học được các quy luật và sự phụ thuộc lẫn nhau giữa các chiều của dữ liệu đầu vào để có thể nén chúng một cách hiệu quả.

*Mục tiêu thực sự của Autoencoder không phải là tái tạo đầu ra, mà là học được một **biểu diễn nén** ( $z$ ) giàu thông tin trong quá trình đó.*

Sau khi huấn luyện xong, chúng ta thường sẽ loại bỏ phần Decoder và chỉ sử dụng phần Encoder như một công cụ trích xuất đặc trưng mạnh mẽ: đưa một câu vào và nhận về một vector biểu diễn dày đặc, giàu ngữ nghĩa.

### 3.2.3. Ứng dụng và Các biến thể trong NLP

#### Lựa chọn Thiết kế cho Autoencoder trong NLP

Việc áp dụng Autoencoder cho văn bản đòi hỏi một số lựa chọn thiết kế quan trọng:

- **Biểu diễn Đầu vào (x):**
  - **Bag-of-Words (BoW):** Đơn giản và phổ biến. Mô hình sẽ học cách tái tạo lại tần suất của các từ trong câu. Biểu diễn tiềm ẩn  $z$  khi đó có xu hướng nắm bắt các chủ đề (topics) chính của văn bản, vì đó là thông tin quan trọng nhất để tái tạo lại túi từ.
  - **TF-IDF:** Tương tự BoW nhưng đã có trọng số cho các từ quan trọng. Điều này có thể giúp mô hình tập trung hơn vào các từ khóa có ý nghĩa.
  - **Lưu ý:** Cả hai phương pháp trên đều làm mất thông tin về trật tự từ.
- **Hàm mất mát (Reconstruction Error):**
  - **Mean Squared Error (MSE):** Phù hợp với đầu vào có giá trị thực như TF-IDF. Tuy nhiên, với BoW (giá trị đếm), nó có thể không phải là lựa chọn lý tưởng nhất về mặt lý thuyết xác suất.
  - **Cross-Entropy:** Một lựa chọn tốt hơn khi xem vector BoW (đã được chuẩn hóa) như một phân phối xác suất đa thức trên từ vựng. Hàm mất mát này sẽ cố gắng làm cho phân phối xác suất tái tạo  $\hat{x}$  gần với phân phối gốc  $x$  nhất có thể, thường cho kết quả tốt hơn trong thực tế.

Sự kết hợp giữa biểu diễn đầu vào và hàm mất mát sẽ định hình những gì mà biểu diễn tiềm ẩn  $z$  học được. Ví dụ, với BoW và Cross-Entropy,  $z$  sẽ là một "bản tóm tắt chủ đề" hiệu quả của văn bản.

Trong NLP, Autoencoder và các biến thể của nó mở ra nhiều ứng dụng quan trọng:

- **Học biểu diễn cho Câu/Đoạn văn:** Đây là ứng dụng cơ bản nhất, tạo ra các "sentence embeddings" chất lượng cao từ dữ liệu không gán nhãn, có thể được dùng cho các tác vụ phân cụm, tìm kiếm ngữ nghĩa, hoặc làm đặc trưng đầu vào cho các mô hình có giám sát.
- **Khởi tạo Trọng số (Pre-training):** Trước ký nguyên Transformer, các Autoencoder xếp chồng (Stacked Autoencoders) được dùng để khởi tạo trọng số cho từng lớp của một mạng nơ-ron sâu. Việc này giúp quá trình huấn luyện hội tụ nhanh hơn và tốt hơn.
- **Sinh Văn bản (Text Generation):** Mặc dù Autoencoder cơ bản là mô hình phân biệt, các biến thể của nó là nền tảng cho các mô hình sinh.

Một số biến thể phổ biến và có sức ảnh hưởng lớn bao gồm:

- **Denoising Autoencoder (DAE) [95]:** Một ý tưởng đơn giản nhưng cực kỳ mạnh mẽ. Thay vì học cách tái tạo lại đầu vào  $x$ , DAE được huấn luyện để tái tạo  $x$  từ một phiên bản đã bị làm hỏng (corrupted) của nó,  $\tilde{x}$  (ví dụ: một vài từ bị xóa hoặc thay thế ngẫu nhiên). Điều này buộc mô hình học được các biểu diễn bền vững (robust) hơn, không chỉ là nén mà còn phải "hiểu" để có thể "sửa lỗi".
- **Variational Autoencoder (VAE) [41]:** Một bước nhảy vọt về mặt lý thuyết, biến Autoencoder thành một mô hình sinh (generative model) thực thụ. Thay vì mã hóa đầu vào thành một vector điểm  $z$  duy nhất, VAE mã hóa nó thành một phân phối xác suất (thường là phân phối Gaussian) trong không gian tiềm ẩn. Sau khi huấn luyện, chúng ta có thể lấy mẫu một vector  $z$  ngẫu nhiên từ phân phối này và đưa qua Decoder để sinh ra một mẫu dữ liệu mới hoàn toàn nhưng vẫn tương tự dữ liệu huấn luyện. Chúng ta sẽ thảo luận kỹ hơn về VAE trong Mục 3.8.1.
- **Sequence-to-Sequence Autoencoder:** Thay vì dùng các lớp kết nối đầy đủ (fully-connected) cho văn bản dạng Bag-of-Words, biến thể này sử dụng các kiến trúc tuần tự như RNN/LSTM cho cả Encoder và Decoder. Encoder đọc một chuỗi từ và nén nó thành một vector "tư tưởng" (thought vector), sau đó Decoder sẽ cố gắng tái tạo lại chính chuỗi từ đó từ vector này. Đây chính là tiền đề cho kiến trúc Encoder-Decoder trong các bài toán Seq2Seq như dịch máy.

Tóm lại, Autoencoder là một công cụ học phi giám sát nền tảng, cung cấp một phương pháp

hiệu quả để học các biểu diễn nén, giàu ngữ nghĩa cho các đơn vị văn bản lớn hơn từ. Ý tưởng về việc học thông qua tái tạo và nén qua "nút cổ chai" là một trong những khái niệm có sức ảnh hưởng sâu rộng trong học sâu hiện đại.

### 3.3. Mạng Nơ-ron Hồi tiếp (RNNs, LSTMs, GRUs)

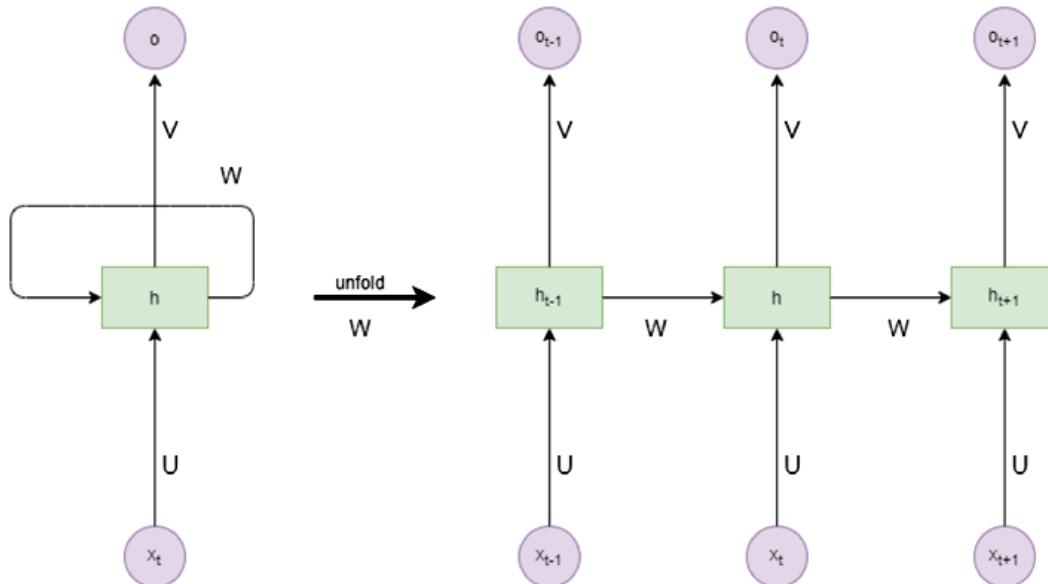
Nếu một mạng nơ-ron truyền thẳng (Feed-Forward Neural Network - FNN) là bộ não xử lý thông tin một cách tức thời, thì Mạng Nơ-ron Hồi tiếp (Recurrent Neural Network - RNN) là bộ não có trí nhớ. Đây là ý tưởng đột phá cho phép mạng nơ-ron xử lý các chuỗi có độ dài thay đổi, một đặc tính cổ hũu của ngôn ngữ.

#### 3.3.1. Mạng Nơ-ron Hồi tiếp Đơn giản (Vanilla RNN)

##### Tư duy cốt lõi: Vòng lặp của Trí nhớ

Một mạng FNN thông thường xử lý mỗi đầu vào một cách độc lập. Nó không có khái niệm về "thời gian" hay "thứ tự". Để hiểu một câu, chúng ta cần xử lý các từ theo đúng thứ tự và ghi nhớ những gì đã đọc.

RNN thực hiện điều này bằng một cơ chế đơn giản nhưng thanh lịch: một **vòng lặp (loop)**. Tại mỗi bước thời gian  $t$ , mạng không chỉ nhận đầu vào hiện tại  $x_t$  (vector của từ thứ  $t$ ), mà còn nhận cả **đầu ra của chính nó từ bước thời gian trước đó**,  $h_{t-1}$ . Đầu ra này, gọi là **trạng thái ẩn (hidden state)**, đóng vai trò như một "bộ nhớ" tóm tắt lại toàn bộ thông tin đã được xử lý cho đến thời điểm đó.



Hình 3.4: Kiến trúc của một RNN. Bên trái là dạng nén với vòng lặp. Bên phải là dạng "trải ra theo thời gian"(unrolled), cho thấy cách trạng thái ẩn  $h$  được truyền từ bước này sang bước tiếp theo.

##### Kiến trúc và Dòng chảy Dữ liệu

Tại mỗi bước thời gian  $t$ :

1. **Đầu vào:** Vector từ  $x_t$  và trạng thái ẩn từ bước trước  $h_{t-1}$ .
2. **Tính toán trạng thái ẩn mới:** Trạng thái ẩn mới,  $h_t$ , được tính bằng cách kết hợp  $x_t$  và  $h_{t-1}$  qua một phép biến đổi tuyến tính và một hàm kích hoạt (thường là 'tanh').

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \quad (3.2)$$

Trong đó,  $W_{hh}$  và  $W_{xh}$  là các ma trận trọng số được **chia sẻ (shared)** trên tất cả các bước thời gian. Đây là điểm mấu chốt: mô hình học một bộ quy tắc duy nhất để cập nhật trí nhớ, bất kể vị trí trong chuỗi.

3. **Tạo đầu ra (tùy chọn):** Một đầu ra  $y_t$  có thể được tạo ra từ trạng thái ẩn  $h_t$ .

$$y_t = W_{hy}h_t + b_y \quad (3.3)$$

#### Ghi chú sâu về Thiết kế của RNN Đơn giản

##### 1. Tại sao phải Chia sẻ Trọng số?

Đây là nguyên lý nền tảng của RNN. Việc sử dụng cùng một bộ trọng số ( $W_{hh}, W_{xh}$ ) ở mọi bước thời gian cho phép mô hình học một quy tắc cập nhật **tổng quát** có thể áp dụng cho bất kỳ vị trí nào trong chuỗi. Nếu mỗi bước thời gian có một bộ trọng số riêng, mô hình sẽ:

- Có số lượng tham số khổng lồ, phụ thuộc vào độ dài chuỗi, khiến việc huấn luyện bất khả thi.
- Không thể tổng quát hóa cho các chuỗi có độ dài khác với dữ liệu huấn luyện. Nó sẽ chỉ học thuộc lòng các mẫu ở các vị trí cố định.

Việc chia sẻ trọng số chính là cách RNN học được một "thuật toán" xử lý chuỗi.

##### 2. Tại sao dùng hàm 'tanh'?

Hàm 'tanh' đưa đầu ra về khoảng  $[-1, 1]$ . Điều này giúp giữ cho các giá trị trong trạng thái ẩn không bị tăng hoặc giảm quá nhanh, tức là kiểm soát dòng chảy thông tin một cách tương đối ổn định. Nếu dùng 'ReLU' (vốn không có giới hạn trên), trạng thái ẩn có thể tăng lên vô hạn, làm cho vấn đề bùng nổ gradient trở nên trầm trọng hơn rất nhiều. 'tanh' hoạt động như một cơ chế "bảo hòa" tự nhiên.

#### Huấn luyện RNN: Lan truyền ngược theo thời gian (BPTT)

Quá trình huấn luyện một RNN cũng dựa trên việc tối thiểu hóa một hàm mất mát, nhưng có một sự phức tạp hơn so với mạng FNN.

- **Hàm Mất mát:** Tại mỗi bước thời gian  $t$ , mô hình tạo ra một dự đoán  $y_t$ . Lỗi tại bước đó,  $\mathcal{L}_t$ , được tính toán bằng cách so sánh  $y_t$  với nhãn thật tương ứng. **Tổng lỗi** cho toàn bộ chuỗi là tổng của các lỗi tại mỗi bước:  $\mathcal{L} = \sum_{t=1}^T \mathcal{L}_t$ .
- **Lan truyền ngược theo thời gian (Backpropagation Through Time - BPTT):** Để tính toán gradient của tổng lỗi  $\mathcal{L}$  theo các trọng số (ví dụ  $W_{hh}$ ), chúng ta phải sử dụng một biến thể của backpropagation gọi là BPTT. Gradient tại một bước thời gian  $t$  không chỉ phụ thuộc vào lỗi tại bước  $t$ , mà còn phụ thuộc vào gradient từ tất cả các bước thời gian *tương lai* ( $t+1, t+2, \dots, T$ ) được truyền ngược lại thông qua trạng thái ẩn  $h$ .
- **Vấn đề của BPTT:** Chính việc gradient phải được truyền ngược qua nhiều bước thời gian này là nguyên nhân gây ra các vấn đề nan giải được thảo luận dưới đây.

#### Vấn đề nan giải: Vanishing và Exploding Gradients

RNN về lý thuyết có thể học các phụ thuộc tầm xa. Tuy nhiên, trong thực tế, chúng gặp rất nhiều khó khăn. Quá trình huấn luyện RNN sử dụng thuật toán lan truyền ngược theo thời gian (Backpropagation Through Time - BPTT). Khi lan truyền gradient lỗi ngược qua nhiều bước thời gian, gradient này phải đi qua phép nhân lặp đi lặp lại với ma trận trọng số  $W_{hh}$ .

- **Vanishing Gradients (Triệt tiêu Gradient):** Nếu các giá trị trong  $W_{hh}$  nhỏ (norm < 1), sau nhiều lần nhân, gradient sẽ co lại và tiến dần về 0.
  - **Hậu quả:** Mạng không thể học được các phụ thuộc giữa các từ ở xa nhau. Tín hiệu lỗi từ tương lai không thể lan truyền đủ xa về quá khứ để cập nhật trọng số một cách có ý nghĩa. RNN trở nên "thiến cận", chỉ nhớ được vài bước gần nhất. Đây là vấn đề nghiêm trọng nhất.

- Exploding Gradients (Bùng nổ Gradient): Nếu các giá trị trong  $W_{hh}$  lớn (norm > 1), gradient sẽ tăng theo cấp số nhân và trở nên khổng lồ.
  - Hậu quả:** Gây ra các bước cập nhật trọng số cực lớn, làm cho quá trình huấn luyện trở nên mất ổn định. Vấn đề này có thể được giải quyết phần nào bằng kỹ thuật *gradient clipping* (cắt bớt gradient nếu nó vượt quá một ngưỡng).

Vấn đề triệt tiêu gradient đã thúc đẩy sự ra đời của các kiến trúc RNN phức tạp hơn, có khả năng kiểm soát dòng chảy thông tin một cách thông minh hơn.

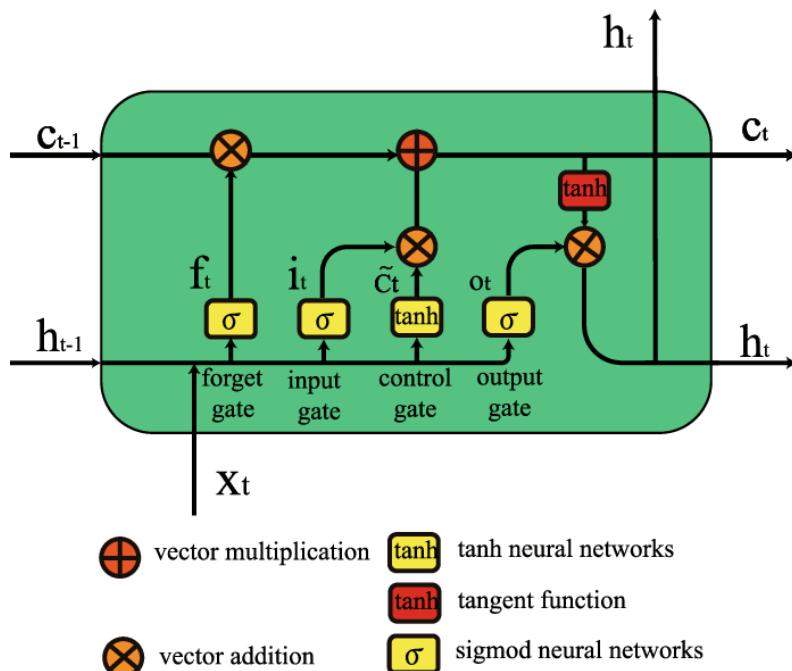
### 3.3.2. Long Short-Term Memory (LSTM)

LSTM, được giới thiệu bởi Hochreiter & Schmidhuber vào năm 1997 [31], là một biến thể đặc biệt của RNN, được thiết kế một cách xuất sắc để giải quyết vấn đề triệt tiêu gradient. Nó đã trở thành kiến trúc thống trị cho các bài toán chuỗi trong gần hai thập kỷ.

#### Tư duy cốt lõi: Các Cổng và Trạng thái Ô nhớ

Ý tưởng thiên tài của LSTM là giới thiệu một "làn đường cao tốc" cho thông tin, gọi là **trạng thái ô nhớ (cell state)**, ký hiệu là  $C_t$ . Trạng thái ô nhớ này chạy dọc theo toàn bộ chuỗi, chỉ với một vài tương tác tuyến tính nhỏ. Thông tin có thể dễ dàng chảy trên làn đường này mà không bị thay đổi nhiều.

LSTM có khả năng thêm hoặc bớt thông tin ra khỏi trạng thái ô nhớ một cách cẩn thận, được điều khiển bởi các cấu trúc gọi là **cổng (gates)**. Cổng là một cách để thông tin có thể đi qua một cách tùy chọn. Chúng bao gồm một lớp mạng sigmoid và một phép nhân theo từng phần tử (element-wise multiplication).



Hình 3.5: Sơ đồ chi tiết của một ô nhớ LSTM. Nó bao gồm 3 cổng chính (Quên, Nạp, Xuất) và một trạng thái ô nhớ  $C_t$  chạy ngang.

Một ô LSTM có ba cổng để bảo vệ và kiểm soát trạng thái ô nhớ:

- Cổng Quên (Forget Gate -  $f_t$ )** Cổng này quyết định xem thông tin nào nên được loại bỏ khỏi trạng thái ô nhớ của bước trước.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Nó nhìn vào  $h_{t-1}$  và  $x_t$ , và xuất ra một vector số từ 0 đến 1 cho mỗi số trong trạng thái ô nhớ  $C_{t-1}$ . Số 1 có nghĩa là "giữ lại hoàn toàn", số 0 có nghĩa là "quên hoàn toàn".

**2. Cổng Nạp (Input Gate -  $i_t$ )** Cổng này quyết định xem thông tin mới nào nên được lưu trữ vào trạng thái ô nhớ. Quá trình này có hai bước:

1. Một lớp sigmoid (cổng nạp) quyết định những giá trị nào chúng ta sẽ cập nhật.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

2. Một lớp 'tanh' tạo ra một vector các giá trị ứng viên mới,  $\tilde{C}_t$ , có thể được thêm vào trạng thái.

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

**3. Cập nhật Trạng thái Ô nhớ** Bây giờ, chúng ta cập nhật trạng thái ô nhớ cũ  $C_{t-1}$  thành trạng thái mới  $C_t$ .

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Ký hiệu  $\odot$  là phép nhân theo từng phần tử. Chúng ta nhân trạng thái cũ với  $f_t$  (quên đi những gì đã quyết định quên), sau đó cộng với  $i_t \odot \tilde{C}_t$  (thêm vào những thông tin mới đã quyết định nạp). Phép cộng này chính là chìa khóa giúp gradient có thể chảy ngược một cách dễ dàng, giải quyết vấn đề triệt tiêu gradient.

**4. Cổng Xuất (Output Gate -  $o_t$ )** Cuối cùng, cổng này quyết định xem chúng ta sẽ xuất ra cái gì. Đầu ra này sẽ dựa trên trạng thái ô nhớ của chúng ta, nhưng sẽ là một phiên bản đã được "lọc".

1. Một lớp sigmoid quyết định phần nào của trạng thái ô nhớ chúng ta sẽ xuất ra.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

2. Chúng ta đưa trạng thái ô nhớ qua hàm 'tanh' (để đưa giá trị về khoảng [-1, 1]) và nhân nó với đầu ra của cổng sigmoid, để chỉ xuất ra những phần chúng ta muốn.

$$h_t = o_t \odot \tanh(C_t)$$

Trạng thái ẩn mới  $h_t$  này sẽ được sử dụng để dự đoán đầu ra  $y_t$  và được truyền tới bước thời gian tiếp theo.

**Phân tích: Tại sao LSTM hoạt động?** Sự kỳ diệu của LSTM nằm ở cách nó tách biệt dòng chảy thông tin và kiểm soát nó một cách tinh vi:

- **Con đường cao tốc Gradient (Gradient Highway):** Nhìn lại công thức cập nhật trạng thái ô nhớ:  $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ . Thành phần quan trọng nhất là phép cộng. Trong quá trình lan truyền ngược, gradient của  $C_t$  đối với  $C_{t-1}$  có một thành phần trực tiếp là  $f_t$ . Do  $f_t$  được điều khiển bởi hàm sigmoid, giá trị của nó thường không bị quá nhỏ hoặc quá lớn. Điều này tạo ra một "con đường" gần như không bị cản trở để gradient có thể chảy ngược qua nhiều bước thời gian mà không bị triệt tiêu hoàn toàn. RNN đơn giản chỉ có phép nhân lặp đi lặp lại, khiến gradient dễ dàng biến mất.
- **Phân vai Trí nhớ:** LSTM tạo ra một sự phân vai rõ ràng:
  - **Trạng thái Ô nhớ ( $C_t$ ):** Đóng vai trò như **trí nhớ dài hạn**. Nó chứa tất cả thông tin tích lũy được. Nó có thể được thay đổi một cách cẩn trọng thông qua các cổng, nhưng bản chất là một "kho lưu trữ".
  - **Trạng thái Ẩn ( $h_t$ ):** Đóng vai trò như **trí nhớ làm việc (working memory)**. Nó là một phiên bản "đã lọc" của trí nhớ dài hạn, chỉ chứa những thông tin liên quan đến quyết định cần đưa ra *tại thời điểm hiện tại*. Việc tách biệt này cho phép mô hình vừa lưu trữ thông tin từ rất xa trong quá khứ ( $C_t$ ), vừa tập trung vào những gì cần thiết cho bước tiếp theo ( $h_t$ ).

### 3.3.3. Gated Recurrent Unit (GRU)

Được giới thiệu bởi Cho và các cộng sự vào năm 2014, GRU [11] là một biến thể của LSTM với kiến trúc đơn giản hơn. Nó kết hợp cổng quên và cổng nạp thành một "cổng cập nhật" duy nhất, và cũng hợp nhất trạng thái ô nhớ và trạng thái ẩn.

**Tư duy cốt lõi** GRU giữ lại tinh thần của LSTM là sử dụng các cổng để điều khiển dòng chảy thông tin, nhưng với ít tham số hơn, giúp việc huấn luyện nhanh hơn và hiệu quả hơn trên các bộ dữ liệu nhỏ.

GRU có hai cổng chính:

**1. Cổng Cập nhật (Update Gate -  $z_t$ )** Cổng này có vai trò tương tự cả cổng quên và cổng nạp trong LSTM. Nó quyết định xem bao nhiêu thông tin từ trạng thái ẩn quá khứ ( $h_{t-1}$ ) nên được giữ lại, và bao nhiêu thông tin mới ( $\tilde{h}_t$ ) nên được thêm vào.

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

**2. Cổng Reset (Reset Gate -  $r_t$ )** Cổng này quyết định xem bao nhiêu phần của trạng thái ẩn quá khứ nên được "quên" đi khi tính toán trạng thái ứng viên mới.

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

#### Tính toán Trạng thái

1. Trạng thái ẩn ứng viên mới  $\tilde{h}_t$  được tính toán, nhưng có sử dụng cổng reset để kiểm soát ảnh hưởng của  $h_{t-1}$ .

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h)$$

Nếu cổng reset  $r_t$  gần bằng 0, mô hình sẽ gần như bỏ qua hoàn toàn thông tin quá khứ và chỉ tập trung vào đầu vào hiện tại  $x_t$  để tạo trạng thái mới.

2. Trạng thái ẩn cuối cùng  $h_t$  là một sự nối suy tuyến tính giữa trạng thái cũ  $h_{t-1}$  và trạng thái ứng viên mới  $\tilde{h}_t$ , được điều khiển bởi cổng cập nhật  $z_t$ .

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

**Phân tích: Sự đánh đổi của GRU** GRU là một ví dụ tuyệt vời về việc đơn giản hóa kiến trúc mà vẫn giữ được hiệu quả cốt lõi.

- **Cổng Reset - Cơ chế "Khởi động lại Ngũ cảnh":** Cổng reset ( $r_t$ ) cho phép GRU bỏ qua hoàn toàn trí nhớ quá khứ khi cần thiết. Hãy tưởng tượng khi đọc một đoạn văn và gặp một dấu chấm hết câu, bắt đầu một câu mới với chủ đề khác. Cổng reset có thể học cách "kích hoạt" (đưa giá trị về gần 0) tại những điểm chuyển tiếp này, cho phép trạng thái ứng viên mới  $\tilde{h}_t$  được hình thành chủ yếu từ đầu vào hiện tại  $x_t$ , tạo ra một khởi đầu mới cho ngũ cảnh.
- **Cổng Cập nhật - Nội suy muộn mà:** Công thức  $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$  hoạt động như một bộ điều khiển "rò rỉ". Khi  $z_t$  gần bằng 0, phần lớn  $h_{t-1}$  sẽ được sao chép trực tiếp sang  $h_t$ , giúp thông tin được bảo tồn qua các bước thời gian dài. Ngược lại, khi  $z_t$  gần bằng 1, thông tin mới sẽ được ưu tiên. Sự cân bằng này giúp GRU mô hình hóa các phụ thuộc dài hạn một cách hiệu quả, dù không có một "trạng thái ô nhớ" riêng biệt.

Việc gộp trạng thái ô nhớ và trạng thái ẩn làm cho GRU mất đi một chút sức mạnh biểu diễn so với LSTM, nhưng chính sự đơn giản này lại giúp nó huấn luyện nhanh hơn và đòi hỏi tổng quát hóa tốt hơn trên các tập dữ liệu nhỏ.

### So sánh LSTM và GRU

LSTM	GRU
Có 3 cổng: Quên, Nạp, Xuất.	Có 2 cổng: Cập nhật, Reset.
Có trạng thái ô nhớ ( $C_t$ ) và trạng thái ẩn ( $h_t$ ) riêng biệt.	Chỉ có một trạng thái ẩn ( $h_t$ ).
Nhiều tham số hơn.	Ít tham số hơn.
Linh hoạt và mạnh mẽ hơn về mặt lý thuyết.	Nhanh hơn, hiệu quả hơn về mặt tính toán và cần ít dữ liệu hơn để tổng quát hóa.
Khi nào dùng? Không có quy tắc tuyệt đối. GRU là một lựa chọn khởi đầu tốt. Nếu bài toán rất phức tạp và có đủ dữ liệu, LSTM có thể cho hiệu năng nhỉnh hơn.	

## 3.4. Mô hình Chuỗi-sang-Chuỗi (Seq2Seq) và Cơ chế Chú ý (Attention)

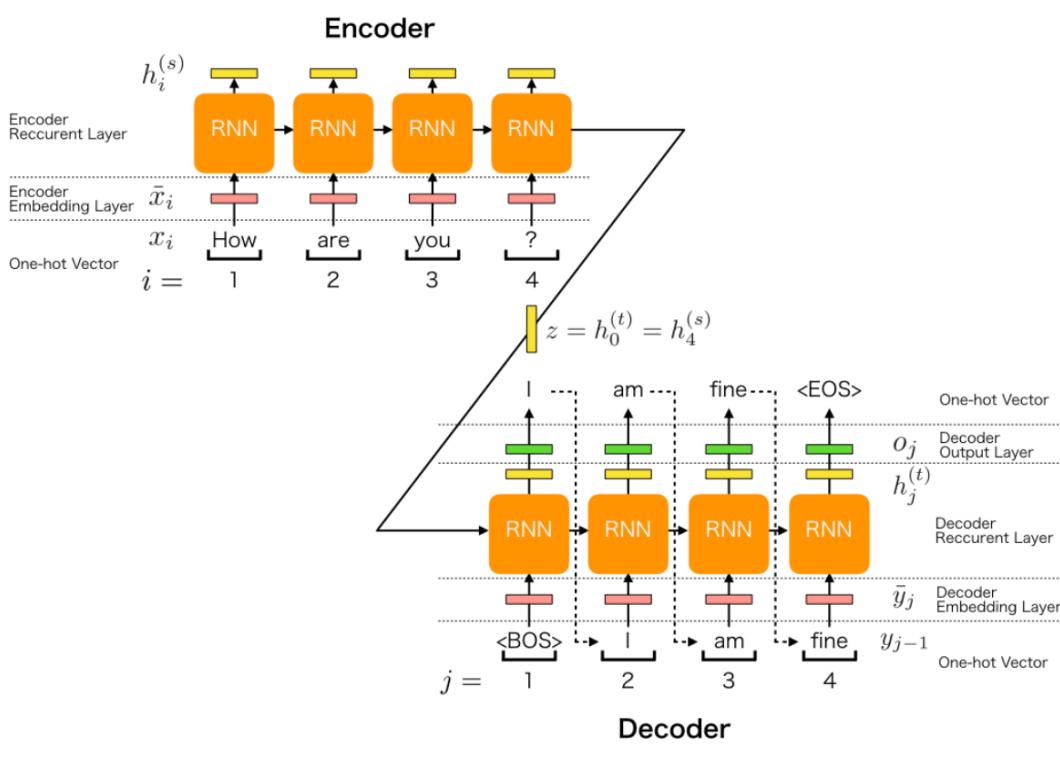
Kiến trúc RNN, LSTM, hay GRU mà chúng ta vừa học có một hạn chế cơ bản: chúng được thiết kế để xử lý các bài toán có đầu vào là một chuỗi và đầu ra là một giá trị duy nhất (phân loại) hoặc một chuỗi có độ dài bằng với chuỗi đầu vào (gán nhãn chuỗi).

Nhưng điều gì sẽ xảy ra với các bài toán như **dịch máy** (một câu tiếng Việt có 10 từ có thể được dịch thành một câu tiếng Anh 8 từ) hoặc **tóm tắt văn bản** (một văn bản 1000 từ được tóm tắt thành một câu 20 từ)? Ở những bài toán này, độ dài chuỗi đầu vào và đầu ra là khác nhau và không có sự tương ứng 1-1.

Mô hình Chuỗi-sang-Chuỗi (Sequence-to-Sequence - Seq2Seq) ra đời để giải quyết chính xác vấn đề này.

### 3.4.1. Kiến trúc Encoder-Decoder của Seq2Seq

Seq2Seq, được giới thiệu gần như đồng thời bởi Sutskever và các cộng sự (2014) [89] và Cho và các cộng sự (2014) [11], dựa trên một kiến trúc thanh lịch gồm hai thành phần chính: **Bộ mã hóa** (Encoder) và **Bộ giải mã** (Decoder).



Hình 3.6: Kiến trúc Encoder-Decoder tổng quát. Encoder ”đọc” và nén chuỗi đầu vào thành một vector ngữ cảnh. Decoder sử dụng vector này để ”viết” ra chuỗi đầu ra.

### Bộ mã hóa (The Encoder)

**Nhiệm vụ** Nhiệm vụ của Encoder là ”đọc” toàn bộ chuỗi đầu vào (ví dụ, một câu tiếng Việt) và nén toàn bộ thông tin, ngữ nghĩa của nó vào một vector có kích thước cố định. Vector này được gọi là **vector ngữ cảnh (context vector)** hay ”thought vector”.

### Cấu tạo và Hoạt động

- Encoder thường là một mạng RNN (hoặc LSTM, GRU).
- Nó nhận đầu vào là một chuỗi các vector từ (word embeddings) của câu nguồn, ví dụ:  $x_1, x_2, \dots, x_T$ .
- Nó xử lý chuỗi này tuần tự, từng từ một. Tại mỗi bước, nó cập nhật trạng thái ẩn của mình.
- Điều quan trọng là chúng ta bỏ qua tất cả các đầu ra (outputs) của Encoder ở mỗi bước. Chúng ta chỉ quan tâm đến trạng thái ẩn cuối cùng ( $h_T$ ). Trạng thái ẩn cuối cùng này chính là vector ngữ cảnh, vì nó được cho là đã tóm tắt thông tin của toàn bộ chuỗi đầu vào.

### Ghi chú sâu về Thiết kế của Encoder

#### 1. Tại sao Encoder thường là Mạng RNN Hai chiều (Bi-RNN)?

Một RNN đơn giản xử lý câu từ trái sang phải. Trạng thái ẩn  $h_t$  tại từ  $x_t$  chỉ chứa thông tin về các từ từ  $x_1$  đến  $x_t$ . Điều này là một hạn chế, vì ngữ nghĩa của một từ thường phụ thuộc vào cả ngữ cảnh đứng trước và đứng sau nó.

- Bi-RNN giải quyết vấn đề này bằng cách sử dụng hai RNN riêng biệt: một RNN chạy xuôi (forward) từ đầu đến cuối câu, và một RNN chạy ngược (backward) từ cuối về đầu.
- Tại mỗi từ  $x_t$ , biểu diễn cuối cùng của nó là **sự kết hợp** (thường là nối - concatenation) của trạng thái ẩn từ cả hai chiều:  $\overrightarrow{h}_t$  và  $\overleftarrow{h}_t$ .
- Vector ngữ cảnh cuối cùng cho mô hình Seq2Seq cơ bản sẽ là nối của trạng thái ẩn cuối cùng của RNN xuôi ( $\overrightarrow{h}_T$ ) và trạng thái ẩn cuối cùng của RNN ngược ( $\overleftarrow{h}_1$ ).

Việc này cung cấp một biểu diễn ngữ cảnh phong phú hơn nhiều cho mỗi từ, và đã trở thành tiêu chuẩn trong hầu hết các mô hình Encoder-Decoder.

#### 2. Tại sao không dùng Pooling thay vì trạng thái ẩn cuối cùng?

Sử dụng trạng thái ẩn cuối cùng là cách đơn giản nhất, nhưng nó có thể bị ảnh hưởng bởi "hiệu ứng gần đây" (recency bias) - các từ cuối chuỗi có thể có ảnh hưởng lớn hơn. Các phương pháp khác như:

- **Average/Max Pooling:** Lấy trung bình hoặc giá trị lớn nhất trên tất cả các trạng thái ẩn (của Bi-RNN). Cách này dân chủ hơn, đảm bảo mọi từ đều có đóng góp, nhưng có thể làm "lu mờ" các chi tiết quan trọng.

Trong thực tế, trước khi Attention ra đời, không có phương pháp nào là hoàn hảo. Sự ra đời của Attention đã giải quyết triệt để vấn đề này bằng cách cho phép Decoder tự quyết định nên tập trung vào đâu, thay vì ép Encoder phải tạo ra một bản tóm tắt duy nhất.

### Bộ giải mã (The Decoder)

**Nhiệm vụ** Nhiệm vụ của Decoder là lấy vector ngữ cảnh do Encoder cung cấp và "giải nén" nó để sinh ra chuỗi đầu ra (ví dụ, câu tiếng Anh tương ứng), từng từ một.

#### Cấu tạo và Hoạt động

- Decoder cũng là một mạng RNN (hoặc LSTM, GRU) khác, thường có kiến trúc tương tự Encoder nhưng với bộ trọng số riêng.
- **Khởi tạo:** Trạng thái ẩn ban đầu của Decoder ( $h_0^{dec}$ ) được khởi tạo bằng chính vector ngữ cảnh (trạng thái ẩn cuối cùng của Encoder,  $h_T^{enc}$ ). Đây là "cây cầu" duy nhất kết nối Encoder và Decoder.
- **Quá trình sinh từ (Generation):** Quá trình này diễn ra tuần tự:
  1. **Bước 1:** Decoder nhận vào một token bắt đầu đặc biệt (ví dụ: '<SOS>' - Start of Sentence) và trạng thái ẩn ban đầu. Nó tính toán trạng thái ẩn mới  $h_1^{dec}$  và tạo ra một phân phối xác suất trên toàn bộ từ vựng của ngôn ngữ đích thông qua một lớp softmax. Từ có xác suất cao nhất được chọn làm từ đầu ra đầu tiên,  $y_1$ .
  2. **Bước 2:** Từ vừa được sinh ra,  $y_1$ , sẽ trở thành **đầu vào cho bước tiếp theo**. Decoder nhận  $y_1$  và trạng thái ẩn  $h_1^{dec}$  để tính  $h_2^{dec}$  và sinh ra từ tiếp theo  $y_2$ .
  3. Quá trình này lặp đi lặp lại (mang tính **tự hồi quy** - auto-regressive) cho đến khi Decoder sinh ra một token kết thúc đặc biệt ('<EOS>' - End of Sentence).

#### Vấn đề của cái "Cổ chai" (The Bottleneck Problem)

Kiến trúc Encoder-Decoder cơ bản này rất thông minh, nhưng nó có một điểm yếu chết người: nó buộc Encoder phải nén toàn bộ thông tin của một câu, dù dài và phức tạp đến đâu, vào một **vector ngữ cảnh** có kích thước cố định.

- Đây là một "cổ chai" thông tin. Với những câu dài, việc nhồi nhét tất cả các chi tiết ngữ nghĩa vào một vector duy nhất là cực kỳ khó khăn, nếu không muốn nói là bất khả thi.
- Decoder chỉ được "nhìn" vào bản tóm tắt này một lần duy nhất lúc bắt đầu. Nó không có cách nào để quay lại và xem xét các phần cụ thể của câu nguồn khi nó đang dịch.

Hãy tưởng tượng bạn yêu cầu một dịch giả nghe toàn bộ một bài phát biểu dài, sau đó phải dịch lại toàn bộ từ trí nhớ mà không được ghi chép. Đó chính là những gì mô hình Seq2Seq cơ bản phải làm. Hạn chế này đã dẫn đến sự ra đời của ý tưởng có sức ảnh hưởng bậc nhất trong lịch sử NLP hiện đại: Cơ chế Chú ý (Attention).

### **Huấn luyện mô hình Seq2Seq**

Mục tiêu huấn luyện của mô hình Seq2Seq là tối đa hóa xác suất của chuỗi đầu ra đúng ( $Y = y_1, \dots, y_{T'}$ ) khi biết chuỗi đầu vào ( $X = x_1, \dots, x_T$ ). Điều này tương đương với việc tối thiểu hóa hàm mất mát Cross-Entropy trên toàn bộ chuỗi đầu ra:

$$\mathcal{L}(\theta) = - \sum_{t=1}^{T'} \log P(y_t | y_1, \dots, y_{t-1}, c)$$

trong đó  $c$  là vector ngữ cảnh từ Encoder. Trong quá trình huấn luyện, lỗi tại mỗi bước của Decoder được tính toán và lan truyền ngược qua cả Decoder và Encoder để cập nhật tất cả các trọng số. Kỹ thuật Teacher Forcing được sử dụng để làm cho quá trình này ổn định và hiệu quả hơn.

### **Huấn luyện và Suy luận: Teacher Forcing và Beam Search**

Có hai kỹ thuật quan trọng liên quan đến cách mô hình Seq2Seq được huấn luyện và sử dụng để sinh văn bản.

**Huấn luyện với Teacher Forcing** Trong quá trình huấn luyện, tại mỗi bước  $t$ , thay vì đưa từ mà Decoder vừa dự đoán ở bước  $t-1$  vào làm đầu vào, chúng ta lại đưa từ **chính xác** (ground truth) trong chuỗi đích vào.

- **Ưu điểm:** Kỹ thuật này giúp quá trình huấn luyện ổn định và hội tụ nhanh hơn rất nhiều. Nó ngăn ngừa việc lỗi bị tích tụ: nếu mô hình dự đoán sai ở bước đầu, lỗi sai đó sẽ không làm ảnh hưởng đến các bước sau trong cùng một chuỗi huấn luyện. Mỗi bước thời gian được coi là một mẫu huấn luyện độc lập.
- **Nhược điểm:** Nó tạo ra một sự khác biệt giữa lúc huấn luyện và lúc suy luận (khi mô hình không có "thầy" để chỉ bảo). Điều này có thể dẫn đến hiệu năng kém khi mô hình phải tự sinh ra một chuỗi dài.

**Suy luận với Beam Search** Trong lúc suy luận, việc luôn chọn từ có xác suất cao nhất tại mỗi bước (gọi là Greedy Search) thường không tạo ra được câu tốt nhất. Một lựa chọn có vẻ tốt ở hiện tại có thể dẫn đến một ngõ cụt. Beam Search là một giải pháp cân bằng.

- Thay vì chỉ giữ lại 1 ứng viên tốt nhất ở mỗi bước, Beam Search sẽ giữ lại  $k$  ứng viên (gọi là *beam width* hoặc *kích thước beam*) có xác suất chuỗi cao nhất.
- Tại bước tiếp theo, nó sẽ mở rộng mỗi ứng viên trong beam với tất cả các từ khả dĩ trong từ vựng, tính toán xác suất của các chuỗi mới, và lại chọn ra  $k$  chuỗi tốt nhất để tiếp tục.
- Quá trình này là một sự đánh đổi giữa chất lượng và chi phí tính toán. Nó hiệu quả hơn nhiều so với Greedy Search và là kỹ thuật tiêu chuẩn để sinh văn bản trong các mô hình Seq2Seq.

#### **3.4.2. Cơ chế Chú ý (Attention Mechanism)**

### Trực giác cốt lõi: Dịch như con người

Khi một dịch giả con người dịch một câu, họ không làm như mô hình Seq2Seq cơ bản. Khi dịch một từ, họ sẽ tập trung (pay attention) vào một hoặc một vài từ liên quan trong câu gốc. Cơ chế Chú ý được thiết kế để mô phỏng chính xác hành vi này.

#### Ý tưởng đột phá của Attention

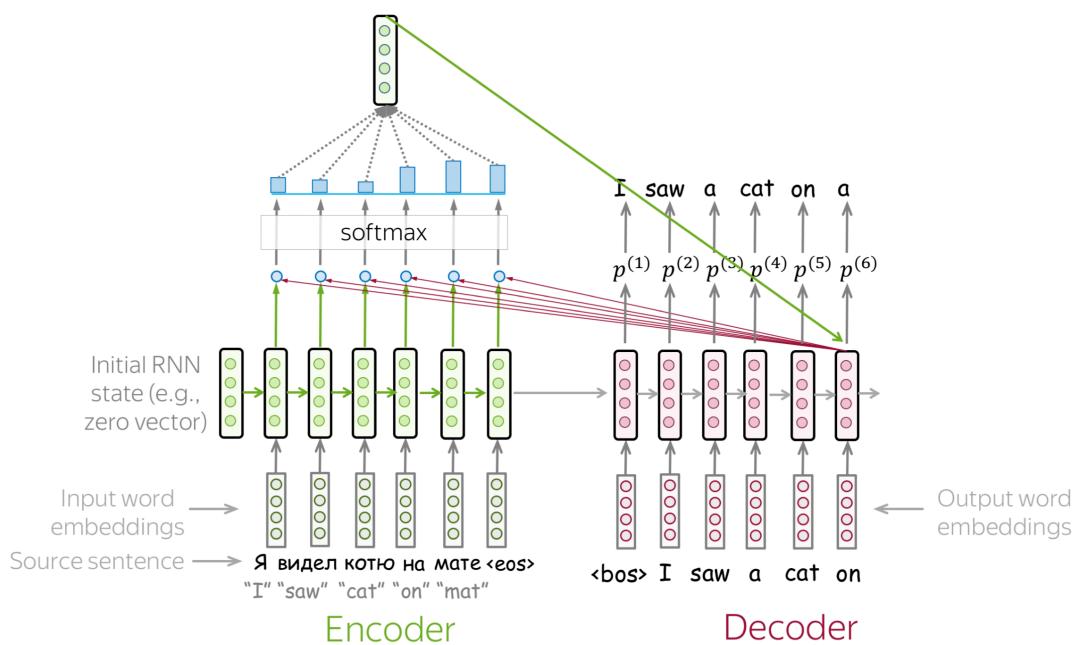
Thay vì ép Encoder tạo ra một vector ngữ cảnh duy nhất, hãy cho phép Decoder, tại **mỗi bước** sinh từ của mình, có khả năng **"nhìn lại"** toàn bộ chuỗi đầu vào và quyết định xem phần nào của chuỗi đầu vào là quan trọng nhất cần phải **"chú ý"** đến ngay tại thời điểm đó.

Attention tạo ra một **"đường tắt"** kết nối trực tiếp Decoder với tất cả các trạng thái ẩn của Encoder, phá vỡ cái **"cỗ chai"** thông tin.

### Cơ chế hoạt động chi tiết

Cơ chế chú ý được tích hợp vào giữa Encoder và Decoder. Thay vì chỉ truyền trạng thái ẩn cuối cùng, Encoder sẽ cung cấp **tất cả các trạng thái ẩn của nó** ( $h_1^{enc}, h_2^{enc}, \dots, h_T^{enc}$ ) cho Decoder.

Bây giờ, tại mỗi bước  $t$  của Decoder, để sinh ra từ  $y_t$ , nó sẽ thực hiện các bước sau:



Hình 3.7: Cơ chế Chú ý trong Seq2Seq. Tại bước sinh từ thứ  $t$  của Decoder, nó tính toán các trọng số chú ý ( $\alpha_t$ ) để tạo ra một vector ngữ cảnh động ( $c_t$ ) từ tất cả các trạng thái ẩn của Encoder.

**Bước 1: Tính điểm chú ý (Attention Scores)** Decoder lấy trạng thái ẩn của nó ở bước trước,  $h_{t-1}^{dec}$ , và so sánh nó với **từng trạng thái ẩn của Encoder**,  $h_i^{enc}$ , để tính ra một **"điểm số tương hợp"** (alignment score). Điểm số này đo lường mức độ **"liên quan"** của từ đầu vào thứ  $i$  đối với việc sinh ra từ đầu ra thứ  $t$ . Có nhiều cách để định nghĩa hàm score( $h_{t-1}^{dec}, h_i^{enc}$ ), trong đó phổ biến là:

- **Additive Attention ([2])**: Sử dụng một mạng nô-ron nhỏ truyền thẳng (feed-forward) có một lớp ẩn. Cách này mạnh mẽ và linh hoạt.

$$\text{score}(h_{t-1}^{dec}, h_i^{enc}) = v_a^T \tanh(W_a[h_{t-1}^{dec}; h_i^{enc}])$$

- **Dot-Product Attention ([53]):** Đơn giản là lấy tích vô hướng giữa hai vector. Cách này rất nhanh và hiệu quả về mặt tính toán, nhưng yêu cầu số chiều của  $h^{dec}$  và  $h^{enc}$  phải bằng nhau.

$$\text{score}(h_{t-1}^{dec}, h_i^{enc}) = (h_{t-1}^{dec})^T h_i^{enc}$$

- **Scaled Dot-Product Attention:** Một biến thể của Dot-Product, được giới thiệu trong bài báo Transformer [94]. Nó chia tích vô hướng cho căn bậc hai của số chiều vector ( $\sqrt{d_k}$ ) để tránh gradient quá nhỏ khi số chiều lớn.

Sự lựa chọn hàm score ảnh hưởng đến độ phức tạp tính toán và hiệu năng của mô hình. Dot-Product và các biến thể của nó đã trở nên rất phổ biến do hiệu quả của chúng, đặc biệt là trong kiến trúc Transformer.

**Bước 2: Chuẩn hóa thành trọng số (Softmax)** Các điểm số vừa tính được sẽ được đưa qua một hàm softmax. Điều này biến các điểm số thành một phân phối xác suất, gọi là **trọng số chú ý (attention weights)**, ký hiệu là  $\alpha_{ti}$ .

$$\alpha_{ti} = \frac{\exp(\text{score}(h_{t-1}^{dec}, h_i^{enc}))}{\sum_{j=1}^T \exp(\text{score}(h_{t-1}^{dec}, h_j^{enc}))}$$

Tổng của tất cả các trọng số  $\alpha_{ti}$  (với  $i$  từ 1 đến  $T$ ) bằng 1. Mỗi  $\alpha_{ti}$  cho biết "mức độ chú ý" mà Decoder nên dành cho từ đầu vào thứ  $i$  khi sinh ra từ đầu ra thứ  $t$ .

**Bước 3: Tính toán vector ngữ cảnh động (Context Vector)** Vector ngữ cảnh  $c_t$  bây giờ không còn cố định nữa. Nó là một **tổng có trọng số (weighted sum)** của tất cả các trạng thái ẩn của Encoder, với các trọng số chính là các trọng số chú ý vừa tính được.

$$c_t = \sum_{i=1}^T \alpha_{ti} h_i^{enc}$$

Vector ngữ cảnh  $c_t$  này được tính toán lại ở **mỗi bước của Decoder**. Nếu các trọng số chú ý  $\alpha_{ti}$  tập trung vào trạng thái ẩn thứ  $j$  của Encoder, thì vector  $c_t$  sẽ chủ yếu chứa thông tin từ  $h_j^{enc}$ .

**Bước 4: Sử dụng vector ngữ cảnh để dự đoán** Cuối cùng, vector ngữ cảnh động  $c_t$  được nối với trạng thái ẩn của Decoder ở bước trước,  $h_{t-1}^{dec}$ , và được đưa vào làm đầu vào cho lớp RNN của Decoder để tính ra trạng thái ẩn hiện tại  $h_t^{dec}$ .

$$h_t^{dec} = \text{RNN}_{\text{dec}}(h_{t-1}^{dec}, c_t)$$

Trạng thái  $h_t^{dec}$  này sau đó được dùng để dự đoán từ đầu ra  $y_t$ . Một cách khác là nối  $c_t$  với  $h_t^{dec}$  trước khi đưa vào lớp softmax cuối cùng.

### Sức mạnh của Attention

- **Phá vỡ nút thắt cổ chai:** Thông tin không còn bị nén vào một vector duy nhất. Decoder có quyền truy cập vào toàn bộ chuỗi đầu vào ở mọi thời điểm.
- **Khả năng diễn giải (Interpretability):** Chúng ta có thể trực quan hóa ma trận trọng số chú ý để xem mô hình đang "nhìn" vào đâu khi dịch. Ví dụ, khi dịch từ "bạn", mô hình sẽ có trọng số chú ý cao nhất ở từ "you". Điều này giúp gỡ lỗi và hiểu mô hình tốt hơn.
- **Hiệu năng vượt trội:** Các mô hình Seq2Seq với Attention đã ngay lập tức thiết lập một tiêu chuẩn mới về chất lượng cho dịch máy và nhiều bài toán chuỗi-sang-chuỗi khác, vượt xa các hệ thống thống kê trước đó.

Cơ chế Chú ý không chỉ là một cải tiến. Nó là một trong những ý tưởng nền tảng và có sức ảnh hưởng lớn nhất trong lịch sử học sâu, là tiền đề trực tiếp cho sự ra đời của kiến trúc Transformer mà chúng ta sẽ tìm hiểu ở chương tiếp theo.

### 3.5. Mạng Nơ-ron Tích chập (CNNs) cho NLP

Khi nhắc đến Mạng Nơ-ron Tích chập (Convolutional Neural Networks - CNNs), người ta thường nghĩ ngay đến các ứng dụng xử lý ảnh (Image Processing) như nhận dạng vật thể hay phân loại hình ảnh. Tuy nhiên, vào khoảng năm 2014, các nhà nghiên cứu như Yoon Kim [39] đã chứng minh rằng kiến trúc này, với một vài điều chỉnh nhỏ, lại có thể đạt được hiệu năng ấn tượng cho các bài toán phân loại văn bản (text classification).

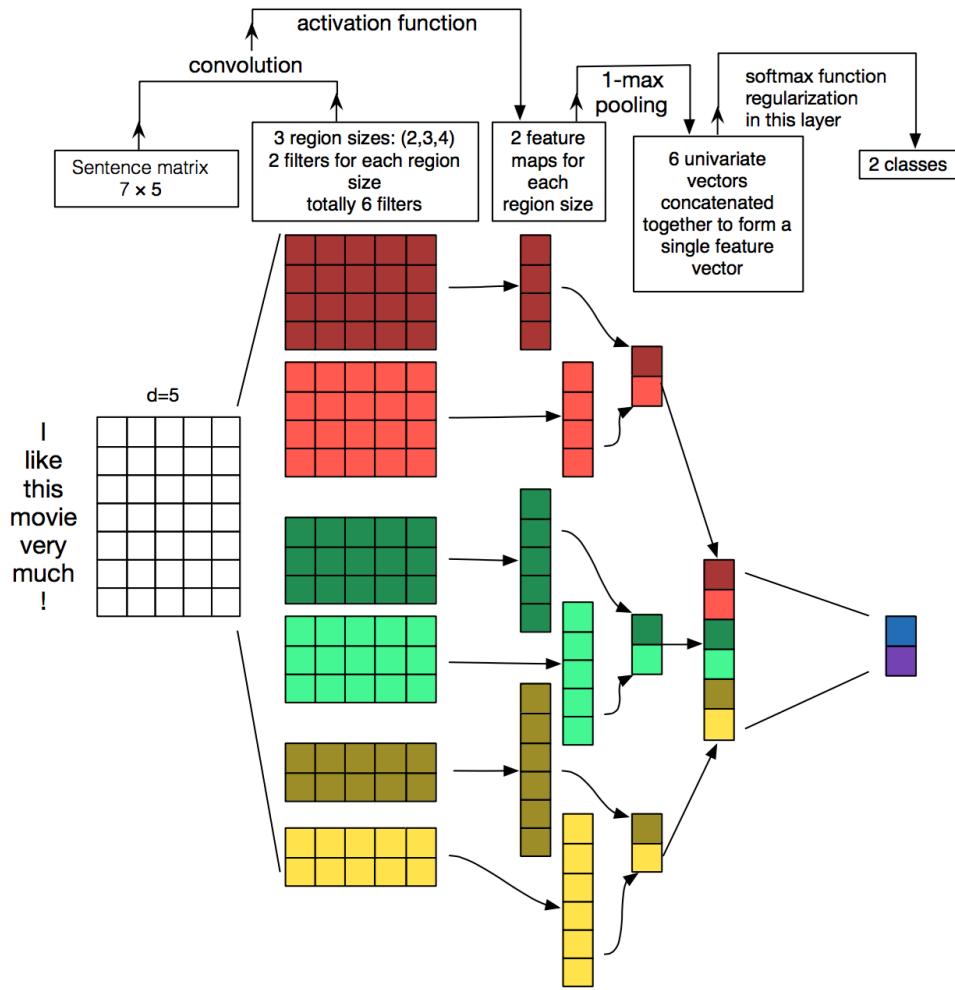
Vậy, làm thế nào một kiến trúc được thiết kế để "nhìn" các điểm ảnh 2D lại có thể "đọc" được văn bản 1D?

#### 3.5.1. Từ Ánh 2D đến Văn bản 1D: Một sự tương đồng

Hãy xem xét sự tương đồng giữa ảnh và văn bản:

- **Ảnh:** Là một ma trận các pixel 2D (chiều cao x chiều rộng). Một CNN sử dụng các **bộ lọc** (filters) hoặc **nhân** (kernels) 2D để trượt qua các vùng ảnh nhỏ, nhằm phát hiện các đặc trưng cục bộ như cạnh, góc, hay các hoa văn.
- **Văn bản:** Sau khi các từ được chuyển thành vector embedding, một câu có thể được xem như một "bức ảnh" 1D. Cụ thể, một câu gồm  $n$  từ, mỗi từ là một vector  $k$  chiều, có thể được biểu diễn bằng một ma trận có kích thước  $n \times k$ .

Dựa trên sự tương đồng này, chúng ta có thể áp dụng các phép tích chập (convolutions) lên văn bản. Tuy nhiên, thay vì các bộ lọc 2D trượt theo cả chiều ngang và dọc, trong NLP, chúng ta sử dụng các **bộ lọc 1D** chỉ trượt theo một chiều duy nhất: chiều dài của câu.



Hình 3.8: Kiến trúc CNN cho phân loại văn bản. Các bộ lọc có kích thước khác nhau (2, 3, 4 từ) trượt dọc theo câu để trích xuất các đặc trưng N-gram. Kết quả sau đó được gộp lại (max-pooling) và đưa vào một lớp FNN để phân loại.

### 3.5.2. Kiến trúc và Dòng chảy Dữ liệu

Một kiến trúc CNN điển hình cho bài toán phân loại văn bản bao gồm bốn thành phần chính:

#### 1. Lớp Nhúng (Embedding Layer)

Đây là bước đầu tiên, biến đổi câu đầu vào từ một chuỗi các chỉ số từ (word indices) thành một ma trận.

- **Đầu vào:** Một câu, ví dụ "NLP rất thú vị và mạnh mẽ".
- **Đầu ra:** Một ma trận có kích thước (**số từ**, chiều\_embedding). Ví dụ, (6, 300).

#### 2. Lớp Tích chập (Convolutional Layer)

Đây là trái tim của mô hình, nơi các đặc trưng được trích xuất.

**Trục giác cốt lõi** Trong NLP, một bộ lọc 1D có thể được xem như một **bộ phát hiện N-gram** (N-gram detector). Một bộ lọc có chiều cao là 2 từ ('height=2') sẽ học cách phát hiện các mẫu bigram

quan trọng. Một bộ lọc có chiều cao là 3 ('height=3') sẽ học cách phát hiện các mẫu trigram quan trọng.

### Hoạt động

- Bộ lọc (Filter):** Một bộ lọc là một ma trận trọng số nhỏ, có kích thước (**kích\_thước\_N-gram, chiều\_embedding**). Ví dụ, một bộ lọc trigram sẽ có kích thước (3, 300).
- Phép tích chập:** Bộ lọc này sẽ **truột (slides)** dọc theo ma trận embedding của câu, từ trên xuống dưới, mỗi lần một bước. Tại mỗi vị trí, nó thực hiện một phép tích vô hướng (dot product) giữa các trọng số của nó và vùng ma trận embedding mà nó bao phủ. Kết quả của mỗi phép tích là một con số duy nhất.
- Bản đồ Đặc trưng (Feature Map):** Sau khi truột hết câu, bộ lọc sẽ tạo ra một chuỗi các con số, gọi là **một bản đồ đặc trưng**. Chuỗi này biểu diễn sự hiện diện của mẫu N-gram mà bộ lọc đó được "chuyên môn hóa" để phát hiện tại các vị trí khác nhau trong câu.

Thông thường, chúng ta không chỉ dùng một bộ lọc. Chúng ta sử dụng **nhiều bộ lọc** cho mỗi kích thước N-gram (ví dụ: 100 bộ lọc cho bigram, 100 bộ lọc cho trigram, 100 bộ lọc cho 4-gram). Mỗi bộ lọc sẽ tự học để phát hiện một loại mẫu N-gram khác nhau (ví dụ: một bộ lọc bigram có thể chuyên phát hiện các mẫu "rất + [tính từ tích cực]", một bộ lọc khác chuyên phát hiện "không + [động từ]").

#### Ghi chú sâu về Lớp Nhúng và Tích chập

##### 1. Fine-tuning Embeddings và Multiple Channels

Một quyết định quan trọng là có nên cập nhật các vector embedding trong quá trình huấn luyện hay không.

- Static (Tĩnh):** Sử dụng embedding đã được huấn luyện trước (pre-trained) và giữ cố định. Đây là lựa chọn tốt khi tập dữ liệu cho tác vụ hiện tại nhỏ, để tránh overfitting.
- Non-static (Động):** Cho phép mô hình "tinh chỉnh" (fine-tune) các embedding. Điều này giúp các vector từ học được những sắc thái ngữ nghĩa đặc thù cho tác vụ đang giải quyết. Ví dụ, trong bài toán phân tích cảm xúc, embedding của từ "tuyệt vời" có thể được đẩy ra xa hơn embedding của từ "tốt".
- Multiple Channels:** Một kỹ thuật mạnh mẽ là sử dụng nhiều "kênh" đầu vào. Ví dụ, một kênh là embedding tĩnh, kênh kia là embedding động. Các bộ lọc tích chập sẽ học cách trích xuất đặc trưng từ cả hai kênh này. Ý tưởng này tương tự như các kênh màu (RGB) trong ảnh, cho phép mô hình tiếp cận dữ liệu từ nhiều "góc nhìn" khác nhau.

##### 2. Tại sao chiều rộng của bộ lọc bằng chiều embedding?

Bộ lọc trong CNN cho NLP được thiết kế để có chiều rộng bằng với chiều của embedding (ví dụ,  $3 \times 300$  cho một trigram detector). Điều này là có chủ ý. Mục đích là để bộ lọc xem xét **toàn bộ thông tin ngữ nghĩa** của một từ tại một thời điểm, chứ không phải chỉ một vài chiều riêng lẻ. Phép tích chập sau đó sẽ kết hợp thông tin từ tất cả các chiều của các từ trong cửa sổ N-gram để tạo ra một giá trị đặc trưng duy nhất. Nó không truột theo chiều embedding, mà chỉ truột theo chiều dài của câu.

### 3. Lớp Gộp (Pooling Layer)

Sau lớp tích chập, chúng ta có rất nhiều bản đồ đặc trưng, mỗi cái là một vector dài. Lớp gộp có nhiệm vụ giảm chiều dữ liệu và tóm tắt thông tin quan trọng nhất từ mỗi bản đồ đặc trưng.

**Trực giác cốt lõi** Lớp gộp trả lời câu hỏi: "Đối với mẫu N-gram mà bộ lọc này phát hiện, liệu nó có xuất hiện một cách **nổi bật** ở đâu đó trong câu không?". Chúng ta không quan tâm nó xuất hiện ở đâu, chỉ cần biết nó có xuất hiện hay không.

**Hoạt động: Max-over-time Pooling** Kỹ thuật gộp phổ biến nhất trong NLP là **Max-over-time Pooling** (gọi tắt là Max-Pooling).

- Đối với mỗi bản đồ đặc trưng (vector), nó chỉ đơn giản là lấy ra **giá trị lớn nhất (maximum value)**.
- Giá trị lớn nhất này đại diện cho tín hiệu mạnh nhất của mẫu N-gram tương ứng trong toàn bộ câu.
- Sau bước này, mỗi bộ lọc chỉ đóng góp một con số duy nhất. Nếu chúng ta có 100 bộ lọc bigram, 100 bộ lọc trigram, và 100 bộ lọc 4-gram, kết quả của lớp gộp sẽ là một vector 300 chiều.

**Tại sao Max-Pooling lại hiệu quả?** Trực giác đầu sau sự thành công của Max-pooling trong phân loại văn bản là nó hoạt động như một cơ chế phát hiện đặc trưng. Mỗi bản đồ đặc trưng là đầu ra của một "bộ phát hiện N-gram" cụ thể. Bằng cách lấy giá trị lớn nhất, chúng ta đang trả lời câu hỏi: "Tín hiệu của N-gram quan trọng này có xuất hiện mạnh mẽ nhất ở đâu đó trong câu không?".

- Nó tạo ra sự **bất biến về vị trí (positional invariance)**. Mô hình không quan tâm cụm từ "rất tệ" xuất hiện ở đầu hay cuối câu, miễn là nó xuất hiện và tạo ra một tín hiệu mạnh.
- Nó tập trung vào các tín hiệu **nổi bật nhất**, bỏ qua các tín hiệu nhiễu hoặc yếu hơn. Trong phân loại cảm xúc, chỉ cần một cụm từ tiêu cực mạnh là đủ để quyết định nhãn của cả câu.
- Average-pooling**, ngược lại, sẽ làm "trung hòa" tín hiệu. Nếu một câu có một cụm từ rất tích cực và nhiều cụm từ trung tính, giá trị trung bình sẽ bị kéo xuống, làm mất đi đặc trưng quan trọng nhất.

#### 4. Lớp Kết nối Đầu đú, Phân loại, và Huấn luyện

Vector đặc trưng cuối cùng (ví dụ, 300 chiều) từ lớp gộp giờ đây đã nắm bắt được những thông tin N-gram quan trọng nhất của câu. Vector này được đưa vào một mạng nô-ron truyền thẳng (FNN) thông thường, thường có một hoặc hai lớp ẩn và một lớp softmax ở cuối để đưa ra dự đoán phân loại cuối cùng.

Quá trình huấn luyện của toàn bộ mạng được thực hiện end-to-end. Hàm mất mát (thường là Cross-Entropy) được tính toán dựa trên dự đoán của lớp softmax và nhãn thật. Sau đó, gradient của lỗi sẽ được lan truyền ngược (**backpropagation**) qua toàn bộ mạng, từ lớp FNN, qua lớp gộp, đến lớp tích chập để cập nhật trọng số của các bộ lọc, và có thể cập nhật cả lớp embedding (nếu ở chế độ non-static).

### 3.5.3. Ưu và Nhược điểm của CNN cho NLP

#### Đánh giá CNN cho NLP

##### Ưu điểm:

- Cực kỳ hiệu quả tính toán:** Phép tích chập có thể được song song hóa ở mức độ cao, làm cho CNN nhanh hơn đáng kể so với RNN/LSTM trong quá trình huấn luyện.
- Trích xuất đặc trưng cục bộ tốt:** CNN rất giỏi trong việc phát hiện các mẫu N-gram quan trọng, bắt kể chúng xuất hiện ở đâu trong câu (nhờ vào lớp gộp). Điều này rất hữu ích cho các bài toán mà sự hiện diện của các cụm từ khóa là yếu tố quyết định, như phân tích cảm xúc hay phân loại chủ đề.
- Kiến trúc đơn giản:** So với LSTM, kiến trúc CNN dễ hiểu và triển khai hơn.

##### Nhược điểm:

- Không nhạy cảm với trật tự từ tầm xa:** Do kích thước của các bộ lọc bị giới hạn, CNN chỉ có thể nắm bắt các phụ thuộc cục bộ (trong phạm vi N-gram). Nó không thể hiểu các mối quan hệ ngữ pháp phức tạp giữa các từ ở hai đầu của một câu dài như cách LSTM có thể làm.
- Không phù hợp cho các tác vụ sinh ngôn ngữ (NLG):** Bản chất của CNN là trích xuất đặc trưng để phân loại, không phải để sinh ra một chuỗi mới.

### 3.5.4. Các Kiến trúc CNN Nâng cao và Ứng dụng

Kiến trúc CNN cơ bản đã rất mạnh mẽ, nhưng có nhiều biến thể nâng cao đã được phát triển để giải quyết các vấn đề phức tạp hơn:

**Mạng CNN Đa lớp (Stacked/Multi-layer CNNs)** Giống như trong xử lý ảnh, chúng ta có thể xếp chồng nhiều lớp tích chập và gộp lên nhau.

- Ý tưởng:** Lớp CNN đầu tiên học các đặc trưng N-gram từ văn bản thô. Lớp CNN thứ hai sẽ thực hiện tích chập trên đầu ra của lớp đầu tiên, do đó nó học được các **tổ hợp** của các N-gram (ví dụ, một quy luật về sự xuất hiện của một bigram tích cực sau một trigram tiêu cực).
- Ứng dụng:** Cho phép mô hình học các đặc trưng ngữ nghĩa ở mức độ trừu tượng cao hơn, hữu ích cho các bài toán phân loại phức tạp.

**Mạng CNN ở Cấp độ Ký tự (Character-level CNNs)** Thay vì áp dụng tích chập trên các từ, chúng ta có thể áp dụng nó trên các ký tự.

- Cơ chế:** Một từ được biểu diễn như một ma trận (`số_ký_tự`, `chiều_embedding_ký_tự`). Một CNN sẽ trượt qua ma trận này để tạo ra một vector biểu diễn cho từ đó. Vector này sau đó có thể được đưa vào một RNN/LSTM ở cấp độ từ.
- Ưu điểm:**
  - Xử lý từ OOV:** Mô hình có thể tạo ra biểu diễn cho bất kỳ từ nào, kể cả từ chưa từng thấy, dựa trên các ký tự cấu thành nó.
  - Hiểu hình thái học:** Nó có thể nhận ra các tiền tố, hậu tố, và gốc từ (ví dụ, "running", "ran", "runner" đều chứa "run"), giúp mô hình hiểu được mối quan hệ giữa các từ cùng họ.

Những kiến trúc nâng cao này cho thấy sự linh hoạt và sức mạnh của phép tích chập như một công cụ trích xuất đặc trưng hiệu quả trong NLP.

### 3.5.5. So sánh CNN và RNN cho NLP

CNN và RNN tiếp cận bài toán xử lý văn bản từ hai góc độ hoàn toàn khác nhau:

- RNN/LSTM xem văn bản như một chuỗi tuần tự, xử lý từng bước một và xây dựng một "trí nhớ" về quá khứ. Nó rất mạnh trong việc mô hình hóa các phụ thuộc tầm xa và các tác vụ yêu cầu hiểu sâu về cấu trúc câu.
- CNN xem văn bản như một "túi các N-gram", sử dụng các bộ lọc để tìm kiếm các mẫu cục bộ quan trọng. Nó rất mạnh trong việc trích xuất các đặc trưng phân loại hiệu quả và nhanh chóng.

Trong thực tế, người ta cũng thường kết hợp cả hai kiến trúc này (ví dụ, sử dụng CNN để trích xuất đặc trưng từ ký tự, sau đó đưa vào một LSTM để xử lý ở cấp độ từ) để tận dụng thế mạnh của cả hai.

### 3.6. Contextualized Word Embeddings: Biểu diễn từ theo ngữ cảnh

Các Word Embeddings truyền thống tạo vector cố định cho mỗi từ, bất kể ngữ cảnh. Ví dụ, từ "bank" trong "river bank" và "investment bank" đều có cùng một vector. Điều này gây ra hạn chế lớn: không thể xử lý đa nghĩa (polysemy).

#### Ý tưởng cơ bản

Contextualized Embeddings giải quyết vấn đề này bằng cách biểu diễn mỗi từ dựa trên ngữ cảnh xung quanh. Vector của từ sẽ thay đổi tùy thuộc vào câu mà nó xuất hiện.

#### 3.6.1. ELMo: Bước đệm tới Embeddings theo ngữ cảnh

**Vấn đề của embedding tĩnh** Các mô hình như Word2Vec, GloVe, FastText gán cho mỗi từ một vector *tĩnh*, không đổi theo ngữ cảnh. Do đó, từ đa nghĩa (*polysemy*) như *bank* sẽ có cùng một vector trong *river bank* và *investment bank*. Điều này giới hạn khả năng biểu diễn ý nghĩa phụ thuộc ngữ cảnh.

**Ý tưởng cốt lõi của ELMo** ELMo (Embeddings from Language Models, 2018) [65] tạo *biểu diễn phụ thuộc ngữ cảnh* cho mỗi từ bằng cách khai thác **các trạng thái ẩn** của một *mô hình ngôn ngữ hai chiều sâu* (biLM). Với mỗi vị trí trong câu, ELMo kết hợp có trọng số các tầng biểu diễn (từ ký tự đến các tầng LSTM) để sinh ra một vector thích nghi với ngữ cảnh cụ thể.

#### Kiến trúc và Pipeline

ELMo gồm ba khối chính:

1. **Biểu diễn cấp ký tự (Char-CNN).** Cho một từ  $w$ , tách thành chuỗi ký tự  $(c_1, \dots, c_{|w|})$ , ánh xạ qua embedding ký tự  $\mathbf{E}_c \in \mathbb{R}^{|\mathcal{C}| \times d_c}$ , thu được ma trận  $\mathbf{X}_w \in \mathbb{R}^{|w| \times d_c}$ . Áp dụng các bộ lọc chập 1D  $\{\mathbf{F}^{(k)}\}_{k=1}^K$  với các kích thước kernel khác nhau (ví dụ 1-7), sau đó *max-over-time pooling* và (thuồng) *Highway layers*:

$$\mathbf{z}_w^{(k)} = \text{pool}(\text{conv1d}(\mathbf{X}_w; \mathbf{F}^{(k)})), \quad \mathbf{x}_w^{(0)} = \text{Highway}([\mathbf{z}_w^{(1)}; \dots; \mathbf{z}_w^{(K)}]).$$

Vector  $\mathbf{x}_w^{(0)}$  là biểu diễn cấp ký tự, giúp xử lý tốt từ hiếm/OOV và thông tin hình thái.

2. **BiLSTM Language Model nhiều tầng.** Cho câu  $\mathbf{w}_{1:T} = (w_1, \dots, w_T)$ , với đầu vào tại vị trí  $t$  là  $\mathbf{x}_t^{(0)}$ :

$$\overrightarrow{\mathbf{h}}_t^{(j)} = \text{LSTM}_{\text{fwd}}^{(j)}(\overrightarrow{\mathbf{h}}_{t-1}^{(j)}, \mathbf{h}_t^{(j-1)}), \quad \overleftarrow{\mathbf{h}}_t^{(j)} = \text{LSTM}_{\text{bwd}}^{(j)}(\overleftarrow{\mathbf{h}}_{t+1}^{(j)}, \mathbf{h}_t^{(j-1)}),$$

với  $j = 1, \dots, L$  là số tầng (thường  $L = 2$ ), và  $\mathbf{h}_t^{(0)} \equiv \mathbf{x}_t^{(0)}$ . Ghép hai hướng:

$$\mathbf{h}_t^{(j)} = [\overrightarrow{\mathbf{h}}_t^{(j)}; \overleftarrow{\mathbf{h}}_t^{(j)}].$$

Mỗi  $\mathbf{h}_t^{(j)}$  mã hoá ngữ cảnh quanh vị trí  $t$  theo hai chiều.

3. **Kết hợp tầng có trọng số (task-specific).** ELMo không chỉ dùng tầng cuối. Thay vào đó, với một tác vụ downstream, học các trọng số tầng  $\{s_j\}_{j=0}^L$  (qua softmax) và một hệ số co giãn  $\gamma > 0$  để kết hợp:

$$\text{ELMo}_t = \gamma \sum_{j=0}^L s_j \mathbf{h}_t^{(j)}, \quad s_j = \frac{\exp(a_j)}{\sum_{k=0}^L \exp(a_k)}.$$

Ở đây  $\mathbf{h}_t^{(0)} \equiv \mathbf{x}_t^{(0)}$  (đầu vào từ Char-CNN). Các tham số  $\{a_j\}$  và  $\gamma$  được *học riêng cho từng tác vụ* trong giai đoạn fine-tuning (trong khi tham số biLM có thể được giữ cố định).

Tóm tắt pipeline:

Văn bản  $\rightarrow$  Char-CNN  $\rightarrow$  BiLSTM (fwd/bwd, nhiều tầng)  $\rightarrow \{\mathbf{h}_t^{(j)}\}_{j=0}^L \rightarrow$  Weighted Sum +  $\gamma \rightarrow \text{ELMo}_t$ .

### Mục tiêu huấn luyện (biLM Objective)

ELMo dựa trên một *mô hình ngôn ngữ hai chiều* được huấn luyện trước trên corpora lớn. Với câu  $\mathbf{w}_{1:T}$ , **forward LM** mô hình hoá:

$$p_{\rightarrow}(\mathbf{w}) = \prod_{t=1}^T p(w_t | w_{1:t-1}),$$

và **backward LM** mô hình hoá:

$$p_{\leftarrow}(\mathbf{w}) = \prod_{t=1}^T p(w_t | w_{t+1:T}).$$

Hàm mục tiêu tổng là *tối đa hoá log-likelihood* của cả hai hướng:

$$\mathcal{L}(\Theta) = \sum_{\mathbf{w} \in \mathcal{D}} \sum_{t=1}^T \left[ \log p_{\rightarrow}(w_t | w_{1:t-1}) + \log p_{\leftarrow}(w_t | w_{t+1:T}) \right],$$

trong đó  $\Theta$  là toàn bộ tham số của Char-CNN, BiLSTM và các đầu ra dự đoán từ vựng (ví dụ qua softmax trên vocab). Sau khi pretrain, ta có thể:

- *Đóng băng*  $\Theta$  và chỉ học  $\{a_j\}, \gamma$  cho tác vụ downstream; hoặc
- *Fine-tune* có kiểm soát một phần tham số (tùy tài nguyên và quy mô dữ liệu).

**Ước lượng phân phối từ tiếp theo.** Tại mỗi hướng, phân phối  $p(w_t | \cdot)$  thường được hiện thực bằng một tầng tuyến tính + softmax trên từ vựng:

$$p(w_t = v | \cdot) = \frac{\exp(\mathbf{u}_v^\top \mathbf{h}_t^{(L)} + b_v)}{\sum_{v' \in \mathcal{V}} \exp(\mathbf{u}_{v'}^\top \mathbf{h}_t^{(L)} + b_{v'})},$$

với  $\mathbf{u}_v$  là vector hàng của ma trận chiếu sang không gian từ vựng,  $\mathcal{V}$  là từ vựng.

### Suy diễn (Inference) và sử dụng trong tác vụ downstream

Cho câu đầu vào, ta tính  $\{\mathbf{h}_t^{(j)}\}_{j=0}^L$  như §3.6.1, sau đó kết hợp:

$$\text{ELMo}_t = \gamma \sum_{j=0}^L s_j \mathbf{h}_t^{(j)}.$$

Vector  $\text{ELMo}_t$  được *nối* vào các đặc trưng đầu vào của mô hình tác vụ (ví dụ: CRF cho NER, BiLSTM/Transformer cho POS, phân loại câu, QA, ...). Vì  $s_j$  và  $\gamma$  được học *cho từng tác vụ*, mô hình có thể “chọn” tầng biểu diễn phù hợp (tầng thấp thường giàu thông tin cú pháp; tầng cao giàu thông tin ngữ nghĩa).

### Tại sao ELMo hiệu quả?

- **Phụ thuộc ngữ cảnh hai chiều:** mỗi vị trí nhận thông tin từ cả lịch sử (trái) và tương lai (phải) trong câu.
- **Kết hợp đa tầng:** thay vì chỉ dùng tầng cuối, ELMo học trọng số để kết hợp cả *đầu vào ký tự* và các *tầng ẩn* – linh hoạt theo từng tác vụ.
- **Xử lý OOV/hình thái:** Char-CNN cung cấp biểu diễn vững cho từ hiếm, từ mới, biến thể hình thái.

### Hạn chế và so sánh với Transformer

- **LSTM khó song song hoá:** suy diễn/chạy huấn luyện chậm hơn kiến trúc self-attention.
- **Quan hệ xa (long-range) hạn chế:** LSTM dựa vào trạng thái tuần tự; Transformer học phụ thuộc xa hiệu quả hơn qua self-attention.
- **BERT và hậu duệ:** BERT thay biLM bằng *masked language modeling* trên *Transformer encoder*, vẫn cho ra embedding ngữ cảnh hoá nhưng hiệu quả và mở rộng tốt hơn.

### Gợi ý thực hành

- **Khi dùng ELMo như đặc trưng bổ sung:** nối  $\text{ELMo}_t$  vào embedding từ (hoặc đầu vào mô hình hiện có). Giữ nguyên tham số biLM và học  $\{a_j\}$ ,  $\gamma$  thường đã đủ mạnh khi dữ liệu tác vụ nhỏ.
- **Điều chỉnh trọng số tầng:** theo dõi  $s_j$  sau khi học để phân tích tác vụ ưu tiên cú pháp hay ngữ nghĩa (tầng thấp hay cao).
- **Tối ưu hoá tài nguyên:** nếu tài nguyên hạn chế, đóng băng toàn bộ biLM; nếu đủ tài nguyên, thử fine-tune một phần (ví dụ chỉ Char-CNN) để thích nghi miền dữ liệu.

### Ví dụ minh họa (giản lược)

Câu	Vector ELMo cho “bank” (rút gọn)
<i>She sat by the river bank.</i>	[0.45, 0.12, -0.20, ...]
<i>He works at an investment bank.</i>	[0.05, -0.55, 0.88, ...]

Sự khác biệt đến từ các trạng thái ẩn  $\{\mathbf{h}_t^{(j)}\}$  do ngữ cảnh khác nhau, sau khi được kết hợp bởi trọng số  $\{s_j\}$  và hệ số  $\gamma$ .

### Tóm tắt công thức trọng tâm

$$(1) \text{ Char-CNN: } \mathbf{x}_t^{(0)} = \text{Highway}\left(\left[\text{pool}(\text{conv1d}(\mathbf{X}_{w_t}; \mathbf{F}^{(k)}))\right]_{k=1}^K\right).$$

$$(2) \text{ BiLSTM tầng } j \in \{1, \dots, L\}: \quad \mathbf{h}_t^{(j)} = [\overrightarrow{\mathbf{h}}_t^{(j)}; \overleftarrow{\mathbf{h}}_t^{(j)}].$$

$$(3) \text{ biLM objective: } \mathcal{L} = \sum_{t=1}^T \log p_{\rightarrow}(w_t | w_{1:t-1}) + \sum_{t=1}^T \log p_{\leftarrow}(w_t | w_{t+1:T}).$$

$$(4) \text{ Kết hợp tầng: } \text{ELMo}_t = \gamma \sum_{j=0}^L s_j \mathbf{h}_t^{(j)}, \quad s_j = \frac{\exp(a_j)}{\sum_{k=0}^L \exp(a_k)}, \quad \gamma > 0.$$

**Kết luận.** ELMo đưa biểu diễn từ vượt khỏi giới hạn tĩnh bằng cách *ngữ cảnh hoá hai chiều* và *kết hợp đa tầng có trọng số*. Nó mở đường cho ký nguyên embedding theo ngữ cảnh dựa trên Transformer (BERT và các họ mô hình sau này), nơi khả năng song song hoá và học phụ thuộc xa được cải thiện đáng kể.

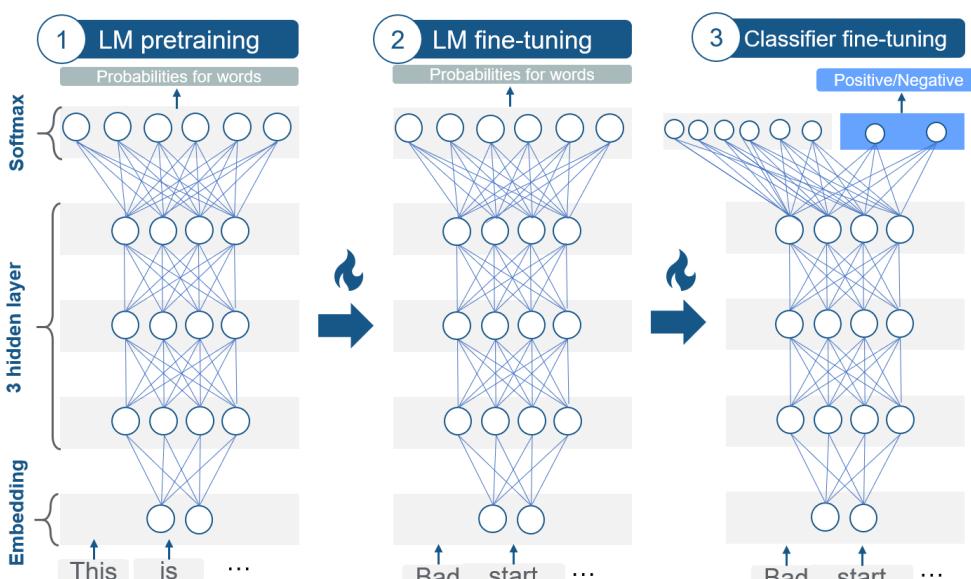
### 3.6.2. ULMFiT: Công thức Vàng cho Học Chuyển giao trong NLP

Trước khi các mô hình Transformer khổng lồ thống trị, một trong những thách thức lớn nhất của NLP là làm sao để áp dụng hiệu quả học chuyển giao (transfer learning) như lĩnh vực Thị giác Máy tính đã làm với ImageNet. Các nỗ lực trước đó thường chỉ chuyển giao các word embeddings tĩnh (Word2Vec, GloVe), trong khi toàn bộ phần còn lại của mô hình phải được huấn luyện lại từ đầu cho mỗi tác vụ.

ULMFiT (Universal Language Model Fine-tuning), được giới thiệu bởi Jeremy Howard và Sebastian Ruder (fast.ai) vào năm 2018 [34], đã tạo ra một cách mới bằng cách đưa ra một **khung phương pháp (framework)** hoàn chỉnh và cực kỳ hiệu quả để tinh chỉnh (fine-tune) một mô hình ngôn ngữ cho các tác vụ phân loại văn bản downstream. ULMFiT đã chứng minh rằng, với các kỹ thuật phù hợp, chúng ta có thể đạt được kết quả SOTA (state-of-the-art) trên nhiều bộ dữ liệu chỉ với khoảng 100 mẫu được gán nhãn.

**Tư duy cốt lõi: Quy trình 3 bước** Ý tưởng của ULMFiT không nằm ở một kiến trúc mạng nơ-ron mới, mà là ở một **quy trình (recipe)** gồm 3 bước, được thiết kế cẩn thận để bảo toàn kiến thức đã học và tránh "sự quên lãng thảm khốc"(catastrophic forgetting).

1. **Huấn luyện trước mô hình ngôn ngữ tổng quát (General-domain LM Pre-training):** Huấn luyện một mô hình ngôn ngữ (cụ thể là AWD-LSTM) trên một kho văn bản lớn và đa dạng (ví dụ: WikiText-103). Mục tiêu của bước này là để mô hình học được các đặc trưng ngôn ngữ phổ quát: ngữ pháp, mối quan hệ ngữ nghĩa, và thậm chí một lượng kiến thức nền về thế giới. Đây là bước tốn kém nhất và thường chỉ cần làm một lần.
2. **Tinh chỉnh mô hình ngôn ngữ trên miền đích (Target-domain LM Fine-tuning):** Đây là bước trung gian quan trọng mà các phương pháp trước đó thường bỏ qua. Trước khi giải quyết tác vụ cuối cùng (ví dụ: phân loại bình luận phim), mô hình ngôn ngữ tổng quát sẽ được tiếp tục huấn luyện trên dữ liệu của chính tác vụ đó (nhưng không dùng nhãn). Bước này giúp mô hình thích nghi với các đặc điểm, thuật ngữ, và phong cách của miền dữ liệu cụ thể (ví dụ: ngôn ngữ "teen code" trong các bình luận phim).
3. **Tinh chỉnh mô hình phân loại cho tác vụ cuối (Target Task Classifier Fine-tuning):** Cuối cùng, lớp đầu ra (output layer) của mô hình ngôn ngữ được thay thế bằng một "đầu phân loại"(classification head) phù hợp với tác vụ. Toàn bộ mô hình sau đó được tinh chỉnh trên dữ liệu có nhãn của tác vụ, nhưng không phải theo cách thông thường, mà bằng các kỹ thuật đặc biệt để tối ưu hóa quá trình học chuyển giao.



Hình 3.9: Quy trình 3 bước của ULMFiT: Pre-training trên dữ liệu chung, Fine-tuning LM trên dữ liệu miền đích, và Fine-tuning bộ phân loại cho tác vụ cuối cùng.

### Các kỹ thuật Tinh chỉnh then chốt

Sự thành công của ULMFiT không chỉ đến từ quy trình 3 bước mà còn từ 3 kỹ thuật tinh chỉnh thông minh được áp dụng trong bước cuối cùng.

**1. Tinh chỉnh phân biệt (Discriminative Fine-tuning)** Các tầng khác nhau của một mạng nơ-ron sâu nắm bắt các loại thông tin khác nhau: tầng đầu tiên học các đặc trưng rất chung chung, trong khi các tầng sau học các đặc trưng ngày càng cụ thể và phức tạp. Do đó, việc tinh chỉnh tất cả các tầng với cùng một tốc độ học (learning rate) là không hợp lý. ULMFiT đề xuất sử dụng các tốc độ học khác nhau cho mỗi tầng.

Giả sử mô hình có  $L$  tầng, ta chọn một tốc độ học cơ sở  $\eta_L$  cho tầng cuối cùng. Tốc độ học cho các tầng thấp hơn sẽ được tính bằng cách chia cho một hằng số (ví dụ: 2.6).

$$\eta_{l-1} = \eta_l / \delta, \quad \text{với } \delta \text{ là một hằng số, ví dụ 2.6.}$$

Khi cập nhật trọng số  $\theta = \{\theta_1, \dots, \theta_L\}$ , mỗi tầng sẽ được cập nhật như sau:

$$\theta_l \leftarrow \theta_l - \eta_l \cdot \nabla_{\theta_l} J(\theta) \quad (3.4)$$

Kỹ thuật này cho phép các tầng trên học hỏi nhanh chóng để thích nghi với tác vụ mới, trong khi các tầng dưới chỉ thay đổi một cách cẩn trọng để không làm mất đi kiến thức ngôn ngữ tổng quát đã học.

**2. Tốc độ học Tam giác Lệch (Slanted Triangular Learning Rates - STLR)** Thay vì giữ một tốc độ học không đổi hoặc giảm dần theo một hàm mũ đơn giản, ULMFiT sử dụng một lịch trình tốc độ học (learning rate schedule) đặc biệt. Trong một epoch, tốc độ học sẽ **tăng tuyến tính** trong một khoảng ngắn lúc đầu (warm-up), sau đó **giảm tuyến tính** trong phần còn lại của quá trình huấn luyện.

Công thức của STLR tại bước huấn luyện  $t$  được định nghĩa như sau:

$$\eta_t = \eta_{\max} \times \begin{cases} \frac{t}{t_{\text{cut}}} & \text{nếu } t < t_{\text{cut}} \\ 1 - \frac{t-t_{\text{cut}}}{T-t_{\text{cut}}} & \text{nếu } t \geq t_{\text{cut}} \end{cases} \quad (3.5)$$

trong đó:

- $T$  là tổng số bước huấn luyện trong một epoch.
- $t_{\text{cut}}$  là bước mà tại đó tốc độ học chuyển từ tăng sang giảm (thường là 10% của  $T$ ).
- $\eta_{\max}$  là tốc độ học tối đa, được chọn cho tầng cuối cùng. Các tầng khác sẽ có  $\eta_{\max}$  riêng theo quy tắc của Tinh chỉnh phân biệt.

Chiến lược này giúp mô hình nhanh chóng hội tụ đến một vùng tham số tốt (giai đoạn tăng) rồi sau đó từ từ tinh chỉnh để tìm điểm tối ưu cục bộ tốt hơn (giai đoạn giảm).

**3. Mở băng Dần dần (Gradual Unfreezing)** Để tránh "sự quên lảng thảm khốc", việc tinh chỉnh toàn bộ mô hình ngay từ đầu trên một tập dữ liệu nhỏ có thể phá hỏng các trọng số đã được huấn luyện cẩn thận. ULMFiT đề xuất một phương pháp an toàn hơn, theo từng giai đoạn:

1. **Giai đoạn 1:** Đóng băng (freeze) tất cả các tầng đã được huấn luyện trước ( $l = 1, \dots, L-1$ ), chỉ huấn luyện các tầng của đầu phân loại mới được thêm vào (tầng  $L$ ) trong một epoch.
2. **Giai đoạn 2:** Mở băng (unfreeze) tầng trên cùng của mô hình gốc (tầng  $L-1$ ) và huấn luyện nó cùng với tầng  $L$  trong một epoch nữa.
3. **Giai đoạn 3 và tiếp theo:** Tiếp tục quá trình này, tuần tự mở băng từng tầng một từ trên xuống dưới ( $L-2, L-3, \dots, 1$ ) và huấn luyện cho đến khi toàn bộ mô hình được tinh chỉnh.

Phương pháp này giống như việc từ từ "rã đông" kiến thức, giúp mô hình thích nghi một cách nhẹ nhàng và hiệu quả, đặc biệt khi dữ liệu tác vụ có hạn.

#### Công thức thành công của ULMFiT

ULMFiT không chỉ là một mô hình, mà là một công thức thực tiễn cho học chuyển giao trong NLP:

- **Kiến trúc nền tảng:** AWD-LSTM (một biến thể LSTM hiệu quả và được điều chỉnh tốt).
- **Quy trình 3 bước:** Pre-train (chung) → Fine-tune LM (miền đích) → Fine-tune Classifier (tác vụ).
- **Bí quyết tinh chỉnh:** Kết hợp Discriminative Fine-tuning, Slanted Triangular Learning Rates, và Gradual Unfreezing.

#### Di sản và Hạn chế

ULMFiT là một cột mốc quan trọng, nó đã "dân chủ hóa" việc áp dụng học chuyển giao trong NLP, cho thấy rằng không cần đến tài nguyên tính toán khổng lồ vẫn có thể đạt được hiệu suất cao. Nó đã đặt nền móng và duy trì cho các phương pháp sau này như BERT và các mô hình dựa trên Transformer, vốn cũng tuân theo quy trình "huấn luyện trước - tinh chỉnh" nhưng với kiến trúc mạnh mẽ hơn.

Tuy nhiên, hạn chế chính của ULMFiT nằm ở kiến trúc LSTM tuần tự, vốn không thể song song hóa hiệu quả và gặp khó khăn trong việc nắm bắt các phụ thuộc tầm xa so với kiến trúc Self-Attention của Transformer.

## 3.7. Các Kiến trúc Nơ-ron Khác

Bên cạnh RNN, CNN và các kiến trúc chính khác, trong NLP còn tồn tại nhiều hướng tiếp cận nơ-ron đặc thù cho các dạng bài toán riêng. Phần này sẽ trình bày hai kiến trúc tiêu biểu: **Mạng Siamese** cho việc học sự tương đồng, và **Mạng Nơ-ron Đồ thị (GNNS)** cho việc xử lý dữ liệu có cấu trúc đồ thị.

### 3.7.1. Mạng Siamese (Siamese Networks) cho việc học sự tương đồng

Các kiến trúc chúng ta đã học như RNN hay CNN thường được huấn luyện cho các tác vụ phân loại (classification) hoặc gán nhãn (labeling). Tuy nhiên, có một lớp bài toán rất quan trọng trong NLP mà các kiến trúc này không trực tiếp giải quyết được: **đo lường sự tương đồng ngữ nghĩa (semantic similarity)** giữa hai câu. Ví dụ:

- **Phát hiện câu hỏi trùng lặp (Duplicate Question Detection):** Trên các diễn đàn như Quora hay Stack Overflow, việc xác định xem một câu hỏi mới có trùng lặp về mặt ý nghĩa với một câu hỏi đã có hay không là rất quan trọng.
- **Tìm kiếm ngữ nghĩa (Semantic Search):** Tìm các tài liệu có ý nghĩa liên quan nhất đến một câu truy vấn, thay vì chỉ khớp từ khóa.
- **Đánh giá câu trả lời của sinh viên (Automatic Short Answer Grading):** So sánh câu trả lời của sinh viên với câu trả lời mẫu để chấm điểm.

Để giải quyết các bài toán này, chúng ta cần một mô hình có khả năng nhận vào hai câu và xuất ra một điểm số thể hiện mức độ tương đồng của chúng. **Mạng Siamese** [58] là một kiến trúc thanh lịch và mạnh mẽ được thiết kế cho chính mục đích này.

#### Tự duy cốt lõi: Học một không gian biểu diễn tốt

Thay vì cố gắng học một hàm phân loại trực tiếp trên cặp câu, ý tưởng cốt lõi của Mạng Siamese là **học một hàm nhúng (embedding function)** có khả năng ánh xạ các câu vào một không gian

vector (embedding space) sao cho trong không gian đó, khoảng cách giữa các câu có thể phản ánh sự tương đồng về mặt ngữ nghĩa của chúng.

Nói một cách cụ thể hơn, Mạng Siamese học cách để:

- Các cặp câu **tương tự** (similar pairs) được kéo lại **gần nhau** trong không gian embedding.
- Các cặp câu **không tương tự** (dissimilar pairs) bị **đẩy ra xa nhau** trong không gian embedding.

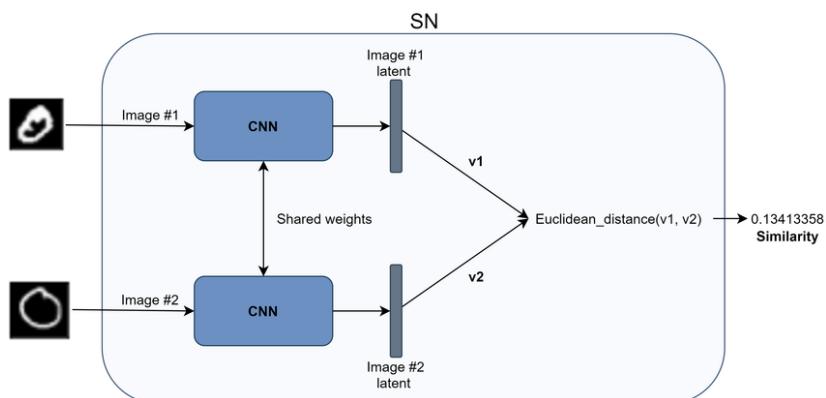
Sau khi đã học được không gian embedding tốt này, việc đo độ tương đồng giữa hai câu mới chỉ đơn giản là ánh xạ chúng vào không gian này và tính khoảng cách Euclidean hoặc độ tương đồng cosine giữa hai vector kết quả.

### Kiến trúc Mạng Siamese

Cái tên "Siamese" (cặp song sinh dính liền) mô tả rất chính xác kiến trúc của nó. Mạng Siamese bao gồm hai "nhánh" (towers) mạng nơ-ron giống hệt nhau, xử lý hai đầu vào một cách song song.

### Hai nhánh, một bộ não

- Mạng bao gồm hai bộ mã hóa (encoder) có **kiến trúc y hệt nhau**.
- Quan trọng nhất, hai bộ mã hóa này **chia sẻ cùng một bộ trọng số** (share weights). Điều này có nghĩa là chúng ta chỉ có một "bộ não" duy nhất, được áp dụng cho cả hai câu đầu vào.
- Việc chia sẻ trọng số đảm bảo rằng hai câu giống hệt nhau sẽ luôn được ánh xạ tới cùng một điểm trong không gian embedding. Nó cũng làm cho mô hình hiệu quả hơn nhiều về mặt tham số.
- Bộ mã hóa này có thể là bất kỳ kiến trúc nào có khả năng biến một chuỗi thành một vector, ví dụ như một mạng LSTM, GRU, hoặc CNN đã được học ở các phần trước.



Hình 3.10: Kiến trúc của Mạng Siamese. Hai mạng con (thường là LSTM hoặc CNN) có cùng kiến trúc và chia sẻ trọng số. Chúng xử lý hai câu đầu vào để tạo ra hai vector embedding. Hàm mất mát sau đó được tính toán dựa trên khoảng cách giữa hai vector này.

### Hàm Mất mát Tương phản (Contrastive Loss)

Kiến trúc chỉ là một nửa câu chuyện. Phép màu thực sự của Mạng Siamese nằm ở hàm mất mát đặc biệt được sử dụng để huấn luyện nó: **Hàm Mất mát Tương phản (Contrastive Loss)** [29].

Hàm mất mát này được thiết kế để thực hiện chính xác mục tiêu mà chúng ta đã đề ra: kéo các cặp tương tự lại gần và đẩy các cặp không tương tự ra xa.

**Dầu vào huấn luyện** Để huấn luyện với Contrastive Loss, chúng ta cần một bộ dữ liệu bao gồm các bộ ba: (câu\_A, câu\_B, nhãn).

- Nhãn  $Y = 1$  (Cặp Tương tự): câu\_A và câu\_B có cùng ý nghĩa.
- Nhãn  $Y = 0$  (Cặp Không Tương tự): câu\_A và câu\_B có ý nghĩa khác nhau.

**Công thức** Cho một cặp đầu vào, Mạng Siamese tạo ra một cặp vector embedding ( $v_A, v_B$ ). Chúng ta định nghĩa  $D_W$  là khoảng cách Euclidean giữa chúng:  $D_W = \|v_A - v_B\|_2$ .

Hàm mất mát tương phản được định nghĩa như sau:

$$\text{Loss}(W, (Y, v_A, v_B)) = Y \cdot \frac{1}{2} D_W^2 + (1 - Y) \cdot \frac{1}{2} \max(0, m - D_W)^2 \quad (3.6)$$

Trong đó  $m$  là một siêu tham số gọi là **lề (margin)**.

**Phân tích hàm mất mát** Hãy xem hàm mất mát này hoạt động như thế nào trong hai trường hợp:

1. Khi cặp câu là **Tương tự** ( $Y = 1$ ):

- Phần thứ hai của công thức bị triệt tiêu (vì  $1 - Y = 0$ ).
- Hàm mất mát trở thành:  $\text{Loss} = \frac{1}{2} D_W^2$ .
- Để tối thiểu hóa loss, mô hình phải làm cho khoảng cách  $D_W$  càng gần 0 càng tốt. Điều này kéo hai vector lại gần nhau.

2. Khi cặp câu là **Không tương tự** ( $Y = 0$ ):

- Phần đầu của công thức bị triệt tiêu (vì  $Y = 0$ ).
- Hàm mất mát trở thành:  $\text{Loss} = \frac{1}{2} \max(0, m - D_W)^2$ .
- Để tối thiểu hóa loss, mô hình phải làm cho  $m - D_W$  nhỏ hơn hoặc bằng 0, tức là  $D_W \geq m$ .
- Điều này **đẩy hai vector ra xa nhau** cho đến khi khoảng cách giữa chúng ít nhất là bằng  $m$ . Nếu chúng đã đủ xa ( $D_W > m$ ), loss sẽ bằng 0 và mô hình không cần phải làm gì thêm.

Lề  $m$  đóng vai trò như một "vùng an toàn", nó yêu cầu các cặp không tương tự phải cách nhau một khoảng tối thiểu, tạo ra sự phân tách rõ ràng trong không gian embedding.

### Triplet Loss: Một sự thay thế nâng cao

Một hàm mất mát phổ biến khác, có liên quan chặt chẽ, là **Triplet Loss** [79]. Thay vì huấn luyện trên các cặp, Triplet Loss huấn luyện trên các bộ ba (triplets) gồm:

- **Anchor (A)**: Một câu neo.
- **Positive (P)**: Một câu tương tự với Anchor.
- **Negative (N)**: Một câu không tương tự với Anchor.

Mục tiêu của Triplet Loss là làm cho khoảng cách từ Anchor đến Positive nhỏ hơn khoảng cách từ Anchor đến Negative, cộng với một lề  $m$ .

$$d(A, P) + m < d(A, N)$$

Hàm mất mát được định nghĩa là:

$$\text{Loss} = \max(0, d(A, P) - d(A, N) + m) \quad (3.7)$$

Triplet Loss thường được coi là mạnh mẽ hơn vì nó trực tiếp tối ưu hóa thứ hạng tương đối giữa các câu, thay vì chỉ xem xét các cặp một cách riêng lẻ. Tuy nhiên, việc tạo ra các bộ ba huấn luyện hiệu quả (hard triplets) là một thách thức kỹ thuật.

### Tổng kết

Mạng Siamese và các hàm mất mát đi kèm (Contrastive/Triplet Loss) là một bộ công cụ cực kỳ mạnh mẽ cho các bài toán so sánh. Chúng không chỉ giới hạn trong NLP mà còn được áp dụng rộng rãi trong xử lý ảnh để nhận dạng khuôn mặt (face verification) hay tìm kiếm ảnh tương tự.

Thay vì học cách phân loại, Mạng Siamese học cách **biểu diễn**, tạo ra các embedding chất lượng cao, có thể tái sử dụng cho nhiều tác vụ hạ nguồn liên quan đến đo lường sự tương đồng.

### 3.7.2. Mạng Nơ-ron Đồ thị (Graph Neural Networks - GNNs) trong NLP

Trong suốt chương này, chúng ta đã xem xét các kiến trúc được thiết kế để xử lý dữ liệu dạng chuỗi (sequences) như RNN và các kiến trúc trích xuất đặc trưng cục bộ như CNN. Tuy nhiên, một sự thật quan trọng là: **ngôn ngữ không phải lúc nào cũng là một chuỗi tuyến tính.** Rất nhiều thông tin quý giá trong ngôn ngữ và thế giới mà nó mô tả lại có cấu trúc của một đồ thị (graph).

Ví dụ:

- Cấu trúc cú pháp của một câu (như Cú pháp Phụ thuộc đã học ở Chương 1) là một đồ thị.
- Một cơ sở tri thức (Knowledge Base) về các thực thể và mối quan hệ của chúng là một đồ thị.
- Một mạng xã hội, với người dùng và mối quan hệ bạn bè, là một đồ thị.
- Một cuộc hội thoại nhiều bên, với các lượt lời và quan hệ trả lời, là một đồ thị.

Các kiến trúc truyền thống như RNN và CNN gặp khó khăn trong việc mô hình hóa các mối quan hệ phức tạp, không tuân tự này. Mạng Nơ-ron Đồ thị (GNNs) [78, 42] ra đời như một giải pháp tổng quát và mạnh mẽ để học trực tiếp trên dữ liệu có cấu trúc đồ thị.

#### Nguyên lý Cốt lõi của GNN: Lan truyền Thông điệp (Message Passing)

Để hiểu GNN, trước hết ta cần hiểu cách nó "suy nghĩ" về một đồ thị. Một đồ thị bao gồm các **nút** (nodes) và các **cạnh** (edges) nối giữa chúng. Trong GNN, mỗi nút có một vector đặc trưng (node feature vector) riêng. Mục tiêu của GNN là học một hàm để tính toán một vector **biểu diễn** (node representation/embedding) cho mỗi nút, sao cho vector này không chỉ chứa thông tin của chính nút đó mà còn tóm tắt được thông tin từ **cấu trúc lân cận** của nó.

GNN thực hiện điều này thông qua một quy trình lặp đi lặp lại gọi là **Lan truyền Thông điệp** (Message Passing) hoặc **Tổng hợp Lân cận** (Neighborhood Aggregation).

#### Trực giác về Lan truyền Thông điệp

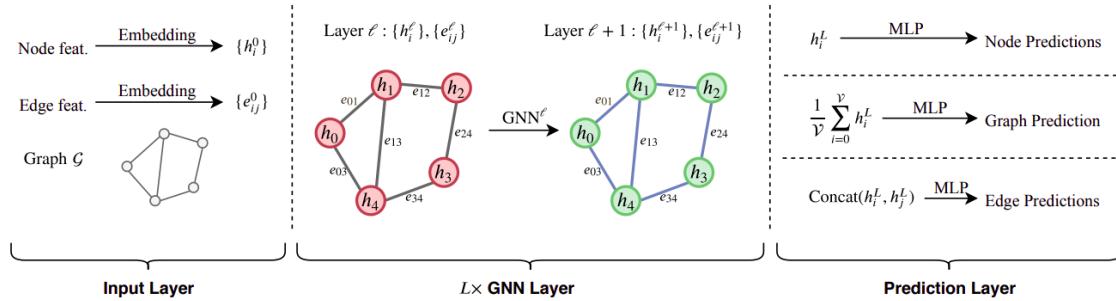
Hãy tưởng tượng mỗi nút trong đồ thị là một người. Tại mỗi vòng, mỗi người sẽ làm hai việc:

1. **Gửi thư:** Gửi một bản sao "trạng thái" hiện tại của mình cho tất cả những người hàng xóm.
2. **Cập nhật bản thân:** Nhận tất cả "thư" từ hàng xóm, tổng hợp chúng lại thành một thông điệp duy nhất, và sau đó kết hợp thông điệp này với trạng thái cũ của mình để tạo ra một trạng thái mới, thông thái hơn.

Khi quá trình này lặp lại nhiều lần, thông tin từ những nút ở xa sẽ dần lan truyền khắp đồ thị. Một nút sau  $k$  vòng lặp sẽ có một vector biểu diễn chứa thông tin từ tất cả các nút cách nó  $k$  bước chân.

Về mặt kỹ thuật, một lớp GNN thực hiện 3 bước:

1. **Thu thập (Gather):** Với mỗi nút, thu thập vector đặc trưng từ tất cả các nút hàng xóm trực tiếp của nó.
2. **Tổng hợp (Aggregate):** Áp dụng một hàm tổng hợp (ví dụ: 'SUM', 'MEAN', 'MAX') lên các vector đã thu thập để tạo ra một "thông điệp" lân cận duy nhất. Hàm này phải không nhạy cảm với thứ tự của các hàng xóm.
3. **Cập nhật (Update):** Kết hợp thông điệp lân cận với vector đặc trưng hiện tại của nút (thường thông qua một mạng nơ-ron nhỏ) để tạo ra vector đặc trưng mới cho nút đó ở lớp tiếp theo.



Hình 3.11: Quy trình Lan truyền Thông điệp trong một lớp GNN. Nút A cập nhật trạng thái của nó bằng cách tổng hợp thông tin từ các hàng xóm B, C, D.

Bằng cách xếp chồng nhiều lớp GNN, một nút có thể tích hợp thông tin từ các hàng xóm ngày càng xa, cho phép mô hình có được "cái nhìn toàn cảnh" về vị trí và vai trò của mỗi nút trong toàn bộ đồ thị.

### Huấn luyện GNN

Quá trình huấn luyện một GNN phụ thuộc vào tác vụ cụ thể ở cấp độ đồ thị, nút, hoặc cạnh.

- **Tạo biểu diễn nút:** Sau  $k$  lớp Message Passing, mỗi nút sẽ có một vector biểu diễn cuối cùng  $h_v^{(k)}$  chứa thông tin từ lân cận  $k$ -bước của nó.
- **Hàm Mất mát:** Vector biểu diễn này sau đó được đưa vào một lớp đầu ra (ví dụ, một lớp FNN) để thực hiện dự đoán.
  - **Phân loại Nút:** Lỗi (ví dụ, Cross-Entropy) được tính toán cho từng nút và lan truyền ngược qua các lớp GNN để cập nhật các trọng số của hàm tổng hợp (Aggregate) và cập nhật (Update).
  - **Phân loại Đồ thị:** Các biểu diễn của tất cả các nút trong đồ thị được tổng hợp lại (ví dụ, qua pooling) thành một vector biểu diễn đồ thị duy nhất trước khi đưa vào lớp phân loại. Tương tự như các mạng khác, toàn bộ quá trình được tối ưu hóa end-to-end thông qua backpropagation.

### Các Ứng dụng then chốt của GNN trong NLP

Với khả năng học trên cấu trúc, GNN mở ra nhiều hướng ứng dụng mạnh mẽ trong NLP.

**Làm việc với Đồ thị Tri thức và Mạng xã hội** Đây là ứng dụng tự nhiên nhất của GNN. Trong một **Đồ thị Tri thức (Knowledge Graph - KG)**, các nút là các thực thể (ví dụ: 'Hà Nội', 'Việt Nam') và các cạnh có nhãn là các mối quan hệ (ví dụ: 'isCapitalOf'). GNN được sử dụng để giải quyết hai bài toán chính:

- **Phân loại Nút (Node Classification):** Dự đoán một thuộc tính còn thiếu của một thực thể. Ví dụ, dự đoán thể loại của một bộ phim dựa trên các diễn viên và đạo diễn (các nút lân cận) của nó.
- **Dự đoán Liên kết (Link Prediction):** Đây là bài toán trích xuất quan hệ và suy luận logic. Mô hình dự đoán xem liệu một cạnh (mối quan hệ) có tồn tại giữa hai nút hay không. Ví dụ, nếu KG có các cạnh '(David Beckham, playsFor, Manchester United)' và '(Manchester United, locatedIn, England)', GNN có thể học cách suy luận ra một liên kết tiềm năng '(David Beckham, hasNationality, English)'. Điều này cho phép tự động hoàn thiện và mở rộng các KG.

**Phân tích Cú pháp dựa trên Đồ thị (Graph-based Parsing)** Như đã đề cập, cây cú pháp phụ thuộc của một câu là một đồ thị. Chúng ta có thể áp dụng GNN trực tiếp lên cấu trúc này.

- **Biểu diễn ngữ nghĩa dựa trên cấu trúc (Structure-aware Semantic Representation):** Thay vì chỉ dựa vào thứ tự tuần tự như LSTM, GNN cho phép một từ được biểu diễn bởi một vector tổng hợp thông tin từ các từ "quản lý" và "phụ thuộc" của nó. Ví dụ, vector của một động từ sẽ được "làm giàu" bởi thông tin từ chủ ngữ và tân ngữ của nó. Các biểu diễn này thường nắm bắt ngữ nghĩa chức năng tốt hơn.
- **Cải thiện các tác vụ hạ nguồn:** Các biểu diễn giàu cú pháp này sau đó có thể được sử dụng để cải thiện hiệu năng của các tác vụ như Trích xuất Quan hệ hay Phân tích Cảm xúc.

**Mô hình hóa Hội thoại (Dialogue Modeling)** Một cuộc hội thoại, đặc biệt là với nhiều người tham gia, không phải là một chuỗi tuyến tính.

- **Xây dựng Đồ thị Hội thoại (Dialogue Graph):** Chúng ta có thể xây dựng một đồ thị trong đó các nút là các lượt lời (utterances), và các cạnh biểu diễn các mối quan hệ như "ai trả lời ai" hoặc "các lượt lời của cùng một người".
- **Ứng dụng GNN:** GNN có thể được chạy trên đồ thị này để học biểu diễn cho mỗi lượt lời, có tính đến toàn bộ lịch sử tương tác. Điều này rất hữu ích cho các tác vụ như tóm tắt hội thoại, theo dõi trạng thái hội thoại (dialogue state tracking), hoặc xác định cảm xúc trong các cuộc trò chuyện phức tạp.

### Các Kiến trúc Lai ghép: Kết hợp GNN và Transformer

Sự ra đời của Transformer (sẽ học ở chương sau) đã thống trị NLP với khả năng nắm bắt phụ thuộc tầm xa trong chuỗi. Tuy nhiên, Transformer vốn không được thiết kế để xử lý thông tin cấu trúc tường minh. Điều này đã dẫn đến một hướng nghiên cứu sôi nổi: **kết hợp GNN và Transformer**.

- **Ý tưởng:** Sử dụng Transformer để có được các biểu diễn từ ban đầu, giàu ngữ cảnh tuần tự. Sau đó, xây dựng một đồ thị (ví dụ: đồ thị cú pháp) lên trên các từ này và sử dụng GNN để "tinh chỉnh" (refine) các biểu diễn đó bằng cách cho chúng lan truyền thông tin qua cấu trúc đồ thị.
- **Kết quả:** Các mô hình lai ghép này thường cho kết quả tốt hơn trên các tác vụ đòi hỏi cả sự hiểu biết về cấu trúc và ngữ cảnh, ví dụ như các bài toán suy luận phức tạp.

### Thách thức và Hướng nghiên cứu Tương lai

Mặc dù rất mạnh mẽ, GNN vẫn phải đối mặt với nhiều thách thức:

- **Khả năng mở rộng (Scalability):** Áp dụng GNN trên các đồ thị web-scale với hàng tỷ nút và cạnh là một thách thức lớn về mặt tính toán và bộ nhớ.
- **Đồ thị Động (Dynamic Graphs):** Hầu hết các GNN giả định một cấu trúc đồ thị tĩnh. Việc xử lý các đồ thị thay đổi liên tục theo thời gian (như mạng xã hội) vẫn là một lĩnh vực nghiên cứu tích cực.
- **Làm mịn quá mức (Over-smoothing):** Khi xếp chồng quá nhiều lớp GNN, các biểu diễn của tất cả các nút có xu hướng hội tụ về cùng một giá trị, làm mất đi thông tin cục bộ. Đây là một vấn đề cố hữu của GNN.
- **Đồ thị nhiều và không hoàn chỉnh:** Làm thế nào để mô hình hoạt động mạnh mẽ khi cấu trúc đồ thị đầu vào bị thiếu hoặc thiếu các cạnh quan trọng?

Bất chấp những thách thức này, GNN đại diện cho một hướng đi quan trọng, giúp NLP vượt ra khỏi giới hạn của chuỗi tuần tự và tiến tới việc mô hình hóa các mối quan hệ phức tạp hơn, một bước gần hơn đến việc thực sự hiểu được ngôn ngữ.

### 3.8. Các Mô hình Sinh Tiền-Transformer (Pre-Transformer Generative Models)

Trước khi Transformer định hình lại toàn bộ lĩnh vực NLP, cộng đồng nghiên cứu đã dành nhiều nỗ lực để xây dựng các **mô hình sinh** (generative models) cho văn bản. Khác với các mô hình phân

loại hay dịch máy vốn chỉ cần *hiểu* hoặc *chuyển đổi* ngôn ngữ, mô hình sinh có mục tiêu tham vọng hơn: **sáng tạo ngôn ngữ mới**, ví dụ như viết một đoạn hội thoại, một bài thơ, hay một câu nhận xét sản phẩm mà vẫn mạch lạc và hợp lý.

Trong giai đoạn 2015–2017, hai hướng tiếp cận chính cho bài toán sinh văn bản nổi bật: **Variational Autoencoders (VAEs)** và **Generative Adversarial Networks (GANs)**. Việc tìm hiểu chúng mang lại giá trị lịch sử và giúp ta thấy rõ những thách thức cố hữu trong việc sinh dữ liệu rời rạc như văn bản.

### 3.8.1. Variational Autoencoder (VAE) cho Văn bản

**Ý tưởng cốt lõi.** Khác với Autoencoder thường (đã trình bày ở Mục 3.2), VAE (Kingma & Welling, 2013) [41] coi không gian ẩn là một *phân phối xác suất*. Thay vì ánh xạ đầu vào  $x$  thành một vector điểm  $z$ , Encoder ánh xạ thành  $q_\phi(z|x) \sim \mathcal{N}(\mu, \sigma^2)$ , từ đó lấy mẫu  $z$  và giải mã bằng Decoder.

**Hàm mất mát ELBO.** Mục tiêu huấn luyện là cực đại hóa cận dưới hợp lý hóa bằng chứng:

$$\mathcal{L}_{VAE}(x) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - KL(q_\phi(z|x) \parallel p(z))$$

Trong đó  $KL(\cdot \parallel \cdot)$  đo khoảng cách giữa phân phối hậu nghiệm xấp xỉ và tiên nghiệm  $p(z)$  (thường là Gaussian chuẩn).

#### Ưu điểm của VAE cho văn bản

- **Khả năng sinh:** lấy mẫu  $z$  từ  $\mathcal{N}(0, I)$  và giải mã ra văn bản mới.
- **Nội suy không gian ẩn:** kết hợp hai vector  $z_1, z_2$  để sinh ra văn bản trung gian.
- **Ứng dụng:** sinh câu đa dạng (Bowman et al., 2016), style transfer.

**Thách thức: Posterior Collapse.** Decoder mạnh (LSTM/GRU) dễ dàng lờ  $z$ , chỉ dự đoán từ tiếp theo dựa vào lịch sử  $\Rightarrow$  khiến không gian ẩn mất ý nghĩa. Đây là một vấn đề được chỉ ra trong các công trình đầu tiên áp dụng VAE cho văn bản [8].

**Giải pháp:**

- KL-annealing (tăng dần trọng số KL).
- Free-bits (ép mỗi chiều KL có đóng góp tối thiểu).
- $\beta$ -VAE (điều chỉnh trade-off giữa likelihood và KL).

### 3.8.2. Generative Adversarial Networks (GANs) cho Văn bản

**Ý tưởng cơ bản.** GAN (Goodfellow et al., 2014) [26] là một trò chơi đối kháng:

- **Generator  $G$ :** sinh văn bản giả từ vector nhiễu  $z$ .
- **Discriminator  $D$ :** phân biệt văn bản thật từ dữ liệu và văn bản giả từ  $G$ .

**Văn đề rời rạc.** Khác ảnh (dữ liệu liên tục), văn bản là **chuỗi rời rạc**. Việc chọn từ (qua `argmax`) không khả vi, gradient từ  $D$  không truyền ngược về  $G$ .

**Các hướng giải quyết:**

- **Học tăng cường:** - SeqGAN (2017): dùng policy gradient. - MaliGAN (2017): cải thiện reward shaping. - LeakGAN (2018): cho  $D$  “rò rỉ” trạng thái cho  $G$ .
- **Xấp xỉ liên tục:** - Gumbel-Softmax để làm tròn `argmax`. - RelGAN (2019): training ổn định hơn.

**Hạn chế thực tiễn.** GAN cho văn bản huấn luyện khó và thiếu ổn định, nên chỉ dừng lại ở mức nghiên cứu, chưa có ứng dụng rộng như GAN trong ảnh.

### 3.8.3. So sánh VAE và GAN trong Sinh Văn bản

Tiêu chí	VAE	GAN
Nguyên lý	Xác suất, tối ưu ELBO (likelihood-based)	Đối kháng, min-max game
Đầu ra	Đa dạng, có thể nội suy latent	Có thể sắc nét, nhưng không ổn định
Thách thức	Posterior collapse	Không khả vi, gradient vanishing
Ứng dụng chính	Sinh câu, style transfer	Chủ yếu học thuật, ít thực tiễn

Bảng 3.1: So sánh VAE và GAN trong sinh văn bản trước Transformer.

### 3.8.4. Chuyển tiếp sang Kỷ nguyên Transformer

VAE và GAN đặt nền móng cho ý tưởng sinh văn bản, nhưng gặp giới hạn về độ ổn định và khả năng mở rộng. Sự ra đời của **Transformer** (Vaswani et al., 2017) đã thay đổi toàn bộ cục diện:

- Self-attention mô hình hóa phụ thuộc dài tốt hơn RNN.
- Huấn luyện likelihood trực tiếp (mô hình ngôn ngữ) ổn định hơn GAN.
- Kiến trúc dễ mở rộng quy mô lớn, tạo tiền đề cho GPT và các LLM sau này.

Do đó, các mô hình sinh tiền-Transformer (VAE, GAN) nên được coi là **bước đệm lịch sử**, giúp rút ra nhiều bài học quan trọng trước khi cộng đồng chuyển sang các mô hình ngôn ngữ sinh hiện đại dựa trên Transformer.

---

## KẾT THÚC CHƯƠNG 3

*Trong chương này, chúng ta đã khám phá một "vườn thú" các kiến trúc mạng nơ-ron đã định hình nền kỷ nguyên học sâu của NLP. Từ khả năng ghi nhớ của RNN/LSTM, đến sức mạnh trích xuất đặc trưng của CNN, và khả năng học trên các cấu trúc phức tạp của GNN. Quan trọng nhất, chúng ta đã chứng kiến sự ra đời của cơ chế Chú ý (Attention) trong mô hình Seq2Seq. Cơ chế Chú ý, ban đầu chỉ là một thành phần hỗ trợ, đã tỏ ra mạnh mẽ đến mức nó đặt ra một câu hỏi mang tính cách mạng: Liệu chúng ta có thể xây dựng một kiến trúc hoàn toàn từ bỏ các vòng lặp hồi tiếp và chỉ dựa vào sự chú ý không? Câu trả lời cho câu hỏi này sẽ mở ra chương tiếp theo và cũng là kỷ nguyên hiện đại của NLP: Kỷ nguyên Transformer.*

### CHƯƠNG 3. CÁC KIẾN TRÚC MẠNG NỔ-RON KINH ĐIỂN

## Chương 4

# KỶ NGUYÊN TRANSFORMER VÀ CÁC MÔ HÌNH NGÔN NGỮ LỚN (LLMs)

Chào mừng bạn đến với chương quan trọng nhất của NLP hiện đại. Nếu như các kiến trúc RNN/LSTM trong chương trước đã giải quyết vấn đề "trí nhớ" trong xử lý chuỗi, chúng vẫn còn một điểm yếu cố hữu: **tính tuần tự (sequentiality)**. Việc phải xử lý các từ lần lượt, từng từ một, đã ngăn cản việc song song hóa quá trình huấn luyện và gây khó khăn cho việc nắm bắt các phụ thuộc tầm rất xa.

Năm 2017, một bài báo có tiêu đề mang tính tuyên ngôn – **"Attention Is All You Need"** – từ các nhà nghiên cứu tại Google đã thay đổi hoàn toàn cuộc chơi. Họ đã giới thiệu một kiến trúc mới, gọi là **Transformer**, hoàn toàn loại bỏ các vòng lặp hồi tiếp và chỉ dựa vào một cơ chế duy nhất: **sự chú ý (attention)**.

Kiến trúc Transformer không chỉ đạt được hiệu năng vượt trội trên các bài toán dịch máy mà còn, quan trọng hơn, mở ra khả năng huấn luyện các mô hình trên một quy mô chưa từng có, khai sinh ra Kỷ nguyên của các Mô hình Ngôn ngữ Lớn (Large Language Models - LLMs) mà chúng ta biết ngày nay.

### 4.1. Kiến trúc Transformer Gốc ("Attention Is All You Need")

Để hiểu được sức mạnh của Transformer, chúng ta cần mổ xẻ từng khối xây dựng nên nó, bắt đầu từ ý tưởng cốt lõi nhất: Self-Attention.

#### 4.1.1. Chi tiết cơ chế Self-Attention: Scaled Dot-Product và Multi-Head Attention

Cơ chế chú ý mà chúng ta đã học trong mô hình Seq2Seq (ở mục 3.4) là *cross-attention*, nơi Decoder "chú ý" đến Encoder. **Self-Attention (Tự chú ý)** là một biến thể đặc biệt, nơi một chuỗi "chú ý" đến chính nó.

**Tự duy cốt lõi** Self-Attention cho phép mỗi từ trong một câu có thể "nhìn" vào tất cả các từ khác trong cùng một câu để tính toán ra một biểu diễn mới cho chính nó. Biểu diễn mới này không chỉ chứa thông tin của bản thân từ đó, mà còn được "làm giàu" bởi ngữ cảnh từ các từ liên quan nhất.

Ví dụ, trong câu "Con mèo không muốn băng qua đường vì nó quá mệt", khi xử lý từ "nó", Self-Attention sẽ giúp mô hình xác định rằng "nó" có liên quan mật thiết đến "mèo" chứ không phải "đường", và tạo ra một biểu diễn cho "nó" mang nhiều thông tin của "mèo".

#### Scaled Dot-Product Attention

Đây là khối xây dựng cơ bản của Self-Attention trong Transformer. Nó hoạt động dựa trên ba khái niệm được lấy cảm hứng từ các hệ thống tìm kiếm thông tin: **Query, Key, và Value**.

### Trực giác về Query, Key, Value

Hãy tưởng tượng bạn đang tìm kiếm video trên YouTube.

- **Query (Truy vấn):** Là cụm từ bạn gõ vào thanh tìm kiếm (ví dụ: "công thức nấu phở"). Nó đại diện cho *thông tin bạn đang cần*.
- **Key (Khóa):** Mỗi video trên YouTube có một bộ các từ khóa (tiêu đề, mô tả) để mô tả nội dung của nó. Nó đại diện cho *thông tin mà một đối tượng có thể cung cấp*.
- **Value (Giá trị):** Chính là nội dung của video. Nó đại diện cho *thông tin thực sự sẽ được lấy ra*.

Cơ chế hoạt động là: bạn lấy **Query** của mình, so sánh nó với **Key** của tất cả các video để tìm ra các video liên quan nhất (tính điểm tương đồng). Sau đó, bạn lấy một "tổng có trọng số" của các **Value** (nội dung video), trong đó các video liên quan hơn sẽ có trọng số cao hơn.

### Tại sao lại cần đến ba vai trò Q, K, V?

Một câu hỏi tự nhiên là: Tại sao mỗi từ đầu vào  $x_i$  lại cần được chiếu thành ba vector  $q_i, k_i, v_i$  riêng biệt? Tại sao không chỉ dùng chính các  $x_i$ ?

- **Sự linh hoạt:** Việc tách thành ba vai trò mang lại sự linh hoạt tối đa. Nó cho phép mô hình học cách so sánh các từ dựa trên một loại đặc trưng (được mã hóa trong  $Q$  và  $K$ ) và sau đó trích xuất một loại đặc trưng khác (được mã hóa trong  $V$ ).
- **Ví dụ trực quan:** Hãy quay lại ví dụ "Con mèo... vì nó quá mệt".
  - Khi tính toán biểu diễn cho "nó", **Query** ( $q_{\text{nó}}$ ) có thể mã hóa thông tin: "Tôi là một đại từ, tôi cần tìm một danh từ làm tham chiếu".
  - **Key** ( $k_{\text{mèo}}$ ) của "mèo" có thể mã hóa: "Tôi là một danh từ giống đặc/cái, số ít". Key này sẽ "khớp" rất tốt với **Query** của "nó".
  - **Value** ( $v_{\text{mèo}}$ ) của "mèo" có thể chứa thông tin ngữ nghĩa thực sự của từ: "một loài động vật bốn chân".

Sau khi  $q_{\text{nó}}$  khớp với  $k_{\text{mèo}}$ , biểu diễn mới của "nó" sẽ nhận được một phần lớn thông tin từ  $v_{\text{mèo}}$ , giúp nó "hiểu" rằng nó đang nói về một con vật. Nếu chỉ dùng  $x_i$  cho cả ba vai trò, mô hình sẽ bị hạn chế trong việc học các mối quan hệ phức tạp như vậy.

Trong Self-Attention, mỗi từ đầu vào sẽ đóng cả ba vai trò này.

**Cơ chế hoạt động chi tiết** Giả sử chúng ta có một chuỗi các vector đầu vào  $X = (x_1, x_2, \dots, x_n)$ .

1. **Tạo Query, Key, Value:** Chúng ta tạo ra ba ma trận trọng số có thể học được:  $W^Q, W^K, W^V$ .

Mỗi vector đầu vào  $x_i$  sẽ được nhân với ba ma trận này để tạo ra ba vector tương ứng:

- $q_i = W^Q x_i$  (Query vector của từ  $i$ )
- $k_i = W^K x_i$  (Key vector của từ  $i$ )
- $v_i = W^V x_i$  (Value vector của từ  $i$ )

Toàn bộ chuỗi sẽ tạo ra các ma trận  $Q, K, V$ .

2. **Tính điểm chú ý (Attention Scores):** Để tính toán biểu diễn mới cho từ thứ  $i$ , chúng ta lấy **Query** của nó ( $q_i$ ) và tính tích vô hướng (dot product) với **Key** của *tất cả* các từ khác (bao gồm cả chính nó),  $k_j$ . Đây chính là bước so sánh "Query" với tất cả các "Key".

$$\text{score}(i, j) = q_i \cdot k_j$$

3. **Chia tỷ lệ (Scaling):** Tích vô hướng có thể tạo ra các giá trị rất lớn, đẩy hàm softmax vào vùng có gradient rất nhỏ, gây khó khăn cho việc huấn luyện. Bài báo đã "scale" (chia) các điểm số này cho căn bậc hai của số chiều của vector **Key**,  $d_k$ .

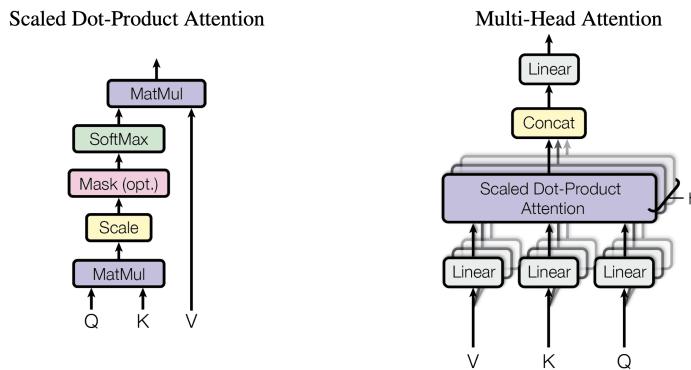
$$\text{scaled\_score}(i, j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

4. Chuẩn hóa (Softmax): Áp dụng hàm softmax trên các điểm số đã chia tỷ lệ để có được các trọng số chú ý  $\alpha_{ij}$ .
5. Tính toán đầu ra: Biểu diễn đầu ra mới cho từ  $i$ , ký hiệu là  $z_i$ , được tính bằng tổng có trọng số của *tất cả* các vector Value.

$$z_i = \sum_{j=1}^n \alpha_{ij} v_j$$

Toàn bộ quá trình này có thể được tóm gọn trong một công thức ma trận duy nhất, cực kỳ hiệu quả cho việc tính toán trên GPU:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (4.1)$$



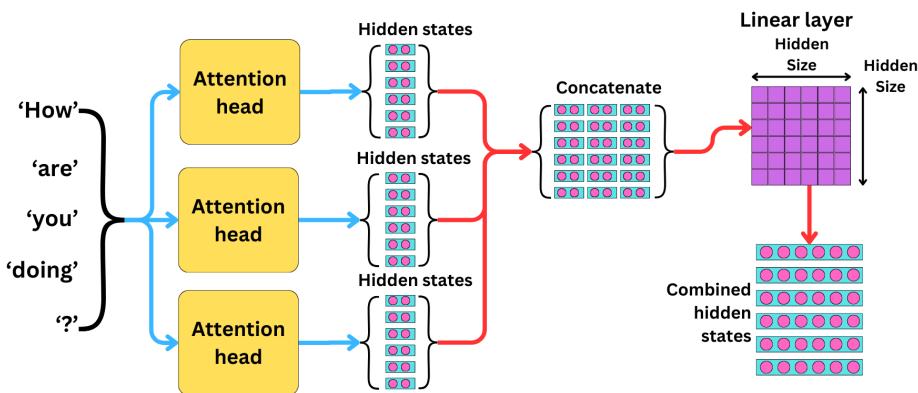
Hình 4.1: Sơ đồ Scaled Dot-Product Attention. Đầu vào là các ma trận  $Q, K, V$  và đầu ra là một ma trận đã được làm giàu ngữ cảnh.

### Muti-Head Attention

**Vấn đề của Single-Head Attention** Chỉ có một bộ Self-Attention duy nhất sẽ bị hạn chế. Nó giống như việc bạn chỉ có thể tập trung vào một khía cạnh của mối quan hệ giữa các từ. Ví dụ, khi xử lý câu "Con mèo ngồi trên chiếu", một "đầu" chú ý (attention head) có thể học cách liên kết "nó" với "mèo" (quan hệ đồng tham chiếu), nhưng có thể sẽ bỏ lỡ các mối quan hệ khác (ví dụ, quan hệ cú pháp).

**Giải pháp: Nhiều "đầu" chú ý song song** Multi-Head Attention giải quyết vấn đề này bằng cách chạy nhiều cơ chế Scaled Dot-Product Attention một cách **song song**.

- Thay vì chỉ có một bộ trọng số  $(W^Q, W^K, W^V)$ , chúng ta có  $h$  bộ (ví dụ,  $h = 8$ ).
- Mỗi bộ trọng số này  $(W_i^Q, W_i^K, W_i^V)$  với  $i = 1, \dots, h$  được gọi là một "đầu" chú ý.
- Mỗi "đầu" sẽ chiếu các vector đầu vào vào một không gian con khác nhau và thực hiện Self-Attention một cách độc lập. Điều này cho phép mỗi đầu có thể học một loại quan hệ khác nhau. Ví dụ, một đầu có thể học quan hệ cú pháp, một đầu khác học quan hệ ngữ nghĩa, một đầu khác học quan hệ đồng tham chiếu...
- Các vector đầu ra từ tất cả các đầu sau đó được **nối lại (concatenated)** với nhau.
- Vector nối này sau đó được chiếu qua một ma trận trọng số cuối cùng  $W^O$  để tạo ra vector đầu ra cuối cùng của lớp Multi-Head Attention.



Hình 4.2: Kiến trúc Multi-Head Attention, bao gồm nhiều "đầu" Scaled Dot-Product Attention chạy song song.

Multi-Head Attention cho phép mô hình cùng lúc chú ý đến thông tin từ các không gian biểu diễn khác nhau tại các vị trí khác nhau. Đây là một trong những nhân tố chính tạo nên sức mạnh biểu diễn của Transformer.

#### 4.1.2. Kiến trúc Encoder-Decoder và các thành phần phụ

Transformer gốc vẫn giữ kiến trúc Encoder-Decoder của Seq2Seq, nhưng thay thế hoàn toàn các lớp RNN bằng các khối Self-Attention và các lớp Feed-Forward.

#### The Encoder Stack

Encoder của Transformer là một chuỗi gồm  $N$  (ví dụ,  $N = 6$ ) khối Encoder giống hệt nhau được xếp chồng lên nhau. Mỗi khối bao gồm hai lớp con chính:

1. **Multi-Head Self-Attention Layer:** Lớp này thực hiện Self-Attention trên chuỗi đầu vào.
2. **Position-wise Feed-Forward Network (FFN):** Đây là một mạng nơ-ron truyền thẳng đơn giản, bao gồm hai lớp tuyến tính với một hàm kích hoạt ReLU ở giữa.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

FFN này được áp dụng một cách **độc lập và giống hệt nhau** cho từng vị trí (position) trong chuỗi.

Ngoài ra, mỗi lớp con này còn được bao bọc bởi hai thành phần quan trọng khác:

- **Residual Connection (Kết nối phần dư):** Đầu ra của mỗi lớp con là  $\text{LayerNorm}(x + \text{Sublayer}(x))$ . Tức là, đầu vào  $x$  của lớp con được cộng trực tiếp vào đầu ra của chính lớp đó ( $\text{Sublayer}(x)$ ). Kết nối phần dư này giúp gradient chảy dễ dàng hơn qua các mạng sâu và chống lại vấn đề triệt tiêu gradient.
- **Layer Normalization (Chuẩn hóa lớp):** Chuẩn hóa đầu ra của mỗi lớp con để ổn định quá trình huấn luyện.

### Vai trò của các Thành phần Phụ trong Khối Transformer

Nếu Self-Attention là "bộ não" thì FFN, Residual Connection và Layer Normalization là "hệ tuần hoàn và hệ thần kinh" giúp bộ não đó hoạt động ổn định và hiệu quả.

- **Position-wise Feed-Forward Network (FFN):**

- **Vai trò:** Lớp Self-Attention chủ yếu thực hiện việc **trộn thông tin (mixing information)** giữa các từ. Sau bước này, lớp FFN được áp dụng riêng lẻ cho từng vị trí để **xử lý sâu hơn (deep processing)** thông tin đã được trộn đó. Nó có thể được xem như một bước "biến đổi phi tuyến" mạnh mẽ, giúp mô hình học các mối quan hệ phức tạp hơn, hoặc thậm chí hoạt động như một bộ nhớ key-value đơn giản để lưu trữ các tri thức đã học.
- **Tại sao "Position-wise"?** Việc áp dụng cùng một FFN cho mọi vị trí tương tự như việc chia sẻ trọng số trong RNN hoặc phép tích chập 1x1 trong CNN. Nó đảm bảo tính nhất quán trong cách xử lý ở các vị trí khác nhau.

- **Residual Connection & Layer Normalization:**

- **Mục đích kép của Residual Connection:** 1) Nó tạo ra một "con đường tắt" cho gradient, giúp chống lại vấn đề triệt tiêu gradient và cho phép huấn luyện các mô hình rất sâu (ví dụ, hàng chục lớp Transformer). 2) Nó đảm bảo rằng ngay cả khi một lớp con (Sublayer) không học được gì hữu ích, thông tin từ lớp trước đó vẫn có thể đi thẳng tới lớp tiếp theo mà không bị mất mát. Mô hình chỉ cần học phần "thay đổi" (delta) so với đầu vào.
- **Layer Normalization:** Nó giúp ổn định quá trình huấn luyện bằng cách giữ cho đầu ra của mỗi lớp con có phân phối ổn định (trung bình 0, phương sai 1). Điều này làm cho quá trình tối ưu hóa trở nên mượt mà hơn và ít nhạy cảm hơn với việc khởi tạo trọng số. Sự kết hợp của Residuals và Norm là công thức vàng để huấn luyện các mạng nơ-ron rất sâu thành công.

### The Decoder Stack

Decoder cũng là một chuỗi gồm  $N$  khối Decoder giống hệt nhau. Mỗi khối Decoder có ba lớp con:

1. **Masked Multi-Head Self-Attention Layer:** Tương tự như Self-Attention của Encoder, nhưng có một sự thay đổi quan trọng: **masking (che giấu)**. Khi dự đoán từ ở vị trí  $i$ , Decoder chỉ được phép "chú ý" đến các từ ở các vị trí trước đó (từ 1 đến  $i$ ).

- **Cơ chế kỹ thuật:** Việc này được thực hiện bằng cách thêm một "ma trận mặt nạ" (mask matrix) vào ma trận điểm chú ý, ngay trước bước softmax. Ma trận này có giá trị là 0 cho các vị trí được phép chú ý và  $-\infty$  cho các vị trí bị cấm.
- **Hệ quả:** Sau khi qua hàm softmax, các vị trí có giá trị  $-\infty$  sẽ có trọng số chú ý bằng 0, đảm bảo rằng thông tin từ các từ tương lai không bị "rò rỉ" vào quá trình tính toán.
- **Mục đích:** Điều này là để đảm bảo tính **tự hồi quy (auto-regressive)** của mô hình, ngăn không cho nó "gian lận" bằng cách nhìn vào các từ mà nó đang cố gắng dự đoán. Nó mô phỏng quá trình sinh văn bản trong thực tế, nơi chúng ta chỉ biết những từ đã viết trước đó.

2. **Multi-Head Cross-Attention Layer:** Đây là nơi Decoder thực sự tương tác với Encoder, và là nơi kiến trúc "chú ý" của Seq2Seq gốc được tái hiện.

- **Dòng chảy thông tin:** Ở lớp này, **Query** được tạo ra từ đầu ra của lớp Masked Self-Attention của Decoder. Nó đại diện cho "câu hỏi" mà Decoder đang muốn hỏi: "Dựa trên những gì tôi đã dịch cho đến nay, tôi nên nhìn vào đâu trong câu gốc để dịch từ tiếp theo?".
- Trong khi đó, **Key** và **Value** được tạo ra từ **đầu ra của khối Encoder cuối cùng**. Chúng đại diện cho "kho thông tin" mà Decoder có thể truy vấn.
- **Kết quả:** Lớp này cho phép mỗi từ trong chuỗi đầu ra chú ý đến tất cả các từ trong chuỗi

đầu vào, giúp mô hình căn chỉnh (align) các từ giữa hai ngôn ngữ một cách hiệu quả.

### 3. Position-wise Feed-Forward Network: Giống hệt như trong Encoder.

Tương tự Encoder, mỗi lớp con của Decoder cũng có Residual Connection và Layer Normalization.

### Huấn luyện và Tối ưu hóa

Kiến trúc Transformer gốc được huấn luyện cho tác vụ dịch máy. Tương tự như mô hình Seq2Seq, mục tiêu là tối thiểu hóa hàm mất mát Cross-Entropy giữa phân phối xác suất đầu ra của Decoder và từ tham chiếu (ground truth).

- **Hàm Mất mát:** Lỗi được tính toán tại mỗi bước sinh từ của Decoder và được tổng hợp lại trên toàn bộ chuỗi.
- **Tối ưu hóa:** Bài báo gốc sử dụng bộ tối ưu hóa Adam [40] với một lịch trình tốc độ học (learning rate schedule) đặc biệt bao gồm một giai đoạn "warmup" (tăng tuyến tính) sau đó là giảm dần theo hàm căn bậc hai nghịch đảo. Lịch trình này được chứng minh là rất quan trọng để huấn luyện Transformer một cách ổn định.
- **Điều chuẩn (Regularization):** Hai kỹ thuật điều chuẩn chính được sử dụng để tránh overfitting là Dropout [86], được áp dụng sau mỗi lớp con (trước kết nối phần dư), và Label Smoothing, một kỹ thuật làm cho các nhãn thật trở nên "mềm" hơn một chút (ví dụ, thay vì  $[0, 0, 1, 0]$ , nhãn có thể là  $[0.025, 0.025, 0.9, 0.025]$ ), giúp mô hình bớt tự tin thái quá và tổng quát hóa tốt hơn.

#### 4.1.3. Positional Encoding: Thêm thông tin về vị trí

Một vấn đề lớn của việc loại bỏ RNN là chúng ta đã làm mất thông tin về **thứ tự của các từ**. Self-Attention, về bản chất, không nhạy cảm với vị trí; nếu ta hoán vị các từ trong câu, ma trận chú ý cũng sẽ bị hoán vị tương ứng mà không thay đổi bản chất.

Để giải quyết vấn đề này, Transformer thêm một vector gọi là **Positional Encoding (Mã hóa Vị trí)** vào các vector embedding đầu vào. Vector này cung cấp thông tin về vị trí tuyệt đối hoặc tương đối của một từ trong chuỗi.

### Absolute Positional Encoding (Sinusoidal)

Đây là phương pháp được sử dụng trong bài báo gốc. Các vector mã hóa vị trí được tạo ra bằng các hàm sin và cos với các tần số khác nhau:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Trong đó  $pos$  là vị trí của từ,  $i$  là chỉ số chiều của vector, và  $d_{\text{model}}$  là số chiều của embedding.

### Tại sao lại dùng sin và cos?

- **Giá trị duy nhất:** Mỗi vị trí có một vector mã hóa duy nhất.
- **Mô hình có thể học quan hệ tương đối:** Có một tính chất toán học quan trọng: mã hóa vị trí của  $pos + k$  có thể được biểu diễn như một hàm tuyến tính của mã hóa vị trí của  $pos$ . Điều này cho phép mô hình dễ dàng học cách chú ý đến các vị trí tương đối, ví dụ "từ cách 2 vị trí về bên phải".
- **Khả năng ngoại suy:** Có thể tạo ra vector mã hóa cho các chuỗi dài hơn những chuỗi đã thấy trong quá trình huấn luyện.

### Relative Positional Encoding

Các phương pháp sau này cho rằng thông tin về khoảng cách tương đối ("từ này cách từ kia bao xa?") quan trọng hơn vị trí tuyệt đối. Các phương pháp mã hóa vị trí tương đối sẽ sửa đổi trực tiếp cơ chế Self-Attention, thêm một số hạng biểu diễn cho khoảng cách tương đối vào lúc tính điểm chú ý.

### Rotary Position Embedding (RoPE)

Đây là một phương pháp mã hóa vị trí tương đối rất thanh lịch và hiệu quả [88], được sử dụng trong nhiều LLM hiện đại như Llama.

- **Ý tưởng:** Thay vì cộng, RoPE xoay (rotates) các vector Query và Key một góc phụ thuộc vào vị trí tuyệt đối của chúng.
- **Hệ quả:** Tích vô hướng giữa hai vector đã được xoay ( $q_m^T k_n$ ) sẽ chỉ phụ thuộc vào vector ban đầu và **khoảng cách tương đối** giữa chúng ( $m - n$ ), chứ không phụ thuộc vào vị trí tuyệt đối  $m, n$ .

RoPE đã chứng tỏ khả năng ngoại suy độ dài ngữ cảnh rất tốt và là một trong những phương pháp mã hóa vị trí hàng đầu hiện nay.

## 4.2. Phân loại Kiến trúc LLMs

Kiến trúc Transformer gốc mà chúng ta vừa phân tích là một mô hình Encoder-Decoder hoàn chỉnh, được thiết kế cho các tác vụ chuỗi-sang-chuỗi (sequence-to-sequence). Tuy nhiên, các nhà nghiên cứu nhanh chóng nhận ra rằng bằng cách chỉ sử dụng một phần của kiến trúc này – hoặc chỉ Encoder, hoặc chỉ Decoder – họ có thể tạo ra các mô hình được chuyên môn hóa cao và cực kỳ hiệu quả cho các loại tác vụ khác nhau.

Sự phân tách này đã khai sinh ra ba "họ" kiến trúc chính, định hình nên toàn bộ bối cảnh của các Mô hình Ngôn ngữ Lớn (LLMs) ngày nay. Hiểu rõ ba họ kiến trúc này là chìa khóa để lựa chọn mô hình phù hợp cho ứng dụng của bạn.

### 4.2.1. Họ Mô hình chỉ Encoder (Encoder-Only): Bậc thầy của Sự Hiểu Ngôn ngữ

Các mô hình thuộc họ này chỉ sử dụng khối Encoder của kiến trúc Transformer. Chúng được thiết kế với một mục tiêu duy nhất: tạo ra các **biểu diễn ngữ cảnh sâu sắc** (deep contextualized representations) cho văn bản đầu vào.

#### Triết lý của Encoder-Only

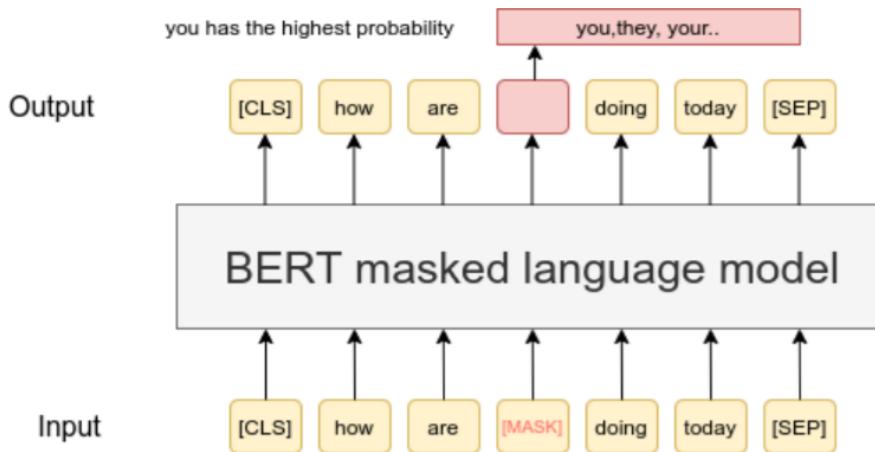
"Để thực sự hiểu ý nghĩa của một từ trong câu, tôi cần phải nhìn vào cả hai phía: những từ đứng trước nó và những từ đứng sau nó."

### Kiến trúc và Mục tiêu Huấn luyện Nền tảng

**Kiến trúc và Tính hai chiều (Bidirectionality)** Đây là đặc điểm quan trọng nhất. Nhờ cơ chế Self-Attention không bị che (unmasked), tại mỗi lớp, mỗi từ có thể "chú ý" đến tất cả các từ khác trong câu, cả bên trái và bên phải. Điều này cho phép mô hình xây dựng một sự hiểu biết toàn diện về ngữ cảnh, điều mà các mô hình hồi tiếp (RNN) hay tự hồi quy (Decoder-Only) không thể làm được. Đầu ra của mô hình là một chuỗi các vector embedding, mỗi vector tương ứng với một token đầu vào, nhưng đã được "làm giàu" bởi thông tin ngữ cảnh hai chiều sâu sắc.

**Mục tiêu Huấn luyện: Masked Language Modeling (MLM)** MLM là mục tiêu huấn luyện cốt lõi của BERT và nhiều mô hình kế nhiệm.

- Cơ chế:** Lấy một câu đầu vào, che (mask) ngẫu nhiên khoảng 15% số token bằng một token đặc biệt '[MASK]'. Sau đó, nhiệm vụ của mô hình là **dự đoán các token gốc đã bị che**, dựa vào các token không bị che xung quanh.
- Tại sao lại hiệu quả?** MLM buộc mô hình phải học các mối quan hệ ngữ nghĩa và cú pháp sâu sắc từ cả hai phía để có thể "điền vào chỗ trống". Nó không chỉ là dự đoán từ tiếp theo, mà là một bài kiểm tra "đọc hiểu" thực sự.



Hình 4.3: Mục tiêu huấn luyện kinh điển của BERT: Masked Language Modeling (MLM). Mô hình phải dự đoán các token bị che dựa trên ngữ cảnh hai chiều.

### Sự phát triển và các biến thể của Họ Encoder-Only

Từ nền tảng của BERT, một hệ sinh thái đa dạng các mô hình Encoder-Only đã ra đời, mỗi loại được tối ưu cho các mục đích khác nhau: hiệu suất, kích thước, tác vụ cụ thể, hay miền kiến thức chuyên biệt.

**1. Gia đình BERT: Những người kế nhiệm trực tiếp** Đây là các mô hình cải tiến trực tiếp từ kiến trúc và phương pháp huấn luyện của BERT.

- RoBERTa (Facebook) [52]:** Không thay đổi kiến trúc, nhưng chứng minh rằng việc tối ưu hóa kỹ lưỡng quá trình huấn luyện có thể mang lại hiệu quả vượt trội. Các cải tiến chính bao gồm: huấn luyện lâu hơn, trên nhiều dữ liệu hơn, với batch size lớn hơn, loại bỏ mục tiêu NSP (Next Sentence Prediction), và sử dụng "che động"(dynamic masking).
- DistilBERT (Hugging Face) [77]:** Một phiên bản "chưng cất" nhỏ hơn, nhanh hơn của BERT. Nó sử dụng kỹ thuật *knowledge distillation*, trong đó một mô hình "học trò"(student) nhỏ hơn học cách bắt chước đầu ra của một mô hình "giáo viên"(teacher) lớn hơn (BERT). DistilBERT giữ lại 97% hiệu năng của BERT nhưng chỉ có 60% kích thước.
- ALBERT (Google) [45]:** Một cách tiếp cận khác để giảm tham số. ALBERT sử dụng hai kỹ thuật chính: *factorized embedding parameterization* (phân tách ma trận embedding từ lớn thành hai ma trận nhỏ) và *cross-layer parameter sharing* (chia sẻ trọng số của các khối Transformer với nhau).
- SpanBERT (Facebook) [38]:** Tối ưu hóa BERT cho các tác vụ cần hiểu các **đoạn văn bản (spans)** liên tục, như Hỏi-đáp trích xuất. Thay vì che các token ngẫu nhiên, SpanBERT che các đoạn ngẫu nhiên và huấn luyện mô hình dự đoán toàn bộ đoạn bị che chỉ từ các token ở biên của đoạn đó.

**2. Các mô hình Biểu diễn Câu: Chuyên gia về Tìm kiếm Ngữ nghĩa** Một hạn chế của BERT gốc là nó không tạo ra các vector biểu diễn câu (sentence embeddings) tốt "ngay lập tức". Việc lấy

embedding của token ‘[CLS]’ hoặc lấy trung bình các token embedding thường cho kết quả tìm kiếm tương đồng không cao. Nhánh mô hình này giải quyết vấn đề đó.

- **SBERT (Sentence-BERT) [73]**: Là mô hình tiên phong. SBERT tinh chỉnh BERT trên các cặp câu bằng cách sử dụng kiến trúc *siamese* hoặc *triplet*. Kiến trúc này buộc mô hình phải tạo ra một không gian vector mà ở đó các câu có ngữ nghĩa giống nhau sẽ có vector gần nhau (cosine similarity cao) và ngược lại.
- **SimCSE (Simple Contrastive Sentence Embedding) [24]**: Một phương pháp học tương phản (contrastive learning) đơn giản nhưng cực kỳ hiệu quả. Để tạo một cặp dương, SimCSE chỉ cần đưa cùng một câu qua mô hình BERT hai lần (với dropout khác nhau). Các câu khác trong batch được coi là cặp âm. Kỹ thuật này tạo ra các embedding câu chất lượng rất cao mà không cần dữ liệu gán nhãn.
- **E5 (Embeddings from bidirectional Encoder Representations [96])**, Contriever, etc.: Là thế hệ mô hình embedding mới nhất, được huấn luyện đặc biệt cho các tác vụ **tìm kiếm (retrieval)**, đặc biệt là tìm kiếm bắt đối xứng (truy vấn ngắn, tài liệu dài). Chúng thường đạt hiệu năng SOTA trên các benchmark tìm kiếm ngữ nghĩa.

**3. Các mô hình cho Miền chuyên biệt và Đa ngôn ngữ** Ý tưởng cốt lõi là một mô hình được huấn luyện trước trên dữ liệu chuyên ngành sẽ hiểu thuật ngữ và ngữ cảnh của ngành đó tốt hơn.

- **SciBERT [4]**, **BioBERT [46]**, **ClinicalBERT [1]**: Được huấn luyện trên các kho văn bản khoa học, y sinh, và y tế lâm sàng.
- **FinBERT**: Chuyên cho lĩnh vực tài chính.
- **LegalBERT**: Chuyên cho lĩnh vực pháp luật.
- **CamemBERT [55] (Pháp)**, **PhoBERT [59] (Việt)**, etc.: Là các mô hình theo kiến trúc RoBERTa nhưng được huấn luyện từ đầu trên kho dữ liệu lớn của một ngôn ngữ cụ thể, cho hiệu năng vượt trội so với các mô hình đa ngôn ngữ trên ngôn ngữ đó.

**4. Các Encoder Hiện đại, Gọn nhẹ và Hiệu quả** Nhánh này tập trung vào việc tạo ra các mô hình nhỏ gọn hơn nhưng vẫn giữ được hiệu năng cao thông qua các cải tiến về kiến trúc và mục tiêu huấn luyện.

- **MiniLM (Microsoft) [97]**: Sử dụng một phương pháp chưng cất tri thức sâu, trong đó mô hình học trò không chỉ bắt chước đầu ra cuối cùng mà còn cả ma trận chú ý và mối quan hệ giữa các vector value từ mô hình giáo viên.
- **DeBERTa (Microsoft) [30]**: Một trong những encoder mạnh nhất hiện nay. DeBERTa giới thiệu cơ chế *disentangled attention*, tách rời việc mã hóa nội dung từ và vị trí tương đối của chúng.
- **MPNet (Microsoft) [84]**: Kết hợp  $\tau\alpha$  tốt nhất của cả hai thế giới: MLM (BERT) và Permutated Language Modeling (XLNet). Nó giải quyết sự không tương thích giữa pre-training và fine-tuning, cho phép mô hình học các phụ thuộc hai chiều một cách toàn diện hơn.

**5. Các Encoder với Mục tiêu Huấn luyện khác MLM** MLM không phải là cách duy nhất để huấn luyện một encoder hai chiều.

- **ELECTRA (Google) [15]**: Sử dụng một tác vụ hiệu quả hơn là *replaced token detection*. Một mô hình “generator” nhỏ sẽ thay thế một số token trong câu, và mô hình “discriminator”(chính là ELECTRA) phải dự đoán xem mỗi token là gốc hay đã bị thay thế. Tác vụ này hiệu quả hơn vì mô hình học từ *tất cả* các token, không chỉ 15.
- **XLNet (Google/CMU) [104]**: Một mô hình *tự hồi quy hoán vị* (*permutation-based autoregressive*). Nó dự đoán các token trong câu theo một thứ tự ngẫu nhiên. Bằng cách này, nó có thể “nhìn thấy” ngữ cảnh từ cả hai phía (giống BERT) trong khi vẫn giữ được các lợi ích của mô hình tự hồi quy.

### Tổng kết: Điểm mạnh và Ứng dụng của Encoder-Only

**Nguyên tắc chung:** Sử dụng mô hình Encoder-Only khi tác vụ của bạn đòi hỏi sự hiểu sâu sắc về toàn bộ văn bản đầu vào và đầu ra không phải là một chuỗi văn bản mới, mạch lạc. Chúng là các mô hình NLU (Natural Language Understanding) chứ không phải NLG (Natural Language Generation).

**Ứng dụng điển hình:**

- **Phân loại văn bản (Text Classification):** Phân tích cảm xúc, phát hiện chủ đề, kiểm duyệt nội dung (toxic detection). Mô hình cần hiểu toàn bộ câu để đưa ra một nhãn duy nhất.
- **Gán nhãn chuỗi (Token Classification):** Nhận dạng Thực thể Tên (NER), Gán nhãn Từ loại (POS Tagging). Mô hình cần hiểu ngữ cảnh xung quanh mỗi từ để gán nhãn cho nó.
- **Tìm kiếm Ngữ nghĩa (Semantic Search):** Biến các câu/tài liệu thành các vector embedding có ý nghĩa (sử dụng các mô hình như SBERT, E5). Các vector này sau đó được lưu trữ và truy vấn bằng các thuật toán tìm kiếm lân cận gần nhất (ANN).
- **Hỏi-đáp trích xuất (Extractive QA):** Tìm và trích xuất một đoạn văn bản (span) làm câu trả lời từ một văn bản ngữ cảnh cho trước.
- **Retrieval-Augmented Generation (RAG):** Đây là một ứng dụng lai. Giai đoạn Retrieval (Truy xuất) trong RAG sử dụng một mô hình Encoder-Only (như E5) để biến truy vấn của người dùng và các tài liệu trong cơ sở tri thức thành vector, sau đó tìm ra các tài liệu liên quan nhất. Các tài liệu này sau đó mới được đưa cho một mô hình sinh (Decoder-Only) để tạo ra câu trả lời.

### 4.2.2. Họ Mô hình chỉ Decoder (Decoder-Only): Bậc thầy của Sự Sáng tạo

Đây là họ kiến trúc đang thống trị thế giới AI hiện nay, là nền tảng của các mô hình như ChatGPT, Llama, và Claude. Chúng chỉ sử dụng khối Decoder của kiến trúc Transformer.

#### Triết lý của Decoder-Only

"Để viết tiếp một câu chuyện, tôi chỉ cần biết những gì đã được viết **trước đó**. Tôi không được phép nhìn vào tương lai."

#### Kiến trúc và Mục tiêu Huấn luyện Nền tảng

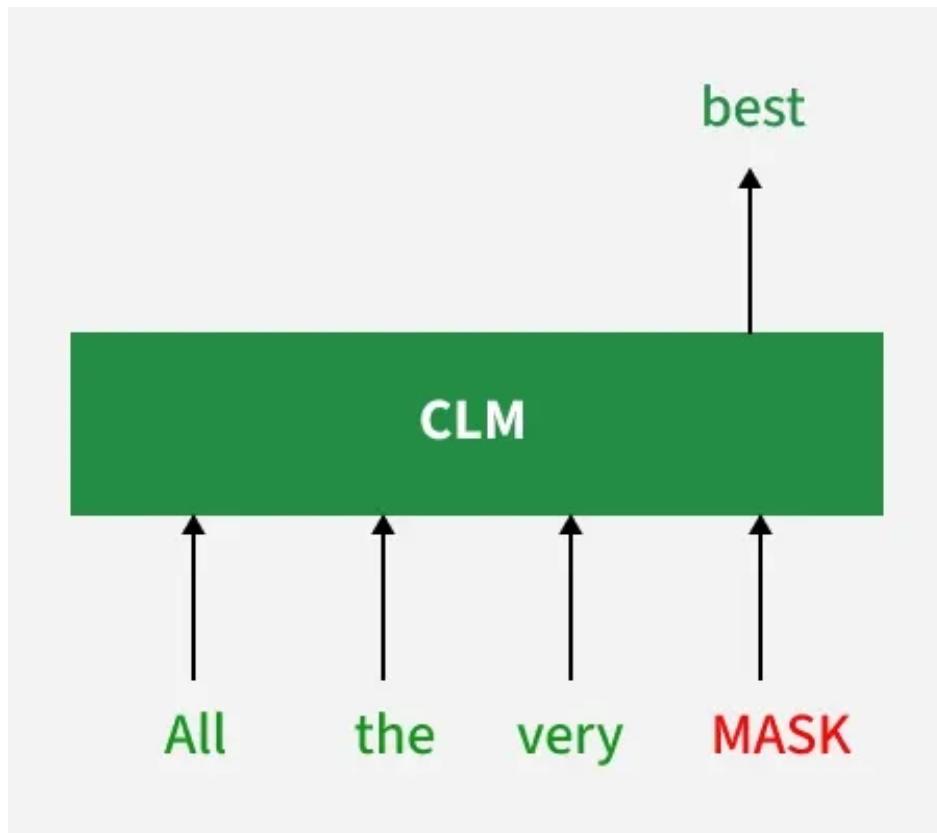
##### Kiến trúc Tự hồi quy (Auto-regressive Architecture)

- **Cấu tạo:** Một chuỗi các khối Decoder của Transformer xếp chồng lên nhau. Quan trọng là, chúng không có Encoder và do đó cũng không có lớp Cross-Attention.
- **Tính một chiều (Unidirectionality):** Đây là đặc điểm nhận dạng. Chúng sử dụng Masked Self-Attention. Tại mỗi vị trí, một từ chỉ có thể "chú ý" đến chính nó và các từ đứng trước nó. Nó không thể nhìn thấy các từ ở phía sau, đảm bảo rằng việc dự đoán chỉ dựa trên thông tin quá khứ.
- **Dòng chảy:** Mô hình nhận vào một chuỗi các từ (prompt) và nhiệm vụ của nó là dự đoán từ tiếp theo có khả năng nhất. Từ được dự đoán sau đó được nối vào chuỗi đầu vào, và quá trình này lặp lại để sinh ra từ tiếp theo, cứ thế tiếp tục.

**Mục tiêu Huấn luyện: Causal Language Modeling (CLM)** Đây là mục tiêu huấn luyện duy nhất và rất đơn giản: **Dự đoán từ tiếp theo (Next Token Prediction)**.

- **Cơ chế:** Mô hình được cho xem một đoạn văn bản và được huấn luyện để, tại mỗi vị trí, dự đoán từ tiếp theo dựa trên tất cả các từ đã có trước đó.

- Sức mạnh của Quy mô: Bằng cách cố gắng dự đoán từ tiếp theo trên một kho văn bản khổng lồ (hàng nghìn tỷ từ), mô hình buộc phải học một lượng kiến thức khổng lồ về thế giới, về ngữ pháp, ngữ nghĩa, và các mẫu hình lý luận để có thể đưa ra dự đoán tốt. Đây là nền tảng cho các "khả năng nổi trội"(emergent abilities) của LLMs.



Hình 4.4: Mục tiêu huấn luyện của GPT: Causal Language Modeling. Tại mỗi bước, mô hình cố gắng dự đoán từ tiếp theo dựa trên các từ trước đó.

### Sự phát triển và các mô hình tiêu biểu

Họ mô hình này đã chứng kiến sự phát triển bùng nổ, từ các mô hình tiên phong đến một hệ sinh thái đa dạng các LLMs mã nguồn mở và đóng.

**1. Gia đình GPT (OpenAI):** Những người tiên phong và dẫn đầu Họ mô hình GPT của OpenAI không chỉ định hình nên kiến trúc Decoder-Only mà còn liên tục đẩy lùi các giới hạn của AI.

- GPT (2018) [69]:** Bài báo gốc chứng minh rằng một mô hình Transformer chỉ-Decoder được huấn luyện trước có thể được tinh chỉnh hiệu quả cho các tác vụ NLU.
- GPT-2 (2019) [70]:** Gây tiếng vang lớn khi cho thấy việc tăng quy mô (lên tới 1.5 tỷ tham số) có thể tạo ra một mô hình có khả năng sinh văn bản mạch lạc đáng kinh ngạc mà không cần tinh chỉnh (zero-shot).
- GPT-3 (2020) [9]:** Mở ra kỷ nguyên LLM thực sự với 175 tỷ tham số. GPT-3 lần đầu tiên thể hiện rõ các khả năng nổi trội (emergent abilities), đặc biệt là **học trong ngữ cảnh (in-context learning)**, cho phép nó thực hiện các tác vụ chỉ bằng cách cung cấp vài ví dụ trong prompt (few-shot).
- InstructGPT & ChatGPT [62]:** Đánh dấu một bước ngoặt từ "mô hình ngôn ngữ" sang "trợ lý AI" bằng cách sử dụng các kỹ thuật căn chỉnh (alignment) như **Instruction Tuning** và **RLHF** (Reinforcement Learning from Human Feedback) để làm cho mô hình hữu ích, trung thực và an toàn hơn.

- **GPT-4 & GPT-4o (2023-2024) [60, 61]**: Là các mô hình SOTA hiện tại, không chỉ mạnh mẽ hơn về ngôn ngữ mà còn trở thành các mô hình **đa phương thức (multimodal)**, có khả năng hiểu và xử lý cả văn bản, hình ảnh, và âm thanh.

**2. Làn sóng LLMs Mã nguồn mở và các đối thủ cạnh tranh** Sự thành công của GPT đã thúc đẩy một làn sóng nghiên cứu và phát triển LLMs trên toàn thế giới, với nhiều mô hình mã nguồn mở mạnh mẽ.

- **LLaMA (Meta) [93]**: Họ mô hình LLaMA (và các phiên bản kế nhiệm LLaMA 2, LLaMA 3) đã "dân chủ hóa" LLMs, cung cấp các mô hình mã nguồn mở có hiệu năng cực kỳ cạnh tranh, tạo điều kiện cho một hệ sinh thái ứng dụng và nghiên cứu khổng lồ.
- **Mistral (Pháp)**: Nổi lên như một đối thủ đáng gờm với các mô hình hiệu năng cao nhưng kích thước nhỏ gọn (ví dụ: Mistral 7B [36]). Họ cũng đi tiên phong trong việc thương mại hóa kiến trúc **Mixture of Experts (MoE)** với mô hình Mixtral, giúp tăng đáng kể số lượng tham số mà không tăng chi phí suy luận tương ứng.
- **Claude (Anthropic)**: Được biết đến với "cửa sổ ngữ cảnh" (context window) rất lớn và tập trung mạnh vào sự an toàn thông qua phương pháp **Constitutional AI** [3].
- **Các mô hình đáng chú ý khác**:
  - **Google**: Gemma (mã nguồn mở), Gemini (mô hình SOTA cạnh tranh với GPT-4).
  - **Cohere**: Command R(+), tập trung vào các ứng dụng doanh nghiệp và Retrieval-Augmented Generation (RAG).
  - **Các mô hình từ Trung Quốc**: Qwen (Alibaba), Yi, Baichuan, InternLM, DeepSeek, thể hiện sự tiến bộ nhanh chóng của cộng đồng AI Trung Quốc.
  - **TII (UAE)**: Falcon.

**3. Các mô hình Decoder-Only chuyên biệt** Bên cạnh các LLM đa dụng, nhiều mô hình được huấn luyện đặc biệt cho các miền cụ thể.

- **Code LLMs**: Các mô hình được huấn luyện chủ yếu trên dữ liệu mã nguồn để hỗ trợ lập trình. Ví dụ: Codex [10] (nền tảng của GitHub Copilot đời đầu), StarCoder [50], CodeLLaMA [75], Phi-3 (Microsoft).
- **Dialogue-focused Models**: Các mô hình được tinh chỉnh đặc biệt cho các tác vụ hội thoại, là tiền thân của các chatbot hiện đại. Ví dụ: DialoGPT (Microsoft) [108], ChatGLM (Trung Quốc).

### Tổng kết: Điểm mạnh và Ứng dụng của Decoder-Only

**Nguyên tắc chung:** Sử dụng mô hình Decoder-Only khi tác vụ của bạn đòi hỏi việc sinh ra một chuỗi văn bản mới, mạch lạc, và có tính sáng tạo dựa trên một ngữ cảnh đầu vào. Chúng là các mô hình NLG (Natural Language Generation) và là nền tảng của AI tạo sinh (Generative AI).

#### Điểm mạnh:

- **Khả năng sinh văn bản xuất sắc:** Là kiến trúc tốt nhất cho mọi tác vụ NLG.
- **Tính linh hoạt (Zero-shot/Few-shot):** Các mô hình lớn có thể thực hiện nhiều tác vụ chỉ thông qua prompt mà không cần fine-tuning, rất linh hoạt và tiết kiệm chi phí gán nhãn.

#### Điểm yếu:

- **Hiểu ngữ cảnh không sâu bằng Encoder-Only:** Do bản chất một chiều, biểu diễn của một từ không được "nhìn thấy" các từ phía sau. Điều này làm chúng kém tối ưu hơn cho các tác vụ NLU thuần túy so với BERT/RoBERTa.
- **Dễ "ảo giác" (Hallucination):** Có xu hướng bịa đặt thông tin một cách tự tin.
- **Quá trình sinh tuân tự là chậm.**

#### Ứng dụng điển hình:

- **Hệ thống Đổi thoại và Trợ lý ảo (Chatbots):** ChatGPT, Claude, Gemini.
- **Sáng tạo nội dung:** Viết văn, làm thơ, soạn email, tóm tắt văn bản (abstractive summarization).
- **Lập trình và Sinh mã (Code Generation):** GitHub Copilot.
- **Giải quyết vấn đề thông qua Prompting:** Bất kỳ tác vụ nào có thể được định dạng dưới dạng "đầu vào -> đầu ra văn bản", từ dịch máy, trả lời câu hỏi, đến lý luận toán học.

### 4.2.3. Họ Mô hình Encoder-Decoder: Sự kết hợp của hai thế giới

Họ mô hình này quay trở lại với kiến trúc Transformer gốc ("Attention is All You Need"), sử dụng cả hai thành phần Encoder và Decoder. Mặc dù không còn là tâm điểm chú ý như các LLM Decoder-Only, chúng vẫn là kiến trúc mạnh mẽ và phù hợp nhất cho nhiều tác vụ chuỗi-sang-chuỗi quan trọng.

#### Triết lý của Encoder-Decoder

"Tôi sẽ dùng một bộ mã hóa hai chiều mạnh mẽ để hiểu trọn vẹn văn bản nguồn, và sau đó dùng một bộ giải mã tự hồi quy để sinh ra văn bản đích dựa trên sự hiểu biết đó."

#### Kiến trúc và Lợi thế Cốt lõi

- **Kiến trúc:** Một Encoder hai chiều (giống BERT) để đọc và hiểu toàn bộ chuỗi đầu vào. Một Decoder một chiều (giống GPT) để sinh ra chuỗi đầu ra. Hai thành phần này được kết nối với nhau thông qua cơ chế **Cross-Attention**, cho phép Decoder "tham khảo" bất kỳ phần nào của chuỗi đầu vào đã được mã hóa ở mỗi bước sinh từ.
- **Lợi thế Cốt lõi:** Sự phân tách vai trò này là một lợi thế không thể thay thế trong các tác vụ đòi hỏi sự hiểu biết sâu sắc về toàn bộ ngữ cảnh nguồn trước khi tạo ra đầu ra. Encoder có thể xây dựng một biểu diễn hai chiều hoàn chỉnh, nắm bắt các mối quan hệ phức tạp trong văn bản nguồn. Decoder sau đó có thể tận dụng biểu diễn phong phú này để tạo ra một đầu ra chính xác và phù hợp hơn.

## Sự phát triển và các mô hình tiêu biểu

Hệ Encoder-Decoder có một lịch sử phát triển phong phú, từ mô hình nền tảng đến các biến thể được tối ưu hóa cao cho các tác vụ chuyên biệt.

### 1. Các mô hình Nền tảng và Đa ngôn ngữ

- **Transformer (Google, 2017) [94]**: Bài báo "Attention Is All You Need" chính là khởi nguồn của tất cả. Nó giới thiệu kiến trúc Encoder-Decoder với Self-Attention và đã thay thế hoàn toàn RNN trong các tác vụ dịch máy SOTA thời bấy giờ.
- **T5 (Google, 2019) [72]**: Phổ biến hóa khung làm việc "Text-to-Text", đổi xử lý mọi bài toán NLP như một tác vụ Seq2Seq, giúp đơn giản hóa và thống nhất quá trình nghiên cứu và ứng dụng.
- **BART (Facebook, 2019) [49]**: Kết hợp các ý tưởng từ BERT (encoder hai chiều) và GPT (decoder tự hồi quy) với mục tiêu huấn luyện khử nhiễu linh hoạt, đặc biệt mạnh mẽ cho các tác vụ sinh văn bản có điều kiện.
- **Các phiên bản Đa ngôn ngữ**: Thành công của các mô hình trên đã dẫn đến các phiên bản đa ngôn ngữ được huấn luyện trên hàng trăm thứ tiếng, bao gồm mT5, mBART, và MarianMT (một mô hình từ Microsoft/Facebook được tối ưu hóa cao cho dịch máy). Gần đây nhất, NLLB (No Language Left Behind) [17] của Meta đã đẩy giới hạn này đi xa hơn, tập trung vào các ngôn ngữ ít tài nguyên.

### 2. Các biến thể với Mục tiêu Huấn luyện và Kiến trúc Sáng tạo

Nhiều mô hình đã cải tiến kiến trúc hoặc mục tiêu huấn luyện để đạt hiệu năng vượt trội trên các tác vụ cụ thể, đặc biệt là tóm tắt văn bản.

- **PEGASUS (Google, 2020) [107]**: Một mô hình cực kỳ mạnh mẽ cho tóm tắt văn bản. Thay vì che các token (MLM) hay các đoạn (T5), PEGASUS có mục tiêu huấn luyện trước là "Gap-Sentences Generation". Nó che đi toàn bộ các câu quan trọng trong một tài liệu và yêu cầu mô hình sinh lại các câu đó từ phần còn lại. Điều này mô phỏng rất gần với tác vụ tóm tắt.
- **ProphetNet (Microsoft, 2020) [67]**: Cải tiến kiến trúc Decoder để có thể dự đoán **nhiều bước** trong tương lai (**future n-gram prediction**) thay vì chỉ một từ tiếp theo. Việc "nhìn xa" này giúp mô hình tránh bị lặp lại và tạo ra các bản tóm tắt chất lượng cao hơn.
- **BERT2BERT [74]**: Một kiến trúc đơn giản nhưng hiệu quả, khởi tạo cả Encoder và Decoder bằng các trọng số của một mô hình BERT đã được huấn luyện trước. Đây là một cách "khởi động nóng" (warm-start) hiệu quả cho các tác vụ Seq2Seq như tóm tắt.
- **ByT5 (Google) [103]**: Một biến thể của T5 xử lý trực tiếp chuỗi các byte UTF-8 thay vì các token. Cách tiếp cận này loại bỏ sự phức tạp và các lỗi tiềm ẩn của quá trình tokenization, giúp mô hình trở nên mạnh mẽ hơn với nhiều và đa ngôn ngữ.

### 3. Hướng tới sự Hợp nhất và Linh hoạt

Các nghiên cứu gần đây tìm cách tạo ra các mô hình linh hoạt hơn, có thể đảm nhận nhiều vai trò khác nhau.

- **UL2 (Google, 2022) [91]**: Giới thiệu một khung làm việc huấn luyện trước **hợp nhất (unified)**. Bằng cách thay đổi các mục tiêu khử nhiễu, UL2 có thể được huấn luyện để hoạt động như một Encoder-Only, Decoder-Only, hoặc Encoder-Decoder, mở ra hướng đi cho các mô hình đa năng.
- **Flan-T5 (Google)**: Là một phiên bản của T5 đã được **tinh chỉnh theo chỉ dẫn (instruction-tuned)** [14] trên hàng trăm tác vụ NLP được định dạng lại. Quá trình này giúp Flan-T5 có khả năng tổng quát hóa zero-shot đáng kinh ngạc cho các tác vụ mới, một khả năng vốn thường thấy ở các mô hình Decoder-Only. Tk-Instruct cũng là một mô hình tương tự được huấn luyện trên một bộ chỉ dẫn còn lớn hơn.

### Khi nào nên sử dụng Họ mô hình Encoder-Decoder?

#### Dánh giá Họ Mô hình Encoder-Decoder

**Nguyên tắc chung:** Sử dụng mô hình Encoder-Decoder khi tác vụ của bạn là một bài toán chuỗi-sang-chuỗi (**sequence-to-sequence**) có điều kiện mạnh mẽ, nơi chất lượng của đầu ra phụ thuộc rất nhiều vào sự hiểu biết sâu sắc và toàn diện về toàn bộ chuỗi đầu vào.

#### Điểm mạnh:

- **Hiệu năng SOTA cho các tác vụ Seq2Seq cổ điển:** Là lựa chọn tự nhiên và thường là tốt nhất cho các tác vụ như dịch máy và tóm tắt văn bản, đặc biệt khi cần độ chính xác cao.
- **Kiểm soát tốt hơn:** Việc tách biệt Encoder và Decoder cho phép kiểm soát tốt hơn quá trình mã hóa thông tin nguồn trước khi sinh ra thông tin đích.

#### Điểm yếu:

- **Phức tạp và nhiều tham số hơn:** Phải huấn luyện cả Encoder và Decoder, làm tăng chi phí tính toán và bộ nhớ.
- **Kém linh hoạt trong các tác vụ zero-shot:** So với các mô hình Decoder-Only lớn, chúng thường không có khả năng "học trong ngữ cảnh" (in-context learning) mạnh mẽ và đòi hỏi phải được tinh chỉnh (fine-tuned) cho từng tác vụ cụ thể (ngoại trừ các phiên bản instruction-tuned như Flan-T5).

#### Ứng dụng điển hình:

- Dịch máy (Machine Translation).
- Tóm tắt văn bản (Summarization).
- Hỏi-đáp sinh (Generative Question Answering).
- Chuyển đổi Dữ liệu-thành-Văn bản (Data-to-text), ví dụ, sinh mô tả từ một bảng dữ liệu có cấu trúc.
- Các tác vụ chỉnh sửa văn bản (Text Editing) và tái cấu trúc câu.

## 4.3. Các Kỹ thuật Tokenization Hiện đại

Trước khi một mô hình Transformer có thể xử lý văn bản, văn bản đó phải được chuyển đổi thành một chuỗi các số nguyên, mỗi số đại diện cho một "token" trong từ vựng của mô hình. Quá trình chia một chuỗi văn bản thành các token này được gọi là **tokenization**.

Lựa chọn tokenizer không phải là một quyết định tầm thường. Nó ảnh hưởng trực tiếp đến:

- **Kích thước từ vựng (Vocabulary Size):** Ảnh hưởng đến kích thước của ma trận embedding và lớp softmax cuối cùng.
- **Độ dài chuỗi (Sequence Length):** Ảnh hưởng đến chi phí tính toán và bộ nhớ.
- **Khả năng xử lý từ hiếm và từ OOV (Out-of-Vocabulary):** Một trong những thách thức lớn nhất.

Các phương pháp tokenization đơn giản như tách từ theo khoảng trắng hoặc theo quy tắc sẽ tạo ra một từ vựng khổng lồ và không thể xử lý các từ mới, từ ghép, hay lỗi chính tả. Để giải quyết vấn đề này, các mô hình ngôn ngữ lớn hiện đại đều sử dụng các thuật toán tokenization dựa trên **subword (dưới từ)**.

#### Triết lý của Subword Tokenization

Ý tưởng cốt lõi là: các từ phức tạp hoặc hiếm có thể được phân rã thành các đơn vị con (subwords) có ý nghĩa và thường xuyên xuất hiện hơn. Bằng cách này, mô hình có thể hiểu và xử lý bất kỳ từ nào, ngay cả những từ chưa từng thấy, bằng cách tổ hợp các subword mà nó đã biết.

### Subword có phải là Hình vị (Morpheme) không?

Các subword được tạo ra bởi các thuật toán thống kê (như ‘re’, ‘play’, ‘ing’ trong ‘replaying’) thường trùng khớp một cách đáng ngạc nhiên với các hình vị (morphemes) – các đơn vị ngôn ngữ nhỏ nhất có ý nghĩa.

- **Tại sao có sự trùng khớp?** Bởi vì các hình vị (tiền tố, hậu tố, gốc từ) xuất hiện lặp đi lặp lại trong nhiều từ khác nhau. Các thuật toán như BPE/WordPiece, vốn được thiết kế để tìm các chuỗi con xuất hiện thường xuyên, sẽ tự động “khám phá” ra các hình vị này một cách tự nhiên mà không cần bất kỳ kiến thức ngôn ngữ học nào.
- **Nhưng chúng không hoàn toàn giống nhau.** Các thuật toán này hoàn toàn dựa trên thống kê. Đôi khi chúng có thể tạo ra các subword không có ý nghĩa về mặt ngôn ngữ học (ví dụ: ‘to’, ‘ken’, ‘ization’ thay vì ‘token’, ‘ization’) nếu sự phân chia đó phổ biến hơn trong kho văn bản.

Do đó, có thể nói rằng các tokenizer subword học được một **sự xấp xỉ dựa trên dữ liệu (data-driven approximation)** của hình thái học ngôn ngữ.

Ví dụ, từ “tokenization” có thể được chia thành các subword như **token** và **##ization**. Từ “hugging” có thể được chia thành **hug** và **##ging**. Dấu **##** (hoặc một ký hiệu đặc biệt khác) cho biết đây là một phần tiếp nối của một từ.

Ba thuật toán subword tokenization chính được sử dụng trong các LLM hiện đại là BPE, WordPiece, và SentencePiece.

#### 4.3.1. Byte-Pair Encoding (BPE)

BPE ban đầu là một thuật toán nén dữ liệu, sau đó được Sennrich và các cộng sự (2015) [81] điều chỉnh cho NLP. Đây là thuật toán nền tảng cho GPT và nhiều mô hình khác.

##### Cơ chế hoạt động: Gộp từ dưới lên (Bottom-up Merging)

BPE hoạt động theo một quy trình lặp đi lặp lại rất trực quan: **tìm cặp token liền kề xuất hiện thường xuyên nhất trong kho văn bản và gộp chúng lại thành một token mới**.

##### Bước 1: Chuẩn bị kho văn bản và Từ vựng ban đầu

1. **Tiền xử lý:** Lấy một kho văn bản lớn (corpus).
2. **Tách từ:** Tách kho văn bản thành các từ, thường theo khoảng trắng. Thống kê tần suất của mỗi từ.
3. **Phân rã thành ký tự:** Tách mỗi từ thành một chuỗi các ký tự. Thêm một ký hiệu kết thúc từ đặc biệt (ví dụ: ‘</w>’) vào cuối mỗi từ. Điều này giúp mô hình biết được ranh giới của từ gốc. Ví dụ: từ ”học” có tần suất 5 lần sẽ trở thành ‘(‘h’, ’o’, ’c’, ‘</w>’): 5’.
4. **Từ vựng ban đầu:** Từ vựng ban đầu bao gồm tất cả các ký tự đơn lẻ xuất hiện trong kho văn bản.

##### Bước 2: Học các quy tắc gộp (Learn Merges) Lặp lại một số lần định trước (ví dụ: 30,000 lần), mỗi lần lặp thực hiện:

1. **Tìm cặp phổ biến nhất:** Duyệt qua toàn bộ kho văn bản (đã được tách thành ký tự) và tìm ra cặp ký tự/token liền kề nào có tần suất xuất hiện cao nhất.
2. **Gộp cặp:** Gộp cặp đó lại thành một token mới.
3. **Thêm vào từ vựng:** Thêm token mới này vào từ vựng.
4. **Cập nhật kho văn bản:** Thay thế tất cả các lần xuất hiện của cặp đã gộp trong kho văn bản bằng token mới.

### Ví dụ 10: Minh họa quá trình học BPE

Giả sử kho văn bản của chúng ta sau khi tiền xử lý có dạng: ‘(‘h’, ’o’, ’c’, ’</w>’): 5, (‘đ’, ’o’, ’c’, ’</w>’): 3, (‘h’, ’o’, ’i’, ’</w>’): 4 ’

Từ vựng ban đầu: ‘h, o, c, </w>, đ, ô, i’

Lần lặp 1:

- Cặp ‘(o, c)’ xuất hiện trong ”học”(5 lần) và ”đọc”(3 lần). Tổng cộng 8 lần. Đây là cặp phổ biến nhất.
- **Gộp:** ‘(o, c)’ → ‘oc’
- **Từ vựng mới:** ‘..., oc’
- **Cập nhật kho văn bản:** ‘(‘h’, ’oc’, ’</w>’): 5, (‘đ’, ’oc’, ’</w>’): 3, (‘h’, ’ô’, ’i’, ’</w>’): 4 ’

Lần lặp 2:

- Cặp ‘(h, oc)’ xuất hiện 5 lần. Cặp ‘(oc, </w>)’ xuất hiện 8 lần. Cặp ‘(đ, oc)’ xuất hiện 3 lần... Cặp ‘(oc, </w>)’ là phổ biến nhất.
- **Gộp:** ‘(oc, </w>)’ → ‘oc</w>’
- **Từ vựng mới:** ‘..., oc, oc</w>’
- **Cập nhật kho văn bản:** ‘(‘h’, ’oc</w>’): 5, (‘đ’, ’oc</w>’): 3, (‘h’, ’ô’, ’i’, ’</w>’): 4 ’

Quá trình này tiếp tục. Cuối cùng, từ vựng sẽ chứa các token từ các ký tự đơn lẻ (‘h’, ‘i’, ...), các subword phổ biến (‘oc’, ‘oc</w>’), và có thể cả các từ hoàn chỉnh nếu chúng đủ phổ biến (‘học</w>’).

**Bước 3: Tokenize một câu mới** Sau khi đã học được một từ vựng và một danh sách các quy tắc gộp theo thứ tự ưu tiên, để tokenize một câu mới:

1. Tách câu thành các ký tự.
2. Áp dụng các quy tắc gộp đã học theo đúng thứ tự ưu tiên cho đến khi không thể gộp được nữa.

### Cái tiến quan trọng: Byte-Level BPE (BBPE)

Các mô hình hiện đại như GPT-2/3/4 và Llama không áp dụng BPE trên các ký tự Unicode mà trên các byte.

- **Cơ chế:** Thay vì xem xét các ký tự (‘a’, ‘b’, ‘á’, ‘à’, ...), BBPE hoạt động trên chuỗi các byte UTF-8 cấu thành nên các ký tự đó.
  - **Từ vựng ban đầu cực nhỏ:** Toàn bộ các ký tự trên thế giới có thể được biểu diễn chỉ bằng 256 byte. Do đó, từ vựng ban đầu của BBPE chỉ có 256 token.
  - **Lợi ích vượt trội:**
    1. **Không bao giờ có token ”Unknown”:** Bất kỳ chuỗi văn bản nào, dù là ngôn ngữ lạ, emoji (❖❖), hay nhiều ngẫu nhiên, đều có thể được biểu diễn bằng một chuỗi các byte. Mô hình sẽ không bao giờ gặp một ký tự ”out-of-vocabulary” ở cấp độ cơ bản nhất.
    2. **Không cần xử lý Unicode phức tạp:** Mô hình không cần quan tâm đến các quy tắc chuẩn hóa Unicode khác nhau, vì nó chỉ làm việc với byte.
    3. **Hiệu quả cho đa ngôn ngữ:** Cùng một tokenizer có thể xử lý hiệu quả nhiều ngôn ngữ mà không cần phải có một từ vựng ban đầu khổng lồ chứa tất cả các ký tự có thể có.
- BBPE là một lựa chọn thiết kế thanh lịch và mạnh mẽ, đã trở thành tiêu chuẩn cho nhiều LLM hàng đầu hiện nay.

#### 4.3.2. WordPiece

WordPiece là một thuật toán tương tự BPE, được phát triển tại Google và sử dụng trong các mô hình như BERT và RoBERTa [102].

### Sự khác biệt chính với BPE: Tiêu chí gộp

Sự khác biệt cốt lõi nằm ở cách nó quyết định cặp nào sẽ được gộp.

- BPE gộp cặp có **tần suất cao nhất**.
- WordPiece gộp cặp có khả năng tối đa hóa "khả năng hợp lý" (likelihood) của dữ liệu huấn luyện nếu chúng ta xem việc tokenization là một mô hình ngôn ngữ.

**Cơ chế hoạt động** WordPiece cũng bắt đầu với một từ vựng gồm các ký tự đơn lẻ. Ở mỗi bước, nó xem xét việc gộp hai token liền kề (ví dụ 'A' và 'B' thành 'AB') và tính toán xem sự thay đổi này làm tăng "khả năng hợp lý" của toàn bộ kho văn bản lên bao nhiêu.

$$\text{score}(A, B) = \frac{\text{count}(AB)}{\text{count}(A) \times \text{count}(B)}$$

Nó sẽ chọn cặp có điểm số (score) cao nhất để gộp. Tiêu chí này ưu tiên các cặp mà các thành phần của nó ít có khả năng xuất hiện độc lập, tức là chúng "thực sự thuộc về nhau".

Trong thực tế, WordPiece thường tạo ra các token có dạng `##` ở đầu (ví dụ `##ization`) để biểu thị các subword không phải là bắt đầu của một từ.

#### 4.3.3. SentencePiece

Được phát triển bởi Google, SentencePiece [43] không phải là một thuật toán tokenization mới, mà là một **thư viện phần mềm** cung cấp một cách triển khai hiệu quả của cả BPE và một thuật toán khác gọi là **Unigram Language Model**, đồng thời giải quyết một số vấn đề thực tiễn.

### Các cải tiến chính

**1. Hoạt động trực tiếp trên văn bản thô** BPE và WordPiece đều yêu cầu văn bản phải được tiền xử lý và tách từ theo khoảng trắng trước. Điều này tạo ra một vấn đề: làm thế nào để xử lý khoảng trắng? Khoảng trắng có phải là một token không? Điều này đặc biệt phức tạp với các ngôn ngữ không dùng khoảng trắng như tiếng Nhật hay tiếng Trung.

SentencePiece giải quyết vấn đề này bằng cách coi khoảng trắng chỉ là một ký tự bình thường. Nó hoạt động trực tiếp trên chuỗi Unicode thô. Ký tự khoảng trắng được thay thế bằng một meta-symbol đặc biệt, ví dụ '`_`' (dấu gạch dưới).

- **Ví dụ:** 'Hello World' → '\_Hello \_World'
- **Lợi ích:** Toàn bộ quá trình tokenization và de-tokenization là hoàn toàn có thể đảo ngược (fully reversible) mà không có bất kỳ sự mơ hồ nào về khoảng trắng.

**2. Tokenization xác suất và Subword Regularization** Ngoài BPE, SentencePiece còn triển khai mô hình Unigram, mang đến một cách tiếp cận linh hoạt hơn. Không giống như BPE và WordPiece chỉ có một cách duy nhất để tokenize một chuỗi, mô hình Unigram có thể tạo ra **nhiều cách tokenization khác nhau cho cùng một chuỗi**, mỗi cách có một xác suất.

- **Cơ chế học (Top-down):** Ngược lại với BPE, mô hình Unigram bắt đầu với một từ vựng rất lớn (ví dụ, tất cả các subword có thể có từ thuật toán Suffix Array). Sau đó, nó sử dụng thuật toán EM (Expectation-Maximization) để lặp đi lặp lại việc ước tính xác suất của mỗi token và loại bỏ các token làm giảm ít nhất "khả năng hợp lý" (likelihood) của toàn bộ kho văn bản, cho đến khi từ vựng đạt kích thước mong muốn.

- **Cơ chế tokenize:**
  - **Chế độ tốt nhất (Best):** Sử dụng thuật toán Viterbi để tìm ra chuỗi token có xác suất cao nhất. Ví dụ, 'Hello World' có thể được tokenize thành '\_Hello', '\_World'.
  - **Chế độ lấy mẫu (Sampling):** Lấy mẫu một cách tokenization từ phân phối xác suất. Ví dụ, cùng câu 'Hello World' có thể được tokenize thành '\_H', 'ell', 'o', '\_W', 'or', 'ld' trong một lần khác.

- Lợi ích (Subword Regularization):** Việc sử dụng chế độ lấy mẫu trong quá trình huấn luyện mô hình ngôn ngữ được gọi là **subword regularization**. Bằng cách cho mô hình thấy nhiều cách phân rã khác nhau của cùng một từ, kỹ thuật này hoạt động như một dạng data augmentation, giúp mô hình trở nên mạnh mẽ hơn trước sự nhiễu và các biến thể hình thái, từ đó cải thiện khả năng tổng quát hóa.

#### Tổng kết các Kỹ thuật Tokenization Hiện đại

Tiêu chí	BPE	WordPiece	SentencePiece (Unigram)
<b>Chiến lược</b>	Bottom-up (gộp)	Bottom-up (gộp)	Top-down (loại bỏ)
<b>Tiêu chí gộp/giữ</b>	Tần suất cặp cao nhất	Tối đa hóa likelihood	Giữ lại token để tối đa hóa likelihood
<b>Dầu ra</b>	Duy nhất (deterministic)	Duy nhất (deterministic)	Có thể là xác suất (probabilistic)
<b>Xử lý khoảng trắng</b>	Yêu cầu tiền xử lý	Yêu cầu tiền xử lý	Coi là ký tự bình thường ( <u>_</u> )
<b>Sử dụng bởi</b>	GPT, RoBERTa, Llama 2	BERT, DistilBERT	T5, ALBERT, Llama 1

## 4.4. Các Biến thể Transformer cho Ngữ cảnh dài (Long-Context Transformers)

### 4.4.1. Vấn đề của Sự chú ý Bậc hai (The Quadratic Bottleneck)

Kiến trúc Transformer gốc, với cơ chế Self-Attention đầy đủ (full self-attention), có một hạn chế nghiêm trọng: **chi phí tính toán và bộ nhớ tăng theo cấp số nhân bậc hai so với độ dài chuỗi** ( $O(n^2)$ ).

Hãy nhớ lại công thức của Self-Attention:  $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ . Phép nhân ma trận  $QK^T$  sẽ tạo ra một **ma trận chú ý (attention matrix)** có kích thước  $n \times n$ , trong đó  $n$  là độ dài chuỗi.

- Nếu  $n = 512$ , ma trận chú ý có  $512^2 \approx 262,000$  phần tử.
- Nếu  $n = 4096$ , ma trận chú ý có  $4096^2 \approx 16.7$  triệu phần tử.
- Nếu  $n = 100,000$  (độ dài một cuốn sách nhỏ), ma trận chú ý sẽ có 10 tỷ phần tử!

Việc lưu trữ và tính toán trên ma trận khổng lồ này là bất khả thi với các phần cứng hiện tại. Đây chính là "nút thắt cổ chai bậc hai", giới hạn hầu hết các mô hình Transformer ban đầu ở độ dài ngữ cảnh là 512 hoặc 1024 token.

Để phá vỡ rào cản này và xử lý các tài liệu dài, các nhà nghiên cứu đã phát triển nhiều phương pháp để làm cho cơ chế chú ý trở nên "hiệu quả" hơn. Hướng tiếp cận phổ biến nhất là **Chú ý Thưa thớt (Sparse Attention)**.

#### Giải pháp từ Kỹ thuật Tối ưu: FlashAttention

Trước khi đi vào các phương pháp xấp xỉ, cần phải nhắc đến một cuộc cách mạng về mặt triển khai: **FlashAttention** (Dao et al., 2022)[18]. FlashAttention không thay đổi kiến trúc Transformer, mà thay đổi cách tính toán cơ chế chú ý trên phần cứng GPU.

- Vấn đề cốt lõi:** Nút thắt cổ chai của attention không chỉ là số phép tính  $O(n^2)$ , mà còn là việc truy cập bộ nhớ. Việc đọc và ghi ma trận chú ý  $n \times n$  khổng lồ từ bộ nhớ chính (HBM) của GPU ra bộ nhớ đệm (SRAM) cực kỳ chậm.
- Giải pháp của FlashAttention:** Nó sử dụng các kỹ thuật thông minh như **tiling** (chia nhỏ ma trận thành các khối) và **recomputation** (tính toán lại một số giá trị thay vì lưu trữ) để thực hiện toàn bộ phép tính chú ý trong bộ nhớ SRAM tốc độ cao của GPU mà không cần phải ghi toàn bộ ma trận chú ý ra HBM.

- Kết quả:** FlashAttention có thể tính toán cơ chế chú ý chính xác (exact attention) nhanh hơn tới 3 lần và yêu cầu bộ nhớ chỉ  $O(n)$  thay vì  $O(n^2)$ .

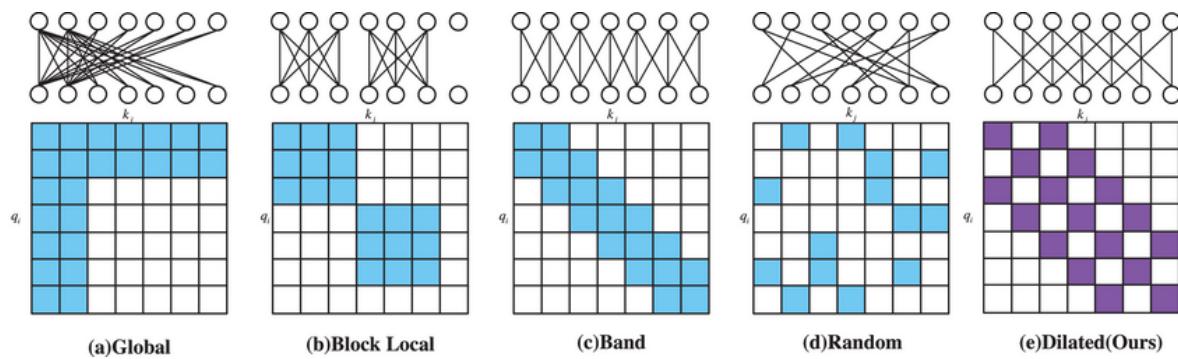
Sự ra đời của FlashAttention và các phiên bản kế nhiệm (FlashAttention-2) đã giúp các mô hình Transformer gốc có thể xử lý ngữ cảnh dài hơn đáng kể (ví dụ, từ 2k lên 8k, 16k, hoặc 32k) mà không cần phải thay đổi kiến trúc. Nó là một trong những nền tảng kỹ thuật quan trọng nhất cho các LLM hiện đại.

#### 4.4.2. Lý thuyết về Chú ý Thưa thoát (Sparse Attention)

##### Trực giác cốt lõi

Ý tưởng trung tâm của Sparse Attention là: **một token không cần phải chú ý đến tất cả các token khác trong chuỗi**. Hầu hết thông tin quan trọng đều đến từ các token lân cận (ngữ cảnh cục bộ) hoặc một vài token "quan trọng" đặc biệt trên toàn chuỗi.

Thay vì tính toán một ma trận chú ý  $n \times n$  dày đặc, các mô hình Sparse Attention chỉ tính toán và lưu trữ một số lượng nhỏ các điểm chú ý, làm cho chi phí giảm từ  $O(n^2)$  xuống gần như tuyến tính,  $O(n \log n)$  hoặc  $O(n)$ .



Hình 4.5: So sánh các mẫu chú ý thưa (sparse attention). (a) Chú ý toàn cục (Global). (b) Chú ý theo khối cục bộ (Block Local). (c) Chú ý theo dải (Band), hay còn gọi là cửa sổ trượt (Sliding Window). (d) Chú ý ngẫu nhiên (Random). (e) Chú ý giãn cách (Dilated). Các mô hình như Longformer và BigBird thường kết hợp các mẫu này.

Hai mô hình tiên phong và tiêu biểu nhất cho trường phái này là Longformer và BigBird.

##### Longformer: Kết hợp Cửa sổ trượt và Chú ý Toàn cục

Longformer (Beltagy et al., 2020) [5] đề xuất một mẫu chú ý kết hợp ba thành phần:

###### 1. Chú ý Cửa sổ trượt (Sliding Window Attention)

- Cơ chế:** Mỗi token chỉ chú ý đến một cửa sổ có kích thước cố định  $w$  gồm các token xung quanh nó (ví dụ:  $w/2$  token bên trái và  $w/2$  token bên phải).
- Mục đích:** Nắm bắt ngữ cảnh cục bộ, tương tự như trong các mô hình CNN. Đây là phần lớn các kết nối chú ý trong mô hình.

###### 2. Chú ý Cửa sổ trượt Giãn cách (Dilated Sliding Window)

- Cơ chế:** Để mở rộng "trường tiếp nhận" (receptive field) mà không tăng chi phí, Longformer sử dụng các cửa sổ trượt có "hở" (gaps) ở các lớp khác nhau. Ví dụ, ở một lớp, cửa sổ có thể chú ý đến các token cách nhau 1 vị trí, ở lớp khác, cách nhau 2 vị trí, v.v.
- Mục đích:** Giúp thông tin từ các token xa hơn có thể được tích hợp vào biểu diễn của một token mà không cần kết nối trực tiếp.

### 3. Chú ý Toàn cục (Global Attention)

- Cơ chế:** Một số ít các token được chọn trước được phép chú ý đến tất cả các token khác trong chuỗi, và ngược lại, tất cả các token khác cũng được phép chú ý đến chúng.
- Mục đích:** Các token toàn cục này hoạt động như một "trung tâm thông tin", tổng hợp và phân phối thông tin trên toàn bộ chuỗi. Các token này thường được chọn một cách đặc biệt, ví dụ như token '[CLS]' trong các tác vụ phân loại.

Bằng cách kết hợp các mẫu này, Longformer giảm độ phức tạp tính toán xuống  $O(n \cdot w)$ , gần như tuyến tính với độ dài chuỗi  $n$ , cho phép nó xử lý các chuỗi lên tới 4096 token hoặc hơn.

### BigBird: Thêm các Mẫu Chú ý Ngẫu nhiên

BigBird (Zaheer et al., 2020) [106] có triết lý tương tự Longformer nhưng bổ sung thêm một thành phần:

- Nó cũng sử dụng **Chú ý Cửa sổ trượt** và **Chú ý Toàn cục**.
- Điểm mới lạ là nó thêm vào **Chú ý Ngẫu nhiên (Random Attention)**. Mỗi token sẽ chú ý đến một số lượng nhỏ  $r$  các token được chọn ngẫu nhiên từ toàn bộ chuỗi.

Các tác giả đã chứng minh về mặt lý thuyết rằng một mô hình kết hợp ba loại chú ý này (cửa sổ, toàn cục, ngẫu nhiên) có thể xấp xỉ được các thuộc tính của cơ chế chú ý đầy đủ và là một bộ xấp xỉ hàm phổ quát (universal function approximator).

#### 4.4.3. Các phương pháp khác: Thoát khỏi Self-Attention Bậc hai

Sparse Attention là một hướng tiếp cận, nhưng không phải là duy nhất. Một hướng đi khác, cấp tiến hơn, là đặt câu hỏi: "Liệu chúng ta có thực sự cần cơ chế chú ý bậc hai không?". Điều này đã dẫn đến sự ra đời của các kiến trúc hoàn toàn mới.

### Cải tiến Mã hóa Vị trí (Positional Encoding)

Một hướng quan trọng khác để giúp Transformer xử lý ngữ cảnh dài là cải tiến cách nó hiểu về vị trí của token.

**ALiBi (Attention with Linear Biases)** Thay vì thêm mã hóa vị trí vào embedding, ALiBi [66] trực tiếp sửa đổi ma trận điểm chú ý.

- Cơ chế:** Trước khi qua softmax, điểm chú ý giữa token  $i$  và  $j$  sẽ bị trừ đi một "án phạt"(penalty) tỷ lệ thuận với khoảng cách của chúng:  $\text{score}(i, j) - m \cdot |i - j|$ . Trong đó,  $m$  là một hệ số học được cho mỗi "đầu" chú ý.
- Trực giác:** "Các từ càng xa nhau thì càng ít liên quan".
- Lợi ích:** ALiBi cho thấy khả năng ngoại suy (extrapolation) ấn tượng. Một mô hình được huấn luyện trên ngữ cảnh 2048 token với ALiBi có thể hoạt động tốt trên ngữ cảnh 4096 token hoặc dài hơn mà không cần fine-tuning.

**RoPE Scaling** Các mô hình sử dụng Rotary Position Embedding (RoPE) như Llama thường gặp khó khăn khi ngoại suy ra các chuỗi dài hơn độ dài huấn luyện. Các kỹ thuật RoPE Scaling giải quyết vấn đề này.

- Vấn đề:** RoPE sử dụng các hàm sin/cos với tần số phụ thuộc vào vị trí. Khi vị trí vượt quá giới hạn huấn luyện, các tần số này bắt đầu lặp lại hoặc tạo ra các mẫu không ổn định.
- Giải pháp (ví dụ:[92], YaRN [63]):** Các kỹ thuật này sửa đổi "buốt sóng"(wavelength) của các hàm sin/cos, về cơ bản là "kéo dãn" không gian vị trí để có thể chứa các chuỗi dài hơn mà không phá vỡ các mối quan hệ tương đối đã học được ở cự ly gần.

### Mô hình Trạng thái (State Space Models - SSMs) và Mamba

Đây là một trong những hướng nghiên cứu thú vị và hứa hẹn nhất gần đây.

**Nguồn gốc từ Lý thuyết Điều khiển** SSMs có nguồn gốc từ lý thuyết điều khiển và xử lý tín hiệu. Chúng được thiết kế để mô hình hóa các hệ thống động liên tục. Ý tưởng cốt lõi là duy trì một **trạng thái ẩn (state)  $h(t)$**  nhỏ, có kích thước cố định, và cập nhật nó một cách liên tục theo thời gian dựa trên đầu vào  $x(t)$ .

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$

$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$$

Trong đó  $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$  là các ma trận tham số.

**Thách thức và Giải pháp** Việc áp dụng trực tiếp SSMs vào học sâu là rất khó khăn về mặt tính toán. Các mô hình như S4 (Structured State Space for Sequence Modeling) [28] đã giải quyết vấn đề này bằng cách tìm ra một cách hiệu quả để "rời rạc hóa" (discretize) các công thức liên tục này, cho phép chúng được tính toán như một phép tích chập lớn trong quá trình huấn luyện (rất nhanh và song song) và như một RNN trong quá trình suy luận (hiệu quả về bộ nhớ).

**Mamba: SSMs có Chọn lọc (Selective SSMs)** Mamba (Gu & Dao, 2023) [27] là một bước đột phá lớn dựa trên SSMs. Nó nhận ra rằng một hạn chế của các SSMs trước đó là các ma trận  $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$  là **cố định**, không phụ thuộc vào đầu vào. Điều này có nghĩa là mô hình không thể thay đổi hành vi của mình dựa trên nội dung cụ thể mà nó đang xử lý.

Mamba đã giải quyết vấn đề này bằng cách làm cho các ma trận  $\mathbf{B}, \mathbf{C}$  và một tham số bước  $\Delta$  **phụ thuộc vào đầu vào (input-dependent)**.

**So sánh SSMs và RNNs truyền thống** Ở chế độ suy luận, SSM hoạt động tuần tự giống như một RNN. Tuy nhiên, có một sự khác biệt cốt lõi:

- **RNNs (LSTM/GRU):** Việc cập nhật trạng thái ẩn đi qua các cổng phi tuyến phức tạp (sigmoid, tanh). Các cổng này, mặc dù mạnh mẽ, lại làm cho gradient khó chảy ngược qua các chuỗi dài và khó song song hóa trong quá trình huấn luyện.
- **SSMs:** Việc cập nhật trạng thái về bản chất là **tuyến tính** ( $h'(t) = \mathbf{A}h(t) + \dots$ ). Bản chất tuyến tính này, kết hợp với các ma trận được cấu trúc đặc biệt, cho phép gradient chảy ngược qua các chuỗi rất dài mà không bị triệt tiêu, đồng thời cho phép nó được biến đổi thành một phép tích chập song song hóa được khi huấn luyện.

Mamba kế thừa những ưu điểm này và thêm vào đó khả năng chọn lọc thông tin, tạo ra một kiến trúc vừa hiệu quả vừa mạnh mẽ.

### Cơ chế Chọn lọc của Mamba

Mamba có khả năng **chọn lọc** thông tin. Dựa trên token đầu vào hiện tại, nó có thể quyết định:

1. **Tập trung:** Nếu một thông tin quan trọng xuất hiện, nó có thể "mở cổng" để cho thông tin đó đi vào trạng thái ẩn.
2. **Lãng quên:** Nếu thông tin hiện tại không còn liên quan, nó có thể "nén" hoặc "quên" đi trạng thái ẩn cũ và đặt lại.

Khả năng chọn lọc này cho phép Mamba néng ngữ cảnh một cách linh hoạt, chỉ giữ lại những gì cần thiết, một khả năng mà các mô hình tích chập hay RNN tuyến tính trước đó không có được.

### Ưu điểm của Mamba so với Transformer

- **Độ phức tạp tuyến tính:** Cả huấn luyện và suy luận đều có độ phức tạp  $O(n)$ , nhanh hơn nhiều so với  $O(n^2)$  của Transformer.
- **Suy luận nhanh:** Tốc độ suy luận (sinh từ mới) nhanh hơn Transformer vì nó không cần tính toán lại ma trận chú ý khổng lồ ở mỗi bước.

Mamba và các kiến trúc dựa trên SSMs đang nổi lên như một đối thủ cạnh tranh thực sự với Transformer, đặc biệt là trong các kịch bản yêu cầu ngữ cảnh cực dài và hiệu quả tính toán cao.

### Các hướng tiếp cận khác

Ngoài ra, còn có các hướng tiếp cận khác như:

- **Tích chập có cửa sổ lớn (Large Kernel Convolutions):** Sử dụng các lớp tích chập 1D nhưng với kích thước kernel rất lớn để mô phỏng sự tương tác tầm xa.
- **Phương pháp đệ quy (Recurrence):** Cố gắng đưa trở lại một số dạng của cơ chế RNN để nén thông tin từ các đoạn (chunks) văn bản dài.

Việc xử lý ngữ cảnh dài vẫn là một trong những lĩnh vực nghiên cứu năng động nhất, hứa hẹn sẽ mở khóa nhiều khả năng mới cho các mô hình ngôn ngữ trong tương lai.

## 4.5. Lý thuyết về Mixture-of-Experts (MoE)

Khi chúng ta nói về việc "tăng quy mô"(scaling up) các mô hình ngôn ngữ, cách tiếp cận thông thường là làm cho các mô hình trở nên "dày đặc" hơn (denser) - tức là tăng số lớp, tăng số chiều ẩn, tăng số đầu chú ý. Tuy nhiên, cách tiếp cận này có một giới hạn: để xử lý mỗi token, **toàn bộ** các tham số của mô hình đều phải được kích hoạt. Điều này dẫn đến chi phí tính toán cực kỳ lớn ở cả quá trình huấn luyện và suy luận.

Mixture-of-Experts (MoE) [83, 22] đưa ra một giải pháp thanh lịch cho vấn đề này, dựa trên một triết lý rất con người:

### Triết lý của Mixture-of-Experts

"Không cần phải huy động toàn bộ bộ não để giải quyết mọi vấn đề. Thay vào đó, hãy có một hội đồng gồm nhiều chuyên gia (experts), mỗi người chuyên về một lĩnh vực khác nhau. Đối với mỗi vấn đề cụ thể, chỉ cần hỏi ý kiến của một vài chuyên gia phù hợp nhất."

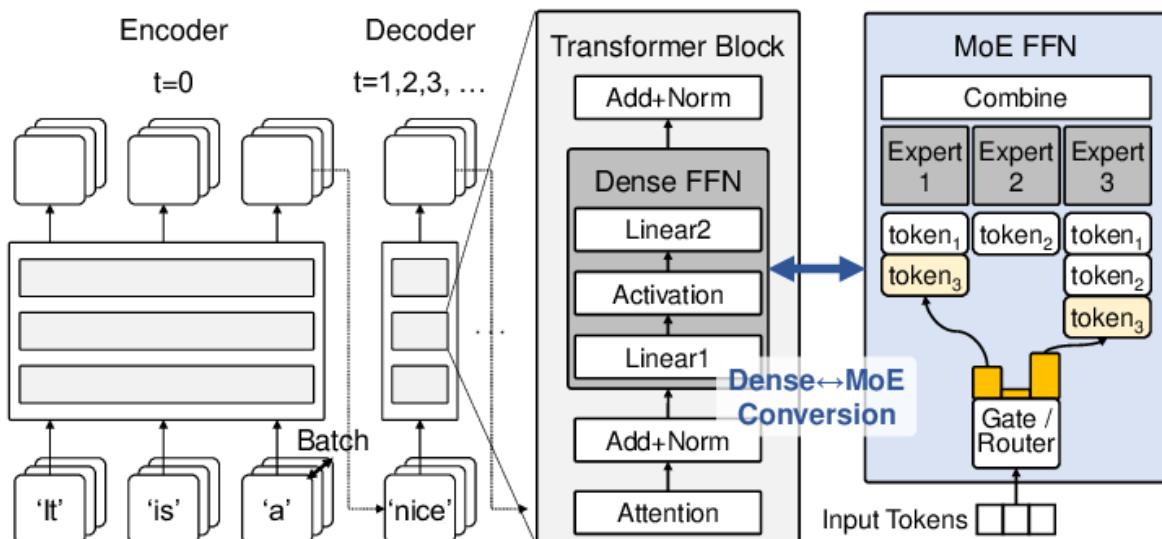
Trong bối cảnh của Transformer, MoE cho phép tăng số lượng tham số của mô hình lên rất lớn, nhưng lại giữ cho chi phí tính toán trên mỗi token không đổi hoặc chỉ tăng nhẹ. Đây được gọi là **tăng quy mô thưa thớt (sparse scaling)**.

### 4.5.1. Kiến trúc và Nguyên lý hoạt động

Kiến trúc MoE không phải là một mô hình hoàn toàn mới. Thay vào đó, nó là một cách để sửa đổi các khối Transformer hiện có. Cụ thể, trong một khối Transformer tiêu chuẩn, lớp Feed-Forward Network (FFN) được thay thế bằng một lớp MoE.

Một lớp MoE bao gồm hai thành phần chính:

1. **Một tập hợp gồm  $N$  mạng FFN độc lập, gọi là các "chuyên gia"(Experts).** Ví dụ,  $N$  có thể là 8, 16, hoặc 64. Mỗi chuyên gia này có bộ trọng số riêng.
2. **Một mạng cổng (Gating Network) nhỏ.** Đây là một mạng nơ-ron có nhiệm vụ quyết định xem nên gửi mỗi token đến chuyên gia nào.



Hình 4.6: So sánh một khối Transformer tiêu chuẩn (trái) và một khối Transformer với lớp MoE (phải). Lớp FFN dày đặc được thay thế bằng một mạng cổng và nhiều chuyên gia FFN thừa thớt.

### Dòng chảy dữ liệu của một token

Khi một token (được biểu diễn bằng vector  $x$ ) đi vào lớp MoE, quá trình sau sẽ xảy ra:

#### Bước 1: Định tuyến bởi Mạng cổng (Routing by the Gating Network)

- Token  $x$  được đưa vào mạng cổng.
- Mạng cổng, thường là một lớp tuyến tính đơn giản, sẽ xuất ra một vector logits có số chiều bằng số chuyên gia ( $N$ ).
- Một hàm ‘softmax’ được áp dụng lên vector logits này để tạo ra một phân phối xác suất,  $G(x) = (g_1, g_2, \dots, g_N)$ , trong đó  $g_i$  là “điểm số” hoặc “mức độ tin cậy” mà mạng cổng dành cho chuyên gia thứ  $i$ .

#### Bước 2: Chọn các Chuyên gia "Top-K"

- Thay vì gửi token đến tất cả các chuyên gia (điều này sẽ làm mất đi tính hiệu quả), hệ thống chỉ chọn ra  $k$  chuyên gia có điểm số cao nhất (Top-K routing). Thông thường,  $k$  là một số rất nhỏ, ví dụ  $k = 1$  hoặc  $k = 2$ .
- Ví dụ, nếu  $k = 2$  và chuyên gia thứ 3 và thứ 5 có điểm số cao nhất, thì token này sẽ chỉ được gửi đến hai chuyên gia này. Tất cả các chuyên gia khác sẽ không được kích hoạt cho token này.

#### Bước 3: Xử lý bởi các Chuyên gia

- Token  $x$  được đưa qua các mạng FFN của  $k$  chuyên gia đã được chọn.
- Kết quả đầu ra là  $k$  vector:  $E_1(x), E_2(x), \dots, E_k(x)$ .

#### Bước 4: Tổng hợp có trọng số

- Đầu ra cuối cùng của lớp MoE,  $y$ , là một tổng có trọng số của đầu ra từ các chuyên gia được chọn. Các trọng số này chính là các điểm số  $g_i$  từ mạng cổng đã được chuẩn hóa.

$$y = \sum_{i \in \text{TopK}} g_i \cdot E_i(x)$$

### Ví dụ: Mixtral 8x7B

Mô hình Mixtral 8x7B nổi tiếng của Mistral AI là một ví dụ điển hình của MoE.

- **8x7B có nghĩa là gì?** Nó có 8 chuyên gia, mỗi chuyên gia có khoảng 7 tỷ tham số. Tổng số tham số của các lớp MoE là  $8 \times 7B = 56B$ . Tuy nhiên, đây là số tham số "thừa thớt".
- **Top-2 Routing:** Tại mỗi lớp MoE, mỗi token chỉ được định tuyến đến 2 trong số 8 chuyên gia.
- **Chi phí tính toán:** Do đó, mặc dù tổng số tham số rất lớn, chi phí tính toán để xử lý một token chỉ tương đương với một mô hình dày đặc khoảng  $2 \times 7B = 14B$  tham số (cộng với một chi phí nhỏ cho mạng cổng và các lớp khác). Mixtral đạt được chất lượng của một mô hình rất lớn với tốc độ và chi phí của một mô hình nhỏ hơn nhiều.

#### Các "Chuyên gia" có thực sự chuyên môn hóa không?

Phép ẩn dụ về "hội đồng chuyên gia" rất hữu ích, nhưng sự chuyên môn hóa của các expert FFN trong thực tế lại trừu tượng hơn nhiều.

- **Không phải là chuyên môn về chủ đề:** Các nghiên cứu đã chỉ ra rằng các expert thường không chuyên môn hóa theo các chủ đề dễ hiểu như "thể thao", "khoa học" hay "lịch sử". Việc định tuyến một token đến expert nào phụ thuộc nhiều vào các đặc trưng trừu tượng của vector biểu diễn của nó.
- **Chuyên môn hóa theo cụm cú pháp/ngữ nghĩa:** Thay vào đó, sự chuyên môn hóa có thể xảy ra ở mức độ thấp hơn. Ví dụ, một số chuyên gia có thể giỏi hơn trong việc xử lý các token là dấu câu, một số khác chuyên về các động từ, một số khác lại tập trung vào các token ở các lớp Transformer cụ thể (ví dụ, các token ở lớp đầu so với lớp cuối).
- **Sự dư thừa có chủ đích:** Có bằng chứng cho thấy các chuyên gia không hoàn toàn độc lập. Thường có sự dư thừa (redundancy) giữa chúng. Điều này có thể là một đặc tính hữu ích, giúp mô hình trở nên mạnh mẽ hơn: nếu một chuyên gia đưa ra kết quả kém, các chuyên gia khác có thể bù đắp.

Vì vậy, hãy xem các "chuyên gia" như các **nhóm tham số (parameter groups)** có thể học được, thay vì các chuyên gia về một lĩnh vực kiến thức cụ thể. Mạng cổng học cách kết hợp các nhóm tham số này một cách tối ưu cho từng token.

#### 4.5.2. Thách thức trong việc Huấn luyện MoE

Mặc dù rất mạnh mẽ, việc huấn luyện MoE không hề đơn giản và đòi hỏi phải giải quyết một số thách thức đặc thù.

**1. Thách thức về sự sụp đổ của việc định tuyến (Routing Collapse)** Đây là vấn đề lớn nhất của MoE. Nếu không được kiểm soát, mạng cổng sẽ nhanh chóng học được một chiến lược "lười biếng": luôn gửi phần lớn các token đến một hoặc vài chuyên gia "yêu thích" mà nó cho là tốt nhất.

- **Hậu quả:**
  1. **Lãng phí tham số:** Các chuyên gia khác không nhận được token, do đó không được huấn luyện và trở thành "dead wood". Mô hình MoE khổng lồ sẽ sụp đổ (collapse) thành một mô hình nhỏ hơn nhiều.
  2. **Mất ổn định huấn luyện:** Gradient sẽ chảy không đều, gây khó khăn cho quá trình tối ưu hóa.

**Giải pháp: Hàm Mất mát Cân bằng Tải (Load Balancing Loss)** Để giải quyết vấn đề này, người ta thêm một hàm mất mát phụ trợ (auxiliary loss) vào hàm mất mát chính của mô hình. Hàm mất mát này được thiết kế để khuyến khích sự phân bổ token đồng đều và thường bao gồm hai thành phần:

- **Một thành phần khuyến khích tầm quan trọng (importance) đồng đều:** Nó phạt nếu tổng các điểm số (logits) mà mạng cổng gán cho mỗi chuyên gia trên toàn bộ một batch là không

đồng đều. Điều này đảm bảo mỗi chuyên gia được coi là "quan trọng" như nhau về mặt tổng thể.

- **Một thành phần khuyến khích số lượng token (load) đồng đều:** Nó phạt nếu số lượng token được gửi đến mỗi chuyên gia trong một batch là không đồng đều. Điều này đảm bảo mỗi chuyên gia có "công việc" để làm.

Việc cân bằng giữa hàm mất mát chính (ví dụ, cross-entropy) và hàm mất mát phụ trợ này là một trong những khía cạnh nghệ thuật nhất khi huấn luyện các mô hình MoE.

**Giải pháp bổ sung: Noisy Top-K Gating** Một kỹ thuật khác là thêm nhiễu ngẫu nhiên (Gaussian noise) vào logits của mạng cổng trong quá trình huấn luyện.

$$\text{logits}_{\text{noisy}} = \text{logits}_{\text{original}} + \text{Noise}$$

Nhiều này buộc mô hình phải "khám phá" (explore) các lựa chọn định tuyến khác nhau thay vì chỉ đi theo con đường quen thuộc, giúp cải thiện sự cân bằng tải một cách tự nhiên.

## 2. Thách thức về Triển khai và Cơ sở hạ tầng

- **Bộ nhớ:** Mặc dù chi phí tính toán trên mỗi token thấp, toàn bộ các tham số của tất cả các chuyên gia vẫn phải được tải vào bộ nhớ của GPU, đòi hỏi các hệ thống có bộ nhớ cực lớn.
- **Giao tiếp mạng:** Trong các hệ thống huấn luyện phân tán, các token từ một GPU có thể cần được gửi đến các chuyên gia nằm trên một GPU khác, đòi hỏi băng thông mạng rất cao.

### 4.5.3. Tổng kết về MoE

Mixture-of-Experts là một sự thay đổi mô hình trong cách chúng ta xây dựng các mô hình ngôn ngữ lớn. Nó cho phép chúng ta tách rời **tổng số tham số khỏi chi phí tính toán trên mỗi token**.

Bằng cách sử dụng các chuyên gia thừa thãi, MoE mở ra một con đường để xây dựng các mô hình có quy mô khổng lồ (hàng nghìn tỷ tham số), có khả năng lưu trữ một lượng kiến thức cực lớn, trong khi vẫn duy trì được tốc độ huấn luyện và suy luận ở mức có thể quản lý được. Đây là một trong những công nghệ nền tảng quan trọng nhất cho thế hệ LLM tiếp theo.

## KẾT THÚC CHƯƠNG 4

*Chương 4 đã là một chuyến đi sâu vào trái tim của NLP hiện đại. Chúng ta đã mở rộng kiến trúc Transformer, hiểu được sức mạnh của cơ chế Tự chú ý, và phân loại ba họ mô hình chính – Encoder-Only, Decoder-Only, và Encoder-Decoder – đã được sinh ra từ nó. Chúng ta cũng đã khám phá các kỹ thuật tiên tiến cho phép các mô hình này đạt đến quy mô khổng lồ và xử lý ngôn ngữ cảnh dài hơn. Về cơ bản, chúng ta đã tìm hiểu cách các "mô hình nền tảng" (foundation models) được xây dựng. Giờ đây, khi đã hiểu rõ cách các mô hình nền tảng khổng lồ này được xây dựng, câu hỏi tiếp theo là: Làm thế nào để chúng ta "thuần hóa" và "chuyên môn hóa" chúng cho các bài toán cụ thể? Chương tiếp theo sẽ tập trung vào các kỹ thuật tinh chỉnh và cẩn chỉnh, biến những gã khổng lồ đa năng này thành những chuyên gia sắc bén.*

## Chương 5

# LÝ THUYẾT VỀ TINH CHỈNH VÀ CĂN CHỈNH MÔ HÌNH

Trong chương 4, chúng ta đã khám phá cách các Mô hình Ngôn ngữ Lớn (LLMs) được **huấn luyện trước** (pre-training) trên một lượng dữ liệu văn bản khổng lồ. Quá trình này trang bị cho chúng một sự hiểu biết sâu sắc về ngôn ngữ, ngữ pháp, ngữ nghĩa và một lượng lớn kiến thức về thế giới. Tuy nhiên, một mô hình được huấn luyện giống như một sinh viên tốt nghiệp xuất sắc nhưng chưa có kinh nghiệm làm việc chuyên sâu: rất thông minh nhưng chưa phải là chuyên gia trong một lĩnh vực cụ thể nào.

Chương này sẽ tập trung vào giai đoạn thứ hai và cũng là giai đoạn quyết định để biến một mô hình nền tảng (foundation model) thành một ứng dụng hữu ích: **giai đoạn thích ứng (adaptation)**. Chúng ta sẽ tìm hiểu các kỹ thuật để "dạy chuyên sâu" cho các LLM, giúp chúng trở nên xuất sắc trong các tác vụ cụ thể, từ phân loại email đến việc tuân theo các chỉ dẫn phức tạp của con người.

### 5.1. Các Phương pháp Tinh chỉnh (Fine-tuning)

#### 5.1.1. Học Chuyển giao (Transfer Learning): Nền tảng Tư duy

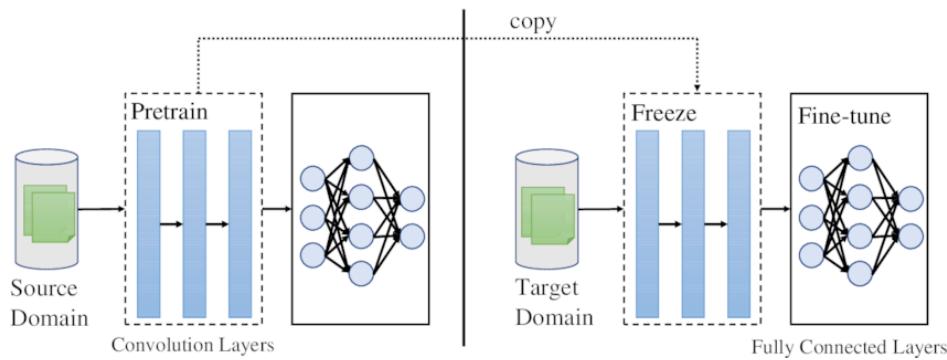
Trước khi đi vào kỹ thuật cụ thể, chúng ta cần hiểu triết lý nền tảng đằng sau nó: **Học Chuyển giao (Transfer Learning)**.

##### Định nghĩa 5: Học Chuyển giao

Học Chuyển giao là một phương pháp học máy trong đó một mô hình được phát triển cho một tác vụ A được tái sử dụng làm điểm khởi đầu cho một mô hình trong tác vụ B.

Trong NLP, Học Chuyển giao đã trở thành một mô hình (paradigm) thống trị:

- Giai đoạn Huấn luyện trước (Pre-training):** Huấn luyện một mô hình lớn (như BERT, GPT) trên một tác vụ chung (ví dụ: Masked Language Modeling, Next Token Prediction) với một kho dữ liệu khổng lồ, không cần gán nhãn. Giai đoạn này giúp mô hình học được các "biểu diễn" ngôn ngữ tổng quát. Đây là phần tốn kém nhất.
- Giai đoạn Tinh chỉnh (Fine-tuning):** Lấy mô hình đã được huấn luyện trước và tiếp tục huấn luyện nó trên một bộ dữ liệu nhỏ hơn, đã được gán nhãn, dành riêng cho tác vụ mục tiêu (ví dụ: phân loại cảm xúc).



Hình 5.1: Mô hình Học Chuyển giao trong NLP: Một mô hình được huấn luyện trước trên dữ liệu lớn, sau đó được tinh chỉnh trên một tác vụ cụ thể.

### Tại sao Học Chuyển giao lại hiệu quả?

- Tiết kiệm tài nguyên:** Chúng ta không cần phải huấn luyện một mô hình khổng lồ từ đầu cho mỗi tác vụ. Chúng ta tận dụng lại kiến thức đã được học từ quá trình pre-training tốn kém.
- Cải thiện hiệu năng:** Các biểu diễn ngôn ngữ tổng quát học được từ dữ liệu lớn cung cấp một điểm khởi đầu tốt hơn nhiều so với việc khởi tạo trọng số ngẫu nhiên, giúp mô hình hội tụ nhanh hơn và đạt độ chính xác cao hơn, đặc biệt khi dữ liệu cho tác vụ cụ thể là có hạn.
- Dân chủ hóa AI:** Cho phép các nhà phát triển và nhà nghiên cứu không có tài nguyên tính toán khổng lồ vẫn có thể xây dựng các ứng dụng NLP state-of-the-art bằng cách tinh chỉnh các mô hình đã được huấn luyện trước.

#### 5.1.2. Tinh chỉnh Toàn bộ (Full Fine-tuning)

Đây là phương pháp tinh chỉnh kinh điển và trực quan nhất.

#### Cơ chế hoạt động

- Bước 1: Tải mô hình đã huấn luyện trước.** Tải toàn bộ kiến trúc và trọng số của một mô hình như BERT hoặc RoBERTa.
- Bước 2: Thay thế "đầu" của mô hình.** Mô hình gốc thường có một "đầu"(head) được thiết kế cho tác vụ pre-training (ví dụ, đầu MLM). Chúng ta sẽ loại bỏ cái đầu này và thay thế nó bằng một "đầu" mới, phù hợp với tác vụ hạ nguồn của mình.
  - Ví dụ (Phân loại câu):** Thêm một lớp tuyến tính (linear layer) đơn giản với một lớp softmax lên trên vector biểu diễn của token '[CLS]'. Số nơ-ron đầu ra của lớp này bằng số lớp cần phân loại.
  - Ví dụ (Gán nhãn chuỗi - NER):** Thêm một lớp tuyến tính lên trên vector biểu diễn của *mỗi* token trong chuỗi.
- Bước 3: Huấn luyện trên dữ liệu của tác vụ mới.** Toàn bộ mô hình (cả phần thân đã được huấn luyện trước và cái đầu mới) được huấn luyện tiếp trên bộ dữ liệu gán nhãn của tác vụ mục tiêu.
- Cập nhật trọng số:** Trong quá trình này, **tất cả các trọng số của mô hình**, từ lớp embedding dưới cùng đến lớp phân loại trên cùng, đều được cập nhật thông qua thuật toán lan truyền ngược để tối thiểu hóa hàm mất mát trên tác vụ mới.

Điều quan trọng là việc huấn luyện ở giai đoạn này thường sử dụng một **tốc độ học (learning rate) rất nhỏ**. Lý do là vì các trọng số đã được huấn luyện trước đã ở một điểm "khá tốt". Chúng ta chỉ muốn "tinh chỉnh" chúng một chút để thích ứng với dữ liệu mới, chứ không muốn phá vỡ các kiến thức quý giá đã học được bằng các bước cập nhật quá lớn.

### Chiến lược Tinh chỉnh Nâng cao

Trong thực tế, việc tinh chỉnh không chỉ đơn giản là chạy lại quá trình huấn luyện. Có một số chiến lược để cải thiện hiệu quả:

#### Huấn luyện theo lớp phân biệt (Discriminative Layer-wise Training)

- **Trực giác:** Các lớp khác nhau của một mô hình học sâu học các loại đặc trưng khác nhau. Các lớp dưới cùng (gần input) học các đặc trưng rất chung (như cú pháp cơ bản), trong khi các lớp trên cùng học các đặc trưng trừu tượng và chuyên biệt hơn.
- **Chiến lược:** Khi tinh chỉnh, chúng ta nên cập nhật các lớp trên cùng nhiều hơn và các lớp dưới cùng ít hơn. Điều này được thực hiện bằng cách áp dụng các **tốc độ học khác nhau cho các lớp khác nhau**. Ví dụ, lớp phân loại mới có thể có learning rate là  $10^{-3}$ , các lớp Transformer trên cùng là  $10^{-4}$ , và các lớp Transformer dưới cùng là  $10^{-5}$ .

#### Rã đông Dần dần (Gradual Unfreezing)

- **Trực giác:** Để tránh "quên thảm khốc" (catastrophic forgetting) - nơi mô hình nhanh chóng quên đi kiến thức đã học trong quá trình pre-training - chúng ta nên giới thiệu dữ liệu mới một cách từ từ.
- **Chiến lược:**
  1. Ban đầu, "đóng băng" (freeze) toàn bộ các lớp đã được huấn luyện trước và chỉ huấn luyện cái đầu phân loại mới trong một vài epoch.
  2. Sau đó, "rã đông" (unfreeze) lớp Transformer trên cùng và huấn luyện cả nó và cái đầu mới.
  3. Tiếp tục quá trình này, rã đông dần dần từng lớp từ trên xuống dưới, cho đến khi toàn bộ mô hình được huấn luyện.

#### Ưu và Nhược điểm của Tinh chỉnh Toàn bộ

##### Đánh giá Tinh chỉnh Toàn bộ (Full Fine-tuning)

###### Ưu điểm:

- **Hiệu năng cao nhất:** Thường mang lại độ chính xác tốt nhất vì toàn bộ mô hình được tối ưu hóa một cách trọn vẹn cho tác vụ mục tiêu.
- **Tương đối đơn giản:** Là một phương pháp đã được thiết lập rõ ràng và dễ triển khai.

###### Nhược điểm:

- **Cực kỳ tốn kém về tài nguyên:** Thách thức lớn nhất trong kỷ nguyên LLM. Nếu bạn có 10 tác vụ khác nhau, bạn phải lưu trữ **10 bản sao đầy đủ** của mô hình khổng lồ đã được tinh chỉnh, mỗi bản sao có thể nặng hàng trăm gigabyte.
- **Dễ bị "quên thảm khốc":** Nếu dữ liệu tinh chỉnh quá khác biệt so với dữ liệu pre-training, mô hình có thể mất đi các khả năng tổng quát của nó.
- **Rủi ro về bảo mật và ổn định:** Việc cập nhật toàn bộ trọng số có thể gây ra các hành vi không mong muốn.

Những nhược điểm này, đặc biệt là về chi phí lưu trữ và tính toán, đã trở nên cực kỳ rõ rệt với sự bùng nổ của các mô hình hàng tỷ tham số. Điều này đã thúc đẩy mạnh mẽ sự phát triển của một hướng tiếp cận mới, hiệu quả hơn nhiều: **Kỹ thuật Tinh chỉnh Hiệu quả Tham số (Parameter-Efficient Fine-Tuning - PEFT)**, mà chúng ta sẽ khám phá chi tiết trong mục tiếp theo.

## 5.2. Kỹ thuật Tinh chỉnh Hiệu quả Tham số (PEFT)

Như đã phân tích ở mục trước, Tinh chỉnh Toàn bộ (Full Fine-tuning) trở nên cực kỳ tốn kém và không thực tế khi làm việc với các Mô hình Ngôn ngữ Lớn (LLMs) có hàng tỷ tham số. Việc lưu trữ

và triển khai hàng chục bản sao đầy đủ của một mô hình 175 tỷ tham số cho các tác vụ khác nhau là một cơn ác mộng về mặt tài chính và kỹ thuật.

Kỹ thuật Tinh chỉnh Hiệu quả Tham số (Parameter-Efficient Fine-Tuning - PEFT) ra đời như một giải pháp cho vấn đề này.

### Triết lý của PEFT

Thay vì cập nhật **tất cả** các trọng số của mô hình lớn, chúng ta hãy **đóng băng** (freeze) toàn bộ mô hình gốc và chỉ huấn luyện một số lượng rất nhỏ các tham số **bổ sung** (additional) hoặc một **tập con** các tham số hiện có.

Mục tiêu là đạt được hiệu năng gần bằng hoặc thậm chí tốt hơn Full Fine-tuning, trong khi chỉ cần huấn luyện và lưu trữ một phần rất nhỏ (thường < 1%) của tổng số tham số. Điều này làm cho việc thích ứng LLM trở nên khả thi và hiệu quả hơn rất nhiều.

Có nhiều họ phương pháp PEFT khác nhau, chúng ta sẽ khám phá những phương pháp tiêu biểu và có ảnh hưởng nhất.

#### 5.2.1. LoRA: Thích ứng Thứ hạng Thấp (Low-Rank Adaptation)

LoRA (Hu et al., 2021) [35] là một trong những kỹ thuật PEFT phổ biến và thành công nhất hiện nay. Nó dựa trên một quan sát sâu sắc về cách các mạng nơ-ron học.

#### Trực giác và Nền tảng Lý thuyết

- Quan sát:** Các nhà nghiên cứu phát hiện ra rằng mặc dù các mô hình ngôn ngữ lớn có số chiều rất cao, nhưng sự thay đổi của các trọng số trong quá trình tinh chỉnh (tức là ma trận  $\Delta W = W_{\text{fine-tuned}} - W_{\text{pre-trained}}$ ) thường có một "thứ hạng nội tại" (intrinsic rank) rất thấp.
- Ý nghĩa:** Điều này có nghĩa là ma trận thay đổi trọng số  $\Delta W$  (kích thước  $d \times d$ ) có thể được xấp xỉ rất tốt bằng tích của hai ma trận nhỏ hơn nhiều,  $A$  (kích thước  $d \times r$ ) và  $B$  (kích thước  $r \times d$ ), trong đó  $r$  (thứ hạng) là một số rất nhỏ so với  $d$  (ví dụ:  $r = 8$ ,  $d = 4096$ ).
- Kết luận:** Thay vì học ma trận  $\Delta W$  khổng lồ với  $d^2$  tham số, chúng ta có thể chỉ cần học hai ma trận nhỏ  $A$  và  $B$  với tổng số tham số chỉ là  $2 \times d \times r$ . Đây là một sự tiết kiệm rất lớn.

#### Cơ chế hoạt động của LoRA

LoRA áp dụng ý tưởng này vào các lớp cần nhiều tham số nhất trong Transformer, thường là các ma trận trọng số của các lớp tuyến tính trong cơ chế Self-Attention ( $W_Q, W_K, W_V, W_O$ ).

- Đóng băng mô hình gốc:** Toàn bộ trọng số của mô hình đã huấn luyện trước,  $W_0$ , được giữ nguyên và không cập nhật trong quá trình huấn luyện.
- Thêm các ma trận phân rã:** Với mỗi ma trận trọng số  $W_0$  mà chúng ta muốn thích ứng (ví dụ,  $W_Q$ ), LoRA thêm vào hai ma trận "adapter" nhỏ là  $A$  (khởi tạo ngẫu nhiên) và  $B$  (khởi tạo bằng 0).
- Sửa đổi quá trình truyền thẳng (Forward Pass):** Đầu ra của lớp được sửa đổi như sau:

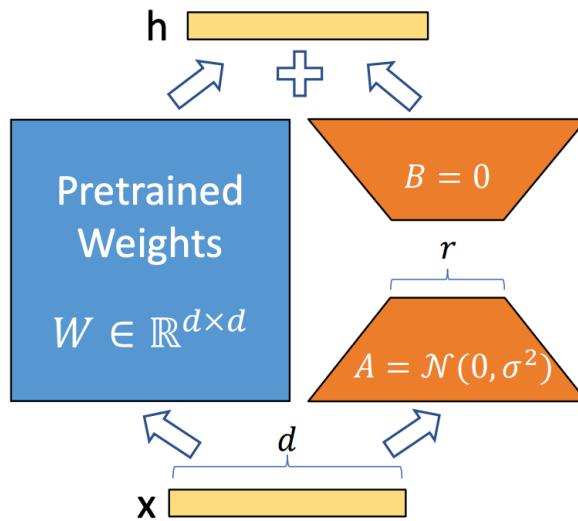
$$h = W_0x + \Delta Wx = W_0x + BAx$$

Vector đầu vào  $x$  sẽ đi qua hai nhánh song song:

- Nhánh chính đi qua ma trận  $W_0$  đã đóng băng.
- Nhánh LoRA đi qua ma trận  $A$  trước (giảm chiều từ  $d$  xuống  $r$ ), sau đó qua ma trận  $B$  (tăng chiều từ  $r$  trở lại  $d$ ).

Kết quả của hai nhánh này sau đó được cộng lại với nhau.

- Huấn luyện:** Chỉ có các ma trận  $A$  và  $B$  được huấn luyện. Tất cả các tham số khác đều bị đóng băng.



Hình 5.2: Cơ chế hoạt động của LoRA. Trọng số gốc  $W_0$  được đóng băng. Một nhánh phụ song song được thêm vào, bao gồm hai ma trận thứ hạng thấp  $A$  và  $B$ . Chỉ có  $A$  và  $B$  được cập nhật trong quá trình huấn luyện.

### Lợi ích của LoRA

- Hiệu quả về lưu trữ:** Thay vì lưu một bản sao 13 tỷ tham số cho mỗi tác vụ, bạn chỉ cần lưu các ma trận LoRA A và B, có thể chỉ vài megabyte.
- Chuyển đổi tác vụ nhanh chóng:** Có thể "cắm" và "rút" các adapter LoRA khác nhau vào cùng một mô hình nền tảng để chuyển đổi giữa các tác vụ mà không cần tốn thời gian tải lại toàn bộ mô hình.
- Không có độ trễ suy luận (Inference Latency):** Sau khi huấn luyện xong, các trọng số LoRA ( $BA$ ) có thể được gộp (merged) vào trọng số gốc ( $W = W_0 + BA$ ). Quá trình suy luận sau đó diễn ra trên một ma trận duy nhất  $W$ , không có thêm bất kỳ chi phí tính toán nào so với mô hình gốc.

#### 5.2.2. QLoRA: Lượng tử hóa LoRA (Quantized LoRA)

QLoRA (Dettmers et al., 2023) [19] là một cải tiến đột phá của LoRA, giúp giảm yêu cầu về bộ nhớ GPU xuống mức cực thấp, cho phép tinh chỉnh các mô hình hàng tỷ tham số trên các GPU dành cho người tiêu dùng.

### Ý tưởng cốt lõi

QLoRA kết hợp LoRA với một kỹ thuật gọi là **lượng tử hóa (quantization)**.

- Vấn đề:** Mặc dù LoRA chỉ huấn luyện một số ít tham số, nó vẫn yêu cầu phải tải **toute bộ mô hình gốc vào bộ nhớ GPU** để thực hiện quá trình truyền thẳng (forward pass). Với các mô hình lớn, các trọng số này (thường ở định dạng 16-bit) vẫn chiếm một lượng VRAM khổng lồ.
- Giải pháp của QLoRA:** Tải mô hình nền tảng đã được **lượng tử hóa xuống định dạng 4-bit**. Tức là, mỗi trọng số chỉ dùng 4 bit để lưu trữ thay vì 16 bit, giảm yêu cầu bộ nhớ đi 4 lần.
- Thách thức:** Việc huấn luyện trực tiếp trên các trọng số 4-bit là không ổn định.
- Sự đột phá của QLoRA:** Nó giới thiệu một kiểu dữ liệu 4-bit mới gọi là **4-bit NormalFloat (NF4)** và kỹ thuật **Double Quantization** để giảm thiểu mất mát thông tin. Quan trọng nhất, trong quá trình truyền thẳng, các trọng số 4-bit của mô hình nền tảng được giải **lượng tử hóa**

(de-quantized) về định dạng 16-bit "ngay khi cần" cho việc tính toán, sau đó bị loại bỏ khỏi bộ nhớ. Gradient vẫn được lan truyền ngược qua các adapter LoRA ở định dạng 16-bit.

Nhờ QLoRA, việc tinh chỉnh một mô hình 65 tỷ tham số có thể được thực hiện trên một GPU duy nhất với 48GB VRAM, một điều không tưởng trước đây.

### 5.2.3. Adapter-tuning

Đây là một trong những họ phương pháp PEFT [33] đầu tiên, có trước cả LoRA.

- **Kiến trúc:** Thay vì thêm các nhánh song song, Adapter-tuning chèn các module nhỏ, gọi là các lớp adapter (**adapter layers**), vào bên trong mỗi khối Transformer, thường là sau lớp Multi-Head Attention và lớp FFN.
- **Cơ chế:** Các lớp adapter này thường có kiến trúc "cổ chai" (bottleneck): một lớp tuyển tính giảm chiều, một hàm kích hoạt phi tuyến, và một lớp tuyển tính tăng chiều trở lại.
- **Huấn luyện:** Tương tự, toàn bộ mô hình gốc được đóng băng và chỉ có các tham số của các lớp adapter này được huấn luyện.
- **So với LoRA:** Adapter có nhược điểm là chúng thêm vào độ trễ suy luận vì đầu ra phải đi qua các lớp bổ sung một cách tuần tự. LoRA khắc phục được điều này nhờ khả năng gộp trọng số.

### 5.2.4. Prompt-based Tuning: Điều chỉnh trên Lớp Đầu vào

Các phương pháp trên can thiệp vào trọng số của mô hình. Một hướng tiếp cận khác, cấp tiến hơn, là: **giữ nguyên hoàn toàn mô hình và chỉ điều chỉnh đầu vào**.

#### Prompt-Tuning

- **Ý tưởng:** Thay vì tinh chỉnh các trọng số khổng lồ, hãy học cách tìm ra "prompt" tốt nhất cho mô hình. Nhưng thay vì tìm kiếm các từ trong ngôn ngữ tự nhiên (hard prompt), Prompt-Tuning [47] học một chuỗi các vector embedding "mềm" (soft prompts) liên tục.
- **Cơ chế:** Một số lượng nhỏ ( $k$ ) các vector prompt có thể huấn luyện được sẽ được chèn vào phía trước chuỗi embedding đầu vào. Toàn bộ mô hình gốc được đóng băng. Trong quá trình huấn luyện, chỉ có các vector prompt này được cập nhật thông qua lan truyền ngược.
- **Lợi ích:** Cực kỳ hiệu quả về tham số (chỉ cần học vài nghìn tham số) và dễ dàng lưu trữ, chia sẻ các "prompt" đã được học cho các tác vụ khác nhau.

#### Prefix-Tuning

- **Ý tưởng:** Tương tự Prompt-Tuning, nhưng thay vì chỉ chèn các vector vào lớp embedding đầu vào, Prefix-Tuning [51] chèn các tiền tố (prefixes) có thể huấn luyện được vào **mỗi lớp Transformer** của mô hình.
- **Cơ chế:** Các tiền tố này hoạt động như một "ngữ cảnh ảo" dành riêng cho tác vụ, giúp điều khiển hành vi của mỗi lớp một cách trực tiếp hơn.
- **Hiệu năng:** Thường cho kết quả tốt hơn Prompt-Tuning nhưng có nhiều tham số hơn để huấn luyện.

Các kỹ thuật PEFT đã thay đổi hoàn toàn cách chúng ta làm việc với LLMs, giúp việc tùy chỉnh các mô hình khổng lồ trở nên khả thi, tiết kiệm và linh hoạt hơn bao giờ hết.

## 5.3. Kỹ thuật Prompt Engineering và Học trong Ngữ cảnh (In-context Learning)

Với sự ra đời của các Mô hình Ngôn ngữ Lớn (LLMs) có quy mô hàng trăm tỷ tham số (ví dụ như họ GPT), một khả năng đáng kinh ngạc đã xuất hiện, được gọi là **Học trong Ngữ cảnh (In-context Learning - ICL)**.

### Định nghĩa 6: Học trong Ngũ cảnh (ICL)

Học trong Ngũ cảnh là khả năng của một LLM có thể học và thực hiện một tác vụ mới chỉ bằng cách được cung cấp một vài ví dụ minh họa ngay trong câu lệnh đầu vào (**prompt**), mà không cần bất kỳ sự cập nhật trọng số nào của mô hình.

Nói cách khác, mô hình "học" bằng cách suy luận từ các ví dụ được đưa ra trong "bộ nhớ ngắn hạn" (ngũ cảnh của prompt), thay vì "học" bằng cách cập nhật "bộ nhớ dài hạn" (trọng số của mô hình) thông qua quá trình fine-tuning.

Khả năng này đã khai sinh ra một lĩnh vực hoàn toàn mới: **Kỹ thuật Thiết kế Câu lệnh (Prompt Engineering)** – nghệ thuật và khoa học của việc thiết kế các prompt hiệu quả để khai thác tối đa tiềm năng của LLMs.

#### 5.3.1. Các cấp độ của Học trong Ngũ cảnh: Zero-shot, One-shot, Few-shot

Khả năng ICL của LLMs thường được phân loại dựa trên số lượng ví dụ ('n') được cung cấp trong prompt.

##### Zero-shot Learning (Học không cần ví dụ, n=0)

- Cơ chế:** Chỉ cung cấp cho mô hình một mô tả về tác vụ và câu hỏi cần trả lời, không kèm theo bất kỳ ví dụ minh họa nào.
- Trực giác:** Đây là bài kiểm tra cuối cùng về khả năng tổng quát hóa của mô hình. Nó phải dựa hoàn toàn vào kiến thức đã học được trong quá trình pre-training để hiểu và thực hiện yêu cầu.
- Ví dụ (Phân tích cảm xúc):**

##### Prompt:

Phân loại cảm xúc của câu sau đây là Tích cực, Tiêu cực, hoặc Trung tính.

Câu: Món ăn này thật tuyệt vời!

Cảm xúc:

Đầu ra mong muốn: 'Tích cực'

##### One-shot Learning (Học với một ví dụ, n=1)

- Cơ chế:** Cung cấp cho mô hình một ví dụ duy nhất về cặp (đầu vào, đầu ra) để làm mẫu, trước khi đưa ra câu hỏi thực sự.
- Trực giác:** Một ví dụ duy nhất giúp mô hình "định vị" được tác vụ, hiểu rõ hơn về định dạng đầu ra mong muốn và bối cảnh cụ thể.
- Ví dụ (Dịch từ vô nghĩa):**

##### Prompt:

Dịch từ "glorp" sang tiếng Việt có nghĩa là "vui vẻ".

Dịch từ "flumph" sang tiếng Việt có nghĩa là gì?

Đầu ra mong muốn: Một từ vô nghĩa khác, thể hiện mô hình đã hiểu "quy luật" của tác vụ.

##### Few-shot Learning (Học với vài ví dụ, n > 1)

- Cơ chế:** Cung cấp cho mô hình **một vài ví dụ** (thường từ 2 đến vài chục) để minh họa cho tác vụ.

- Trực giác:** Càng nhiều ví dụ, mô hình càng có nhiều thông tin để suy luận ra quy luật và mẫu hình của tác vụ, từ đó đưa ra câu trả lời chính xác hơn. Đây là phương pháp phổ biến và hiệu quả nhất trong ICL.
- Ví dụ (Phép cộng):**

Prompt:

Q:  $2 + 2 = ?$

A: 4

Q:  $5 + 8 = ?$

A: 13

Q:  $12 + 3 = ?$

A:

**Đầu ra mong muốn: '15'**

Hiệu năng của mô hình thường tăng lên cùng với số lượng ví dụ, nhưng sẽ đạt đến một điểm bão hòa và bị giới hạn bởi độ dài ngữ cảnh của mô hình.

### 5.3.2. Chuỗi Suy luận (Chain-of-Thought - CoT) Prompting

#### Vấn đề của Prompting Tiêu chuẩn

Với các bài toán đòi hỏi suy luận nhiều bước (ví dụ: các bài toán logic, toán học), phương pháp few-shot tiêu chuẩn thường thất bại. Mô hình có thể cố gắng "đoán" câu trả lời cuối cùng một cách trực tiếp và thường đưa ra kết quả sai.

#### Giải pháp của CoT: "Hãy suy nghĩ từng bước"

Chain-of-Thought (CoT) Prompting (Wei et al., 2022) là một kỹ thuật đơn giản nhưng cực kỳ mạnh mẽ.

#### Trực giác của Chain-of-Thought

Thay vì chỉ cung cấp các ví dụ (Câu hỏi, Trả lời), chúng ta hãy cung cấp các ví dụ bao gồm cả **quá trình suy luận từng bước (step-by-step reasoning)** dẫn đến câu trả lời cuối cùng. Điều này khuyến khích mô hình "bắt chước" quá trình suy nghĩ đó, phân rã một bài toán phức tạp thành các bước trung gian đơn giản hơn.

#### Ví dụ 11: So sánh Standard Prompting và CoT Prompting

**Bài toán:** Roger có 5 quả bóng tennis. Anh ấy mua thêm 2 hộp, mỗi hộp có 3 quả. Hỏi bây giờ anh ấy có tất cả bao nhiêu quả bóng?

##### 1. Standard Few-shot Prompting (Thường thất bại):

Q: Một người bán hoa quả có 23 quả táo. Nếu anh ta dùng 20 quả để làm bánh và mua thêm 6 quả, anh ta còn lại bao nhiêu quả? A: 9 quả.

Q: Roger có 5 quả bóng tennis. Anh ấy mua thêm 2 hộp, mỗi hộp có 3 quả. Hỏi bây giờ anh ấy có tất cả bao nhiêu quả bóng? A:

→ Mô hình có thể trả lời sai, ví dụ '10'.

##### 2. Chain-of-Thought Prompting (Hiệu quả hơn):

Q: Một người bán hoa quả có 23 quả táo. Nếu anh ta dùng 20 quả để làm bánh và mua thêm 6 quả, anh ta còn lại bao nhiêu quả? A: Ban đầu anh ta có 23 quả táo. Anh ta dùng 20 quả nên còn  $23 - 20 = 3$  quả. Sau đó anh ta mua thêm 6 quả, vậy anh ta có  $3 + 6 = 9$  quả. Câu trả lời là 9.

Q: Roger có 5 quả bóng tennis. Anh ấy mua thêm 2 hộp, mỗi hộp có 3 quả. Hỏi bây giờ anh ấy có tất cả bao nhiêu quả bóng? A:

→ Bị "mờ" bởi ví dụ trên, mô hình có khả năng cao sẽ sinh ra chuỗi suy luận: 'Roger ban đầu có 5 quả bóng. Anh ấy mua thêm 2 hộp, mỗi hộp 3 quả, tức là thêm  $2 * 3 = 6$  quả. Tổng cộng anh ấy có  $5 + 6 = 11$  quả. Câu trả lời là 11.'

**Zero-shot-CoT** Một khám phá thú vị sau đó là chúng ta thậm chí không cần cung cấp ví dụ. Chỉ cần thêm một câu thần chú đơn giản như "Let's think step by step." (Hãy suy nghĩ từng bước.) vào cuối prompt cũng đủ để kích hoạt khả năng suy luận theo chuỗi của các LLM lớn.

### 5.3.3. Các kỹ thuật nâng cao dựa trên CoT

CoT là một ý tưởng nền tảng, và nhiều kỹ thuật khác đã được xây dựng dựa trên nó để tăng cường hơn nữa độ tin cậy và chính xác.

#### Tự Nhất quán (Self-Consistency)

- **Vấn đề:** Ngay cả với CoT, quá trình sinh của LLM vẫn mang tính ngẫu nhiên (do việc lấy mẫu từ phân phối xác suất). Nếu chạy cùng một prompt nhiều lần, mô hình có thể tạo ra các chuỗi suy luận khác nhau, dẫn đến các câu trả lời khác nhau.
- **Giải pháp của Self-Consistency (Wang et al., 2022) [98]:**
  1. Chạy cùng một prompt CoT nhiều lần (ví dụ: 10 lần) với nhiệt độ (temperature) lớn hơn 0 để tạo ra các chuỗi suy luận đa dạng.
  2. Trích xuất câu trả lời cuối cùng từ mỗi chuỗi suy luận.
  3. Chọn câu trả lời xuất hiện thường xuyên nhất (majority vote) làm câu trả lời cuối cùng.
- **Trực giác:** Nếu có nhiều con đường suy luận khác nhau nhưng đều dẫn đến cùng một câu trả lời, thì câu trả lời đó có khả năng cao là đúng. Kỹ thuật này giúp giảm thiểu ảnh hưởng của các lỗi suy luận ngẫu nhiên.

#### Cây Suy tưởng (Tree of Thoughts - ToT)

- **Vấn đề:** CoT và Self-Consistency vẫn chỉ khám phá các con đường suy luận một cách độc lập. Chúng không có khả năng "quay lại", "đánh giá" hay "so sánh" các bước suy luận trung gian.
- **Giải pháp của Tree of Thoughts (Yao et al., 2023) [105]:**
  1. **Tạo cây:** Thay vì một chuỗi duy nhất, mô hình sẽ chủ động tạo ra một cây các "suy tưởng"(thoughts). Tại mỗi bước, nó sẽ đề xuất nhiều bước tiếp theo có thể có, tạo thành các nhánh mới.
  2. **Đánh giá:** Một "bộ đánh giá"(có thể là chính LLM đó) sẽ cho điểm mỗi nhánh (mỗi "suy tưởng" trung gian), đánh giá xem nó có hứa hẹn dẫn đến giải pháp cuối cùng hay không.
  3. **Tìm kiếm:** Sử dụng các thuật toán tìm kiếm trên cây (như tìm kiếm theo chiều rộng hoặc chiều sâu) để khám phá cây suy tưởng một cách có hệ thống, ưu tiên các nhánh có điểm số cao và loại bỏ các nhánh không hứa hẹn.
- **Trực giác:** ToT mô phỏng cách con người giải quyết các vấn đề phức tạp hơn: chúng ta không chỉ đi theo một luồng suy nghĩ duy nhất, mà còn xét nhiều khả năng, đánh giá chúng, và quay lại nếu một hướng đi có vẻ sai lầm. ToT cho phép mô hình có khả năng tự đánh giá và lập kế hoạch một cách tường minh.

Các kỹ thuật prompt engineering, từ few-shot đơn giản đến các cấu trúc suy luận phức tạp như ToT, đang liên tục phát triển, biến việc "trò chuyện" với LLM trở thành một lĩnh vực kỹ thuật thực sự.

## 5.4. Tinh chỉnh theo chỉ dẫn (Instruction Tuning)

### 5.4.1. Vấn đề của các LLM cơ sở (Base LLMs)

Một mô hình ngôn ngữ cơ sở (base model) được huấn luyện trước trên mục tiêu Causal Language Modeling (dự đoán từ tiếp theo) có một kiến thức khổng lồ. Tuy nhiên, nó có một hạn chế lớn về mặt tương tác: nó được huấn luyện để **bắt chước văn phong** của dữ liệu pre-training, chứ không phải để **tuân theo mệnh lệnh** của người dùng.

#### Ví dụ 12: Hành vi của một Base LLM

Nếu bạn đưa cho một base LLM (như GPT-3 gốc) một prompt:

**Prompt:** 'Dịch câu sau sang tiếng Pháp: "Hello, how are you?"'

Mô hình có thể sẽ tiếp tục prompt này theo cách nó thường thấy trên Internet, ví dụ:

**Đầu ra có thể có của Base LLM:** 'Dịch câu sau sang tiếng Đức: "Good morning."' 'Dịch câu sau sang tiếng Tây Ban Nha: "Thank you."' 'Liệt kê 10 thành phố lớn nhất thế giới.'

Nó hiểu rằng đây là một mẫu câu hỏi trong một bài kiểm tra hoặc một trang web học ngôn ngữ, và nó chỉ đơn giản là "hoàn thành" cái mẫu đó, thay vì thực sự thực hiện yêu cầu dịch. Nó thiếu khả năng "hiểu ý định" (intent understanding).

Để biến một base LLM thành một trợ lý hữu ích, chúng ta cần dạy cho nó một kỹ năng mới: **tuân theo chỉ dẫn** (instruction following). Đây chính là mục tiêu của Tinh chỉnh theo chỉ dẫn.

### 5.4.2. Tư duy cốt lõi và Cơ chế hoạt động

#### Triết lý của Instruction Tuning

Chúng ta sẽ tiếp tục tinh chỉnh (fine-tune) mô hình ngôn ngữ cơ sở trên một bộ dữ liệu lớn, đa dạng bao gồm các cặp (**Chỉ dẫn, Kết quả mong muốn**). Bằng cách cho mô hình xem hàng ngàn ví dụ về cách thực hiện các mệnh lệnh, nó sẽ học được một siêu-kỹ năng tổng quát: "Khi người dùng đưa ra một chỉ dẫn, nhiệm vụ của tôi là thực hiện nó một cách hữu ích và trực tiếp, thay vì chỉ hoàn thành câu."

Về mặt kỹ thuật, Instruction Tuning là một quá trình fine-tuning có giám sát. Mô hình được huấn luyện để tối thiểu hóa hàm mất mát (thường là Cross-Entropy) trên phần "kết quả mong muốn" của các ví dụ. Quá trình này điều chỉnh các trọng số của mô hình để nó có xu hướng sinh ra các câu trả lời tuân theo chỉ dẫn.

### 5.4.3. Xây dựng Bộ dữ liệu Chỉ dẫn: Chìa khóa thành công

Chất lượng và sự đa dạng của bộ dữ liệu chỉ dẫn là yếu tố **quyết định tuyệt đối** đến thành công của quá trình này. Một bộ dữ liệu tốt phải bao quát một loạt các khía cạnh.

#### Các thành phần của một mẫu dữ liệu

Một mẫu dữ liệu chỉ dẫn lý tưởng thường có ba phần:

- Chỉ dẫn (Instruction):** Một mô tả rõ ràng về tác vụ cần thực hiện.
- Đầu vào (Input) (tùy chọn):** Ngữ cảnh hoặc dữ liệu cụ thể để thực hiện chỉ dẫn.
- Đầu ra (Output) / Kết quả (Response):** Câu trả lời mong muốn mà mô hình nên tạo ra.

### Ví dụ 13: Các loại mẫu dữ liệu chỉ dẫn

#### 1. Mẫu không có đầu vào (Input-free):

- Instruction:** "Viết một bài thơ haiku về mùa thu."
- Output:** "Lá vàng rơi nhẹ,  
Heo may se sắt se lòng,  
Chờ đông lạnh về."

#### 2. Mẫu có đầu vào (With Input):

- Instruction:** "Tóm tắt đoạn văn sau thành một câu."
- Input:** "[Một đoạn văn dài về lịch sử của Trí tuệ Nhân tạo...]"
- Output:** "Trí tuệ Nhân tạo đã phát triển qua nhiều giai đoạn, từ logic biểu tượng đến học sâu, với Transformer là kiến trúc thống trị hiện nay."

### Yêu cầu về sự đa dạng

Để mô hình có khả năng tổng quát hóa, bộ dữ liệu phải cực kỳ đa dạng:

- Đa dạng về Tác vụ:** Bao gồm tất cả các loại tác vụ NLP có thể tương ứng được: tóm tắt, dịch thuật, hỏi-đáp (mở, đóng), phân loại, trích xuất thông tin, viết sáng tạo, viết lại văn bản, suy luận logic, giải toán, sinh mã nguồn, v.v.
- Đa dạng về Ngôn ngữ:** Bao gồm các chỉ dẫn được diễn đạt theo nhiều cách khác nhau (ví dụ: "Tóm tắt đoạn sau", "Cho tôi một bản tóm tắt", "Nội dung chính của đoạn này là gì?").
- Đa dạng về Độ dài và Độ phức tạp:** Từ các chỉ dẫn ngắn gọn đến các yêu cầu phức tạp, nhiều bước.

### Các phương pháp xây dựng bộ dữ liệu

1. **Sử dụng các bộ dữ liệu học thuật hiện có** Các nhà nghiên cứu đã tổng hợp hàng trăm bộ dữ liệu NLP công khai (như SQuAD cho hỏi-đáp, MNLI cho suy luận) và biến đổi chúng thành định dạng (instruction, input, output). Đây là nguồn dữ liệu chất lượng cao nhưng có thể không đa dạng về mặt diễn đạt.

2. **Thu thập thủ công bởi con người (Human Annotation)** Đây là phương pháp tốn kém nhưng cho chất lượng cao nhất. Con người được yêu cầu viết ra các chỉ dẫn sáng tạo và các câu trả lời tương ứng. Bộ dữ liệu Alpaca nổi tiếng của Stanford [90] được tạo ra theo một cách tương tự, nhưng thông minh hơn.

3. **Tự sinh dữ liệu bằng LLM (Self-Instruct)** Kỹ thuật này, được tiên phong bởi bài báo Self-Instruct [99], là một bước đột phá giúp giảm chi phí tạo dữ liệu.

- Bước 1 (Tạo hạt giống):** Bắt đầu với một vài ví dụ chỉ dẫn được viết thủ công.
- Bước 2 (Sinh chỉ dẫn):** Đưa các ví dụ này vào một LLM mạnh (như GPT-3.5) và yêu cầu nó sinh ra thêm các chỉ dẫn mới, đa dạng về tác vụ.
- Bước 3 (Sinh đầu vào/đầu ra):** Với mỗi chỉ dẫn mới, lại yêu cầu LLM sinh ra một cặp (Input, Output) phù hợp.
- Bước 4 (Lọc và Tinh chỉnh):** Lọc bỏ các chỉ dẫn chất lượng thấp, trùng lặp, hoặc không thể thực hiện.

Bằng cách lặp lại quy trình này, người ta có thể tạo ra một bộ dữ liệu hàng chục nghìn mẫu chỉ dẫn từ một vài hạt giống ban đầu. Bộ dữ liệu Alpaca (52,000 mẫu) đã được tạo ra bằng cách sử dụng kỹ thuật này trên mô hình text-davinci-003 của OpenAI.

#### 5.4.4. Vai trò và Tác động của Instruction Tuning

Instruction Tuning có một tác động sâu sắc, làm thay đổi cơ bản hành vi của LLM.

### Tăng cường khả năng tuân theo mệnh lệnh và "Tính hữu ích"

Đây là tác động rõ ràng nhất. Sau khi được tinh chỉnh, mô hình học được cách trở thành một "trợ lý". Nó hiểu rằng khi người dùng đưa ra một prompt, đó là một yêu cầu cần được thực hiện, chứ không phải một chuỗi cần được hoàn thành. Điều này làm cho mô hình trở nên **hữu ích (helpful)** hơn rất nhiều.

### Cải thiện khả năng Tổng quát hóa Zero-shot

Một kết quả đáng ngạc nhiên là Instruction Tuning trên một tập hợp các tác vụ đa dạng sẽ cải thiện đáng kể hiệu năng của mô hình trên các **tác vụ hoàn toàn mới, chưa từng thấy (unseen tasks)** ở dạng zero-shot.

- **Lý do:** Bằng cách học cách tuân theo hàng ngàn loại chỉ dẫn khác nhau, mô hình không chỉ học cách làm từng tác vụ riêng lẻ, mà còn học được một "meta-skill" về cách "học để học" từ chỉ dẫn. Nó học được cách suy luận ra ý định đằng sau một chỉ dẫn mới và thực hiện nó một cách hợp lý.

### Cầu nối đến sự Căn chỉnh (Alignment)

Instruction Tuning là bước đầu tiên và thiết yếu trong một quy trình lớn hơn gọi là **căn chỉnh (alignment)** – làm cho hành vi của LLM phù hợp với các giá trị và mục tiêu của con người. Bằng cách dạy mô hình tuân theo chỉ dẫn, chúng ta đã đặt nền móng để sau này có thể dạy nó tuân theo các chỉ dẫn phức tạp hơn, chẳng hạn như "Hãy trả lời một cách trung thực" hoặc "Đừng đưa ra các nội dung độc hại".

Tóm lại, Instruction Tuning là một bước chuyển đổi, biến một mô hình ngôn ngữ thô thành một nền tảng có khả năng tương tác và thực thi, mở đường cho các ứng dụng thực tế và là tiền đề cho các kỹ thuật căn chỉnh phức tạp mà chúng ta sẽ khám phá trong phần tiếp theo.

## 5.5. Các phương pháp Căn chỉnh (Alignment) với Con người

Sau khi đã qua bước Tinh chỉnh theo chỉ dẫn (Instruction Tuning), mô hình của chúng ta đã học được cách tuân theo mệnh lệnh. Tuy nhiên, nó vẫn có thể tạo ra các kết quả không mong muốn:

- **Không hữu ích (Unhelpful):** Trả lời "Tôi không biết" quá thường xuyên, hoặc đưa ra các câu trả lời không liên quan.
- **Không trung thực (Dishonest) / Bịa đặt (Hallucination):** Tự tin bịa đặt thông tin, trích dẫn các nguồn không tồn tại.
- **Độc hại (Harmful):** Tạo ra các nội dung nguy hiểm, thiên vị, hoặc xúc phạm, ngay cả khi không được yêu cầu trực tiếp.

Vấn đề là, các khái niệm như "hữu ích", "trung thực", "vô hại" rất phức tạp, mơ hồ và phụ thuộc vào ngữ cảnh. Chúng ta không thể định nghĩa chúng bằng một hàm mất mát đơn giản. Chúng ta cần một cách để truyền đạt những sở thích (preferences) tinh tế của con người vào mô hình. Đây là lúc các phương pháp Căn chỉnh (Alignment) phát huy tác dụng.

### 5.5.1. Mục tiêu Căn chỉnh: Bộ ba H (The 3 H's)

Mục tiêu cuối cùng của quá trình căn chỉnh thường được tóm gọn trong bộ ba nguyên tắc, được đề xuất bởi các nhà nghiên cứu tại Anthropic, thường được gọi là "HHH":

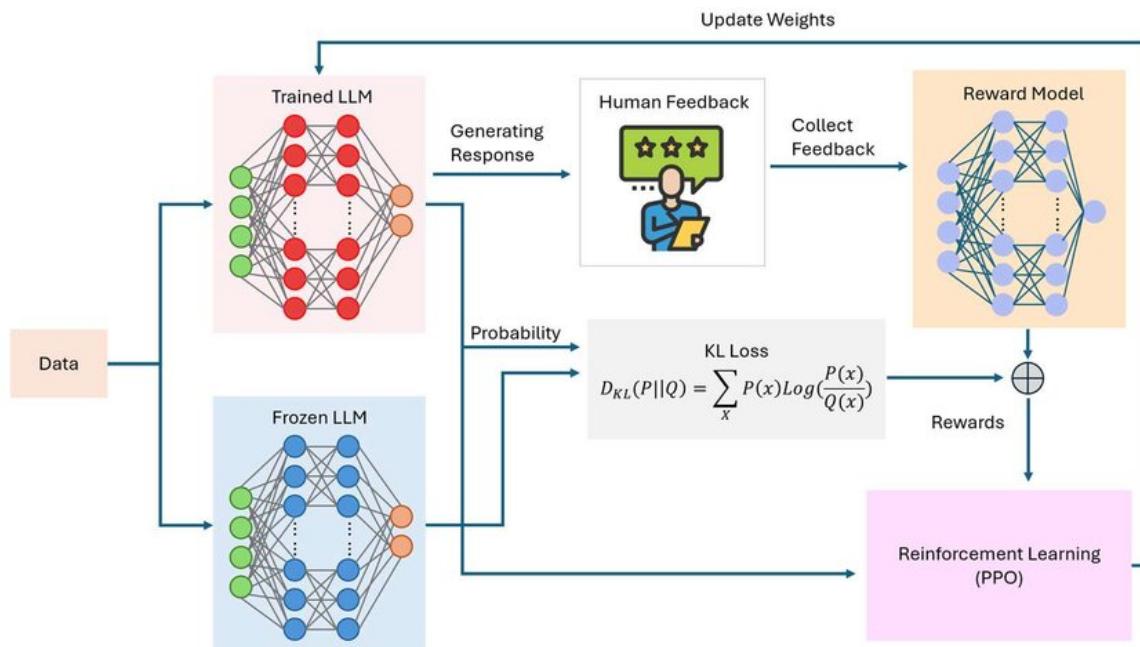
1. **Helpful (Hữu ích):** Mô hình nên cố gắng thực hiện ý định của người dùng một cách hiệu quả và đầy đủ. Nó nên tuân theo chỉ dẫn, thay vì lảng tránh hoặc từ chối một cách không cần thiết.
2. **Honest (Trung thực):** Mô hình không nên cố ý lừa dối hoặc bịa đặt thông tin. Khi không chắc chắn, nó nên thể hiện sự không chắc chắn đó thay vì đưa ra một câu trả lời sai một cách tự tin.

**3. Harmless (Vô hại):** Mô hình không nên gây ra tác hại, dù là cho người dùng trực tiếp hay cho xã hội nói chung. Nó phải từ chối các yêu cầu tạo ra nội dung nguy hiểm, phi đạo đức, hoặc bất hợp pháp.

Việc cân bằng ba mục tiêu này là một thách thức lớn (ví dụ, một câu trả lời hoàn toàn trung thực có thể gây tổn thương và không "vô hại"). Các phương pháp căn chỉnh dưới đây là nỗ lực để đạt được sự cân bằng này.

### 5.5.2. Học Tăng cường từ Phản hồi Con người (RLHF)

Reinforcement Learning from Human Feedback (RLHF) [13, 87] là phương pháp tiên phong và nổi tiếng nhất để căn chỉnh LLM, là công nghệ cốt lõi đằng sau thành công của ChatGPT. Đây là một quy trình gồm ba bước phức tạp.



Hình 5.3: Quy trình ba bước của RLHF: (1) Tinh chỉnh có giám sát (SFT), (2) Huấn luyện Mô hình Phần thưởng (RM), (3) Tối ưu hóa bằng Học tăng cường (PPO).

#### Bước 1: Tinh chỉnh có giám sát (Supervised Fine-Tuning - SFT)

- Mục tiêu:** Dạy cho mô hình văn phong và định dạng của một trợ lý đối thoại.
- Quy trình:** Đây chính là bước Instruction Tuning mà chúng ta đã học ở mục 5.4. Một nhóm nhỏ những người gán nhãn chuyên nghiệp (labelers) sẽ viết các prompt chất lượng cao và các câu trả lời mẫu lý tưởng. Mô hình ngôn ngữ cơ sở (base LLM) sau đó được fine-tune trên bộ dữ liệu này.
- Kết quả:** Một mô hình SFT có khả năng tuân theo chỉ dẫn, nhưng chưa được căn chỉnh sâu về các sở thích tinh tế.

#### Bước 2: Huấn luyện một Mô hình Phần thưởng (Reward Model - RM)

Đây là bước cốt lõi của RLHF.

- Mục tiêu:** Dạy cho một mô hình khác (gọi là Mô hình Phần thưởng) cách đánh giá chất lượng của một câu trả lời, mô phỏng lại sở thích của con người. RM sẽ nhận vào một cặp (prompt, response) và trả về một điểm số duy nhất thể hiện mức độ "tốt" của câu trả lời đó.
- Quy trình thu thập dữ liệu:**
  - Lấy một prompt từ bộ dữ liệu.

2. Cho mô hình SFT (từ bước 1) sinh ra nhiều câu trả lời khác nhau (ví dụ: 4 đến 9 câu).
3. Yêu cầu người gán nhãn **xếp hạng (rank)** các câu trả lời này từ tốt nhất đến tệ nhất, dựa trên các tiêu chí HHH.

Việc yêu cầu con người xếp hạng thay vì cho điểm trực tiếp sẽ dễ dàng và nhất quán hơn nhiều.

- **Quy trình huấn luyện RM:**

- Dữ liệu huấn luyện của RM là các cặp so sánh. Ví dụ, với một prompt, nếu câu trả lời  $A$  được xếp hạng cao hơn câu trả lời  $B$ , chúng ta có một điểm dữ liệu '(prompt, response\_A, response\_B)' với nhãn 'A > B'.
- RM (thường là một mô hình Encoder-Only như RoBERTa hoặc một LLM đã được loại bỏ lớp cuối) được huấn luyện để dự đoán xác suất một câu trả lời sẽ được con người ưa thích hơn. Cụ thể, nó được huấn luyện để gán một điểm số  $r(x, y)$  cho mỗi cặp (prompt  $x$ , response  $y$ ) sao cho với các cặp được xếp hạng,  $r(x, y_{\text{thích}}) > r(x, y_{\text{kém}})$ .
- **Kết quả:** Một Mô hình Phần thưởng có khả năng "chấm điểm" bất kỳ câu trả lời nào, hoạt động như một "giám khảo AI" được đào tạo từ sở thích của con người.

### Bước 3: Tối ưu hóa bằng Học tăng cường (Reinforcement Learning)

- **Mục tiêu:** Sử dụng Mô hình Phần thưởng để tinh chỉnh chính sách của mô hình SFT.
- **Thiết lập Học tăng cường (RL):**
  - **Tác tử (Agent):** Chính là mô hình SFT.
  - **Chính sách (Policy):** Là chính mô hình SFT, quyết định xác suất sinh ra một token khi biết trạng thái.
  - **Không gian hành động (Action Space):** Toàn bộ từ vựng của mô hình.
  - **Môi trường (Environment):** Người dùng đưa ra một prompt.
  - **Phần thưởng (Reward):** Điểm số được tính bởi Mô hình Phần thưởng (từ bước 2).
- **Quy trình huấn luyện:**
  1. Lấy một prompt từ bộ dữ liệu.
  2. Mô hình SFT (chính sách hiện tại) sinh ra một câu trả lời.
  3. Mô hình Phần thưởng chấm điểm câu trả lời này để tạo ra một phần thưởng.
  4. Phần thưởng này được sử dụng làm tín hiệu để cập nhật các trọng số của mô hình SFT thông qua một thuật toán tối ưu hóa chính sách như **Proximal Policy Optimization (PPO)** [80].
  5. Một thành phần quan trọng được thêm vào hàm phần thưởng là một **hình phạt KL-divergence**. Nó đo lường sự khác biệt giữa chính sách hiện tại và chính sách của mô hình SFT gốc. Hình phạt này ngăn không cho mô hình "tối ưu quá mức" cho Mô hình Phần thưởng và đi quá xa so với những gì nó đã học, giúp nó không bị "quên" các kiến thức ngôn ngữ cơ bản.
- **Kết quả:** Một mô hình đã được căn chỉnh cuối cùng, có khả năng sinh ra các câu trả lời vừa tuân theo chỉ dẫn, vừa được con người ưa thích.

#### 5.5.3. Các phương pháp thay thế: Hướng tối sự Đơn giản và Ổn định

RLHF cực kỳ mạnh mẽ nhưng cũng rất phức tạp, tốn kém và không ổn định để huấn luyện. Điều này đã thúc đẩy việc tìm kiếm các phương pháp thay thế đơn giản hơn.

##### Direct Preference Optimization (DPO)

- **Ý tưởng cốt lõi (Rafailov et al., 2023) [71]:** Tại sao phải huấn luyện một Mô hình Phần thưởng riêng biệt rồi lại dùng RL để tối ưu hóa nó? Chúng ta có thể tối ưu hóa trực tiếp trên dữ liệu sở thích của con người chỉ bằng một hàm mất mát duy nhất.
- **Cơ chế:** DPO chứng minh rằng hàm mất mát của RLHF có thể được viết lại thành một hàm mất mát đơn giản, có thể được tối ưu hóa trực tiếp bằng cách fine-tuning có giám sát.

- **Quy trình:** DPO chỉ cần dữ liệu so sánh (prompt, câu trả lời được thích hơn, câu trả lời kém hơn) và fine-tune mô hình SFT trực tiếp trên dữ liệu này. Nó tối ưu hóa mô hình để tăng xác suất của các câu trả lời được ưa thích và giảm xác suất của các câu trả lời không được ưa thích, trong khi vẫn duy trì một khoảng cách nhất định với mô hình tham chiếu.
- **Lợi ích:** Đơn giản hơn rất nhiều, ổn định hơn, và thường cho kết quả tương đương hoặc tốt hơn RLHF. DPO đang nhanh chóng trở thành một tiêu chuẩn mới cho việc căn chỉnh.

### Constitutional AI (CAI)

- **Ý tưởng cốt lõi** (Bai et al., 2022) [3]: Giảm sự phụ thuộc vào phản hồi của con người, đặc biệt là trong việc làm cho mô hình trở nên vô hại. Thay vì yêu cầu con người gán nhãn các nội dung độc hại, hãy để mô hình tự học cách tuân theo một bộ "hiến pháp" (constitution) gồm các nguyên tắc cơ bản.
- **Quy trình (Giai đoạn RL from AI Feedback - RLAIF):**
  1. **Phê bình (Critique):** Cho mô hình SFT xem một prompt có thể gây hại. Yêu cầu chính mô hình (hoặc một mô hình khác) phê bình câu trả lời của nó dựa trên các nguyên tắc trong hiến pháp.
  2. **Sửa đổi (Revision):** Yêu cầu mô hình viết lại câu trả lời dựa trên những lời phê bình đó để làm cho nó tuân thủ hiến pháp hơn.
  3. **Huấn luyện:** Cập (câu trả lời gốc, câu trả lời đã sửa đổi) bây giờ có thể được sử dụng làm dữ liệu sở thích (sửa đổi > gốc) để huấn luyện một Mô hình Phản thường hoặc sử dụng DPO.
- **Lợi ích:** Tự động hóa quá trình tạo dữ liệu căn chỉnh về tính vô hại, giảm bớt gánh nặng và sự phơi nhiễm của con người với các nội dung độc hại.

Quá trình căn chỉnh là bước cuối cùng và cũng là một trong những bước phức tạp nhất, biến một công cụ mạnh mẽ thành một đối tác đáng tin cậy.

---

## KẾT THÚC CHƯƠNG 5

*Chương này đã trang bị cho bạn những kỹ thuật thiết yếu để biến một mô hình nền tảng thành một công cụ hữu ích và an toàn. Chúng ta đã đi từ các phương pháp tinh chỉnh kinh điển, đến các kỹ thuật PEFT hiệu quả, và khám phá sức mạnh của việc "trò chuyện" với mô hình thông qua Prompt Engineering. Cuối cùng, chúng ta đã tìm hiểu các quy trình phức tạp như RLHF và DPO để căn chỉnh hành vi của mô hình với sở thích của con người. Với một mô hình đã được tinh chỉnh và căn chỉnh, chúng ta đã có một "bộ não" AI mạnh mẽ. Tuy nhiên, để giải quyết các vấn đề phức tạp trong thế giới thực, bộ não này cần có khả năng truy cập kiến thức bên ngoài, sử dụng công cụ, và thậm chí tương tác với các phương thức khác như hình ảnh. Chương tiếp theo sẽ khám phá các hệ thống và kiến trúc nâng cao kết hợp LLM với các thành phần bên ngoài, tạo ra các ứng dụng AI thực sự toàn diện.*



## Chương 6

# CÁC HỆ THỐNG VÀ KIẾN TRÚC NÂNG CAO

Sau khi đã nắm vững các kiến trúc LLM nền tảng và cách tinh chỉnh chúng, chúng ta sẽ bước vào một lĩnh vực cao hơn: làm thế nào để xây dựng các **hệ thống (systems)** hoàn chỉnh, có khả năng kết hợp LLM với các nguồn kiến thức bên ngoài.

Một trong những hạn chế lớn nhất của LLM là:

1. **Kiến thức bị giới hạn (Knowledge Cutoff):** Kiến thức của chúng bị đóng băng tại thời điểm huấn luyện và không thể tự cập nhật thông tin mới.
2. **Ảo giác (Hallucination):** Chúng có xu hướng bị đặt thông tin một cách tự tin khi không biết câu trả lời.
3. **Thiếu khả năng truy cập dữ liệu riêng tư:** Chúng không thể trả lời các câu hỏi về các tài liệu nội bộ của một công ty hay dữ liệu cá nhân của người dùng.

Để giải quyết những vấn đề này, một kiến trúc đã nổi lên như một tiêu chuẩn vàng trong ngành: **Tìm kiếm và Sinh Tăng cường (Retrieval-Augmented Generation - RAG).**

### 6.1. Tìm kiếm và Sinh Tăng cường (Retrieval-Augmented Generation - RAG)

#### Trực giác cốt lõi của RAG

Thay vì buộc LLM phải "nhớ" mọi thứ, hãy cho nó làm một "bài kiểm tra mở sách" (open-book exam). Khi nhận được một câu hỏi, thay vì trả lời ngay lập tức từ trí nhớ, hệ thống RAG trước hết sẽ **tìm kiếm (retrieve)** các thông tin liên quan nhất từ một cơ sở tri thức bên ngoài (ví dụ: tài liệu công ty, Wikipedia, các file PDF). Sau đó, nó sẽ cung cấp các thông tin đã tìm được này, cùng với câu hỏi gốc, cho LLM để **sinh ra (generate)** một câu trả lời dựa trên các bằng chứng đó.

RAG biến LLM từ một "nhà hiền triết" biết tuốt (nhưng đôi khi đăng trí và bịa chuyện) thành một "nhà nghiên cứu" tài ba, có khả năng tìm kiếm, tổng hợp và trả lời dựa trên các nguồn thông tin xác thực.

#### 6.1.1. Tại sao và Khi nào chúng ta cần RAG?

Trước khi đi sâu vào kỹ thuật, điều quan trọng là phải hiểu rõ các vấn đề mà RAG giải quyết và khi nào nó là công cụ phù hợp.

**Các vấn đề cố hữu của LLM** Các mô hình ngôn ngữ lớn, dù mạnh mẽ đến đâu, vẫn đối mặt với ba thách thức lớn:

- **Bịa đặt thông tin (Hallucination):** LLM có xu hướng tự "sáng tạo" ra các thông tin không có thật một cách rất tự tin, gây ra rủi ro nghiêm trọng trong các ứng dụng đòi hỏi tính chính xác.

- **Giới hạn tri thức (Knowledge Cutoff):** Tri thức của LLM bị đóng băng tại thời điểm nó được huấn luyện. Nó không biết về các sự kiện, dữ liệu hoặc thông tin mới xuất hiện sau ngày đó.
- **Thiếu tính kiểm chứng (Lack of Verifiability):** Khi LLM đưa ra một câu trả lời, rất khó để truy ngược lại nguồn thông tin mà nó đã sử dụng, khiến việc xác thực trở nên bất khả thi.

RAG được thiết kế để giải quyết trực tiếp cả ba vấn đề này bằng cách cung cấp cho LLM các bằng chứng cập nhật và có thể kiểm chứng tại thời điểm trả lời.

### Khi nào nên sử dụng RAG?

- Khi ứng dụng của bạn cần truy cập vào **kho tri thức riêng tư, lớn, hoặc thay đổi liên tục** (tài liệu nội bộ, cơ sở dữ liệu sản phẩm, văn bản pháp luật mới). Việc fine-tune lại LLM liên tục là không khả thi và tốn kém.
- Khi **tính chính xác và khả năng trích dẫn nguồn** là yêu cầu bắt buộc (trợ lý pháp lý, tư vấn y tế, tra cứu thông tin tài chính).
- Khi mục tiêu chính là **giảm thiểu rủi ro bịa đặt thông tin** và đảm bảo câu trả lời được neo vào các nguồn xác thực.

#### 6.1.2. Kiến trúc RAG Cơ bản (The Naive RAG Pipeline)

Một hệ thống RAG cơ bản, thường được gọi là "Naive" hay "Vanilla" RAG, bao gồm hai giai đoạn chính: **Lập chỉ mục (Indexing)** và **Truy vấn (Querying)**. Đây là nền tảng để hiểu tất cả các kỹ thuật nâng cao sau này.

**Giai đoạn 1: Lập chỉ mục (Offline)** Giai đoạn này được thực hiện một lần hoặc định kỳ để chuẩn bị cơ sở tri thức. Nó giống như việc chúng ta sắp xếp và tạo mục lục cho một thư viện.

1. **Tải Dữ liệu (Load):** Tải các tài liệu từ nguồn (PDF, TXT, HTML, v.v.).
2. **Phân mảnh (Chunk/Split):** Chia các tài liệu dài thành các đoạn nhỏ hơn, gọi là các "chunk". Trong kiến trúc cơ bản, đây thường là *fixed-size chunking* (chia thành các đoạn có kích thước cố định).
3. **Nhúng (Embed):** Sử dụng một mô hình nhúng (embedding model, ví dụ: các bi-encoder từ 'sentence-transformers') để chuyển mỗi chunk thành một vector số (embedding). Vector này nắm bắt ngữ nghĩa của chunk đó.
4. **Lưu trữ (Store):** Lưu trữ các chunk và các vector embedding tương ứng của chúng vào một **Cơ sở dữ liệu Vector** (Vector Database) như FAISS, ChromaDB, hoặc pgvector.

**Giai đoạn 2: Truy vấn (Online)** Giai đoạn này xảy ra mỗi khi người dùng đặt câu hỏi.

1. **Nhúng Câu hỏi (Query Embedding):** Câu hỏi của người dùng được nhúng thành một vector truy vấn, sử dụng cùng một **embedding model** đã dùng ở giai đoạn lập chỉ mục.
2. **Tìm kiếm (Retrieve):** Vector truy vấn được sử dụng để thực hiện một *tìm kiếm tương đồng* (*similarity search*) trong Vector Database, nhằm tìm ra  $k$  chunk có vector gần nhất (tương đồng nhất về mặt ngữ nghĩa).
3. **Tăng cường Prompt (Augment):** Các chunk đã tìm được được ghép vào prompt cùng với câu hỏi gốc của người dùng.

Ngữ cảnh được cung cấp:

[Nội dung của chunk 1...]

[Nội dung của chunk 2...]

[...]

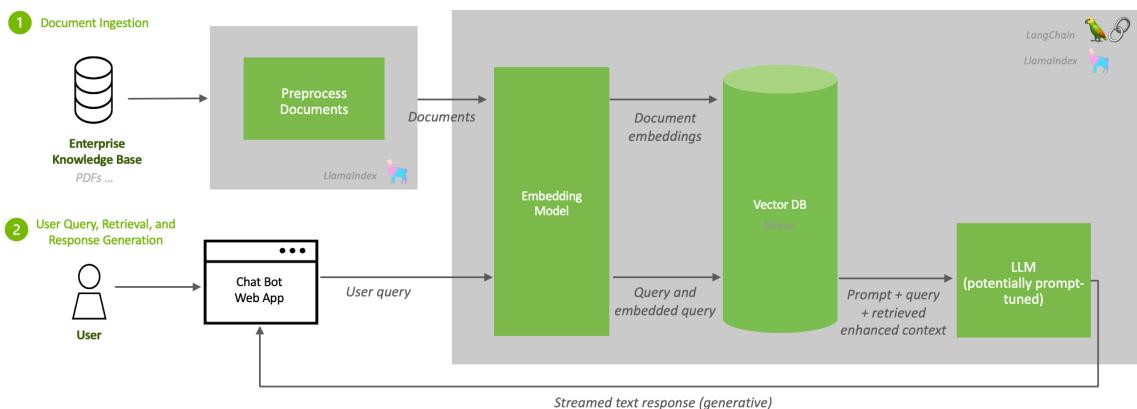
Dựa vào ngữ cảnh trên, hãy trả lời câu hỏi sau:

[Câu hỏi gốc của người dùng]

Trả lời:

- Sinh Câu trả lời (Generate): Prompt đã được tăng cường này được gửi đến một LLM. LLM sẽ tổng hợp thông tin từ ngữ cảnh được cung cấp để tạo ra câu trả lời cuối cùng.

### Retrieval Augmented Generation (RAG) Sequence Diagram



Hình 6.1: Sơ đồ kiến trúc của một hệ thống RAG, bao gồm giai đoạn Lập chỉ mục (bên trên) và giai đoạn Truy vấn (bên dưới).

#### 6.1.3. Bài học từ các Công cụ Tìm kiếm: Kiến trúc Phân tầng của Google

Để hiểu tại sao các hệ thống RAG hiện đại được thiết kế theo một cách cụ thể, sẽ rất hữu ích khi nhìn vào “người khổng lồ” trong lĩnh vực tìm kiếm: Google. Kiến trúc của Google Search là một ví dụ điển hình về một hệ thống retrieval đa tầng (multi-stage), nơi mỗi tầng sẽ lọc và tinh chỉnh kết quả, cân bằng giữa tốc độ và độ chính xác.

**Kiến trúc Phân tầng (A Tiered Architecture)** Một truy vấn trên Google không chỉ đơn giản là “tìm và trả về”. Nó đi qua một “phễu” tinh vi gồm nhiều giai đoạn:

- Crawl & Indexing (Thu thập và Lập chỉ mục):** Trước khi bất kỳ truy vấn nào được xử lý, Googlebot liên tục crawl web, tải về nội dung và phân tích cấu trúc trang. Dữ liệu sau đó được chuẩn hóa và lưu trong nhiều loại chỉ mục:
  - Inverted Index (Sparse Index):* ánh xạ từ khoá → danh sách tài liệu chứa từ đó.
  - Dense Index (Vector Index):* biểu diễn văn bản dưới dạng embedding để phục vụ tìm kiếm ngữ nghĩa.
  - Ngoài ra, các siêu dữ liệu (metadata), đồ thị liên kết (PageRank), và thông tin đa phương tiện (hình ảnh, video) cũng được lưu trữ song song.
- Query Understanding (Hiểu Truy vấn):** Khi người dùng nhập một truy vấn, Google áp dụng các mô hình ngôn ngữ tiên tiến (BERT, MUM, T5) để phân tích ý định và ngữ nghĩa. Các bước xử lý điển hình gồm:
  - Chuẩn hóa, sửa lỗi chính tả, mở rộng từ đồng nghĩa.
  - Nhận diện thực thể (entity recognition) và giải quyết nhập nhằng (disambiguation).
  - Viết lại truy vấn (query rewriting) và cá nhân hóa dựa trên ngôn ngữ, vị trí, lịch sử tìm kiếm.
- Candidate Generation (Tạo Ứng viên - Nhanh và Rộng):**
  - Mục tiêu:** Từ hàng tỷ tài liệu, nhanh chóng lọc ra một tập hợp lớn (10,000–100,000) ứng viên tiềm năng.
  - Công nghệ:** Kết hợp Sparse Retrieval (BM25/TF-IDF) với Dense Retrieval (bi-encoders + ANN search). Giai đoạn này ưu tiên Recall – đảm bảo không bỏ sót kết quả quan trọng.
- Initial Ranking (Xếp hạng ban đầu):**

- **Mục tiêu:** Thu hẹp tập ứng viên từ hàng chục nghìn xuống còn vài nghìn tài liệu có triển vọng.
- **Công nghệ:** Sử dụng *Learning-to-Rank models* (boosted trees, shallow neural nets) để kết hợp nhiều tín hiệu: điểm BM25, PageRank, độ mới (freshness), anchor text, độ phổ biến của tài liệu. Đây là lớp lọc trung gian quan trọng trước khi áp dụng mô hình ngôn ngữ nặng nề.

### 5. Neural Re-ranking & Final Touches (Xếp hạng lại bằng mô hình Neural và Tinh chỉnh cuối cùng):

- **Mục tiêu:** Đạt độ chính xác cao nhất trong top kết quả hiển thị (thường là vài chục link).
- **Công nghệ:** Các *Cross-Encoders* hoặc mô hình seq2seq (như T5, BERT, MUM) được áp dụng cho vài trăm ứng viên. Các mô hình này đánh giá trực tiếp cặp (*truy vấn, tài liệu*) nên rất chính xác, dù tốn kém về tính toán.
- **Tinh chỉnh cuối cùng:** Diversification (đa dạng hoá kết quả), lọc spam, tích hợp Knowledge Graph, snippet, và kết hợp các nguồn media (tin tức, hình ảnh, video).
- Ngoài ra, các tín hiệu hành vi người dùng (CTR, dwell time) được khai thác để điều chỉnh thứ hạng theo thời gian.

#### Bài học cốt lõi cho RAG từ Google

Xây dựng một hệ thống retrieval mạnh mẽ không phải là việc chọn một thuật toán duy nhất. Đó là việc xây dựng một **pipeline đa tầng**. Chúng ta chấp nhận sử dụng các phương pháp "đủ tốt" và nhanh ở giai đoạn đầu để lọc số lượng lớn, sau đó mới đầu tư tài nguyên tính toán vào các phương pháp chính xác cao cho một số lượng nhỏ các ứng viên hứa hẹn nhất. Triết lý "lọc rộng rồi tinh chỉnh sâu" này chính là nền tảng cho các hệ thống RAG hiện đại.

#### 6.1.4. Tối ưu hóa Giai đoạn 1: Lập chỉ mục Thông minh (Smarter Indexing)

Chất lượng của các chunk được lập chỉ mục ảnh hưởng trực tiếp đến khả năng tìm thấy thông tin chính xác. "Rác vào, Rác ra". Các kỹ thuật dưới đây giúp cải thiện "chất lượng đầu vào" cho hệ thống RAG.

#### Chiến lược Phân mảnh Nâng cao (Advanced Chunking Strategies)

Việc chia văn bản một cách "mù quáng" theo kích thước cố định (fixed-size chunking) thường làm phá vỡ ngữ nghĩa của câu hoặc đoạn văn.

- **Chunking theo Ngữ nghĩa/Cấu trúc (Semantic/Structural Chunking):** Thay vì chia theo số ký tự, chúng ta chia dựa trên các ranh giới tự nhiên của văn bản như dấu ngắt câu, đoạn văn, tiêu đề (Markdown headers), hoặc các thẻ HTML. Điều này giúp các chunk giữ được sự mạch lạc và trọn vẹn về ý nghĩa.
- **Chunking có chồng lấn (Overlap Chunking):** Để tránh mất ngữ cảnh ở điểm nối giữa hai chunk, một phần nhỏ của chunk trước được lặp lại ở đầu chunk sau. Ví dụ: chunk 1 chứa các câu [1, 2, 3, 4], chunk 2 sẽ chứa các câu [4, 5, 6, 7].

#### Kỹ thuật Small-to-Big và Sentence-Windowing

Đây là một kỹ thuật mạnh mẽ để cân bằng giữa độ chính xác khi tìm kiếm và đầy đủ ngữ cảnh cho LLM.

- **Vấn đề:** Các chunk lớn (nguyên đoạn văn) chứa nhiều ngữ cảnh nhưng khó khớp chính xác với một câu hỏi cụ thể. Các chunk nhỏ (một câu) thì dễ khớp chính xác nhưng lại thiếu ngữ cảnh xung quanh.
- **Giải pháp "Small-to-Big":**
  1. **Lập chỉ mục (Index):** Chia tài liệu thành các đơn vị nhỏ (ví dụ: từng câu) và nhúng chúng.

2. **Tìm kiếm (Retrieve):** Tìm kiếm trên các câu đơn lẻ để có độ chính xác cao nhất.
3. **Mở rộng (Expand):** Sau khi tìm được câu phù hợp nhất, thay vì chỉ trả về câu đó, hệ thống sẽ lấy ra một "cửa sổ" văn bản lớn hơn xung quanh câu đó (ví dụ: 3 câu trước và 3 câu sau, hoặc toàn bộ đoạn văn chứa câu đó) để cung cấp cho LLM.  
Kỹ thuật này giúp đạt được sự chính xác trong tìm kiếm và sự đầy đủ về ngữ cảnh khi sinh câu trả lời.

### Tâm quan trọng của Metadata

Metadata là "thông tin về thông tin". Khi lập chỉ mục, việc lưu trữ metadata cùng với mỗi chunk là cực kỳ quan trọng cho các ứng dụng thực tế.

- **Ví dụ về Metadata:** Tên file nguồn, số trang, ngày xuất bản, tác giả, tiêu đề chương/mục, URL.
- **Tại sao nó quan trọng?**
  - **Trích dẫn nguồn:** Giúp LLM có thể trích dẫn nguồn gốc của thông tin một cách chính xác trong câu trả lời (ví dụ: "Theo tài liệu X, trang 5...").
  - **Lọc trước khi tìm kiếm (Pre-filtering):** Cho phép thu hẹp không gian tìm kiếm. Ví dụ: "Chỉ tìm kiếm trong các tài liệu được xuất bản sau năm 2023".
  - **Gỡ lỗi (Debugging):** Dễ dàng truy vết xem chunk gây ra câu trả lời sai đến từ tài liệu nào.

### 6.1.5. Tối ưu hóa Giai đoạn 2: Tìm kiếm và Xếp hạng (Advanced Retrieval & Ranking)

#### Mục tiêu

Retriever là trái tim của RAG. Giai đoạn này quyết định "chất lượng nguyên liệu đầu vào" cho LLM. Một hệ thống RAG chỉ có thể tốt bằng thành phần retriever của nó. Các kỹ thuật dưới đây giúp tìm được những mảnh ghép thông tin (chunk) chính xác, liên quan và đa dạng nhất.

### 6.1.6. Tối ưu hóa Giai đoạn 2: Tìm kiếm và Xếp hạng (Advanced Retrieval & Ranking)

Lấy cảm hứng từ kiến trúc của Google, giờ đây chúng ta sẽ áp dụng triết lý đa tầng vào hệ thống RAG của mình.

#### Tìm kiếm Lai (Hybrid Search): Giai đoạn Candidate Generation của RAG

Đây chính là giai đoạn 1 của chúng ta. Thay vì chỉ dùng một phương pháp, ta kết hợp cả hai:

- **Cơ chế:** Hệ thống chạy song song cả Sparse Retriever (BM25) và Dense Retriever (Vector Search). Sau đó, điểm số từ hai hệ thống được kết hợp lại (ví dụ: dùng thuật toán Reciprocal Rank Fusion - RRF) để tạo ra một danh sách ứng viên (candidate list) chất lượng cao.
- **Lợi ích:** Tận dụng khả năng khớp từ khóa chính xác của Sparse và khả năng hiểu ngữ nghĩa sâu sắc của Dense. Đây được coi là phương pháp mặc định để có được recall tốt nhất.

#### Xếp hạng lại (Re-ranking) với Cross-Encoders

- **Vấn đề:** Hybrid Search rất nhanh và hiệu quả để lọc ra top 100-200 ứng cử viên tiềm năng từ hàng triệu chunk. Nhưng trong top 100 này, đâu mới là những chunk thực sự tốt nhất?
- **Giải pháp:** Sử dụng một mô hình thứ hai, mạnh hơn nhưng chậm hơn, gọi là Cross-Encoder.
  - **Cơ chế:** Thay vì tạo vector riêng cho câu hỏi và tài liệu, Cross-Encoder nhận vào cặp (câu hỏi, chunk) và đưa ra một điểm số tương đồng duy nhất. Việc xem xét cả hai cùng lúc cho phép nó hiểu được sự tương tác tinh vi hơn.
  - **Quy trình:** Hybrid Search tìm ra top-K ứng viên (K lớn). Sau đó, Cross-Encoder sẽ đánh giá và xếp hạng lại top-K này để chọn ra top-N cuối cùng (N nhỏ) gửi cho LLM.

### 6.1.7. Tối ưu hóa Giai đoạn 3: Tăng cường và Sinh (Smarter Augmentation & Generation)

Sau khi đã có những chunk chất lượng nhất, chúng ta cần đảm bảo chúng được sử dụng một cách hiệu quả và trung thực.

#### Nén Ngữ cảnh (Context Compression)

Đôi khi, các chunk được tìm thấy vẫn chứa nhiều thông tin thừa so với câu hỏi. Việc "nhồi nhét" tất cả vào prompt có thể làm LLM bị nhiễu.

- **Cơ chế:** Trước khi đưa các chunk vào prompt cuối cùng, một mô hình nhỏ hơn sẽ chạy qua và chỉ giữ lại những câu, những ý có liên quan trực tiếp đến câu hỏi của người dùng.
- **Lợi ích:** Giảm độ dài của prompt (tiết kiệm chi phí), tăng sự tập trung cho LLM vào các bằng chứng quan trọng nhất.

#### Kiểm soát việc Sinh: Buộc Trích dẫn và Chống Bịa đặt (Hallucination Guardrails)

Đây là bước cuối cùng và quan trọng nhất để đảm bảo tính tin cậy.

- **Instruction Tuning:** Huấn luyện hoặc chỉ dẫn LLM một cách rõ ràng thông qua prompt: *"Hãy trả lời câu hỏi chỉ dựa trên ngữ cảnh được cung cấp. Với mỗi thông tin đưa ra, hãy trích dẫn nguồn [doc\_id, chunk\_id]. Nếu thông tin không có trong ngữ cảnh, hãy trả lời 'Tôi không tìm thấy thông tin trong tài liệu'."*
- **Giảm Temperature:** Khi sinh câu trả lời, đặt tham số 'temperature' của LLM ở mức thấp (ví dụ: 0.1) để giảm tính ngẫu nhiên và sáng tạo, buộc nó bám sát vào văn bản nguồn.

### 6.1.8. Các Mô hình RAG Nâng cao (Advanced RAG Patterns)

Khi các bài toán trở nên phức tạp hơn, chúng ta cần các kiến trúc RAG tinh vi hơn.

- **Biến đổi Truy vấn (Query Transformation):** Câu hỏi của người dùng thường không phải là dạng tốt nhất để tìm kiếm. Các kỹ thuật này sẽ biến đổi nó trước:
  - **Viết lại Truy vấn (Query Rewriting):** Là nhóm kỹ thuật tập trung vào việc sửa đổi, mở rộng, hoặc diễn đạt lại văn bản câu hỏi gốc mà không làm thay đổi ý định tìm kiếm cốt lõi. Mục tiêu là cải thiện sự tương thích giữa từ vựng của người dùng và từ vựng trong kho tài liệu, qua đó tăng cả *recall* và *precision*. Các phương pháp này được chia thành ba hướng tiếp cận chính:
    - \* **Viết lại dựa trên Từ vựng (Lexical Rewriting):** Các phương pháp này hoạt động ở cấp độ từ hoặc ký tự, dựa trên các quy tắc, từ điển, hoặc các chuẩn hóa ngôn ngữ cơ bản.
      - **Mở rộng Từ đồng nghĩa (Synonym/Thesaurus Expansion):** Bổ sung các từ hoặc cụm từ đồng nghĩa (ví dụ: luật pháp → pháp luật, quy định) để tăng khả năng khớp, giúp bắt được các tài liệu sử dụng cách diễn đạt khác nhau.
      - **Chuẩn hóa Hình thái học (Morphological Normalization):** Đưa các biến thể của một từ về dạng gốc (stemming hoặc lemmatization). Ví dụ: các từ xử lý, được xử lý, sự xử lý đều được đưa về gốc xử lý.
      - **Sửa lỗi Chính tả/Gõ nhầm (Spelling/Typo Correction):** Tự động phát hiện và sửa các lỗi sai trong truy vấn (luật lao động → luật lao động) để tránh tìm kiếm thất bại.
      - **Mở rộng Từ viết tắt (Abbreviation Expansion):** Chuyển các từ viết tắt thành dạng đầy đủ của chúng (ví dụ: TP.HCM → Thành phố Hồ Chí Minh, LHQ → Liên Hợp Quốc).
      - **Xử lý Stopword (Stopword Handling):** Loại bỏ các từ phổ biến, ít mang nghĩa (như là, của, và) để tập trung vào từ khóa chính, hoặc ngược lại, giữ lại chúng khi cần thiết cho các truy vấn yêu cầu sự chính xác về ngữ pháp (phrase queries).

- \* **Viết lại dựa trên Thống kê (Statistical Rewriting):** Các kỹ thuật này tận dụng thông tin thống kê từ kho tài liệu hoặc từ kết quả tìm kiếm ban đầu để mở rộng truy vấn một cách tự động.
  - **Phản hồi Giả liên quan (Pseudo-Relevance Feedback - PRF):** Giả định rằng các tài liệu top-k trong kết quả tìm kiếm ban đầu là liên quan. Hệ thống sẽ trích xuất các thuật ngữ quan trọng từ các tài liệu này để bổ sung vào truy vấn gốc và thực hiện tìm kiếm lại. (Các thuật toán tiêu biểu: Rocchio, RM3).
  - **Mở rộng dựa trên Tần suất Cùng xuất hiện (Term Co-occurrence Expansion):** Tìm và thêm vào truy vấn các thuật ngữ thường xuyên xuất hiện cùng với các thuật ngữ gốc trong toàn bộ kho tài liệu (Dựa trên các độ đo như Pointwise Mutual Information - PMI).
  - **Mở rộng dựa trên Mô hình Chủ đề (Topic-Model-Based Expansion):** Sử dụng các mô hình như LDA (Latent Dirichlet Allocation) để xác định các chủ đề chính của truy vấn, sau đó bổ sung các từ khóa đặc trưng của những chủ đề đó.
  - **Mô hình Liên quan (Relevance Models):** Ước tính một mô hình xác suất của các thuật ngữ sẽ xuất hiện trong một tài liệu liên quan, sau đó lấy mẫu từ mô hình này để tạo ra truy vấn mới.
- \* **Viết lại dựa trên Ngữ nghĩa (Semantic Rewriting):** Các phương pháp hiện đại, tập trung vào việc nắm bắt và mở rộng ý nghĩa, ngữ cảnh của truy vấn thay vì chỉ dựa trên từ vựng bề mặt.
  - **Dựa trên Vector Embedding (Embedding-based Nearest Neighbors):** Biểu diễn truy vấn dưới dạng vector và tìm các từ/cụm từ có vector gần nhất trong không gian embedding để mở rộng (Ví dụ: sử dụng Word2Vec, GloVe, fastText).
  - **Diễn đạt lại bằng LLM (LLM-based Paraphrasing):** Sử dụng các mô hình ngôn ngữ lớn (LLM) như GPT-4 hay T5 để tạo ra một phiên bản câu hỏi hoàn toàn mới nhưng giữ nguyên ý nghĩa, giúp vượt qua rào cản từ vựng.
  - **Tạo nhiều Truy vấn song song (Multi-Query Rewriting):** Yêu cầu LLM sinh ra nhiều biến thể khác nhau của câu hỏi gốc. Hệ thống sẽ thực hiện tìm kiếm với tất cả các biến thể này và tổng hợp kết quả.
  - **Mở rộng bằng Đồ thị Tri thức (Knowledge-Graph Expansion):** Liên kết các thực thể trong truy vấn (tên người, địa điểm, tổ chức) với một đồ thị tri thức. Sau đó, mở rộng truy vấn bằng cách thêm vào các thông tin, thuộc tính, hoặc các thực thể liên quan được lấy từ đồ thị.
  - **Mở rộng theo Ngữ cảnh Hội thoại (Contextual Expansion):** Tận dụng lịch sử của cuộc trò chuyện để làm rõ và bổ sung thông tin cho truy vấn hiện tại. Ví dụ, câu hỏi "còn anh ấy thì sao?" sẽ được mở rộng với thông tin về "anh ấy" đã được đề cập trước đó.
- **Biến đổi Biểu diễn Truy vấn (Query Representation Transformation):** Khác với việc viết lại văn bản, nhóm kỹ thuật này giữ nguyên câu hỏi gốc nhưng biến đổi cách nó được *biểu diễn* (thường là dưới dạng vector) để tối ưu hóa việc so khớp trong không gian embedding.
  - \* **HyDE (Hypothetical Document Embeddings):** Một kỹ thuật đột phá, yêu cầu LLM tự sinh một tài liệu trả lời *giả định* cho câu hỏi. Thay vì nhúng vector của câu hỏi ngắn gọn và thiếu ngữ cảnh, hệ thống sẽ nhúng vector của tài liệu giả định này. Vector này thường giàu thông tin và nằm gần các tài liệu liên quan thực tế trong không gian embedding, do đó cải thiện đáng kể chất lượng tìm kiếm.
  - \* **Query2Doc (Q2D):** Một phương pháp tiền thân của HyDE, cũng biến đổi truy vấn thành một dạng trông giống tài liệu hơn, nhưng thường sử dụng các phương pháp heuristic hoặc các mô hình sinh văn bản gọn nhẹ (như T5) thay vì các LLM lớn.
  - \* **Tăng cường Embedding Truy vấn (Query Embedding Augmentation):** Cải thiện vector truy vấn gốc bằng cách kết hợp nó với các thông tin bổ sung.

- **Pooling với các Mở rộng (Average Pooling with Expansions):** Tạo embedding cho truy vấn gốc và các phiên bản mở rộng (ví dụ: từ đồng nghĩa), sau đó kết hợp chúng lại (ví dụ: lấy trung bình) để tạo ra một vector đại diện toàn diện hơn.
- **Hợp nhất Embedding (Embedding Fusion):** Thực hiện tìm kiếm song song với nhiều hệ thống retriever (ví dụ: một sparse retriever như BM25 và một dense retriever). Sau đó, vector truy vấn cuối cùng được điều chỉnh hoặc kết hợp dựa trên thông tin từ các kết quả tìm kiếm này.
- \* **Biểu diễn Truy vấn Tương phản (Contrastive Query Representation):** Tinh chỉnh (fine-tune) mô hình encoder để nó học được cách tạo ra các embedding tốt hơn. Cụ thể, mô hình được huấn luyện để vector của truy vấn gần với vector của tài liệu trả lời đúng (positive sample) và xa các tài liệu không liên quan (negative samples).
- \* **Biến đổi Ngầm định qua Kiến trúc (Implicit Transformation via Architecture):** Một số kiến trúc retriever không tạo ra một vector duy nhất cho truy vấn. Thay vào đó, chúng thực hiện một sự biến đổi ngầm thông qua tương tác sâu hơn giữa truy vấn và tài liệu.
  - **ColBERT (Late Interaction):** Mô hình này tạo ra embedding cho mỗi token trong cả truy vấn và tài liệu. Quá trình so khớp tính toán điểm tương đồng giữa mỗi token của truy vấn với tất cả các token của tài liệu, cho phép một sự tương tác chi tiết và linh hoạt hơn là so khớp hai vector đơn lẻ.
- **Phân tích và Cấu trúc hóa Truy vấn (Query Understanding & Structuring):** Nhóm kỹ thuật này không chỉ "hiểu" ý định của truy vấn mà còn "dịch" nó thành một định dạng có cấu trúc, cho phép tìm kiếm chính xác trên các nguồn dữ liệu có schema rõ ràng (structured/semi-structured data) như cơ sở dữ liệu hoặc kho tri thức.
  - \* **Tự truy vấn (Self-Querying):** LLM được cung cấp thông tin về siêu dữ liệu (metadata) của kho tài liệu (ví dụ: **năm xuất bản, tác giả, thể loại**). Khi nhận câu hỏi, LLM tự động phân tích và trích xuất các bộ lọc tương ứng. Ví dụ, câu hỏi "các bài báo của Hinton về mạng nơ-ron sau 2015" sẽ được dịch thành một truy vấn tìm kiếm nội dung "mạng nơ-ron" kèm theo hai bộ lọc: **tác giả = "Hinton"** và **năm > 2015**.
  - \* **Phân tích Ngữ nghĩa (Semantic Parsing):** Chuyển đổi hoàn toàn câu hỏi ngôn ngữ tự nhiên thành một truy vấn trong một ngôn ngữ truy vấn chính thức như SQL (cho cơ sở dữ liệu quan hệ), SPARQL (cho đồ thị tri thức), hoặc GraphQL.
  - \* **Phân rã Truy vấn (Query Decomposition / Multi-Hop):** Đối với các câu hỏi phức tạp đòi hỏi phải kết hợp thông tin từ nhiều nguồn, kỹ thuật này chia câu hỏi lớn thành một chuỗi các câu hỏi con đơn giản hơn. Kết quả của câu hỏi con trước được dùng làm đầu vào cho câu hỏi con tiếp theo.
  - \* **Phân định Nghĩa (Query Disambiguation):** Xác định ý nghĩa chính xác của các từ đa nghĩa dựa trên ngữ cảnh. Ví dụ, hệ thống cần phân biệt "Apple" là công ty công nghệ hay là một loại trái cây để có thể tìm kiếm đúng hướng.
  - \* **Phân loại Ý định và Trích xuất Thực thể (Intent Classification & Slot Filling):** Một cách tiếp cận phổ biến trong các hệ thống chatbot, trong đó hệ thống xác định mục đích chung của người dùng (intent) và trích xuất các thông tin chi tiết (entities/slots). Ví dụ, trong câu "đặt vé máy bay đến Hà Nội vào ngày mai", intent là "đặt vé", và các slot là **đích đến = "Hà Nội"** và **thời gian = "ngày mai"**.
  - \* **Phân loại theo Khía cạnh (Query Faceting):** Tự động nhận diện và tách câu hỏi thành nhiều khía cạnh hoặc thuộc tính khác nhau, thường được dùng trong các hệ thống tìm kiếm thương mại điện tử. Ví dụ, truy vấn "điện thoại Samsung dưới 10 triệu pin trâu" có thể được tách thành các facet: **thương hiệu = "Samsung"**, **giá < 10,000,000 VNĐ**, **tính năng = "pin dung lượng cao"**.
- **Viết lại Truy vấn dựa trên Học máy (Learning-based Query Rewriting):** Nhóm kỹ thuật

này coi việc tạo ra truy vấn tối ưu là một bài toán cần học từ dữ liệu, thay vì dựa trên các quy tắc hoặc heuristic cố định. Các mô hình được huấn luyện để tự động tạo ra các phiên bản truy vấn hiệu quả nhất.

- \* **Viết lại dựa trên Học tăng cường (Reinforcement Learning - RL):** Xây dựng một "tác nhân"(agent) thông minh có khả năng ra quyết định viết lại truy vấn thông qua thử và sai. Agent học một "chính sách"(policy) để tạo ra các truy vấn tốt nhất dựa trên "phản thưởng"(reward) nhận được từ môi trường.
- **Học để Truy xuất (Learning to Retrieve with RL):** Phần thưởng được tính trực tiếp dựa trên chất lượng của tập tài liệu truy xuất được (ví dụ: recall, precision, MRR). Agent được khuyến khích tạo ra các truy vấn tìm thấy nhiều tài liệu liên quan hơn.
- **RL dựa trên Chất lượng Câu trả lời (Answer-based RL):** Một cách tiếp cận sâu hơn, trong đó phần thưởng không chỉ dựa trên kết quả truy xuất mà còn dựa trên chất lượng của câu trả lời cuối cùng do mô hình sinh tạo ra. Điều này giúp tối ưu toàn bộ pipeline RAG một cách end-to-end.
- **Viết lại Chủ động (Active Query Rewriting):** Agent có khả năng tự quyết định xem có cần thực hiện thêm một bước viết lại nữa hay không. Nếu kết quả tìm kiếm ban đầu đã tốt, agent có thể dừng lại để tiết kiệm tài nguyên tính toán.
- **Học theo Lộ trình (Curriculum RL):** Agent được huấn luyện theo một lộ trình có cấu trúc, bắt đầu từ những câu hỏi đơn giản và dần dần tăng độ khó. Cách tiếp cận này giúp agent học các chiến lược viết lại một cách ổn định và hiệu quả hơn.
- **Học Tương tác (Interactive RL):** Tận dụng phản hồi trực tiếp từ người dùng (ví dụ: nhấn "thích"hoặc báo cáo kết quả không chính xác) như một nguồn tín hiệu phần thưởng bổ sung, giúp agent cá nhân hóa và thích ứng nhanh chóng.
- **Học Phân cấp (Hierarchical RL):** Sử dụng một kiến trúc RL đa tầng. Một "meta-agent" cấp cao sẽ quyết định *loại* chiến lược viết lại cần dùng (ví dụ: mở rộng từ đồng nghĩa, thêm bộ lọc, hay phân rã câu hỏi), trong khi một agent cấp thấp sẽ thực thi chi tiết chiến lược đó.
- \* **Truy xuất Khả vi End-to-End (End-to-End Differentiable Retrieval):** Các phương pháp này tích hợp bộ phận retriever vào trong quá trình huấn luyện của mô hình ngôn ngữ lớn, cho phép tối ưu hóa việc biểu diễn truy vấn một cách trực tiếp.
- **Học Biểu diễn Truy vấn (DPR, ColBERT, Retro):** Encoder của truy vấn được huấn luyện trực tiếp từ tín hiệu lỗi (loss) của tác vụ cuối cùng (ví dụ: trả lời câu hỏi). Quá trình này "ép" mô hình phải học cách tạo ra các embedding truy vấn hiệu quả nhất cho tác vụ đó.
- **Huấn luyện Tăng cường Truy xuất (Retrieval-Augmented Training):** Các mô hình như RAG hay Atlas tối ưu đồng thời cả retriever và generator. Retriever học cách tìm kiếm tài liệu hữu ích, và generator học cách tận dụng các tài liệu đó để sinh ra câu trả lời tốt hơn, tạo thành một vòng lặp tối ưu hóa lẫn nhau.
- **Định tuyến và Biến đổi Meta (Query Routing & Meta-Transformation):** Trong các hệ thống RAG phức tạp sử dụng nhiều công cụ hoặc nhiều nguồn dữ liệu, nhóm kỹ thuật này đóng vai trò như một "bộ điều phối" thông minh, quyết định cách xử lý truy vấn một cách hiệu quả nhất.
  - \* **Định tuyến Truy vấn (Query Routing):** Dựa trên nội dung của câu hỏi, hệ thống sẽ quyết định gửi nó đến retriever phù hợp nhất. Ví dụ, các câu hỏi chứa từ khóa rõ ràng có thể được gửi đến BM25 (sparse), trong khi các câu hỏi trùng tượng hơn sẽ được gửi đến dense retriever. Tương tự, hệ thống có thể định tuyến truy vấn đến các kho tri thức chuyên ngành khác nhau (ví dụ: y tế, pháp luật).
  - \* **Tái định dạng Truy vấn cho Công cụ (Query Reformulation for Tools):** Trong các hệ thống RAG dựa trên agent, LLM đóng vai trò trung tâm, có khả năng tự biến đổi

- câu hỏi của người dùng thành các lời gọi hàm hoặc các truy vấn API hợp lệ để tương tác với các công cụ bên ngoài (ví dụ: công cụ tìm kiếm web, máy tính, API thời tiết).
- \* **Hợp nhất Truy vấn (Query Fusion):** Thay vì chọn một retriever duy nhất, hệ thống gửi truy vấn đến nhiều retriever song song và sau đó thông minh hợp nhất các kết quả lại. Các phương pháp phổ biến bao gồm bỏ phiếu (voting), hợp nhất có trọng số (weighted fusion), hoặc Reciprocal Rank Fusion (RRF).
  - \* **Chiến lược Truy vấn Thích ứng (Adaptive Query Strategy):** Đây là một phương pháp động, trong đó hệ thống tự quyết định chiến lược biến đổi nào là tốt nhất cho một truy vấn cụ thể tại một thời điểm nhất định. Ví dụ, hệ thống có thể thử một truy vấn mở rộng đơn giản trước, nếu kết quả không tốt, nó sẽ tự động chuyển sang một chiến lược phức tạp hơn như HyDE hoặc phân rã câu hỏi (multi-hop).
  - **RAG Đa bước (Multi-Hop RAG):** Được thiết kế để giải quyết các câu hỏi phức tạp đòi hỏi phải suy luận theo chuỗi (reasoning chain) và tổng hợp bằng chứng từ nhiều tài liệu hoặc nguồn thông tin khác nhau. Thay vì một lần truy xuất duy nhất, hệ thống thực hiện nhiều "bước nhảy"(hops) để thu thập đủ thông tin cần thiết.
- Ví dụ, với câu hỏi: *"Thủ tướng Anh hiện tại có quan điểm gì về chính sách của tổng thống Pháp?"*, một hệ thống multi-hop sẽ không tìm kiếm trực tiếp mà sẽ phân rã và thực hiện theo các bước. Các phương pháp để thực hiện điều này rất đa dạng:
- **Dựa trên Phân rã (Decomposition-based):** Đây là hướng tiếp cận trực tiếp nhất, trong đó câu hỏi phức tạp được chia thành các câu hỏi con đơn giản hơn.
    - \* **Phân rã Truy vấn (Query Decomposition):** Sử dụng LLM để tự động phân rã câu hỏi gốc. Ví dụ trên sẽ được chia thành: (1) "Thủ tướng Anh hiện tại là ai?", (2) "Tổng thống Pháp hiện tại là ai?", và (3) "Quan điểm của [kết quả 1] về chính sách của [kết quả 2] là gì?".
    - \* **Truy xuất Lặp (Iterative Retrieval):** Hệ thống thực hiện truy xuất tuần tự. Kết quả của bước truy xuất trước được sử dụng để tạo ra câu hỏi cho bước tiếp theo. Đây chính là cơ chế hoạt động cốt lõi của ví dụ về thủ tướng Anh - tổng thống Pháp.
  - **Dựa trên Đồ thị (Graph-based):** Phương pháp này biến đổi toàn bộ kho tài liệu thành một Đồ thị Tri thức (Knowledge Graph), trong đó các thực thể là các nút (nodes) và mối quan hệ là các cạnh (edges).
    - \* **Tìm kiếm Đường đi (Path Finding):** Câu hỏi của người dùng được chuyển thành một bài toán tìm đường đi trên đồ thị. Ví dụ: "Ai đã đạo diễn bộ phim mà Tom Hanks đóng vai chính?" sẽ trở thành việc tìm một đường đi từ nút **Tom Hanks** đến một nút **Đạo diễn** thông qua một nút **Bộ phim**. (Phương pháp này rất phổ biến trong các bộ dữ liệu QA phức tạp như HotpotQA).
    - **Tăng cường bằng Suy luận (Reasoning-augmented):** Các phương pháp này tích hợp chặt chẽ quá trình suy luận của LLM vào vòng lặp truy xuất, làm cho quá trình tìm kiếm trở nên linh hoạt và thông minh hơn.
      - \* **Chain-of-Thought (CoT) RAG:** LLM thực hiện suy luận từng bước một ("think out loud"). Tại mỗi bước, nếu thấy thiếu thông tin, nó sẽ chủ động tạo ra một truy vấn con để gửi đến retriever, sau đó sử dụng kết quả để tiếp tục chuỗi suy luận.
      - \* **Tree-of-Thoughts / Graph-of-Thoughts RAG:** Thay vì một chuỗi suy luận tuyến tính duy nhất, hệ thống tạo ra và khám phá nhiều nhánh suy luận song song. Nếu một nhánh đi vào ngõ cụt, hệ thống có thể quay lại và thử một hướng khác, giúp tăng cường sự mạnh mẽ và khả năng tìm ra lời giải cho các câu hỏi cực kỳ phức tạp.
    - **Tăng cường bằng Bộ nhớ (Memory-augmented):** Được thiết kế để giải quyết các chuỗi suy luận dài, nơi việc duy trì ngữ cảnh là tối quan trọng.
      - \* **Bộ nhớ Làm việc (Working Memory):** Mỗi kết quả trung gian, bằng chứng, hoặc suy luận được sinh ra trong quá trình multi-hop sẽ được lưu vào một "bộ nhớ đệm" hoặc "bảng nháp"(scratchpad). Tại các bước sau, LLM có thể tham chiếu lại thông tin trong bộ nhớ này, giúp tránh truy xuất lại thông tin đã có, giữ ngữ cảnh xuyên suốt,

và cho phép thực hiện các chuỗi suy luận dài và phức tạp hơn.

- **RAG dạng Tác tử (Agentic RAG):** Đây là một bước tiến hóa của RAG, nơi LLM không còn là một thành phần thụ động trong một pipeline cố định mà trở thành một "tác tử"(agent) tự chủ, có khả năng lập kế hoạch, ra quyết định và thực thi một chiến lược truy xuất linh hoạt. Thay vì chỉ trả lời dựa trên những gì được cung cấp, agent sẽ chủ động điều khiển toàn bộ quá trình thu thập thông tin.
  - **RAG Tăng cường bằng Công cụ (Tool-augmented RAG):** LLM được trao quyền truy cập vào một bộ công cụ đa dạng, không chỉ giới hạn ở một retriever duy nhất. Dựa vào bản chất của câu hỏi, agent có thể tự quyết định:
    - \* Khi nào cần gọi đến retriever nội bộ (để tìm trong kho tri thức).
    - \* Khi nào cần sử dụng công cụ tìm kiếm bên ngoài (như Google Search) để lấy thông tin mới nhất.
    - \* Khi nào cần truy vấn một cơ sở dữ liệu có cấu trúc (qua SQL).
    - \* Khi nào cần sử dụng một công cụ tính toán (calculator) cho các câu hỏi số học.
  - **Truy xuất Thích ứng (Adaptive Retrieval):** Agent có khả năng điều chỉnh chiến lược truy xuất một cách linh hoạt ngay trong quá trình xử lý một câu hỏi, thay vì tuân theo một quy trình cứng nhắc.
    - \* **Độ sâu Truy xuất Độ rộng (Dynamic Retrieval Depth):** Sau khi truy xuất một vài tài liệu ban đầu, agent tự đánh giá xem thông tin đã đủ để trả lời hay chưa. Nếu chưa, nó sẽ quyết định truy xuất thêm; nếu đủ, nó sẽ dừng lại để tiết kiệm tài nguyên.
    - \* **Lựa chọn/Định tuyến Retriever (Retriever Selection/Routing):** Agent phân tích câu hỏi và chọn công cụ retriever phù hợp nhất từ nhiều lựa chọn có sẵn: dùng BM25 cho các truy vấn dựa trên từ khóa, dùng dense retriever cho các câu hỏi ngữ nghĩa, hoặc dùng retriever trên đồ thị tri thức cho các câu hỏi về mối quan hệ.
  - **Kiến trúc Lập kế hoạch - Thực thi (Planner-Executor Framework):** Tách biệt quá trình tư duy và hành động.
    - \* **Planner (Người lập kế hoạch):** LLM phân tích câu hỏi phức tạp và tạo ra một kế hoạch hành động từng bước. Ví dụ: "Đầu tiên, tìm thông tin về A. Sau đó, dùng kết quả đó để tìm thông tin về B. Cuối cùng, tổng hợp cả hai để trả lời."
    - \* **Executor (Người thực thi):** Một thành phần khác (hoặc chính LLM trong một vai trò khác) sẽ thực hiện từng bước trong kế hoạch (gọi tool, truy vấn retriever) và trả kết quả về cho Planner. Planner sẽ cập nhật kế hoạch dựa trên kết quả mới nhận được.
  - **RAG Tác tử Tương tác (Interactive Agentic RAG):** Agent có khả năng tương tác với người dùng để cải thiện quá trình tìm kiếm, tạo ra một vòng lặp phản hồi (human-in-the-loop).
    - \* **Làm rõ Truy vấn (Query Clarification):** Nếu câu hỏi của người dùng mơ hồ hoặc không đủ chi tiết, thay vì đoán mò, agent sẽ chủ động đặt câu hỏi ngược lại để làm rõ. Ví dụ: "Khi bạn nói về 'Apple', bạn đang muốn hỏi về công ty công nghệ hay về một loại trái cây?"
  - **RAG Tác tử Tăng cường bằng Học máy (Learning-enhanced Agentic RAG):** Chiến lược ra quyết định của agent không phải là cố định mà có thể được cải thiện thông qua học máy, đặc biệt là Học tăng cường (RL).
    - \* **Học Chính sách Truy xuất (Learning Retrieval Policies):** Agent học được một "chính sách"(policy) tối ưu qua thời gian. Ví dụ, nó có thể học được rằng: "Với các câu hỏi so sánh, chiến lược multi-hop thường hiệu quả hơn", hoặc "Khi người dùng hỏi về tin tức, hãy ưu tiên sử dụng công cụ tìm kiếm web".

#### 6.1.9. Đánh giá và Triển khai Hệ thống RAG

Làm thế nào để biết hệ thống RAG của bạn hoạt động tốt?

### Các chỉ số Dánh giá

Việc đánh giá được chia làm hai cấp độ:

- **Đánh giá Retriever (Component-level):** Đánh giá riêng thành phần tìm kiếm. Các chỉ số phổ biến là Hit Rate (tỷ lệ câu hỏi có chunk chứa câu trả lời nằm trong top K) và Mean Reciprocal Rank (MRR).
- **Đánh giá End-to-End (System-level):** Đánh giá chất lượng câu trả lời cuối cùng. Các framework như RAGAS cung cấp các chỉ số quan trọng:
  - **Faithfulness (Tính trung thực):** Câu trả lời có bám sát và được hỗ trợ bởi ngữ cảnh đã cung cấp không?
  - **Answer Relevancy (Độ liên quan):** Câu trả lời có đi thẳng vào vấn đề của câu hỏi không?
  - **Context Precision & Recall:** Các chunk được tìm thấy có thực sự cần thiết và đầy đủ để trả lời câu hỏi không?

### Checklist khi đưa vào Vận hành (Production Checklist)

Xây dựng một hệ thống RAG cho sản phẩm không chỉ dừng lại ở mô hình. Dưới đây là một vài yếu tố cần cân nhắc:

- **Tốc độ (Latency):** Toàn bộ quá trình từ lúc nhận câu hỏi đến lúc trả về câu trả lời phải đủ nhanh để người dùng không phải chờ đợi.
- **Chi phí (Cost):** Theo dõi chi phí cho mỗi lượt gọi API embedding và LLM.
- **Giám sát (Monitoring):** Xây dựng dashboard để theo dõi các chỉ số đánh giá, latency, chi phí, và các câu hỏi mà hệ thống trả lời sai.
- **Vòng lặp Phản hồi (Feedback Loop):** Thu thập phản hồi từ người dùng (ví dụ: nút "thích"/"không thích") để tìm ra các trường hợp lỗi và liên tục cải thiện hệ thống.

## 6.2. Lý thuyết về Tối ưu hóa Mô hình

Các Mô hình Ngôn ngữ Lớn (LLMs), đúng như tên gọi của chúng, thường cực kỳ lớn, đòi hỏi phần cứng đắt đỏ và có độ trễ suy luận (inference latency) cao. Điều này tạo ra một rào cản lớn cho việc triển khai chúng trong các ứng dụng thực tế, đặc biệt là các ứng dụng cần phản hồi thời gian thực hoặc cần chạy trên các thiết bị có tài nguyên hạn chế (như điện thoại di động).

Để giải quyết vấn đề này, một loạt các kỹ thuật tối ưu hóa mô hình đã được phát triển. Mục tiêu chung của chúng là: **giảm kích thước mô hình, tăng tốc độ suy luận, và giảm yêu cầu về bộ nhớ, trong khi cố gắng duy trì hiệu năng (độ chính xác) ở mức cao nhất có thể.**

Chúng ta sẽ khám phá ba họ kỹ thuật tối ưu hóa chính: **Chưng cất, Lượng tử hóa, và Cắt tỉa.**

### 6.2.1. Chưng cất Tri thức (Knowledge Distillation)

#### Trực giác cốt lõi: Thầy dạy Trò

Hãy tưởng tượng bạn có một mạng nơ-ron rất lớn, phức tạp nhưng chính xác (gọi là **mô hình thầy - teacher model**), và một mạng nơ-ron nhỏ hơn, nhanh hơn nhiều (gọi là **mô hình trò - student model**). Chưng cất Tri thức là quá trình "dạy" cho mô hình trò bắt chước hành vi của mô hình thầy.

#### Triết lý của Chưng cất Tri thức

Thay vì chỉ dạy cho mô hình trò học từ các nhãn cứng (hard labels) của dữ liệu (ví dụ: "chó" hoặc "mèo"), chúng ta hãy dạy nó bắt chước cả phân phối xác suất đầu ra "mềm" (soft probabilities) của mô hình thầy. Các xác suất mềm này chứa đựng những thông tin "đen tối" (dark knowledge) vô cùng phong phú về cách mô hình thầy "suy nghĩ" và "khái quát hóa".

Ví dụ, khi nhìn vào ảnh một chiếc xe tải, mô hình thầy có thể dự đoán: "xe tải": 0.9, "xe buýt": 0.08, "ô tô": 0.015, "máy bay": 0.005. Thông tin rằng nó "hơi giống xe buýt" nhưng "không hề giống máy bay" là một tín hiệu huấn luyện quý giá hơn nhiều so với nhãn cứng "xe tải".

### Cơ chế hoạt động

Quá trình chưng cất bao gồm việc huấn luyện mô hình trò trên một hàm mất mát kết hợp:

- Tạo các nhãn mềm:** Cho cùng một đầu vào, chúng ta lấy đầu ra logits (đầu ra trước lớp softmax) của mô hình thầy và đưa nó qua một hàm softmax đã được "làm mềm" bằng một tham số gọi là **nhiệt độ (temperature - T)**.

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

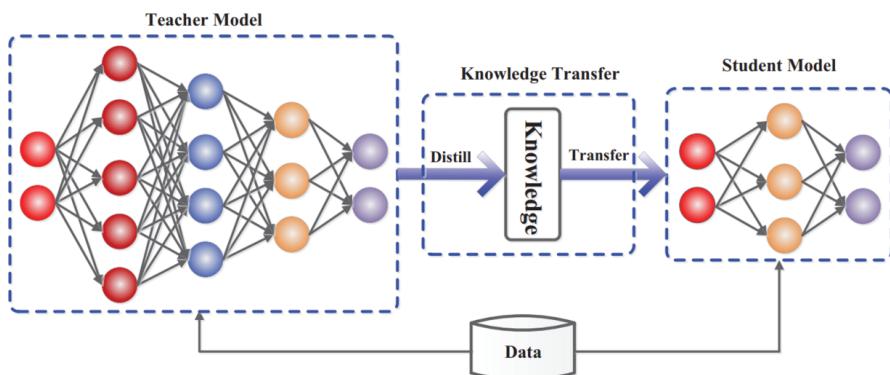
Khi  $T > 1$ , phân phối xác suất sẽ trở nên "mềm" hơn, gán xác suất cao hơn cho các lớp không phải là lớp tốt nhất, từ đó để lộ ra "dark knowledge".

- Huấn luyện mô hình trò:** Mô hình trò được huấn luyện để tối thiểu hóa một hàm mất mát tổng hợp:

$$\mathcal{L}_{\text{distill}} = \alpha \cdot \mathcal{L}_{\text{CE}}(y_{\text{true}}, y_{\text{student}}) + (1 - \alpha) \cdot \mathcal{L}_{\text{KL}}(p_{\text{teacher}}^T, p_{\text{student}}^T) \quad (6.1)$$

Trong đó:

- Thành phần 1 (Hard Loss):** Là hàm mất mát Cross-Entropy thông thường giữa dự đoán của mô hình trò và nhãn thật ( $y_{\text{true}}$ ). Phần này đảm bảo mô hình trò vẫn học tốt trên tác vụ.
- Thành phần 2 (Soft Loss):** Là hàm mất mát KL-Divergence (hoặc Cross-Entropy) giữa phân phối xác suất mềm của mô hình trò và phân phối xác suất mềm của mô hình thầy (cả hai đều được tính với nhiệt độ  $T$ ). Phần này buộc mô hình trò phải bắt chước cách "suy nghĩ" của mô hình thầy.
- $\alpha$  là một siêu tham số để cân bằng giữa hai thành phần.



Hình 6.2: Quy trình Chưng cất Tri thức. Mô hình trò học đồng thời từ nhãn cứng của dữ liệu và nhãn mềm (được tạo ra bởi mô hình thầy với nhiệt độ  $T$ ).

Các mô hình như DistilBERT đã sử dụng kỹ thuật này để tạo ra một phiên bản BERT nhỏ hơn

40

### 6.2.2. Lượng tử hóa (Quantization)

#### Trục giác cốt lõi: Giảm độ chính xác của Số

Lượng tử hóa là quá trình giảm số lượng bit được sử dụng để biểu diễn một con số. Trong học sâu, các trọng số và giá trị kích hoạt của mô hình thường được lưu trữ dưới dạng số thực dấu phẩy động 32-bit (FP32) hoặc 16-bit (FP16). Lượng tử hóa sẽ ánh xạ các giá trị này sang các định dạng có độ chính xác thấp hơn, ví dụ như số nguyên 8-bit (INT8) hoặc thậm chí 4-bit.

### Triết lý của Lượng tử hóa

"Chúng ta có thực sự cần đến 32 bit độ chính xác để lưu trữ mỗi trọng số không? Liệu việc làm tròn các trọng số thành một tập hợp các giá trị rời rạc, có độ chính xác thấp hơn có làm ảnh hưởng nhiều đến hiệu năng cuối cùng không?"

Câu trả lời, một cách đáng ngạc nhiên, thường là "không nhiều". Các mạng nơ-ron sâu có một sự dư thừa (redundancy) nhất định và có khả năng chống chịu khá tốt với nhiều và sự mất mát độ chính xác.

### Các phương pháp Lượng tử hóa

**Post-Training Quantization (PTQ) - Lượng tử hóa sau Huấn luyện** Đây là phương pháp đơn giản nhất.

1. Lấy một mô hình đã được huấn luyện đầy đủ.
2. Chuyển đổi các trọng số của nó từ FP32/FP16 sang INT8. Quá trình này bao gồm việc tìm ra một "tỷ lệ"(scale) và một "điểm không"(zero-point) để ánh xạ khoảng giá trị thực sang khoảng giá trị nguyên 8-bit.
3. Thường yêu cầu một bộ dữ liệu hiệu chỉnh nhỏ (calibration dataset) để tìm ra các tham số ánh xạ tối ưu.

PTQ rất nhanh để thực hiện nhưng có thể gây ra sụt giảm độ chính xác đáng kể.

**Quantization-Aware Training (QAT) - Huấn luyện Nhận biết Lượng tử hóa** Đây là phương pháp phức tạp nhưng cho kết quả tốt hơn.

1. Trong quá trình fine-tuning (hoặc huấn luyện từ đầu), mô hình sẽ mô phỏng (simulate) ảnh hưởng của việc lượng tử hóa.
2. Tức là, trong quá trình truyền thẳng, các trọng số sẽ được "làm tròn"(quantized) về định dạng có độ chính xác thấp, nhưng trong quá trình truyền ngược, gradient vẫn được tính toán và cập nhật trên các trọng số có độ chính xác đầy đủ.
3. Điều này cho phép mô hình học cách "thích ứng"với sự mất mát độ chính xác sẽ xảy ra sau khi lượng tử hóa, giúp giảm thiểu sụt giảm hiệu năng.

Lợi ích của Lượng tử hóa là rất rõ ràng: giảm kích thước mô hình (INT8 nhỏ hơn FP32 4 lần) và tăng tốc độ suy luận (các phép toán trên số nguyên nhanh hơn nhiều so với số thực trên hầu hết các phần cứng). Chúng ta đã thấy một ứng dụng của nó trong QLoRA (mục 5.2.2).

### 6.2.3. Cắt tia (Pruning)

#### Trực giác cốt lõi: Loại bỏ các Kết nối Thừa

Các mạng nơ-ron lớn thường được "tham số hóa quá mức"(over-parameterized), nghĩa là chúng có nhiều trọng số và nơ-ron hơn mức cần thiết. Cắt tia là quá trình xác định và loại bỏ các trọng số hoặc các nơ-ron "ít quan trọng"nhất trong mạng.

### Triết lý của Cắt tia

"Một mạng nơ-ron giống như một cái cây. Sau khi nó phát triển, chúng ta có thể cắt tia đi những cành lá không cần thiết mà không làm ảnh hưởng đến sức sống và khả năng ra quả của cây, thậm chí còn giúp cây khỏe mạnh hơn."

### Các phương pháp Cắt tia

**Cắt tia theo Độ lớn (Magnitude Pruning)** Đây là phương pháp phổ biến nhất.

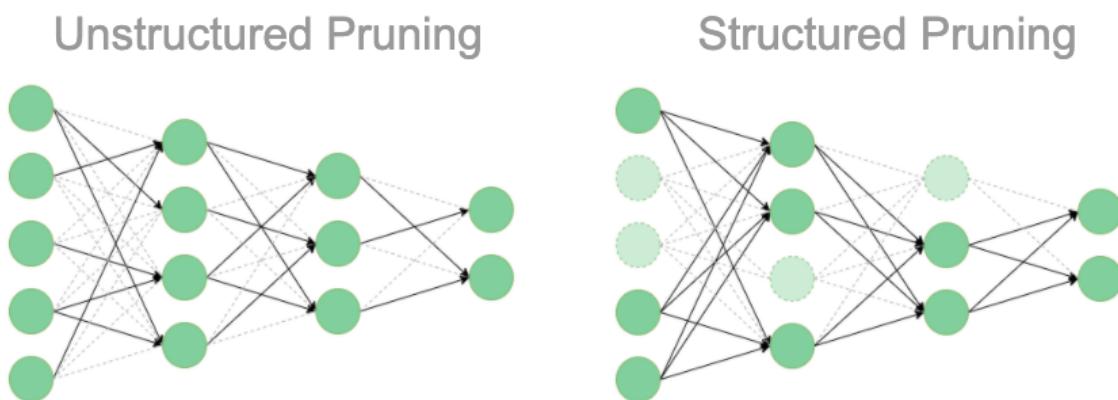
- **Giả định:** Các trọng số có giá trị tuyệt đối gần bằng 0 là ít quan trọng nhất.

- **Quy trình:**

1. Huấn luyện một mô hình dày đặc đến khi hội tụ.
2. Xóa (đặt bằng 0) tất cả các trọng số có giá trị tuyệt đối nhỏ hơn một ngưỡng nhất định. Điều này tạo ra một ma trận trọng số "thưa thớt"(sparse).
3. (Tùy chọn nhưng quan trọng) Fine-tune lại mô hình thừa thớt này trong một vài epoch để cho phép các trọng số còn lại "bù đắp" cho các trọng số đã bị xóa.

#### Cắt tia có Cấu trúc vs. Không có Cấu trúc (Structured vs. Unstructured Pruning)

- **Không có cấu trúc:** Xóa các trọng số riêng lẻ ở bất kỳ đâu trong ma trận. Điều này tạo ra các ma trận thừa thớt, có thể giảm đáng kể số lượng tham số nhưng không phải lúc nào cũng tăng tốc độ trên phần cứng thông thường (như GPU) vốn được tối ưu cho các phép toán ma trận dày đặc.
- **Có cấu trúc:** Xóa toàn bộ các cấu trúc lớn hơn, ví dụ như xóa toàn bộ một nơ-ron (một cột trong ma trận trọng số) hoặc toàn bộ một bộ lọc trong CNN. Phương pháp này có thể không giảm được nhiều tham số bằng, nhưng kết quả là một mô hình nhỏ hơn, dày đặc hơn, có thể chạy nhanh hơn trên phần cứng hiện có mà không cần các thư viện đặc biệt.



Hình 6.3: So sánh Cắt tia Không có Cấu trúc (xóa các trọng số riêng lẻ, tạo ma trận thừa thớt) và Cắt tia Có Cấu trúc (xóa toàn bộ cột/nơ-ron, tạo ma trận nhỏ hơn).

Các kỹ thuật tối ưu hóa này thường được sử dụng kết hợp với nhau (ví dụ, một mô hình được chưng cất, sau đó được cắt tia và cuối cùng được lượng tử hóa) để đạt được sự cân bằng tốt nhất giữa kích thước, tốc độ và độ chính xác.

### 6.3. Mô hình Đa phương thức (Multimodal Models)

Con người trải nghiệm thế giới thông qua nhiều giác quan cùng một lúc: chúng ta nhìn thấy một hình ảnh, nghe một âm thanh, và đọc một đoạn văn bản mô tả về nó. Các mô hình mà chúng ta đã thảo luận cho đến nay chủ yếu là đơn phương thức (unimodal) - chúng chỉ xử lý văn bản. Để AI thực sự tiến gần hơn đến trí thông minh của con người, chúng cần có khả năng hiểu và lý luận trên nhiều loại dữ liệu khác nhau một cách đồng thời. Đây chính là mục tiêu của các **Mô hình Đa phương thức (Multimodal Models)**.

Lĩnh vực này đã có những bước nhảy vọt phi thường, đặc biệt là trong việc kết hợp hai phương thức quan trọng nhất: **Ngôn ngữ (Language)** và **Thị giác (Vision)**. Phần này sẽ đi sâu vào các kiến trúc nền tảng đã tạo nên cuộc cách mạng này.

### 6.3.1. Nền tảng: Xây dựng cầu nối giữa Ảnh và Chữ

Để một mô hình có thể hiểu cả ảnh và chữ, nó cần một "ngôn ngữ chung" mà không gian biểu diễn nối cả ảnh và chữ đều có thể được ánh xạ vào và so sánh với nhau. Hai đột phá nền tảng đã tạo ra cây cầu này là Vision Transformer (ViT) và Contrastive Learning (CLIP).

#### Vision Transformer (ViT): "Đọc" ảnh như đọc một câu

**Vấn đề với CNN** Các mạng CNN (đã học ở mục 3.5) xử lý ảnh bằng cách trượt các bộ lọc nhỏ, xây dựng các đặc trưng từ cục bộ (cạnh, góc) đến toàn cục (mắt, mũi, khuôn mặt). Cách tiếp cận này rất hiệu quả nhưng lại có một sự khác biệt về kiến trúc so với Transformer đang thống trị NLP.

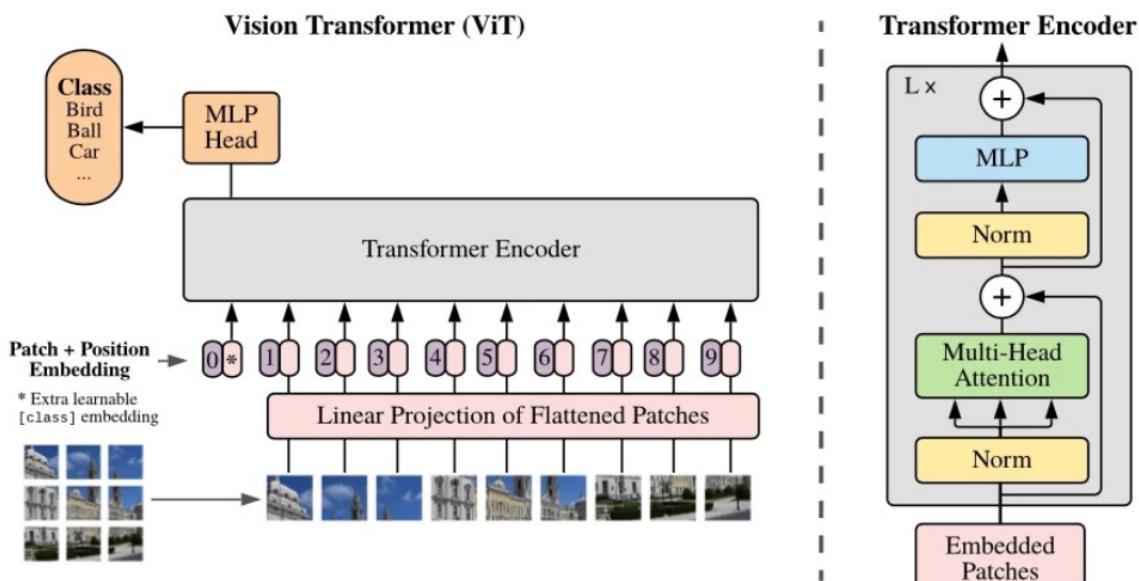
**Giải pháp của ViT:** Biến ảnh thành một chuỗi các "từ" Vision Transformer (Dosovitskiy et al., 2020) đã đưa ra một ý tưởng cấp tiến:

#### Trực giác của Vision Transformer

"Điều gì sẽ xảy ra nếu chúng ta coi một bức ảnh không phải là một lưới pixel, mà là một chuỗi các mảnh vá (sequence of patches)? Nếu vậy, chúng ta có thể đưa chuỗi các mảnh vá này vào một kiến trúc Transformer tiêu chuẩn, giống hệt như cách chúng ta xử lý một chuỗi các từ."

#### Cơ chế hoạt động

1. **Phân mảnh Ảnh (Image Patching):** Một bức ảnh đầu vào (ví dụ:  $224 \times 224$  pixel) được chia thành một lưới các mảnh vá không chồng chéo có kích thước cố định (ví dụ:  $16 \times 16$  pixel). Điều này tạo ra một chuỗi gồm  $(224/16)^2 = 14 \times 14 = 196$  mảnh vá.
2. **Nhúng Mảnh vá (Patch Embedding):** Mỗi mảnh vá được "làm phẳng" (flatten) thành một vector dài và sau đó được chiếu qua một lớp tuyến tính để tạo ra một "patch embedding" có số chiều phù hợp với Transformer (ví dụ: 768 chiều).
3. **Thêm Token Phân loại và Mã hóa Vị trí:** Tương tự như BERT, một token '[CLS]' có thể học được sẽ được chèn vào đầu chuỗi. Sau đó, các vector mã hóa vị trí (positional encodings) được cộng vào mỗi patch embedding để cung cấp thông tin về vị trí tương đối của các mảnh vá.
4. **Đưa vào Transformer Encoder:** Chuỗi các patch embedding này giờ đây được xử lý bởi một chuỗi các khối Transformer Encoder tiêu chuẩn. Cơ chế Self-Attention sẽ cho phép mỗi mảnh vá "chú ý" đến tất cả các mảnh vá khác để học các mối quan hệ không gian.



Hình 6.4: Kiến trúc của Vision Transformer (ViT). Ảnh được chia thành các mảnh vá, mỗi mảnh được nhúng và xử lý như một "token" trong một chuỗi bởi một Transformer Encoder tiêu chuẩn.

ViT đã chứng minh rằng một kiến trúc dựa hoàn toàn trên Transformer có thể đạt được hiệu năng state-of-the-art trong các tác vụ thị giác, tạo ra một sự thống nhất về mặt kiến trúc giữa NLP và CV. Nó cung cấp một cách mạnh mẽ để có được các bộ mã hóa hình ảnh (Image Encoders) tạo ra các biểu diễn giàu ngữ cảnh cho ảnh.

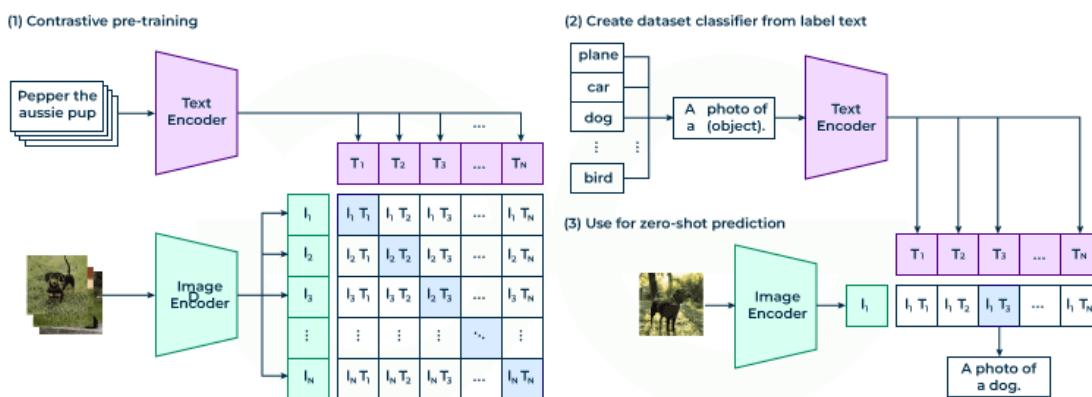
### CLIP: Học sự tương đồng giữa Ảnh và Chữ bằng Contrastive Learning

**Vấn đề** Chúng ta đã có Image Encoder (ViT) và Text Encoder (Transformer). Làm thế nào để chúng "nói chuyện" với nhau? Làm thế nào để mô hình biết rằng bức ảnh một chú chó và câu "a photo of a dog" là cùng một khái niệm?

**Giải pháp của CLIP (Contrastive Language-Image Pre-training)** CLIP (Radford et al., 2021) từ OpenAI đã giải quyết vấn đề này bằng một phương pháp huấn luyện quy mô lớn dựa trên **Học Tương phản (Contrastive Learning)**, một ý tưởng tương tự như Mạng Siamese (mục ??).

#### Cơ chế hoạt động

1. **Kiến trúc:** CLIP bao gồm hai bộ mã hóa riêng biệt:
  - Một **Image Encoder** (ví dụ: ViT).
  - Một **Text Encoder** (ví dụ: một Transformer).
2. **Dữ liệu Huấn luyện:** Một bộ dữ liệu khổng lồ gồm 400 triệu cặp (ảnh, văn bản mô tả) được thu thập từ Internet.
3. **Mục tiêu Huấn luyện:** Huấn luyện cả hai bộ mã hóa từ đầu để chúng ánh xạ các cặp (ảnh, văn bản) tương ứng vào **cùng một vị trí** trong một không gian embedding đa phương thức chung.
4. **Quy trình trong một batch:**
  - (a) Lấy một batch gồm  $N$  cặp (ảnh, văn bản).
  - (b) Đưa  $N$  ảnh qua Image Encoder để có được  $N$  vector ảnh ( $I_1, \dots, I_N$ ).
  - (c) Đưa  $N$  văn bản qua Text Encoder để có được  $N$  vector văn bản ( $T_1, \dots, T_N$ ).
  - (d) Tính toán độ tương đồng cosine giữa tất cả các cặp vector ảnh và văn bản có thể có, tạo ra một ma trận tương đồng  $N \times N$ .
  - (e) **Hàm mất mát tương phản:** Mục tiêu là làm cho độ tương đồng của các cặp **khớp đúng** (ví dụ:  $I_i$  và  $T_i$ ) trên đường chéo chính của ma trận là cao nhất, và độ tương đồng của tất cả các cặp **không khớp** ( $I_i$  và  $T_j$  với  $i \neq j$ ) là thấp nhất.



Hình 6.5: Quy trình huấn luyện của CLIP. Mô hình học cách tối đa hóa độ tương đồng của các cặp (ảnh, văn bản) chính xác trên đường chéo chính và tối thiểu hóa độ tương đồng của các cặp không chính xác.

Sau khi huấn luyện, CLIP có khả năng zero-shot đáng kinh ngạc trong việc phân loại ảnh. Ví dụ, để phân loại một ảnh, bạn chỉ cần nhúng ảnh đó và so sánh độ tương đồng của nó với các vector văn bản được tạo từ các prompt như "a photo of a dog", "a photo of a cat",... và chọn prompt có độ tương đồng cao nhất.

Quan trọng hơn, CLIP đã tạo ra một **không gian embedding đa phương thức được căn chỉnh** (aligned multimodal embedding space) mạnh mẽ, là nền tảng cho các mô hình thế hệ tiếp theo.

### 6.3.2. Ví dụ Kiến trúc: Kết hợp LLM với Thị giác

Với các khối xây dựng là Image Encoder và không gian đa phương thức được căn chỉnh, các nhà nghiên cứu đã tạo ra các Mô hình Ngôn ngữ Lớn Đa phương thức (Multimodal Large Language Models - MLLMs).

#### Flamingo: Thêm Cross-Attention vào một LLM đã đóng băng

- Kiến trúc** (Alayrac et al., 2022): Flamingo giữ một LLM văn bản (ví dụ: Chinchilla) và một Vision Encoder (ví dụ: ViT) đã được huấn luyện trước và **đóng băng** (frozen) chúng.
- Cơ chế kết nối:** Nó chèn các lớp Cross-Attention đặc biệt vào giữa các lớp của LLM.
- Dòng chảy dữ liệu:** Khi xử lý văn bản, các token văn bản sẽ đi qua các lớp Transformer của LLM như bình thường. Tại các lớp Cross-Attention, các token văn bản này sẽ đóng vai trò là **Query**, trong khi các đặc trưng hình ảnh từ Vision Encoder sẽ đóng vai trò là **Key** và **Value**. Điều này cho phép các token văn bản "nhìn" vào hình ảnh và rút ra các thông tin thị giác liên quan khi cần thiết.
- Lợi ích:** Rất hiệu quả về mặt tham số vì chỉ cần huấn luyện các lớp Cross-Attention mới.

#### LLaVA: Ánh xạ Trực tiếp vào Không gian Từ của LLM

LLaVA (Large Language and Vision Assistant - Liu et al., 2023) đưa ra một cách tiếp cận đơn giản hơn nhưng lại cực kỳ hiệu quả.

- Kiến trúc:** Sử dụng một Vision Encoder (cụ thể là ViT từ CLIP) và một LLM (cụ thể là Vicuna, một phiên bản của Llama).
- Cơ chế kết nối:**
  - Ảnh được đưa qua Vision Encoder để tạo ra một chuỗi các patch embedding.
  - Một lớp chiếu tuyến tính (projection layer) nhỏ duy nhất được huấn luyện để **ánh xạ** (map) các patch embedding này vào cùng một không gian với các word embedding của LLM.
  - Các patch embedding đã được chiếu này sau đó được coi như các "từ" đặc biệt và được **chèn trực tiếp** vào chuỗi văn bản đầu vào của LLM.
- Quá trình huấn luyện 2 giai đoạn:**
  - Giai đoạn 1 (Căn chỉnh đặc trưng):** Đóng băng cả Vision Encoder và LLM, chỉ huấn luyện lớp chiếu tuyến tính trên một bộ dữ liệu lớn gồm các cặp (ảnh, mô tả). Giai đoạn này dạy cho lớp chiếu cách "dịch" đặc trưng ảnh thành "ngôn ngữ" mà LLM có thể hiểu.
  - Giai đoạn 2 (Tinh chỉnh End-to-End):** Mở đóng băng các trọng số của LLM và fine-tune toàn bộ mô hình (hoặc chỉ lớp chiếu và LLM) trên một bộ dữ liệu nhỏ hơn, chất lượng cao hơn gồm các chỉ dẫn đa phương thức phức tạp (ví dụ: các cuộc trò chuyện về hình ảnh).
- Lợi ích:** Kiến trúc cực kỳ đơn giản, dễ triển khai và tận dụng được sức mạnh của các LLM và Vision Encoder đã có sẵn. LLaVA đã trở thành một trong những kiến trúc nền tảng cho các MLLM mã nguồn mở.

Sự kết hợp giữa Ngôn ngữ và Thị giác đang mở ra một kỷ nguyên mới của AI, nơi các mô hình có thể tương tác với thế giới một cách phong phú và tự nhiên hơn rất nhiều.

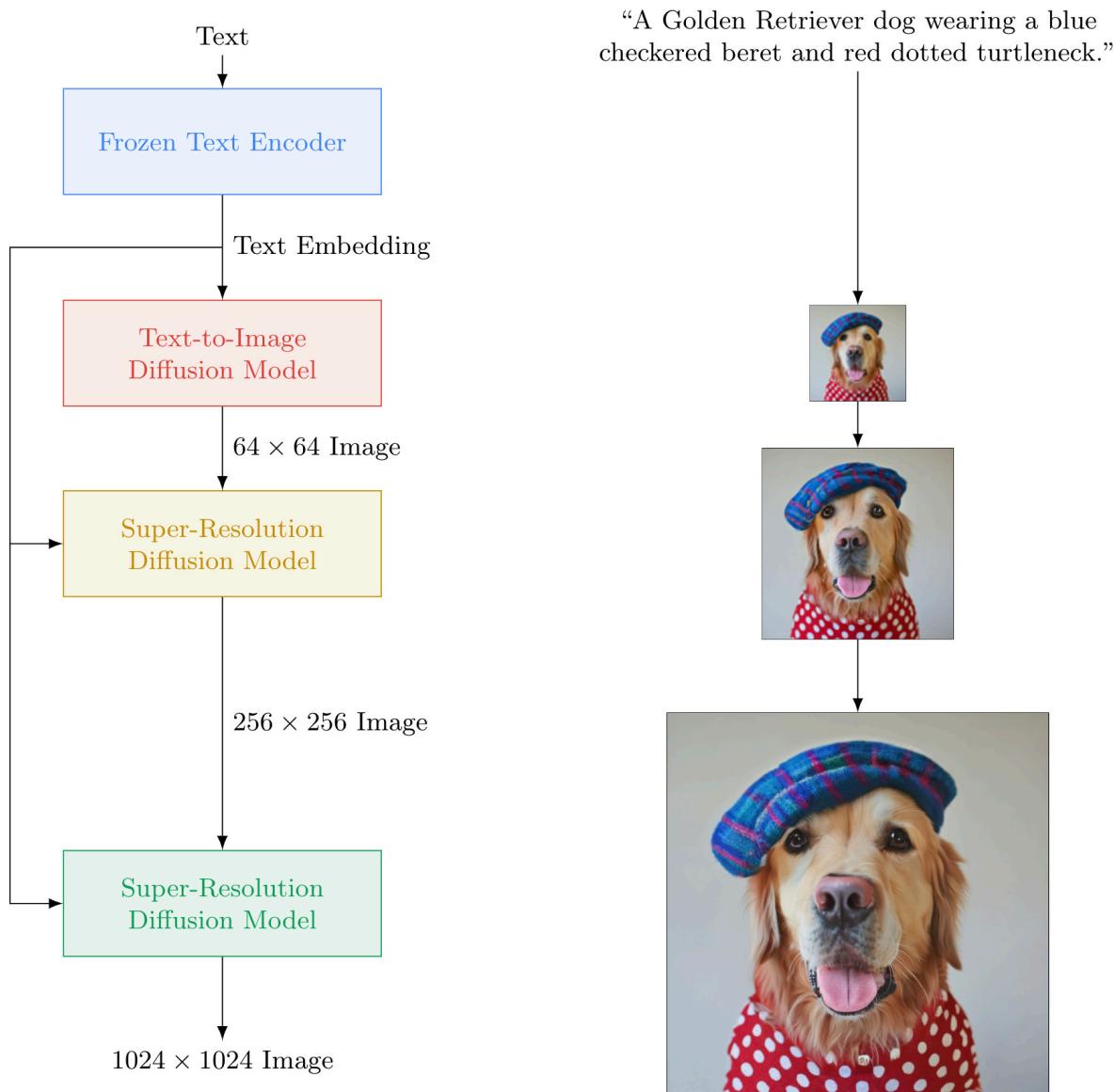
### 6.3.3. Sinh Đa phương thức (Multimodal Generation)

Nếu các mô hình ở mục trước tập trung vào việc *hiểu* đầu vào đa phương thức (ví dụ, mô tả một bức ảnh bằng văn bản), thì một trong những bước đột phá ngoạn mục nhất của AI hiện đại là khả năng *sinh* ra một phương thức từ một phương thức khác. Nổi bật nhất trong số đó là các tác vụ sinh từ văn bản, biến những mô tả trừu tượng thành các sản phẩm thị giác cụ thể.

#### Sinh Ảnh từ Văn bản (Text-to-Image)

Đây là lĩnh vực đã tạo ra một cuộc cách mạng trong ngành công nghiệp sáng tạo, cho phép bất kỳ ai cũng có thể tạo ra các tác phẩm nghệ thuật kỹ thuật số chỉ bằng ngôn ngữ.

- **Bài toán:** Cho một câu mô tả bằng văn bản (prompt), ví dụ "một phi hành gia đang cưỡi ngựa trên sao Hỏa theo phong cách siêu thực", mô hình phải tạo ra một bức ảnh hoàn toàn mới khớp với mô tả đó.
- **Kiến trúc cốt lõi: Mô hình Khuếch tán (Diffusion Models)** Các mô hình thành công nhất hiện nay như DALL-E 2, Midjourney, và Stable Diffusion thường dựa trên kiến trúc này.
  1. **Mã hóa Văn bản:** Prompt được đưa qua một bộ mã hóa văn bản mạnh mẽ (thường là một phần của mô hình CLIP) để tạo ra một vector embedding giàu ngữ nghĩa.
  2. **Quá trình Khuếch tán Ngược có Điều kiện:** Mô hình bắt đầu với một bức ảnh chỉ toàn nhiễu ngẫu nhiên (random noise). Sau đó, trong một chuỗi nhiều bước, một mạng nơ-ron (thường là U-Net) sẽ dần dần khử nhiễu (denoise) cho bức ảnh. Ở mỗi bước, quá trình khử nhiễu này được *dẫn dắt* (conditioned) bởi vector embedding của văn bản. Quá trình này dần dần làm hiện ra một hình ảnh mượt mà và khớp với prompt.



Hình 6.6: Mô hình Khuếch tán cho việc sinh ảnh từ văn bản. Bắt đầu từ nhiều, mô hình dần dần khử nhiễu theo sự dẫn dắt của embedding văn bản để tạo ra ảnh cuối cùng.

### Sinh Video từ Văn bản (Text-to-Video)

Là một bước tiến tự nhiên nhưng khó hơn rất nhiều so với sinh ảnh.

- Thách thức chính:** Ngoài việc tạo ra các khung hình (frames) chất lượng cao, mô hình còn phải đảm bảo tính **quán về mặt thời gian (temporal consistency)**. Các đối tượng và bối cảnh phải duy trì sự logic và liền mạch từ khung hình này sang khung hình tiếp theo.
- Hướng tiếp cận:** Các mô hình tiên tiến như Sora (OpenAI) hay Lumiere (Google) mở rộng ý tưởng của mô hình khuếch tán sang không-thời gian. Chúng không xử lý từng khung hình riêng lẻ mà xử lý một loạt các khung hình hoặc toàn bộ "patch" không-thời gian cùng lúc để đảm bảo sự mượt mà và nhất quán của chuyển động và bối cảnh.

Việc làm chủ khả năng sinh đa phương thức không chỉ mở ra những ứng dụng sáng tạo mới mà còn là một bước tiến quan trọng hướng tới các hệ thống AI có khả năng tương tác và sáng tạo trong một thế giới đa dạng về phương thức như thế giới của chúng ta.

### 6.3.4. Kiểm soát Quá trình Sinh (Controlled Generation)

Trong nhiều ứng dụng thực tế, yêu cầu không chỉ dừng lại ở việc sinh văn bản mạch lạc và tự nhiên, mà còn phải đảm bảo tuân theo các ràng buộc (constraints) hoặc thuộc tính (attributes) cụ thể. Ví dụ:

- Sinh bài đánh giá sản phẩm với *sentiment* tích cực.
- Viết tóm tắt nhưng giữ nguyên thuật ngữ chuyên ngành.
- Sinh hội thoại chatbot với giọng điệu thân thiện.

Đây là bài toán về việc “lái” quá trình sinh ngôn ngữ của LLM theo mong muốn của người dùng hoặc hệ thống.

#### Các kỹ thuật kinh điển

##### CTRL (Conditional Transformer Language Model)

- Ý tưởng: thêm các mã kiểm soát (control codes) đặc biệt vào đầu prompt để chỉ định thuộc tính mong muốn.
- Ví dụ: **Reviews Rating: 5.0** The product was ... để mô hình sinh tiếp một đánh giá tích cực.
- Ưu điểm: đơn giản, trực quan.
- Nhược điểm: đòi hỏi mô hình được huấn luyện lại từ đầu với dữ liệu gắn kèm mã kiểm soát; khó áp dụng cho nhiều thuộc tính phức tạp.

##### PPLM (Plug and Play Language Models)

- Ý tưởng: sử dụng một mô hình thuộc tính nhỏ (ví dụ: bộ phân loại cảm xúc) để hướng dẫn LLM đã đóng băng.
- Cách làm: tại mỗi bước sinh, gradient từ mô hình thuộc tính sẽ điều chỉnh trạng thái ẩn của LLM để tăng xác suất sinh ra từ phù hợp.
- Ưu điểm: không cần huấn luyện lại LLM lớn; dễ áp dụng cho nhiều thuộc tính.
- Nhược điểm: chi phí tính toán cao do phải tính gradient trong lúc sinh; có thể làm giảm độ mạch lạc.

##### FUDGE (Future Discriminators for Generation)

- Thay vì chỉnh trực tiếp trạng thái ẩn, FUDGE huấn luyện một bộ phân loại để ước lượng khả năng tiếp diễn thỏa mãn thuộc tính, sau đó kết hợp với xác suất sinh từ mô hình chính.
- Ưu điểm: linh hoạt, không cần huấn luyện lại LLM lớn.
- Nhược điểm: vẫn bị trade-off giữa kiểm soát và chất lượng.

#### Các kỹ thuật dựa trên Prompt

Các phương pháp này thuộc nhóm PEFT (Parameter-Efficient Fine-Tuning) và hiện được ứng dụng rộng rãi do hiệu quả cao.

#### Prompt Tuning

- Học các prompt mềm (soft prompts) – tức là các vector embedding có thể học được – để hướng dẫn mô hình.
- Ví dụ: để điều khiển văn bản theo giọng điệu “lịch sự”, ta học một soft prompt tương ứng và nối nó vào trước input.
- Ưu điểm: nhẹ, không cần tinh chỉnh toàn bộ mô hình.
- Nhược điểm: hiệu quả hạn chế nếu số lượng thuộc tính cần kiểm soát quá nhiều.

### Prefix Tuning

- Ý tưởng: học các vector embedding “tiền tố” và chèn vào mỗi lớp Transformer.
- Nhờ vậy, prompt không chỉ ảnh hưởng ở input mà còn tác động trực tiếp lên attention của mọi tầng.
- Hiệu quả hơn Prompt Tuning trong nhiều tác vụ yêu cầu kiểm soát tinh vi.

### LoRA (Low-Rank Adaptation) và Adapter Tuning

- LoRA chèn thêm ma trận hạng thấp vào trọng số attention, trong khi Adapter Tuning chèn “mô-đun phụ” vào giữa các lớp Transformer.
- Cả hai cho phép học thêm một số ít tham số để kiểm soát phong cách, giọng điệu hoặc thuộc tính sinh.

### In-Context Control

- Không cần tinh chỉnh mô hình, chỉ cần đưa ví dụ minh họa (few-shot) vào prompt.
- Ví dụ: nếu muốn mô hình sinh phản hồi ngắn gọn, ta có thể thêm vào prompt vài ví dụ “câu trả lời ngắn”.
- Đây là phương pháp thực tế nhất trong triển khai API, tuy nhiên không ổn định khi yêu cầu phức tạp.

### Các hướng mở rộng hiện đại

- RLHF (Reinforcement Learning from Human Feedback)**: dùng con người để đánh giá và tinh chỉnh chính sách sinh, giúp mô hình tuân theo các chuẩn mực xã hội.
- Direct Preference Optimization (DPO)**: thay thế RLHF truyền thống bằng tối ưu trực tiếp trên dữ liệu so sánh, đơn giản hơn.
- Control bằng ràng buộc hình thức**: ví dụ đảm bảo output phải là JSON hợp lệ, hoặc đảm bảo không vi phạm regex.
- Multi-attribute control**: điều khiển đồng thời nhiều thuộc tính (ví dụ: văn bản vừa “tích cực” vừa “ngắn gọn”).

#### 6.3.5. Sinh Nhận biết Cấu trúc (Structure-aware Generation)

Một thách thức lớn khác là sinh ngôn ngữ tự nhiên từ dữ liệu có cấu trúc (*structured data*) như **bảng**, **cơ sở dữ liệu**, hoặc **đồ thị tri thức**. Đây là nền tảng cho nhiều ứng dụng: báo cáo tự động, sinh mô tả sản phẩm, hoặc trợ lý áo truy vấn dữ liệu.

##### Sinh Văn bản từ Bảng (Table-to-text)

- Bài toán**: Cho một bảng dữ liệu, sinh ra đoạn văn mô tả hoặc tóm tắt thông tin.
- Hướng tiếp cận**:
  - Tuyến tính hoá (linearization)**: biến bảng thành chuỗi 1D với các token đặc biệt cho hàng/cột/ ô, rồi dùng Seq2Seq (T5, BART).
  - Schema-guided generation**: tận dụng nhãn cột/hàng để hướng dẫn mô hình.
  - Pretraining đặc thù**: ví dụ TAPAS (Google) huấn luyện trực tiếp trên bảng + câu hỏi.

##### Sinh Văn bản từ Đồ thị (Graph-to-text)

- Bài toán**: Cho một đồ thị tri thức (Knowledge Graph), sinh ra mô tả bằng ngôn ngữ tự nhiên.
- Hướng tiếp cận**:
  - Mạng nơ-ron đồ thị (GNN)** để mã hoá cấu trúc quan hệ thành embedding.
  - Transformer attend lên embedding này để sinh chuỗi văn bản.
  - Một số mô hình tiêu biểu: *GraphWriter*, *KG-to-Text*.

### Sinh Văn bản từ CSDL (Data-to-text)

- **Ví dụ:** Sinh báo cáo tài chính tự động từ số liệu.
- **Cách làm:** Tương tự table-to-text nhưng bổ sung thêm cơ chế **content planning** để quyết định thông tin nào cần nhắc đến trước.

## 6.4. Hệ thống Tác tử (Agentic Systems) với LLMs

Cho đến nay, chúng ta chủ yếu xem LLM như một bộ não xử lý thông tin và sinh ra văn bản. Nhưng điều gì sẽ xảy ra nếu chúng ta trao cho bộ não đó "tay chân" để tương tác với thế giới bên ngoài? Đây chính là ý tưởng đằng sau các **Hệ thống Tác tử (Agentic Systems)**.

Một Tác tử LLM (LLM Agent) không chỉ đơn thuần trả lời một câu hỏi. Nó là một hệ thống tự hành có khả năng:

- **Phân rã (Decompose)** một mục tiêu phức tạp thành các bước nhỏ hơn.
- **Lập kế hoạch (Plan)** một chuỗi các hành động để thực hiện các bước đó.
- **Sử dụng công cụ (Use Tools)** như truy cập API, chạy mã nguồn, hoặc tìm kiếm trên web để thu thập thông tin và thực hiện hành động.
- **Quan sát (Observe)** kết quả của các hành động và **tự sửa lỗi (self-correct)** nếu cần.

Về cơ bản, chúng ta đang cố gắng xây dựng một vòng lặp, trong đó LLM đóng vai trò là "bộ não" ra quyết định, và các công cụ bên ngoài là "cơ thể" thực thi.

### 6.4.1. Vòng lặp Suy luận-Hành động: Khung sườn ReAct

Một trong những khung sườn (frameworks) nền tảng và có ảnh hưởng nhất cho việc xây dựng Tác tử là **ReAct (Reasoning and Acting)** (Yao et al., 2022).

#### Triết lý của ReAct

ReAct đề xuất rằng để hoàn thành một mục tiêu, một Tác tử nên xen kẽ một cách linh hoạt giữa hai hành vi nguyên thủy: **Suy luận (Reasoning)** và **Hành động (Acting)**. Việc suy luận giúp Tác tử xây dựng và điều chỉnh kế hoạch, trong khi việc hành động giúp nó thu thập thông tin mới từ môi trường bên ngoài để cung cấp cho các suy luận tiếp theo.

#### Cơ chế hoạt động

ReAct hoạt động bằng cách dạy cho LLM sinh ra các "lượt lời" có một định dạng đặc biệt, bao gồm:

1. **Thought (Suy tưởng):** Một đoạn văn bản mô tả dòng suy nghĩ của Tác tử, phân tích tình hình hiện tại, và quyết định bước tiếp theo. Đây là phần suy luận.
2. **Action (Hành động):** Một lệnh gọi đến một công cụ cụ thể. Ví dụ: 'search[thủ đô của Pháp]' hoặc 'calculate[2\*5]'.
3. **Observation (Quan sát):** Sau khi hệ thống thực thi hành động (ví dụ: chạy công cụ tìm kiếm), kết quả trả về sẽ được chèn vào prompt dưới dạng một "quan sát".

LLM sau đó sẽ lặp lại vòng lặp '(Thought, Action, Observation)' này cho đến khi nó có đủ thông tin để đưa ra câu trả lời cuối cùng.

#### Ví dụ 14: Minh họa vòng lặp ReAct

**Mục tiêu:** "Tác giả của cuốn sách 'The Hobbit' sinh ở quốc gia nào?"

**Vòng 1:**

- **Thought:** Tôi cần tìm tác giả của 'The Hobbit' trước. Sau đó tôi sẽ tìm nơi sinh của tác giả đó.
- **Action:** 'search[tác giả của The Hobbit]'

- **Observation:** (Hệ thống chạy tìm kiếm và trả về) "J.R.R. Tolkien là tác giả của The Hobbit."

Vòng 2:

- **Thought:** OK, tác giả là J.R.R. Tolkien. Böyle giờ tôi cần tìm nơi ông ấy sinh ra.
- **Action:** 'search[nơi sinh của J.R.R. Tolkien]'
- **Observation:** (Hệ thống chạy tìm kiếm và trả về) "J.R.R. Tolkien sinh tại Bloemfontein, Orange Free State (nay là một phần của Nam Phi)"

Vòng 3:

- **Thought:** Tôi đã có câu trả lời. J.R.R. Tolkien sinh ra ở nơi mà bây giờ là Nam Phi. Tôi có thể kết thúc.
- **Action:** 'finish[J.R.R. Tolkien, tác giả của 'The Hobbit', sinh ra ở Nam Phi.]'

ReAct đã chứng minh rằng việc buộc LLM phải "nói ra suy nghĩ của mình" (verbalize its thoughts) giúp nó lập kế hoạch và phục hồi sau lỗi tốt hơn đáng kể.

### 6.4.2. Lập kế hoạch và Sử dụng Công cụ (Planning & Tool Use)

Đây là hai khả năng cốt lõi của một Tác tử.

#### Lập kế hoạch (Planning)

- **Nhiệm vụ:** Phân rã một mục tiêu cấp cao thành một chuỗi các nhiệm vụ con có thể thực hiện được.
- **Các cách tiếp cận:**
  - **Suy luận một bước (Single-step reasoning):** Như trong ReAct, Tác tử chỉ quyết định hành động tiếp theo tại mỗi bước. Cách này linh hoạt nhưng có thể không tối ưu.
  - **Lập kế hoạch nhiều bước (Multi-step planning):** Trước khi thực hiện bất kỳ hành động nào, Tác tử sẽ cố gắng tạo ra một kế hoạch hoàn chỉnh gồm nhiều bước. Sau đó, nó sẽ thực thi kế hoạch đó. Cách này có thể tối ưu hơn nhưng kém linh hoạt khi gặp các tình huống bất ngờ.
  - **Lập kế hoạch có phản hồi (Planning with feedback):** Kết hợp cả hai. Tác tử tạo một kế hoạch ban đầu, thực thi từng bước, và sau mỗi bước, nó quan sát kết quả và có thể điều chỉnh lại kế hoạch nếu cần.

#### Sử dụng Công cụ (Tool Use)

- **Định nghĩa:** Công cụ là bất kỳ nguồn tài nguyên nào bên ngoài mà Tác tử có thể tương tác, ví dụ:
  - **API:** API thời tiết, API đặt vé máy bay, API của các ứng dụng khác.
  - **Cơ sở dữ liệu:** Truy vấn một cơ sở dữ liệu SQL.
  - **Môi trường thực thi mã nguồn:** Viết và chạy một đoạn mã Python để thực hiện các phép tính phức tạp.
  - **Các hệ thống RAG:** Sử dụng tìm kiếm trên tài liệu như một công cụ.
- **Cơ chế hoạt động:** LLM được dạy (qua fine-tuning hoặc prompting) cách sinh ra các lệnh gọi công cụ theo một định dạng cụ thể (ví dụ: JSON hoặc lời gọi hàm). Một bộ điều phối (orchestrator) bên ngoài sẽ phân tích đầu ra của LLM, thực thi lệnh gọi công cụ tương ứng, và trả kết quả về cho LLM dưới dạng một "Observation".
- **Function Calling:** Các API LLM hiện đại (như của OpenAI, Google) đã tích hợp sẵn khả năng "Function Calling", giúp chuẩn hóa và đơn giản hóa quá trình này.

### 6.4.3. Bộ nhớ cho Tác tử (Agent Memory)

Để thực hiện các nhiệm vụ kéo dài và học hỏi từ các tương tác trong quá khứ, Tác tử cần có bộ nhớ.

### Bộ nhớ Ngắn hạn (Short-term Memory)

- **Bản chất:** Đây chính là **cửa sổ ngữ cảnh (context window)** của LLM.
- **Cơ chế:** Toàn bộ lịch sử của cuộc hội thoại hiện tại (bao gồm các vòng lặp Thought-Action-Observation) được đưa vào prompt ở mỗi lượt.
- **Hạn chế:** Bị giới hạn bởi độ dài ngữ cảnh của LLM. Khi cuộc hội thoại quá dài, các thông tin cũ sẽ bị mất đi.

### Bộ nhớ Dài hạn (Long-term Memory)

- **Bản chất:** Cung cấp cho Tác tử khả năng lưu trữ và truy xuất thông tin qua nhiều cuộc hội thoại khác nhau.
- **Cơ chế:**
  1. **Lưu trữ (Storage):** Sau mỗi tương tác, các thông tin quan trọng (ví dụ: các cặp hỏi-đáp, các tóm tắt, các bài học kinh nghiệm) được trích xuất và lưu trữ vào một cơ sở dữ liệu bên ngoài, thường là một **Cơ sở dữ liệu Vector**.
  2. **Truy xuất (Retrieval):** Ở đầu mỗi tương tác mới, hệ thống sẽ lấy câu hỏi hoặc mục tiêu hiện tại của người dùng và sử dụng nó để **tìm kiếm (retrieve)** các ký ức liên quan nhất từ bộ nhớ dài hạn.
  3. **Tích hợp:** Các ký ức đã được truy xuất này sau đó được chèn vào prompt (bộ nhớ ngắn hạn), cung cấp cho Tác tử ngữ cảnh về các tương tác trong quá khứ.
- **Kết luận:** Về cơ bản, bộ nhớ dài hạn của Tác tử thường được triển khai bằng chính kiến trúc RAG.

#### 6.4.4. Tự sửa lỗi (Self-correction)

Một Tác tử thông minh không chỉ thực thi, mà còn có khả năng nhận ra và sửa chữa lỗi lầm của mình.

- **Cơ chế:** Đây là một vòng lặp meta-suy luận.
  1. **Tạo đầu ra ban đầu:** Tác tử tạo ra một kế hoạch hoặc một câu trả lời.
  2. **Tự phê bình (Self-critique):** Sau đó, hệ thống sẽ đưa chính đầu ra đó trở lại cho LLM với một prompt khác, yêu cầu nó đóng vai một "người phê bình" và tìm ra các lỗi, điểm yếu, hoặc các giả định sai lầm trong đầu ra ban đầu.
  3. **Sửa đổi (Refinement):** Dựa trên những lời phê bình, Tác tử sẽ được yêu cầu sửa đổi lại kế hoạch hoặc câu trả lời của mình.
- **Lợi ích:** Kỹ thuật này giúp cải thiện đáng kể độ tin cậy và chất lượng của các hệ thống Tác tử, đặc biệt là trong các nhiệm vụ phức tạp đòi hỏi sự chính xác cao.

Việc kết hợp các thành phần này – ReAct, Lập kế hoạch, Sử dụng Công cụ, Bộ nhớ, và Tự sửa lỗi – cho phép chúng ta xây dựng các hệ thống Tác tử ngày càng tự chủ và mạnh mẽ, đánh dấu một bước tiến lớn trong hành trình hướng tới Trí tuệ Nhân tạo Tổng quát.

## 6.5. Mô hình Thế giới (World Models) và Mô phỏng

Khi chúng ta đi đến cuối chương về các hệ thống nâng cao, một câu hỏi sâu sắc và cơ bản được đặt ra: Khi một LLM dự đoán từ tiếp theo một cách chính xác trên một lượng dữ liệu khổng lồ, nó thực sự đang học được điều gì? Liệu nó có chỉ đơn thuần học các quy luật thống kê bề mặt của ngôn ngữ, hay nó đang âm thầm xây dựng một thứ gì đó sâu sắc hơn – một **mô hình nội tại (internal model)** về thế giới mà ngôn ngữ đó mô tả?

Đây chính là ý tưởng cốt lõi đằng sau khái niệm **Mô hình Thế giới (World Models)**.

### 6.5.1. Khái niệm về khả năng LLM xây dựng mô hình nội tại

#### Định nghĩa 7: Mô hình Thế giới (trong bối cảnh LLM)

Một Mô hình Thế giới là một biểu diễn nén, có thể thực thi (executable), và trùu tượng về các thực thể, mối quan hệ, và các quy luật vật lý/xã hội của thế giới, được hình thành bên trong các trọng số của một mô hình ngôn ngữ lớn thông qua quá trình huấn luyện trên dữ liệu văn bản.

#### Triết lý của Mô hình Thế giới

Giả thuyết cho rằng: để có thể dự đoán văn bản một cách hiệu quả và nhất quán trên nhiều lĩnh vực, cách tối ưu nhất cho mô hình không phải là ghi nhớ tất cả các chuỗi ký tự, mà là xây dựng một mô hình nén của các quy luật sinh ra văn bản đó. Vì văn bản là sự phản ánh của thế giới, nên một mô hình tốt của văn bản cũng phải là một mô hình tốt của thế giới.

#### Các bằng chứng gián tiếp

Mặc dù chúng ta không thể "nhìn" trực tiếp vào các trọng số và thấy một "mô hình thế giới" rõ ràng, có nhiều bằng chứng gián tiếp cho thấy các LLM lớn đang học được những biểu diễn như vậy:

#### Biểu diễn không gian và thời gian

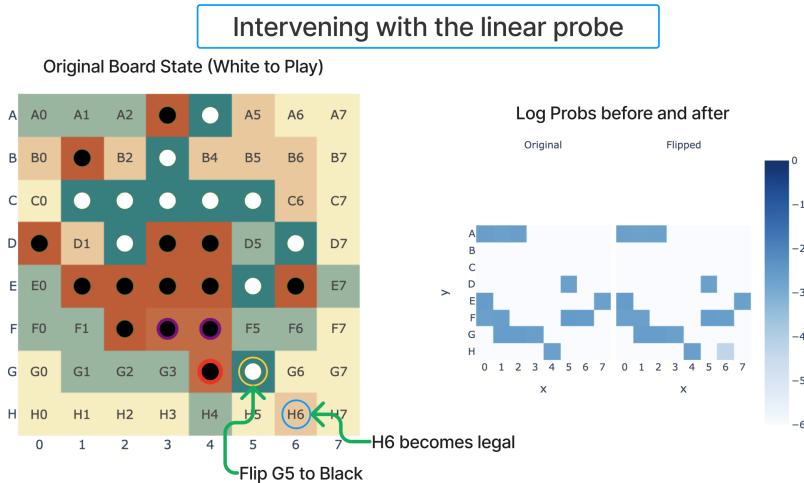
- Các nhà nghiên cứu đã phát hiện ra các nơ-ron hoặc các hướng (directions) trong không gian kích hoạt của LLM dường như tương ứng với các khái niệm trong thế giới thực. Ví dụ, có những nơ-ron chỉ "kích hoạt" mạnh khi mô hình xử lý các văn bản liên quan đến một địa điểm địa lý cụ thể như "Tháp Eiffel" hoặc một mốc thời gian cụ thể.
- Yann LeCun đã đưa ra ví dụ: để dự đoán từ cuối cùng trong câu "John cầm quả bóng, đi vào phòng khách, đặt quả bóng xuống, rồi đi vào bếp. Quả bóng đang ở...", mô hình phải có một biểu diễn nội tại về không gian, về các đối tượng và vị trí của chúng.

#### Mô phỏng các thực thể

- Khi một LLM tương tác trong một vai trò (ví dụ: "hãy đóng vai Napoleon"), nó có thể duy trì sự nhất quán về kiến thức, tính cách, và các mối quan hệ của nhân vật đó trong một cuộc hội thoại dài. Điều này cho thấy nó không chỉ lấy ra các thông tin rời rạc, mà dường như đang "mô phỏng" một mô hình về thực thể "Napoleon".

**Mô hình Othello-GPT** Một nghiên cứu nổi tiếng đã huấn luyện một Transformer nhỏ chỉ để chơi cờ Othello (cờ lật). Mô hình chỉ nhận vào chuỗi các nước đi hợp lệ và dự đoán nước đi hợp lệ tiếp theo.

- Kết quả đáng kinh ngạc:** Sau khi huấn luyện, các nhà nghiên cứu đã tìm thấy một biểu diễn rõ ràng của **trạng thái bàn cờ Othello** bên trong các kích hoạt của mô hình. Tức là, mô hình đã tự học cách xây dựng một "mô hình thế giới" (bàn cờ) để có thể chơi tốt, mặc dù nó chưa bao giờ được "nhìn thấy" bàn cờ một cách trực tiếp.



Hình 6.7: Nghiên cứu Othello-GPT cho thấy một mô hình Transformer, chỉ được huấn luyện trên chuỗi nước đi, đã tự học cách biểu diễn trạng thái của bàn cờ bên trong nó.

### 6.5.2. Hướng nghiên cứu và Tiềm năng

Việc các LLM có thể xây dựng mô hình thế giới nội tại mở ra những hướng đi cực kỳ hứa hẹn cho tương lai của AI.

#### Cải thiện khả năng Suy luận và Lập kế hoạch

- Nếu một Tác tử AI có một mô hình thế giới nội tại tốt, nó có thể thực hiện "mô phỏng trong tâm trí"(mental simulation).
- Trước khi thực hiện một hành động trong thế giới thực, nó có thể "chạy thử" hành động đó trên mô hình thế giới của mình để dự đoán kết quả.
- Điều này cho phép nó lập kế hoạch một cách hiệu quả hơn nhiều, lường trước các hậu quả, và tránh các sai lầm tốn kém, tương tự như cách con người suy nghĩ "nếu... thì...".

#### Mô hình Thế giới Đa phương thức

- Hướng nghiên cứu hiện tại đang tập trung vào việc xây dựng các mô hình thế giới không chỉ từ văn bản, mà còn từ **video**.
- Bằng cách xem hàng triệu giờ video, một mô hình có thể học được các quy luật vật lý trực quan (intuitive physics) – ví dụ, một vật thể không thể đi xuyên qua vật thể khác, trọng lực làm cho mọi thứ rơi xuống.
- Các mô hình như **Sora** của OpenAI, có khả năng sinh ra các video cực kỳ thực tế và nhất quán về mặt vật lý, được coi là một bước tiến lớn trong việc xây dựng các mô hình thế giới có khả năng mô phỏng.

#### Hướng tới Trí tuệ Nhân tạo Tổng quát (AGI)

- Nhiều nhà nghiên cứu hàng đầu tin rằng khả năng xây dựng và sử dụng các mô hình thế giới là một trong những thành phần cốt lõi của trí thông minh thực sự.
- Một hệ thống AGI trong tương lai có thể sẽ bao gồm một LLM lớn đóng vai trò là "giao diện" ngôn ngữ, kết hợp với một mô hình thế giới mạnh mẽ cho phép nó hiểu sâu, suy luận, và lập kế hoạch trong môi trường phức tạp.

Mặc dù vẫn còn là một lĩnh vực đầy tranh cãi và câu hỏi mở, giả thuyết về Mô hình Thế giới cung cấp một lăng kính mạnh mẽ để hiểu về những khả năng đáng kinh ngạc của LLM. Nó gợi ý rằng chúng ta có thể đang vô tình xây dựng được những thứ phức tạp hơn nhiều so với những gì

chúng ta nghĩ ban đầu, mở ra một tương lai đầy tiềm năng cho các hệ thống AI có khả năng lý luận và tương tác với thế giới một cách sâu sắc.

## 6.6. Các Kỹ thuật Biểu diễn Nâng cao cho Truy xuất

### 6.6.1. Biểu diễn Đa ngôn ngữ (Cross-lingual Embeddings)

#### Vấn đề

Các mô hình như Word2Vec hay GloVe khi huấn luyện riêng biệt trên từng ngôn ngữ sẽ tạo ra các không gian vector độc lập, không có mối liên hệ. Ví dụ: vector của từ "máy tính" (tiếng Việt) và "computer" (tiếng Anh) có thể nằm ở vị trí hoàn toàn ngẫu nhiên trong hai không gian khác nhau, dù ý nghĩa giống nhau.

#### Mục tiêu

**Biểu diễn đa ngôn ngữ** tìm cách xây dựng một **không gian vector chung** (shared vector space), trong đó các từ/câu có nghĩa tương đương ở các ngôn ngữ khác nhau sẽ được ánh xạ gần nhau. Điều này cho phép:

- Tìm kiếm thông tin đa ngôn ngữ (cross-lingual information retrieval).
- Phân loại tài liệu nhiều ngôn ngữ với một mô hình duy nhất.
- Dịch máy hoặc gợi ý từ/câu dựa trên ngữ nghĩa thay vì chỉ dựa vào từ vựng.

#### Một "Ngôn ngữ chung" của các Vector

Hãy tưởng tượng một không gian nói vector của "mèo" (vi), "cat" (en), "neko" (ja) và "gato" (es) đều hội tụ về cùng một vùng lân cận. Đây là sức mạnh của biểu diễn đa ngôn ngữ – tạo ra một "cầu nối" toán học giữa các ngôn ngữ tự nhiên.

#### Các hướng tiếp cận chính

1. **Mapping-based (Học ma trận ánh xạ)** Huấn luyện một ma trận  $W$  sao cho khi nhân với embedding của ngôn ngữ nguồn, ta nhận được embedding "khớp" với ngôn ngữ đích:

$$W \cdot E_{\text{src}} \approx E_{\text{tgt}}$$

- MUSE (Lample et al.): Tìm  $W$  bằng cách tối ưu khoảng cách giữa các cặp từ song ngữ. Có thể: - *Giám sát*: Dùng từ điển song ngữ nhỏ. - *Phi giám sát*: Sử dụng huấn luyện đối nghịch (adversarial training) để khớp phân phối embedding.
- 2. **Joint training (Huấn luyện chung nhiều ngôn ngữ)** Huấn luyện một encoder duy nhất trên dữ liệu song song nhiều ngôn ngữ để ép embedding về chung một không gian. - LASER: Sử dụng BiLSTM encoder + max-pooling, huấn luyện trên hơn 90 ngôn ngữ. - LaBSE: Thay BiLSTM bằng Transformer (BERT), tối ưu độ tương đồng cosine giữa cặp câu dịch chuẩn và giảm độ tương đồng với cặp câu ngẫu nhiên.
- 3. **Multilingual Language Models** Huấn luyện mô hình ngôn ngữ đa ngôn ngữ trực tiếp: - mBERT, XLM-R: Học embedding từ pretraining masked language modeling trên nhiều ngôn ngữ. - mT5, BLOOMZ: Mô hình sinh đa ngôn ngữ, có thể dùng embedding tầng encoder.

#### Thách thức

- **Polysemy**: Từ đa nghĩa có thể gây nhầm lẫn nếu không có ngữ cảnh.
- **Resource imbalance**: Một số ngôn ngữ có ít dữ liệu song song → khó huấn luyện tốt.
- **Domain shift**: Khác biệt miền dữ liệu (ví dụ: tin tức vs hội thoại) làm giảm chất lượng ánh xạ.

### Ứng dụng minh họa

- Tìm kiếm học thuật: Gõ từ khoá tiếng Việt nhưng vẫn tìm được bài báo tiếng Anh/Nhật.
- Phân tích cảm xúc đa ngôn ngữ: Một mô hình duy nhất cho nhiều ngôn ngữ.
- Dịch máy zero-shot: Mô hình chưa từng thấy cặp ngôn ngữ nhưng vẫn dịch được nhờ embedding chung.

### 6.6.2. Biểu diễn Thưa và Tối ưu cho Truy xuất (Sparse & Retriever-specific Embeddings)

#### Bối cảnh

Trong tìm kiếm thông tin (Information Retrieval – IR), có hai trường phái chính:

- **Sparse retrieval** (ví dụ: TF-IDF, BM25): Biểu diễn tài liệu và truy vấn như vector thưa với số chiều bằng kích thước từ vựng. Nhanh, dễ lập chỉ mục, nhưng yếu về ngữ nghĩa – chỉ khớp chính xác từ khóa.
- **Dense retrieval** (ví dụ: BERT embeddings): Biểu diễn tài liệu và truy vấn bằng vector dày đặc, có khả năng nắm bắt ý nghĩa, nhưng tìm kiếm trên hàng triệu vector tốn tài nguyên (phải dùng ANN – Approximate Nearest Neighbor).

#### Vấn đề

Dense embeddings tuy mạnh về ngữ nghĩa nhưng:

1. Bỏ lỡ các khớp từ khóa chính xác (*exact match*), đặc biệt quan trọng với các tên riêng, thuật ngữ kỹ thuật.
2. Chi phí lưu trữ và tìm kiếm cao khi số tài liệu rất lớn.

#### Mục tiêu

Tạo ra các **retriever-specific embeddings** kết hợp được:

- Sức mạnh khớp từ khóa của sparse retrieval.
- Hiểu ngữ nghĩa sâu của dense retrieval.

#### Các phương pháp tiêu biểu

##### SPLADE (SPARse Lexical AnD Expansion model)

- Huấn luyện một Transformer (BERT/DistilBERT) để đầu ra là vector thưa có số chiều bằng kích thước từ vựng.
- Mỗi chiều tương ứng với một token trong từ vựng, giá trị là mức độ quan trọng (*term weight*).
- **Hỗ trợ term expansion**: gán trọng số cho cả từ không xuất hiện trong văn bản nhưng có liên quan ngữ nghĩa.
- Hàm loss: thường là *margin ranking loss* hoặc *cross-entropy loss* trên cặp truy vấn–tài liệu (relevant vs non-relevant).
- **Ưu điểm**: tương thích với **inverted index**, tốc độ truy xuất gần như BM25, nhưng giàu ngữ nghĩa hơn.

##### ColBERT (Contextualized Late Interaction over BERT)

- Thay vì nén cả tài liệu thành một vector, ColBERT tạo embedding dày đặc cho từng token.
- Khi truy vấn, với mỗi token trong câu hỏi, tính độ tương đồng tối đa với tất cả token embedding của tài liệu.
- Hàm loss: tối đa hóa độ tương đồng giữa truy vấn và tài liệu liên quan, đồng thời giảm độ tương đồng với tài liệu không liên quan.
- **Ưu điểm**: giữ được chi tiết cấp từ, cho độ chính xác cao hơn dense retriever thuần.
- **Nhược điểm**: tốn bộ nhớ và chậm hơn do số lượng embedding lớn.

### Kết hợp trong hệ thống RAG

- SPLADE: tốt để mở rộng tập ứng viên (candidate set) nhờ độ phủ cao và khả năng truy xuất nhanh.
- ColBERT: tốt để rerank tập ứng viên nhỏ, nhờ giữ được ngữ cảnh chi tiết.
- Chiến lược phổ biến: SPLADE → lấy top-1000 tài liệu → ColBERT rerank top-50 → đưa vào LLM.

### Thách thức

- Tối ưu tốc độ khi kết hợp nhiều tầng retriever.
- Cân bằng giữa recall (độ phủ) và precision (độ chính xác).
- Chi phí lưu trữ embedding lớn với mô hình late-interaction.

## KẾT THÚC CHƯƠNG 6

Chúng ta đã đi đến cuối hành trình khám phá các hệ thống AI tiên tiến. Chương này đã mở rộng tầm nhìn của chúng ta vượt ra ngoài một LLM đơn lẻ, cho thấy cách kết hợp nó với các nguồn kiến thức bên ngoài (RAG), các phương thức khác (Multimodality), và các công cụ (Agentic Systems) để tạo ra các giải pháp mạnh mẽ. Chúng ta cũng đã tìm hiểu các kỹ thuật tối ưu hóa để đưa những mô hình này vào thực tế và suy ngẫm về khái niệm "Mô hình Thế giới". Chúng ta đã xây dựng nên những hệ thống AI vô cùng phức tạp và mạnh mẽ. Nhưng một câu hỏi tối quan trọng vẫn còn đó: Làm thế nào để chúng ta đo lường và so sánh chúng một cách khoa học? Liệu mô hình A có thực sự tốt hơn mô hình B không, và dựa trên tiêu chí nào? Chương cuối cùng của Phần 1 sẽ đi sâu vào lý thuyết và thực hành của việc đánh giá, cung cấp cho chúng ta những công cụ cần thiết để định lượng sự tiến bộ và hiểu rõ những giới hạn của các mô hình này.

## Chương 7

# LÝ THUYẾT ĐÁNH GIÁ VÀ CÁC BENCHMARK

Xuyên suốt các chương vừa qua, chúng ta đã khám phá một loạt các kiến trúc và kỹ thuật để xây dựng các mô hình NLP ngày càng mạnh mẽ. Tuy nhiên, một câu hỏi nền tảng vẫn còn đó: "Làm thế nào để chúng ta đo lường sự 'tốt' của một mô hình một cách khách quan?". Việc xây dựng các mô hình mà không có phương pháp đánh giá rõ ràng cũng giống như lái một con tàu mà không có lá bàn.

### 7.1. Các Metric kinh điển

Đây là những công cụ đầu tiên và cơ bản nhất trong bộ công cụ của bất kỳ nhà thực hành NLP nào. Mỗi metric được thiết kế để đo lường một khía cạnh cụ thể của hiệu năng mô hình.

#### 7.1.1. Perplexity: Đo lường chất lượng của Mô hình Ngôn ngữ

- **Dùng cho tác vụ nào?** Đánh giá chất lượng nội tại của các **Mô hình Ngôn ngữ** (Language Models) (ví dụ: các mô hình được huấn luyện để dự đoán từ tiếp theo).
- **Trực giác cốt lõi:** Một mô hình ngôn ngữ tốt là mô hình ít "ngạc nhiên" (less surprised) hơn khi nhìn thấy một câu văn thực tế. Perplexity (PPL), hay còn gọi là Độ bối rối, chính là một cách để định lượng sự "ngạc nhiên" này. **Perplexity càng thấp, mô hình càng tốt.**
- **Cơ chế hoạt động:** Perplexity có mối quan hệ trực tiếp với hàm mất mát Cross-Entropy. Đối với một chuỗi văn bản  $W = (w_1, w_2, \dots, w_N)$ , Perplexity được định nghĩa là:

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}} = \left( \prod_{i=1}^N \frac{1}{P(w_i|w_1, \dots, w_{i-1})} \right)^{1/N} \quad (7.1)$$

Về mặt toán học, nó là nghịch đảo của xác suất trung bình trên mỗi từ (được lấy căn bậc  $N$ ).

- **Hạn chế:**
  - Chỉ đo lường khả năng dự đoán xác suất, không trực tiếp đo lường chất lượng ngôn ngữ nghĩa hay sự mạch lạc của văn bản được sinh ra.
  - Rất nhạy cảm với kích thước từ vựng. Không thể so sánh Perplexity của hai mô hình có từ vựng khác nhau.

#### 7.1.2. Precision, Recall, và F1-Score: Nền tảng của các bài toán Phân loại

Đây là bộ ba metric không thể thiếu cho hầu hết các bài toán **phân loại** (classification) và **trích xuất thông tin** (information extraction) như NER.

#### Các thành phần cơ bản

Để hiểu bộ ba này, chúng ta cần định nghĩa bốn kết quả có thể xảy ra khi phân loại một điểm dữ liệu:

- **True Positive (TP):** Dự đoán là **Dương tính**, và thực tế là **Dương tính**. (Dự đoán đúng)
- **True Negative (TN):** Dự đoán là **Âm tính**, và thực tế là **Âm tính**. (Dự đoán đúng)
- **False Positive (FP):** Dự đoán là **Dương tính**, nhưng thực tế là **Âm tính**. (Lỗi Loại I - "báo động giả")
- **False Negative (FN):** Dự đoán là **Âm tính**, nhưng thực tế là **Dương tính**. (Lỗi Loại II - "bỏ sót")

		Predicted	
		NEGATIVE	POSITIVE
Actual	NEGATIVE	Count of <b>TN</b>	Count of <b>FP</b>
	POSITIVE	Count of <b>FN</b>	Count of <b>TP</b>

Hình 7.1: Ma trận nhầm lẫn (Confusion Matrix) minh họa bốn kết quả của một bài toán phân loại nhị phân.

### Định nghĩa các Metric

#### Precision (Độ chính xác)

- **Câu hỏi nó trả lời:** "Trong số tất cả những gì mô hình dự đoán là Dương tính, có bao nhiêu cái thực sự là Dương tính?"
- **Công thức:**

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Mục tiêu:** Tối đa hóa Precision có nghĩa là cố gắng giảm thiểu số lượng báo động giả (FP). Rất quan trọng trong các ứng dụng như phát hiện spam (thà bỏ sót một email spam còn hơn là đưa một email quan trọng vào hòm thư rác).

#### Recall (Độ phủ, hay Độ nhạy)

- **Câu hỏi nó trả lời:** "Trong số tất cả những cái thực sự là Dương tính, mô hình đã tìm thấy được bao nhiêu cái?"
- **Công thức:**

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **Mục tiêu:** Tối đa hóa Recall có nghĩa là cố gắng giảm thiểu số lượng bỏ sót (FN). Rất quan trọng trong các ứng dụng như chẩn đoán y tế (thà chẩn đoán nhầm một người khỏe mạnh là có bệnh còn hơn là bỏ sót một người thực sự có bệnh).

#### F1-Score (Điểm F1)

- **Vấn đề:** Precision và Recall thường có sự đánh đổi (trade-off). Một mô hình rất "cẩn trọng" có thể có Precision cao nhưng Recall thấp, và ngược lại. F1-score được tạo ra để kết hợp cả hai thành một chỉ số duy nhất.
- **Công thức:** F1-score là trung bình điều hòa (harmonic mean) của Precision và Recall.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Mục tiêu:** F1-score cao khi cả Precision và Recall đều cao. Nó là một thước đo cân bằng, được sử dụng rộng rãi trong các tác vụ như NER, nơi chúng ta quan tâm đến cả việc không nhận dạng sai và không bỏ sót thực thể.

Khi đánh giá trên nhiều lớp, người ta thường tính ‘macro-F1’ (tính F1 cho mỗi lớp rồi lấy trung bình) hoặc ‘micro-F1’ (tính tổng TP, FP, FN trên tất cả các lớp rồi mới tính F1).

### 7.1.3. BLEU: Đánh giá Dịch máy

- **Dùng cho tác vụ nào?** Chủ yếu là **Dịch máy** (Machine Translation), nhưng đôi khi cũng dùng cho các tác vụ sinh văn bản khác.
- **Trực giác cốt lõi:** Một bản dịch tốt nên có nhiều N-gram trùng lặp với các bản dịch tham khảo chất lượng cao do con người tạo ra.
- **Cơ chế hoạt động:** BLEU (Bilingual Evaluation Understudy) tính toán một điểm số dựa trên hai yếu tố:
  1. **Độ chính xác N-gram đã được sửa đổi (Modified N-gram Precision):** Tính toán tỷ lệ các N-gram (thường từ 1-gram đến 4-gram) trong câu do máy dịch tạo ra cũng xuất hiện trong các câu tham khảo. Nó được “sửa đổi” để ngăn mô hình lặp lại một từ đúng nhiều lần để gian lận điểm.
  2. **Hình phạt cho câu ngắn (Brevity Penalty - BP):** Nếu câu do máy dịch tạo ra ngắn hơn đáng kể so với các câu tham khảo, nó sẽ bị phạt. Điều này ngăn mô hình chỉ sinh ra một vài từ rất chính xác để có Precision cao.

$$\text{BLEU} = \text{BP} \cdot \exp \left( \sum_{n=1}^N w_n \log p_n \right)$$

Trong đó  $p_n$  là độ chính xác của n-gram, và  $w_n$  thường là  $1/N$ .

- **Hạn chế:**
  - **Chỉ quan tâm đến Precision, không quan tâm đến Recall:** Nó không phạt nếu bản dịch bỏ sót các thông tin quan trọng.
  - **Không hiểu ngữ nghĩa:** Một bản dịch có thể dùng từ đồng nghĩa hoàn hảo nhưng vẫn bị điểm BLEU thấp nếu không khớp chính xác N-gram.
  - **Gặp khó khăn với trật tự từ và sự sáng tạo:** Không đánh giá tốt các bản dịch có cấu trúc câu khác biệt nhưng vẫn đúng.

### 7.1.4. ROUGE: Đánh giá Tóm tắt Văn bản

- **Dùng cho tác vụ nào?** Chủ yếu là **Tóm tắt văn bản (Summarization)**.
- **Trực giác cốt lõi:** ROUGE (Recall-Oriented Understudy for Gisting Evaluation) hoạt động ngược lại với BLEU. Nó cho rằng một bản tóm tắt tốt nên “bao phủ” được các N-gram quan trọng có trong các bản tóm tắt tham khảo của con người.
- **Cơ chế hoạt động:** ROUGE là một họ các metric, trong đó phổ biến nhất là:
  - **ROUGE-N:** Dựa trên sự trùng lặp N-gram, nhưng tính toán dựa trên Recall.

$$\text{ROUGE-N} = \frac{\text{Số N-gram trong tham khảo cũng xuất hiện trong tóm tắt máy}}{\text{Tổng số N-gram trong tham khảo}}$$

Thường dùng ‘ROUGE-1’ (unigram) và ‘ROUGE-2’ (bigram).

- **ROUGE-L:** Dựa trên Chuỗi con chung dài nhất (Longest Common Subsequence - LCS). Nó không yêu cầu các từ phải liền kề nhau, do đó nắm bắt được sự tương đồng về trật tự từ tốt hơn.
- **Hạn chế:** Tương tự như BLEU, ROUGE cũng chỉ dựa trên sự khớp bề mặt của từ và không hiểu được ngữ nghĩa sâu sắc. Một bản tóm tắt có thể diễn đạt lại ý tưởng một cách hoàn hảo nhưng vẫn bị điểm ROUGE thấp.

Mặc dù có nhiều hạn chế, các metric kinh điển này vẫn được sử dụng rộng rãi vì chúng tự động, nhanh chóng và cung cấp một thước đo khách quan (dù không hoàn hảo) để so sánh các mô hình với nhau trong quá trình phát triển.

## 7.2. Các Benchmark Kinh điển và Hiện đại

Một metric riêng lẻ chỉ đo lường một khía cạnh của mô hình. Để có một đánh giá toàn diện, cộng đồng nghiên cứu đã tạo ra các **benchmark** – những bộ sưu tập các bộ dữ liệu và tác vụ được tiêu chuẩn hóa. Các benchmark đóng vai trò như những "kỳ thi Olympic" cho các mô hình NLP, cho phép các nhà nghiên cứu so sánh một cách công bằng và theo dõi sự tiến bộ của toàn ngành.

Sự phát triển của các benchmark cũng phản ánh sự tiến hóa của chính các mô hình NLP.

### 7.2.1. Kỷ nguyên Pre-LLM: GLUE và SuperGLUE

Trước khi các LLM lớn với khả năng zero-shot ra đời, các mô hình (như BERT) thường được đánh giá bằng cách tinh chỉnh (fine-tune) trên một loạt các tác vụ Hiểu Ngôn ngữ Tự nhiên (NLU) đa dạng. GLUE và SuperGLUE là hai benchmark tiêu biểu nhất cho kỷ nguyên này.

#### GLUE: Một Bài kiểm tra Năng lực Ngôn ngữ Tổng quát

- **Tên đầy đủ:** General Language Understanding Evaluation.
- **Triết lý:** Một mô hình thực sự hiểu ngôn ngữ phải hoạt động tốt trên nhiều loại tác vụ khác nhau, từ suy luận, phân tích cảm xúc, đến nhận dạng sự tương đồng.
- **Cấu trúc:** GLUE bao gồm 9 tác vụ NLU khác nhau, được nhóm thành các loại:
  - Suy luận Ngôn ngữ Tự nhiên (NLI): MNLI, QNLI, RTE.
  - Tương đồng Nghĩa và Diễn giải: MRPC, QQP, STS-B.
  - Phân loại câu đơn: SST-2 (cảm xúc), CoLA (tính đúng ngữ pháp).
- **Đánh giá:** Một mô hình được fine-tune riêng cho từng tác vụ. Điểm số cuối cùng là điểm trung bình trên tất cả các tác vụ.
- **Tác động:** GLUE đã trở thành một "thuốc đo vàng" để đánh giá các mô hình như BERT, RoBERTa. Tuy nhiên, các mô hình nhanh chóng đạt đến và vượt qua cả **hiệu năng của con người** trên benchmark này vào khoảng năm 2019, cho thấy nó đã trở nên quá "dễ".

#### SuperGLUE: Nâng cấp Độ khó

- **Động lực:** Do GLUE đã bị "chinh phục", SuperGLUE được tạo ra với các tác vụ khó hơn, đòi hỏi khả năng suy luận phức tạp hơn.
- **Cấu trúc:** Nó cũng là một bộ sưu tập các tác vụ đa dạng, nhưng tập trung nhiều hơn vào:
  - Hỏi-dáp yêu cầu suy luận: BoolQ, MultiRC.
  - Suy luận nhân quả: COPA.
  - Giải quyết đồng tham chiếu: WSC.
- **Tác động:** SuperGLUE đã tiếp tục là một thách thức trong một thời gian, nhưng rồi các mô hình lớn hơn cũng dần chinh phục được nó. Sự bão hòa của các benchmark này cho thấy một giới hạn của mô hình fine-tuning và sự cần thiết của các phương pháp đánh giá mới.

### 7.2.2. Kỷ nguyên LLM: Đánh giá các Khả năng Nổi trội (Emergent Abilities)

Với sự ra đời của các LLM có khả năng few-shot, mô hình đánh giá đã thay đổi. Thay vì fine-tuning, các benchmark mới tập trung vào việc đo lường kiến thức và khả năng suy luận của mô hình trong cài đặt zero-shot và few-shot.

### MMLU: Bài kiểm tra Kiến thức Đa chuyên ngành

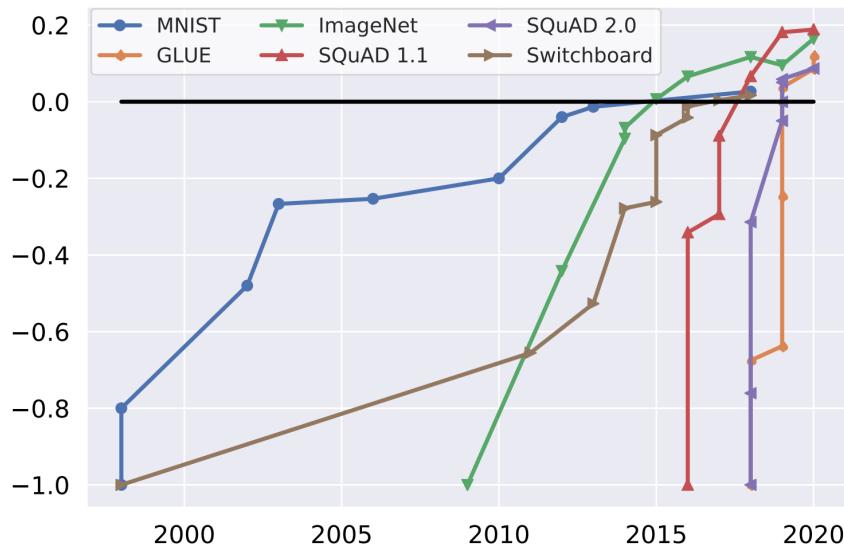
- **Tên đầy đủ:** Massive Multitask Language Understanding.
- **Triết lý:** Một LLM mạnh phải có kiến thức sâu rộng trên nhiều lĩnh vực chuyên môn, tương tự như các kỳ thi chuẩn hóa của con người.
- **Cấu trúc:** MMLU bao gồm gần 16,000 câu hỏi trắc nghiệm được lấy từ các kỳ thi thực tế, bao phủ 57 môn học khác nhau, từ các môn cơ bản (toán tiểu học) đến các môn chuyên ngành cấp độ đại học và chuyên nghiệp (luật, y, kỹ thuật).
- **Đánh giá:** Các mô hình được đánh giá ở chế độ **few-shot** (thường là 5-shot). Prompt sẽ bao gồm 5 câu hỏi và câu trả lời mẫu từ một môn học, sau đó là câu hỏi thực sự cần trả lời.
- **Tác động:** MMLU đã trở thành một trong những benchmark quan trọng nhất để đo lường kiến thức và khả năng suy luận của các LLM hàng đầu. Việc một mô hình đạt điểm cao trên MMLU là một minh chứng mạnh mẽ cho năng lực của nó.

### Big-Bench: Khám phá Giới hạn của LLMs

- **Tên đầy đủ:** Beyond the Imitation Game benchmark.
- **Triết lý:** Các benchmark hiện có vẫn còn hạn chế. Chúng ta cần một bộ sưu tập các tác vụ cực kỳ đa dạng và sáng tạo để thăm dò các khả năng và những điểm yếu bất ngờ của LLM.
- **Cấu trúc:** Một sự hợp tác khổng lồ của cộng đồng, tạo ra hơn 200 tác vụ rất đa dạng, bao gồm logic, lý luận toán học, hiểu biết vật lý trực quan, nhận biết định kiến xã hội, và nhiều tác vụ sáng tạo khác mà máy tính trước đây chưa từng làm tốt.
- **Đánh giá:** Chủ yếu ở chế độ **zero-shot**, để kiểm tra khả năng giải quyết vấn đề "ngay từ đầu" của mô hình.
- **Tác động:** Big-Bench đã giúp các nhà nghiên cứu xác định được các "khả năng nổi trội" (emergent abilities) – các khả năng chỉ xuất hiện khi mô hình đạt đến một quy mô đủ lớn.

### HumanEval: Đánh giá Năng lực Sinh mã nguồn

- **Dùng cho tác vụ nào?** Đánh giá khả năng viết và hoàn thiện mã nguồn (code generation) của LLM.
- **Cấu trúc:** Bao gồm 164 bài toán lập trình. Mỗi bài toán được đưa ra dưới dạng một docstring (mô tả hàm) và một vài test case.
- **Đánh giá:** Mô hình được yêu cầu sinh ra phần thân của hàm. Mã nguồn được sinh ra sau đó sẽ được thực thi và kiểm tra xem nó có vượt qua được các test case hay không. Metric chính là **Pass@k**, tức là xác suất một giải pháp đúng được tạo ra trong  $k$  lần thử.
- **Tác động:** HumanEval đã trở thành một tiêu chuẩn để đo lường năng lực lập trình của các mô hình như Codex, AlphaCode, và các LLM hiện đại khác.



Hình 7.2: Sơ đồ thời gian thể hiện sự phát triển của các benchmark

Việc chuyển từ các benchmark như GLUE sang MMLU/Big-Bench không chỉ là một sự thay đổi về quy mô, mà còn là một sự thay đổi cơ bản trong cách chúng ta hiểu và đo lường "trí thông minh" của các mô hình ngôn ngữ.

### 7.3. Thách thức trong Đánh giá LLM

Mặc dù chúng ta đã có một bộ các metric và benchmark ngày càng tinh vi, việc đánh giá các Mô hình Ngôn ngữ Lớn vẫn là một bài toán mở với vô vàn thách thức. Việc chỉ nhìn vào một con số duy nhất trên một bảng xếp hạng (leaderboard) có thể rất sai lầm và không phản ánh được hiệu năng thực sự của mô hình trong thế giới thực.

Mục này sẽ đi sâu vào những thách thức cố hữu và những cạm bẫy tiềm ẩn trong quá trình đánh giá LLM hiện nay.

#### 7.3.1. Nhiễm bẩn Dữ liệu (Data Contamination)

##### Vấn đề cốt lõi

"Chúng ta đang kiểm tra khả năng suy luận của mô hình, hay đang kiểm tra khả năng ghi nhớ của nó?"

**Định nghĩa** Nhiễm bẩn dữ liệu xảy ra khi dữ liệu của bộ **kiểm tra** (test set) hoặc bộ **phát triển** (dev set) của một benchmark vô tình xuất hiện trong bộ **dữ liệu huấn luyện** (training set) của mô hình.

##### Nguyên nhân

- Dữ liệu Pre-training khổng lồ:** Các LLM được huấn luyện trên một phần lớn của Internet (ví dụ: Common Crawl). Rất có khả năng các bộ dữ liệu benchmark phổ biến (vốn cũng thường được đăng tải công khai trên các trang như GitHub, arXiv) đã nằm sẵn trong kho dữ liệu pre-training khổng lồ này.
- Sự trùng lặp không chủ ý:** Ngay cả khi không có sự trùng lặp trực tiếp, các câu hỏi và câu trả lời trong benchmark có thể có các phiên bản diễn giải lại (paraphrased versions) hoặc các thảo luận về chúng nằm trên các trang web như Stack Overflow hay Reddit, và mô hình đã học được chúng.

### Hậu quả

- Điểm số bị thổi phồng một cách giả tạo:** Khi mô hình gặp một câu hỏi trong benchmark mà nó đã "nhìn thấy" trong quá trình huấn luyện, nó không cần phải suy luận. Nó chỉ đơn giản là "nhớ" lại và đưa ra câu trả lời. Điều này làm cho điểm số của mô hình cao hơn nhiều so với năng lực thực sự của nó.
- Mất đi tính giá trị của benchmark:** Nếu một benchmark bị nhiễm bẩn nặng, nó không còn là một thước đo đáng tin cậy cho khả năng tổng quát hóa nữa.

**Hướng giải quyết** Đây là một vấn đề rất khó giải quyết triệt để. Các hướng tiếp cận bao gồm:

- Lọc dữ liệu cẩn thận:** Cố gắng loại bỏ các dữ liệu từ các benchmark đã biết khỏi tập pre-training.
- Phát triển các benchmark mới liên tục:** Tạo ra các bộ dữ liệu mới, được giữ kín cho đến khi công bố.
- Các phương pháp phát hiện nhiễm bẩn:** Phát triển các kỹ thuật để ước tính mức độ một mô hình đã "học thuộc lòng" một benchmark.

### 7.3.2. Đánh giá bởi Con người vs. Đánh giá Tự động (Human vs. Auto Evaluation)

#### Hạn chế của Đánh giá Tự động (Auto-Eval)

Như đã thảo luận ở mục 7.1, các metric tự động như BLEU, ROUGE có những hạn chế nghiêm trọng:

- Hời hợt:** Chúng chỉ dựa trên sự trùng lặp bề mặt của từ hoặc N-gram.
- Không hiểu ngữ nghĩa:** Chúng không thể đánh giá sự sáng tạo, tính mạch lạc, hay sự chính xác về mặt factual. Một câu trả lời có thể sai hoàn toàn về mặt thông tin nhưng vẫn đạt điểm ROUGE cao nếu nó sử dụng lại các từ trong tài liệu tham khảo.
- Không đo lường được các khía cạnh tinh tế:** Không thể đo lường được "tính hữu ích", "sự thú vị", hay "mức độ an toàn" của một câu trả lời.

#### Ưu điểm và Thách thức của Đánh giá bởi Con người (Human-Eval)

- Tiêu chuẩn vàng:** Đánh giá của con người là tiêu chuẩn vàng vì con người có thể hiểu được các khía cạnh tinh tế mà các metric tự động bỏ qua.
- Thách thức:**
  - Tốn kém và chậm:** Cần phải thuê và huấn luyện một đội ngũ những người đánh giá chuyên nghiệp.
  - Thiểu nhất quán:** Nhiều người đánh giá khác nhau có thể có những sở thích và tiêu chuẩn khác nhau, dẫn đến sự không nhất quán trong kết quả (low inter-annotator agreement).
  - Thiên vị (Bias):** Người đánh giá có thể bị ảnh hưởng bởi các yếu tố như độ dài của câu trả lời (thiên vị cho câu trả lời dài hơn) hoặc văn phong của mô hình.

Do đó, thực tế tốt nhất thường là sử dụng kết hợp cả hai: dùng các metric tự động để theo dõi nhanh trong quá trình phát triển và dùng đánh giá của con người một cách định kỳ cho các mốc quan trọng.

### 7.3.3. Định luật Goodhart và "Dạy cho Bài kiểm tra"(Goodhart's Law & "Teaching to the Test")

#### Định luật Goodhart

"When a measure becomes a target, it ceases to be a good measure."

(Khi một thước đo trở thành một mục tiêu, nó sẽ không còn là một thước đo tốt nữa.)

### Ý nghĩa trong bối cảnh LLM

- Khi một benchmark cụ thể (ví dụ: MMLU) trở nên quá quan trọng và trở thành mục tiêu tối thượng của các phòng thí nghiệm AI, các nhà phát triển sẽ bắt đầu **tối ưu hóa mô hình một cách trực tiếp cho benchmark đó (over-optimizing)**.
- Họ có thể vô tình hoặc cố ý đưa các dữ liệu có văn phong tương tự như benchmark vào quá trình huấn luyện, hoặc tinh chỉnh các siêu tham số của mô hình để nó hoạt động tốt nhất trên một phân phối dữ liệu rất hẹp của benchmark đó.
- Hậu quả:** Mô hình đạt được điểm số rất cao trên benchmark, nhưng sự cải thiện này không thực sự chuyển thành sự cải thiện về khả năng suy luận tổng quát trong các tình huống thực tế. Mô hình chỉ đơn giản là đã học cách "làm bài kiểm tra" rất giỏi.

Điều này tạo ra một cuộc chạy đua vũ trang, nơi các bảng xếp hạng có thể không còn phản ánh đúng sự tiến bộ thực sự của ngành, và chúng ta cần phải liên tục phát triển các phương pháp đánh giá mới và đa dạng hơn để tránh rơi vào cái bẫy của Định luật Goodhart.

Những thách thức này cho thấy rằng việc đánh giá LLM không chỉ là một vấn đề kỹ thuật, mà còn là một vấn đề khoa học sâu sắc, đòi hỏi sự cẩn trọng, tư duy phản biện, và một cách tiếp cận đa diện.

## 7.4. Đánh giá dựa trên LLM (LLM-as-a-Judge)

Như chúng ta đã thấy, việc đánh giá các kết quả đầu ra của LLM, đặc biệt là trong các tác vụ mở như đối thoại hay tóm tắt sáng tạo, là một thách thức lớn. Các metric tự động (BLEU, ROUGE) thì quá hời hợt, trong khi đánh giá bởi con người lại quá tốn kém và chậm chạp.

Trong bối cảnh đó, một hướng tiếp cận mới đã nổi lên, tận dụng chính sức mạnh của các mô hình ngôn ngữ lớn: **sử dụng một LLM mạnh mẽ để đóng vai một "giám khảo"(judge) và đánh giá các câu trả lời do các mô hình khác tạo ra**.

### 7.4.1. Nguyên lý và Cách tiếp cận

#### Trực giác cốt lõi

Nếu một LLM đủ mạnh (như GPT-4) có khả năng hiểu các chỉ dẫn phức tạp và tạo ra các câu trả lời chất lượng cao, thì nó cũng có khả năng hiểu các **tiêu chí đánh giá (evaluation criteria)** phức tạp và áp dụng chúng để chấm điểm hoặc so sánh các câu trả lời. Về cơ bản, chúng ta đang biến bài toán đánh giá thành một bài toán Hỏi-dáp hoặc Suy luận khác cho một LLM giám khảo.

Có hai cách tiếp cận chính để sử dụng LLM-as-a-Judge:

#### Đánh giá dựa trên Điểm số (Score-based Evaluation)

- Cơ chế:** LLM giám khảo được cung cấp một bộ các tiêu chí đánh giá (ví dụ: độ chính xác, sự mạch lạc, tính hữu ích), một thang điểm (ví dụ: từ 1 đến 10), cùng với prompt và câu trả lời cần đánh giá. Sau đó, nó được yêu cầu đưa ra một điểm số cho mỗi tiêu chí và một lời giải thích cho điểm số đó.
- Ví dụ Prompt cho Giám khảo:**

Bạn là một giám khảo AI công tâm. Hãy đánh giá câu trả lời sau đây dựa trên các tiêu chí: Mạch lạc, Trung thực, Hữu ích, trên thang điểm từ 1 đến 10. Hãy đưa ra lý do cho điểm số của bạn.

**Câu hỏi:** "Trình bày những lợi ích chính của Học Tăng cường từ Phản hồi Con người (RLHF)."

**Câu trả lời cần đánh giá:** "RLHF là một phương pháp giúp mô hình tốt hơn. Nó sử dụng con người để chấm điểm và làm cho AI an toàn hơn."

**Đánh giá của bạn (duới dạng JSON):**

- Đầu ra mong muốn của Giám khảo:

```
{
  "scores": {
    "coherence": 8,
    "honesty": 9,
    "helpfulness": 5
  },
  "reasoning": "Câu trả lời mạch lạc và trung thực nhưng ..."
}
```

### Dánh giá dựa trên So sánh Cặp (Pairwise Comparison)

- Cơ chế:** Thay vì cho điểm một câu trả lời một cách độc lập (việc này rất khó và không nhất quán), LLM giám khảo được cung cấp cùng một prompt và hai câu trả lời khác nhau (ví dụ: từ Model A và Model B). Sau đó, nó được yêu cầu cho biết câu trả lời nào tốt hơn và tại sao.
- Lợi ích:** Giống như con người, việc so sánh hai đối tượng và chọn ra cái tốt hơn thường dễ dàng và đáng tin cậy hơn nhiều so với việc cho một điểm số tuyệt đối.
- Ví dụ Prompt cho Giám khảo:**

Bạn là một giám khảo AI công tâm. Dưới đây là một câu hỏi và hai câu trả lời từ hai trợ lý AI khác nhau. Hãy so sánh chúng và quyết định câu trả lời nào tốt hơn. Hãy giải thích chi tiết cho lựa chọn của bạn.

**Câu hỏi:** [Câu hỏi gốc...]

[Bắt đầu Câu trả lời A] [Nội dung câu trả lời của Model A...] [Kết thúc Câu trả lời A]

[Bắt đầu Câu trả lời B] [Nội dung câu trả lời của Model B...] [Kết thúc Câu trả lời B]

**Phân tích và Quyết định của bạn:**

- Kết quả:** Sau khi thực hiện nhiều so sánh cặp, chúng ta có thể sử dụng các hệ thống xếp hạng như Elo để tính ra một bảng xếp hạng tổng thể cho các mô hình. Các bảng xếp hạng LLM lớn như **Chatbot Arena Leaderboard** hoạt động dựa trên nguyên tắc này (nhưng với sự so sánh của con người).

#### 7.4.2. Ứng dụng và Tiềm năng

- Tăng tốc Chu trình Đánh giá:** LLM-as-a-Judge nhanh hơn và rẻ hơn đánh giá của con người hàng nghìn lần, cho phép các nhà phát triển nhanh chóng lặp lại và thử nghiệm các ý tưởng mới.
- Tự động hóa việc tạo Dữ liệu Sở thích:** Như đã thấy trong Constitutional AI (mục 5.5.3), LLM có thể được dùng để tự động tạo ra các cặp dữ liệu sở thích (câu trả lời tốt hơn > câu trả lời kém hơn), sau đó dữ liệu này có thể được dùng để tinh chỉnh mô hình bằng các phương pháp như DPO.

- **Đánh giá các khía cạnh khó đo lường:** Cho phép đánh giá các thuộc tính như "sự sáng tạo", "sự đồng cảm", hay "tính an toàn" mà các metric tự động truyền thống không thể chạm tới.

#### 7.4.3. Thách thức và Những điều cần Lưu ý

Việc sử dụng LLM-as-a-Judge không phải là một viên đạn bạc và đi kèm với những thách thức riêng.

**Thiên vị của Giám khảo (Judge's Bias)** LLM giám khảo cũng có những thiên vị của riêng nó, có thể ảnh hưởng đến kết quả đánh giá:

- **Thiên vị về vị trí (Position Bias):** Có xu hướng ưu tiên câu trả lời xuất hiện ở vị trí đầu tiên.
- **Thiên vị về sự dài dòng (Verbosity Bias):** Có xu hướng cho điểm cao hơn cho các câu trả lời dài hơn và chi tiết hơn, ngay cả khi chúng không chính xác hơn.
- **Thiên vị về văn phong (Style Bias):** Có xu hướng ưa thích các câu trả lời có văn phong giống với văn phong của chính nó. Điều này đặc biệt có vấn đề khi một mô hình (ví dụ: GPT-4) được dùng để đánh giá chính nó hoặc các biến thể của nó.

**Hạn chế về Kiến thức và Suy luận** Nếu một bài toán đòi hỏi kiến thức chuyên môn sâu hoặc khả năng suy luận phức tạp (ví dụ: một bài toán toán học khó), LLM giám khảo có thể không đủ năng lực để xác định câu trả lời nào là đúng. Nó có thể bị "đánh lừa" bởi một câu trả lời sai nhưng được trình bày một cách rất thuyết phục và có cấu trúc.

**Sự cần thiết của việc Hiệu chỉnh (Calibration)** Các nghiên cứu đã chỉ ra rằng điểm số từ LLM-as-a-Judge thường có độ tương quan cao với đánh giá của con người, nhưng không phải là hoàn hảo. Việc sử dụng các kỹ thuật như prompting cẩn thận, cung cấp các ví dụ minh họa (few-shot examples of good evaluation), và tổng hợp kết quả từ nhiều lần chạy là rất quan trọng để có được kết quả đáng tin cậy.

Mặc dù có những thách thức, LLM-as-a-Judge đã trở thành một công cụ không thể thiếu trong việc phát triển và đánh giá các mô hình ngôn ngữ thế hệ mới, cung cấp một giải pháp cân bằng giữa tốc độ của các metric tự động và chiều sâu của đánh giá bởi con người.

---

## KẾT THÚC CHƯƠNG 7

Trong chương cuối cùng của hành trình lý thuyết này, chúng ta đã giải quyết một trong những câu hỏi quan trọng nhất: "Làm thế nào để đo lường sự tiến bộ?". Chúng ta đã đi từ các metric kinh điển như F1-score và BLEU, khám phá các "kỳ thi" tiêu chuẩn như GLUE và MMLU, và quan trọng hơn, đã phát triển một tư duy phản biện về những thách thức sâu sắc trong việc đánh giá, từ niềm bẩn dũi liệu đến Định luật Goodhart. Việc hiểu rõ cách đánh giá và những hạn chế của nó cũng quan trọng như việc xây dựng mô hình. Đây chính là chiếc lá bàn giúp chúng ta định hướng trong quá trình phát triển, phân biệt giữa tiến bộ thực sự và những con số được tối ưu hóa quá mức. Với nền tảng lý thuyết đã hoàn thiện, chúng ta đã sẵn sàng để bước sang một giai đoạn mới.

# KẾT THÚC PHẦN 1

---

*Phần 1 của giáo trình đã cung cấp cho bạn một nền tảng lý thuyết toàn diện, từ những nguyên lý ngôn ngữ học cơ bản, các mô hình thống kê, các kiến trúc nơ-ron kinh điển, cho đến sự thống trị của Transformer và các kỹ thuật tiên tiến để tinh chỉnh, căn chỉnh, và đánh giá các Mô hình Ngôn ngữ Lớn. Với nền tảng này, bạn đã sẵn sàng để bước sang Phần 2, nơi chúng ta sẽ biến những lý thuyết này thành các kỹ năng và quy trình thực chiến để xây dựng các ứng dụng NLP thực tế.*

---

## Tóm lược nội dung Phần 1:

- **Chương 1 – Nhập môn và Nguyên lý nền tảng:** Định nghĩa NLP, mối quan hệ với AI, các kỹ nguyên phát triển, kiến thức ngôn ngữ học, toán học, và các vấn đề đạo đức trong NLP.
- **Chương 2 – Biểu diễn văn bản và Mô hình thống kê:** Các kỹ thuật BoW, TF-IDF, N-gram, smoothing, mô hình phân loại, học chuỗi, và LDA.
- **Chương 3 – Các kiến trúc mạng nơ-ron kinh điển:** Word embeddings (Word2Vec, GloVe, FastText), autoencoder, RNN/LSTM/GRU, seq2seq với attention, CNN cho NLP, contextualized embeddings, và các mô hình sinh pre-Transformer.
- **Chương 4 – Kỹ nguyên Transformer và LLMs:** Kiến trúc Transformer gốc, phân loại LLMs, tokenization hiện đại, biến thể cho ngữ cảnh dài, MoE.
- **Chương 5 – Tinh chỉnh và Căn chỉnh mô hình:** Fine-tuning, PEFT (LoRA, QLoRA, Adapter), prompt engineering, instruction tuning, RLHF và các kỹ thuật alignment khác.
- **Chương 6 – Hệ thống và Kiến trúc nâng cao:** RAG, tối ưu hóa mô hình (distillation, quantization, pruning), mô hình đa phương thức, hệ thống tác tử, world models, và biểu diễn nâng cao cho truy xuất.
- **Chương 7 – Đánh giá và Benchmark:** Các metric kinh điển (perplexity, F1, BLEU, ROUGE), benchmark (GLUE, SuperGLUE, đánh giá emergent abilities), thách thức trong đánh giá, và xu hướng “LLM-as-a-Judge”.



## **Phần II**

# **CẨM NANG KỸ THUẬT NLP THỰC CHIẾN**



---

*Chào mừng bạn đến với Phần 2: Cẩm nang kỹ thuật NLP thực chiến. Mục tiêu của Phần này là trang bị cho bạn những kỹ năng, công cụ và quy trình cần thiết để xây dựng, đánh giá và triển khai các ứng dụng NLP trong thực tế. Nếu Phần 1 tập trung vào việc trả lời “tại sao” và “như thế nào” của lý thuyết, thì Phần 2 sẽ hướng dẫn bạn từng bước thực hành: từ thu thập và xử lý dữ liệu, huấn luyện và đánh giá mô hình, tối ưu hóa và triển khai, cho đến việc áp dụng các công thức (recipes) để giải quyết các bài toán phổ biến. Mỗi chương là một hướng dẫn chi tiết, đi từ cơ bản đến nâng cao, giúp bạn hiểu rõ quy trình thực tế và tự tin áp dụng các kỹ thuật NLP trong các dự án thật sự. Hãy sẵn sàng để chuyển từ lý thuyết sang thực chiến.*



# Chương 1

## QUY TRÌNH LÀM VIỆC VÀ CÔNG CỤ XỬ LÝ DỮ LIỆU

Nếu như các chương trước đã trang bị cho bạn kho "vũ khí" lý thuyết, thì chương này sẽ dạy bạn cách trở thành một "chiến binh" thực thụ trên chiến trường NLP. Một mô hình tinh vi nhất cũng sẽ trở nên vô dụng nếu không có dữ liệu chất lượng cao để học hỏi và một quy trình làm việc bài bản để định hướng.

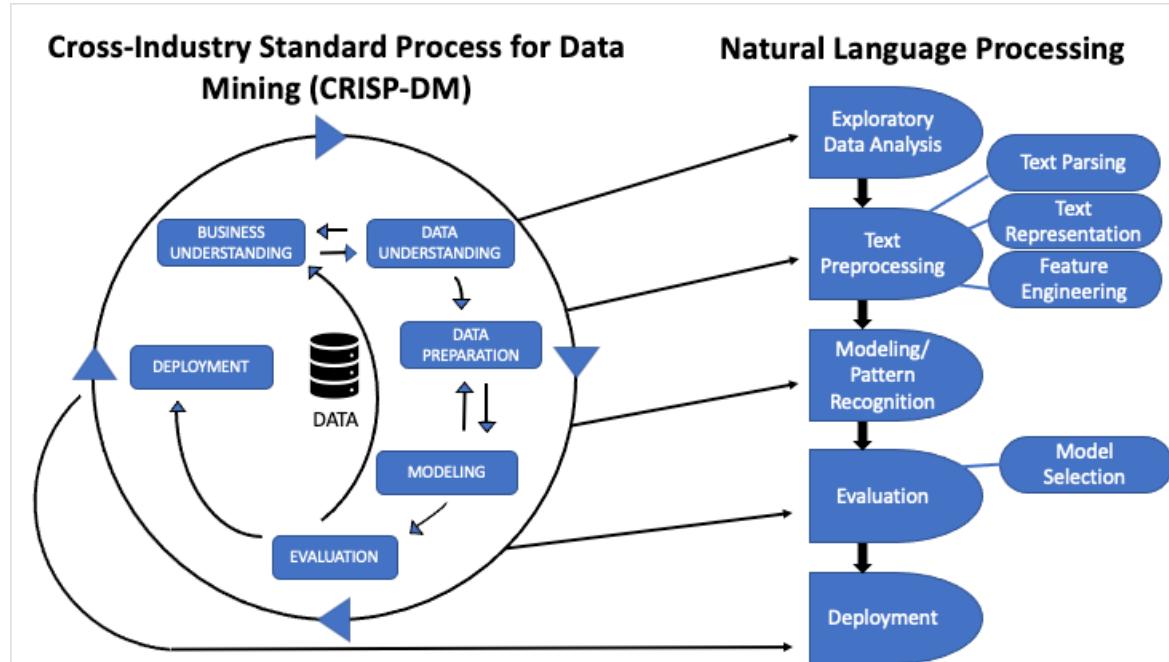
Trong chương mở đầu của Phần 2, chúng ta sẽ tập trung vào nền móng của mọi dự án thành công: **Dữ liệu và Quy trình**. Chúng ta sẽ phác thảo một vòng đời dự án NLP tinh gọn, từ khâu lên ý tưởng, thiết lập baseline, cho đến khi triển khai và giám sát. Quan trọng hơn, chúng ta sẽ đi sâu vào các kỹ thuật và công cụ thiết yếu để thu thập, làm sạch, gán nhãn, và quản lý phiên bản dữ liệu – những công việc chiếm đến 80% thời gian của một dự án thực tế. Nắm vững những kỹ năng này là bước đi đầu tiên và quan trọng nhất để biến lý thuyết thành những sản phẩm có giá trị.

### 1.1. Phác thảo Quy trình Dự án NLP Tinh gọn

Chào mừng bạn đến với phần thực chiến của giáo trình. Trước khi chúng ta lao vào việc viết code và huấn luyện các mô hình phức tạp, điều quan trọng nhất là phải có một "tấm bản đồ- một quy trình làm việc có hệ thống để dẫn dắt dự án từ một ý tưởng mơ hồ đến một sản phẩm hữu ích.

Việc thiếu một quy trình rõ ràng là một trong những nguyên nhân hàng đầu khiến các dự án học máy thất bại. Các đội nhóm có thể tốn hàng tháng trời để xây dựng một mô hình phức tạp chỉ để nhận ra rằng nó không giải quyết đúng bài toán kinh doanh, hoặc dữ liệu không đủ chất lượng.

Quy trình dưới đây được thiết kế theo một vòng lặp tinh gọn, nhấn mạnh vào việc tạo ra giá trị nhanh chóng, thu thập phản hồi, và lặp lại. Nó không phải là một thác nước cứng nhắc, mà là một chu trình linh hoạt.



Hình 1.1: Quy trình Dự án NLP Tinh gọn: Một chu trình lặp đi lặp lại từ việc xác định bài toán đến triển khai và giám sát.

### 1.1.1. Vòng lặp 0: Xác định Bài toán và Thiết lập Đường cơ sở (Baseline)

Đây là bước quan trọng nhất, quyết định sự thành bại của toàn bộ dự án.

#### 1. Hiểu rõ Mục tiêu (Problem Framing)

- Câu hỏi cần trả lời:** Chúng ta đang cố gắng giải quyết vấn đề kinh doanh/nghiên cứu nào? Tác động mong muốn là gì? (Ví dụ: "giảm 30% thời gian phản hồi của nhân viên hỗ trợ", "tự động phân loại 80% tin tức đầu vào").
- Xác định Input và Output:** Đầu vào của mô hình là gì (văn bản thô, bình luận, email)? Đầu ra mong muốn là gì (một nhãn, một đoạn văn bản tóm tắt, một câu trả lời)?
- Định hình bài toán NLP:** Ánh xạ vấn đề kinh doanh thành một bài toán NLP cụ thể. Ví dụ: "Giảm thời gian phản hồi" → "Xây dựng mô hình phân loại email để tự động định tuyến đến đúng phòng ban".

#### 2. Định nghĩa Metric Thành công

- Metric Máy học (Offline):** Chọn các metric kỹ thuật để đánh giá mô hình trong quá trình phát triển (ví dụ: F1-score, Accuracy, BLEU). Metric này phải phản ánh được mục tiêu kinh doanh.
- Metric Kinh doanh (Online):** Xác định cách bạn sẽ đo lường tác động thực tế của mô hình sau khi triển khai (ví dụ: tỷ lệ click, thời gian giải quyết ticket, mức độ hài lòng của khách hàng).

#### 3. Thiết lập một Đường cơ sở (Baseline) Đơn giản

Lời khuyên quan trọng: Bắt đầu một cách "ngu ngốc"

Đừng bao giờ bắt đầu một dự án bằng việc xây dựng một mô hình Transformer phức tạp. Hãy bắt đầu với giải pháp đơn giản nhất có thể. Một baseline tốt không chỉ cho bạn một điểm để so sánh, mà còn có thể đủ tốt để giải quyết 80% vấn đề với 20% công sức.

- Các ý tưởng cho Baseline:**

- **Không dùng ML:** Một hệ thống dựa trên từ khóa hoặc biểu thức chính quy (regex).
- **Mô hình Thống kê kinh điển:** Sử dụng ‘TF-IDF’ kết hợp với ‘Logistic Regression’ hoặc ‘Naive Bayes’. Các thư viện như ‘Scikit-learn’ giúp việc này trở nên cực kỳ nhanh chóng.
- **Mô hình Zero-shot:** Sử dụng API của một LLM lớn (như GPT-4) với một prompt zero-shot. Đây là một baseline rất mạnh mẽ trong kỷ nguyên hiện đại.

Baseline này sẽ là thước đo để bạn biết liệu những nỗ lực phức tạp hơn của mình có thực sự mang lại giá trị hay không.

## 1.1.2. Vòng lặp 1: Phát triển Mô hình (Model Development)

Đây là vòng lặp cốt lõi của việc xây dựng và cải tiến mô hình.

### 4. Thu thập và Chuẩn bị Dữ liệu

Đây thường là phần tốn nhiều thời gian nhất.

- **Thu thập (Collection):** Lấy dữ liệu từ các nguồn (cơ sở dữ liệu, API, web scraping).
- **Làm sạch (Cleaning):** Xử lý các vấn đề như mã hóa ký tự, loại bỏ các thẻ HTML, chuẩn hóa văn bản.
- **Gán nhãn (Labeling):** Nếu là bài toán có giám sát, đây là bước cực kỳ quan trọng. Sử dụng các công cụ như ‘Label Studio’ hoặc các kỹ thuật gán nhãn yếu (weak supervision).
- **Phân chia Dữ liệu (Splitting):** Chia dữ liệu thành các tập Huấn luyện (Train), Đánh giá (Validation/Dev), và Kiểm tra (Test). **Không bao giờ được** “nhìn” vào tập Test trong quá trình phát triển.

### 5. Huấn luyện Mô hình

- **Lựa chọn Mô hình:** Dựa trên baseline và yêu cầu của bài toán, chọn một kiến trúc phù hợp (ví dụ: một mô hình từ Hugging Face Hub).
- **Huấn luyện/Fine-tuning:** Thực hiện quá trình huấn luyện.
- **Theo dõi Thí nghiệm (Experiment Tracking):** Sử dụng các công cụ như ‘Weights & Biases’ hoặc ‘MLflow’ để ghi lại mọi thứ: phiên bản code, siêu tham số, metric, và các mô hình đã được huấn luyện. Việc này là tối quan trọng để đảm bảo tính tái lập (reproducibility) và so sánh các thử nghiệm.

### 6. Phân tích Lỗi (Error Analysis)

Đây là bước mà nhiều người bỏ qua, nhưng nó lại là chìa khóa để cải tiến mô hình một cách thông minh.

- **Mục tiêu:** Không chỉ nhìn vào con số metric tổng thể (ví dụ: F1 = 85%), mà phải hiểu tại sao mô hình lại sai ở 15% còn lại.
- **Quy trình:**
  1. Lấy một mẫu các dự đoán sai trên tập validation.
  2. Phân loại các lỗi này thành các nhóm. Ví dụ: “mô hình sai ở các câu bị phủ định”, “mô hình nhầm lẫn giữa hai lớp A và B”, “lỗi do dữ liệu gán nhãn sai”.
  3. **Ưu tiên:** Tập trung nỗ lực vào việc giải quyết nhóm lỗi lớn nhất và có tác động nhất. Ví dụ, nếu mô hình thường sai ở các câu phủ định, bạn có thể cần thu thập thêm dữ liệu về loại câu này hoặc sử dụng kỹ thuật tăng cường dữ liệu.

Vòng lặp (Huấn luyện → Phân tích lỗi → Cải thiện dữ liệu/mô hình) được lặp lại cho đến khi mô hình đạt được hiệu năng mong muốn trên tập validation.

## 1.1.3. Vòng lặp 2: Triển khai và Giám sát (Deployment & Monitoring)

Một mô hình chỉ thực sự tạo ra giá trị khi nó được đưa vào sử dụng.

## 7. Đóng gói và Triển khai (Deployment)

- **Tối ưu hóa:** Áp dụng các kỹ thuật như lượng tử hóa (quantization) hoặc chưng cất (distillation) để làm cho mô hình nhỏ hơn và nhanh hơn.
- **Đóng gói (Packaging):** Đóng gói mô hình và các thành phần phụ thuộc của nó vào một định dạng có thể triển khai, ví dụ như một container Docker.
- **Phục vụ (Serving):** Triển khai mô hình như một API endpoint (sử dụng các framework như 'FastAPI' hoặc 'BentoML') để các ứng dụng khác có thể gọi đến.

## 8. Giám sát và Bảo trì (Monitoring & Maintenance)

Công việc không kết thúc sau khi triển khai.

- **Giám sát Hiệu năng:** Theo dõi các metric của mô hình trong môi trường thực tế.
- **Phát hiện Trôi dạt Dữ liệu (Data Drift):** Thế giới thực luôn thay đổi. Cách người dùng viết và các chủ đề họ quan tâm có thể thay đổi theo thời gian. Cần có cơ chế để phát hiện khi nào phân phối dữ liệu đầu vào thực tế bắt đầu khác biệt so với dữ liệu huấn luyện.
- **Thu thập Phản hồi và Huấn luyện lại:** Thiết lập một vòng lặp để thu thập các dự đoán sai từ môi trường production, gán nhãn lại chúng, và sử dụng chúng để định kỳ huấn luyện lại (re-train) mô hình afin d'améliorer sa performance au fil du temps.

Quy trình tinh gọn này đảm bảo rằng bạn luôn tập trung vào việc tạo ra giá trị, học hỏi từ dữ liệu và các lỗi sai, và xây dựng các hệ thống NLP mạnh mẽ và bền vững.

## 1.2. Kỹ thuật Thu thập Dữ liệu

Dữ liệu là nguồn sống của các mô hình học máy. Trước khi có thể làm sạch hay gán nhãn, chúng ta phải có được dữ liệu thô. Trong nhiều trường hợp, dữ liệu không có sẵn trong một cơ sở dữ liệu gọn gàng, mà nằm rải rác trên Internet. Mục này sẽ giới thiệu các công cụ và kỹ thuật phổ biến để thu thập dữ liệu văn bản từ các nguồn trực tuyến.

### 1.2.1. Sử dụng APIs: Cách tiếp cận "Lịch sự" và Đáng tin cậy

- **API là gì?** Giao diện Lập trình Ứng dụng (Application Programming Interface - API) là một tập hợp các quy tắc và giao thức cho phép các ứng dụng phần mềm khác nhau "nói chuyện" với nhau. Rất nhiều nền tảng lớn (Twitter, Reddit, Wikipedia, các trang tin tức) cung cấp API để các nhà phát triển có thể truy cập dữ liệu của họ một cách có cấu trúc và được cho phép.
- **Tại sao nên ưu tiên?**
  - **Có cấu trúc:** Dữ liệu trả về thường ở định dạng chuẩn như JSON, rất dễ phân tích.
  - **Đáng tin cậy:** Ít bị ảnh hưởng bởi những thay đổi về giao diện của trang web.
  - **"Lịch sự":** Bạn đang tuân thủ các quy tắc do nhà cung cấp dịch vụ đặt ra, tránh được các vấn đề pháp lý và kỹ thuật.
- **Công cụ:** Thư viện 'requests' của Python là công cụ tiêu chuẩn để thực hiện các yêu cầu HTTP đến các API.

Ví dụ 15: Lấy dữ liệu từ một API đơn giản với 'requests'

Giả sử chúng ta muốn lấy một sự thật ngẫu nhiên về loài mèo từ API 'catfact.ninja'.  
baselinestretch

```
1 import requests
2 import json
3
4 # Địa chỉ của API endpoint
5 url = "https://catfact.ninja/fact"
```

```

6
7     try:
8         # Gửi một yêu cầu GET đến API
9         response = requests.get(url)
10
11        # Kiểm tra xem yêu cầu có thành công không (status code 200)
12        response.raise_for_status()
13
14        # Phân tích dữ liệu JSON trả về
15        data = response.json()
16        fact = data['fact']
17
18        print(f"Sự thật về loài mèo: {fact}")
19
20    except requests.exceptions.RequestException as e:
21        print(f'Lỗi khi gọi API: {e}')
22

```

**Lưu ý quan trọng:** Hầu hết các API thương mại sẽ yêu cầu bạn đăng ký để nhận một "API key" (khóa xác thực) và có thể giới hạn số lượng yêu cầu bạn có thể thực hiện trong một khoảng thời gian (rate limiting).

### 1.2.2. Web Scraping: Trích xuất Dữ liệu từ HTML

Khi không có API, chúng ta phải dùng đến kỹ thuật **web scraping** (cào dữ liệu web) - quá trình tự động tải về và trích xuất thông tin từ các trang web.

#### Cảnh báo về Web Scraping

Web scraping là một công cụ mạnh mẽ nhưng cần được sử dụng một cách có trách nhiệm. Trước khi cào dữ liệu từ một trang web, hãy luôn:

1. **Kiểm tra file 'robots.txt':** Truy cập '[tên\_trang\_web]/robots.txt' để xem các quy tắc của trang web về việc các bot tự động có được phép truy cập vào các phần nào của trang.
2. **Tôn trọng Điều khoản Dịch vụ (Terms of Service):** Đọc và tuân thủ các quy định của trang web.
3. **Tránh làm quá tải máy chủ:** Gửi các yêu cầu một cách từ từ, có độ trễ giữa các lần gọi, và tránh gửi quá nhiều yêu cầu trong một thời gian ngắn.

#### BeautifulSoup: Phân tích HTML một cách dễ dàng

'BeautifulSoup' là một thư viện Python giúp việc phân tích cú pháp (parsing) các tài liệu HTML và XML trở nên cực kỳ đơn giản. Nó không tự tải trang web, mà hoạt động trên nội dung HTML mà bạn cung cấp (thường là từ thư viện 'requests').

- **Cơ chế hoạt động:** Nó biến một tài liệu HTML phức tạp thành một cây các đối tượng Python, cho phép bạn điều hướng và tìm kiếm các phần tử HTML dựa trên tên thẻ (tag), thuộc tính (attributes) như 'id' hoặc 'class'.

#### Ví dụ 16: Trích xuất tiêu đề bài viết với 'requests' và 'BeautifulSoup'

baselinestretch

```

1 import requests
2 from bs4 import BeautifulSoup
3
4 # URL của trang web cần cào
5 url =
6   ↪ "https://en.wikipedia.org/wiki/Natural_language_processing"
7
8 try:
9     # 1. Tải nội dung HTML của trang
10    response = requests.get(url)
11    response.raise_for_status()
12
13    # 2. Tạo một đối tượng BeautifulSoup để phân tích HTML
14    soup = BeautifulSoup(response.content, 'html.parser')
15
16    # 3. Tìm phần tử HTML chứa tiêu đề
17    # Bằng cách "Inspect Element" trên trình duyệt, ta thấy tiêu
18    # đề
19    # nằm trong một thẻ <h1> với class "firstHeading".
20    title_element = soup.find('h1', class_='firstHeading')
21
22    # 4. Trích xuất nội dung văn bản từ phần tử
23    if title_element:
24        title_text = title_element.get_text()
25        print(f"Tiêu đề của trang là: {title_text}")
26    else:
27        print("Không tìm thấy tiêu đề.")
28
29 except requests.exceptions.RequestException as e:
30     print(f"Lỗi khi tải trang: {e}")

```

'BeautifulSoup' rất phù hợp cho các tác vụ cào dữ liệu quy mô nhỏ và các dự án học tập.

### Scrapy: Một Framework Cào dữ liệu Toàn diện

Khi bạn cần cào dữ liệu từ hàng nghìn hoặc hàng triệu trang web, một kịch bản đơn giản với 'requests' và 'BeautifulSoup' là không đủ. 'Scrapy' là một framework (khung làm việc) hoàn chỉnh cho việc cào dữ liệu quy mô lớn.

- **Kiến trúc bất đồng bộ (Asynchronous):** Scrapy được xây dựng trên 'Twisted', một thư viện mạng bất đồng bộ, cho phép nó xử lý hàng nghìn yêu cầu cùng lúc mà không cần đợi yêu cầu trước hoàn thành. Điều này làm cho nó cực kỳ nhanh.
- **Các thành phần chính:** Một dự án Scrapy bao gồm:
  - **Spiders (Nhện):** Là các lớp Python bạn viết để định nghĩa cách cào một trang web cụ thể: bắt đầu từ URL nào, đi theo các liên kết nào, và trích xuất dữ liệu gì.
  - **Items:** Định nghĩa cấu trúc của dữ liệu bạn muốn trích xuất (ví dụ: một item "Bài viết" có các trường "tiêu đề", "tác giả", "nội dung").
  - **Pipelines:** Xử lý các item sau khi chúng được trích xuất (ví dụ: làm sạch dữ liệu, lưu vào cơ sở dữ liệu, loại bỏ các mục trùng lặp).
  - **Middlewares:** Can thiệp vào quá trình gửi yêu cầu và nhận phản hồi (ví dụ: để xoay vòng proxy, thay đổi user-agent).
- **Khi nào nên dùng Scrapy?** Khi bạn cần xây dựng một hệ thống cào dữ liệu mạnh mẽ, có khả năng mở rộng, và cần xử lý một khối lượng lớn các trang web một cách hiệu quả.

Việc học Scrapy đòi hỏi nhiều công sức hơn, nhưng nó là một công cụ chuẩn công nghiệp cho các

tác vụ thu thập dữ liệu quy mô lớn.

### 1.3. Làm sạch, Tiền xử lý và Gán nhãn Dữ liệu

Dữ liệu thô thu thập được từ thế giới thực thường rất "bẩn": đầy lỗi chính tả, các ký tự lạ, thẻ HTML, và thiếu cấu trúc. Giai đoạn làm sạch, tiền xử lý và gán nhãn là quá trình biến mở dữ liệu hỗn độn này thành một tài sản quý giá, sẵn sàng cho việc huấn luyện mô hình. Đây thường là giai đoạn tốn nhiều thời gian và công sức nhất, nhưng cũng là giai đoạn mang lại lợi tức đầu tư (ROI) cao nhất.

#### 1.3.1. Thao tác Dữ liệu quy mô lớn: Pandas, NumPy, và Polars

Khi làm việc với các bộ dữ liệu văn bản lớn (từ hàng trăm nghìn đến hàng triệu dòng), việc sử dụng các cấu trúc dữ liệu cơ bản của Python (như list hay dict) sẽ trở nên rất chậm chạp. Các thư viện tính toán hiệu năng cao là không thể thiếu.

**NumPy** Là thư viện nền tảng cho tính toán khoa học trong Python. Nó cung cấp đối tượng mảng đa chiều ‘ndarray’ mạnh mẽ, các phép toán tuyến tính và các hàm toán học hiệu quả. Mặc dù không trực tiếp xử lý văn bản, NumPy là “xương sống” của các thư viện khác như Pandas.

**Pandas** Là công cụ tiêu chuẩn trong khoa học dữ liệu để thao tác và phân tích dữ liệu dạng bảng.

- **Cấu trúc dữ liệu chính:** ‘DataFrame’, một cấu trúc dữ liệu 2 chiều giống như một bảng tính hoặc một bảng SQL, với các hàng và các cột có nhãn.
- **Điểm mạnh:** Cung cấp một API cực kỳ phong phú và linh hoạt để:
  - Đọc và ghi nhiều định dạng file (CSV, Excel, JSON, Parquet...).
  - Lọc, chọn, và truy vấn dữ liệu một cách mạnh mẽ.
  - Xử lý các giá trị bị thiếu (missing values).
  - Áp dụng các hàm tùy chỉnh lên từng hàng hoặc cột để làm sạch và tiền xử lý văn bản.
  - Gộp (merge) và nối (join) các bảng dữ liệu khác nhau.
- **Hạn chế:** Pandas hoạt động trên một lõi CPU duy nhất và tải toàn bộ dữ liệu vào bộ nhớ RAM. Với các bộ dữ liệu cực lớn (hàng chục gigabyte), nó có thể trở nên chậm hoặc gây lỗi tràn bộ nhớ.

**Polars: Một sự thay thế Hiện đại và Nhanh hơn** Polars là một thư viện DataFrame mới hơn, được viết bằng Rust, và được thiết kế để giải quyết các hạn chế của Pandas.

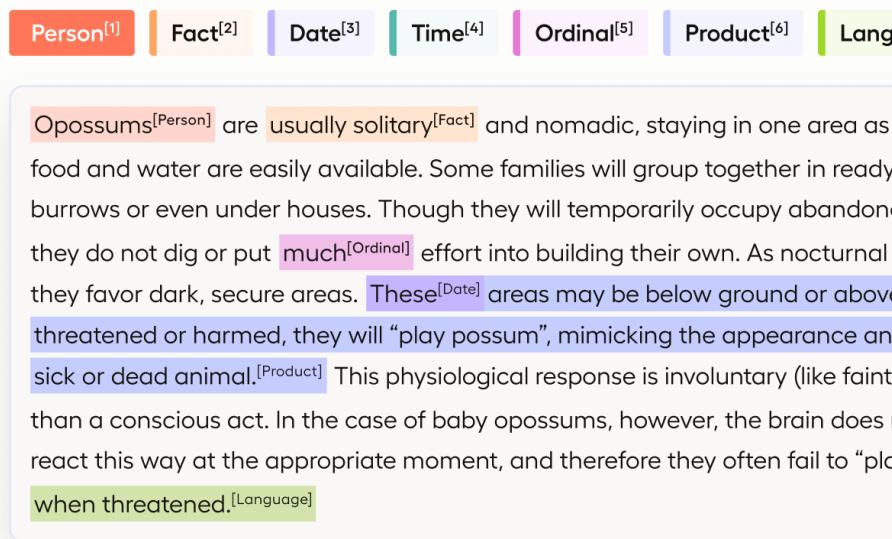
- **Điểm mạnh chính:**
  - **Thực thi song song (Parallel Execution):** Polars có thể tự động tận dụng tất cả các lõi CPU có sẵn trên máy của bạn, giúp các phép toán nhanh hơn đáng kể.
  - **Tối ưu hóa Truy vấn (Query Optimization):** Nó có một bộ tối ưu hóa truy vấn thông minh, có thể sắp xếp lại và kết hợp các phép toán để thực thi một cách hiệu quả nhất (lazy execution).
  - **Hiệu quả về bộ nhớ:** Được thiết kế để xử lý các bộ dữ liệu lớn hơn RAM của máy.
- **Khi nào nên dùng Polars?** Khi bạn làm việc với các bộ dữ liệu lớn và thấy rằng Pandas đang trở thành một “nút thắt cổ chai” về hiệu năng.

#### 1.3.2. Gán nhãn Dữ liệu: Từ Thủ công đến Tự động

Gán nhãn là quá trình thêm thông tin giám sát vào dữ liệu. Đây là một bước cực kỳ quan trọng và tốn kém cho các bài toán học có giám sát.

### Công cụ Gán nhãn Thủ công: Label Studio

- Mục đích:** Cung cấp một giao diện người dùng trực quan để con người có thể gán nhãn dữ liệu một cách hiệu quả và nhất quán.
- Label Studio là gì?** Là một công cụ gán nhãn dữ liệu mã nguồn mở và cực kỳ linh hoạt.
- Các tính năng chính:**
  - Hỗ trợ rất nhiều loại dữ liệu (văn bản, ảnh, âm thanh, video) và nhiều loại tác vụ NLP (phân loại, NER, trích xuất quan hệ).
  - Giao diện có thể tùy chỉnh cao.
  - Hỗ trợ làm việc nhóm, cho phép nhiều người cùng gán nhãn và quản lý chất lượng.
  - Tích hợp với các mô hình học máy để "gán nhãn bán tự động" (semi-automated labeling), nơi mô hình đưa ra gợi ý và con người chỉ cần sửa lại.



Hình 1.2: Giao diện gán nhãn Nhận dạng Thực thể Tên (NER) trong Label Studio.

### Gán nhãn có Lập trình: Snorkel và Weak Supervision

- Vấn đề:** Gán nhãn thủ công hàng triệu điểm dữ liệu là bất khả thi.
- Tư duy của Weak Supervision:** Thay vì gán nhãn từng điểm dữ liệu một, hãy để các chuyên gia lĩnh vực (Subject Matter Experts - SMEs) viết ra các **hàm gán nhãn (Labeling Functions - LFs)** – các quy tắc, heuristics, hoặc các mô hình đơn giản để gán nhãn dữ liệu một cách tự động nhưng có thể "nhiều".
- Snorkel là gì?** Là một framework mã nguồn mở để xây dựng các bộ dữ liệu huấn luyện bằng cách sử dụng weak supervision.
- Cơ chế hoạt động:**
  - Viết các Hàm gán nhãn (LFs):** Người dùng viết nhiều LFs. Ví dụ, để phát hiện email spam, các LFs có thể là: "Nếu email chứa từ 'khuyến mãi', gán nhãn SPAM", "Nếu email đến từ một địa chỉ trong danh bạ, gán nhãn NOT\_SPAM", "Sử dụng một mô hình Naive Bayes đơn giản để gán nhãn".
  - Mô hình Sinh (Generative Model):** Snorkel sẽ phân tích đầu ra của tất cả các LFs trên dữ liệu không nhãn. Nó sẽ học cách ước tính độ chính xác và sự tương quan giữa các LFs, ngay cả khi chúng mâu thuẫn với nhau.
  - Tạo nhãn xác suất:** Dựa trên những gì đã học, mô hình sinh sẽ kết hợp các tín hiệu yếu từ các LFs để tạo ra một nhãn xác suất (probabilistic label) duy nhất cho mỗi điểm dữ liệu.

4. **Huấn luyện Mô hình cuối:** Các nhãn xác suất này sau đó được sử dụng để huấn luyện một mô hình phân biệt mạnh mẽ cuối cùng (ví dụ: RoBERTa).
- **Lợi ích:** Cho phép tạo ra các bộ dữ liệu huấn luyện quy mô lớn một cách nhanh chóng bằng cách tận dụng kiến thức chuyên môn thay vì công sức gán nhãn thủ công.

### Sử dụng LLM để Sinh và Gán nhãn Dữ liệu

Đây là một mô hình mới nổi, tận dụng sức mạnh của các LLM lớn như GPT-4.

- **Few-shot Labeling:** Cung cấp cho LLM một chỉ dẫn rõ ràng và một vài ví dụ (few-shot examples) về cách gán nhãn. Sau đó, yêu cầu nó gán nhãn cho hàng nghìn điểm dữ liệu mới. Cách này nhanh và cho chất lượng đáng kinh ngạc.
- **Data Generation:** LLM không chỉ có thể gán nhãn, mà còn có thể **sinh ra** các mẫu dữ liệu hoàn toàn mới. Ví dụ, để huấn luyện một mô hình phát hiện ý định, bạn có thể yêu cầu LLM: "Hãy sinh ra 20 cách khác nhau mà một người dùng có thể hỏi về tình trạng đơn hàng".
- **Cảnh báo:** Dữ liệu do LLM tạo ra có thể thiếu sự đa dạng và chứa các thiên vị (biases) của chính LLM đó. Cần phải kiểm tra và lọc kỹ lưỡng.

### 1.3.3. Xử lý Dữ liệu Mất cân bằng (Handling Imbalanced Data)

- **Vấn đề:** Trong nhiều bài toán thực tế (phát hiện gian lận, chẩn đoán bệnh hiếm), số lượng mẫu của một lớp (lớp thiểu số - minority class) ít hơn rất nhiều so với các lớp khác (lớp đa số - majority class). Một mô hình được huấn luyện trên dữ liệu này có thể chỉ đơn giản là học cách "luôn dự đoán lớp đa số" để đạt độ chính xác cao, nhưng lại hoàn toàn vô dụng.
- **Các kỹ thuật xử lý:**
  - **Lấy mẫu lại (Resampling):**
    - \* **Under-sampling:** Xóa bỏ các mẫu từ lớp đa số. Có nguy cơ làm mất thông tin.
    - \* **Over-sampling:** Tạo ra các bản sao của các mẫu từ lớp thiểu số. Có nguy cơ gây quá khớp (overfitting).
    - \* **SMOTE (Synthetic Minority Over-sampling Technique):** Một kỹ thuật over-sampling thông minh hơn, tạo ra các mẫu "tổng hợp" mới cho lớp thiểu số bằng cách nội suy giữa các mẫu hiện có.
  - **Sử dụng Trọng số Lớp (Class Weights):** Trong quá trình huấn luyện, điều chỉnh hàm mất mát để "phạt" nặng hơn các lỗi mắc phải trên lớp thiểu số. Hầu hết các framework học máy đều hỗ trợ tham số 'class\_weight'.
  - **Thay đổi Metric Đánh giá:** Thay vì dùng 'Accuracy', hãy tập trung vào các metric như 'F1-score', 'Precision-Recall Curve (PRC)', hoặc 'AUC-ROC' vốn ít bị ảnh hưởng bởi sự mất cân bằng.

Việc lựa chọn kỹ thuật phù hợp phụ thuộc vào đặc điểm cụ thể của bộ dữ liệu và bài toán.

## 1.4. Kỹ thuật Tăng cường Dữ liệu (Data Augmentation)

### Vấn đề: "Còn sói" Dữ liệu

Các mô hình học sâu, đặc biệt là các mô hình Transformer lớn, rất "đói" dữ liệu. Khi được fine-tune trên một bộ dữ liệu nhỏ, chúng có nguy cơ cao bị **quá khớp (overfitting)** – tức là chúng "học thuộc lòng" các ví dụ trong tập huấn luyện thay vì học quy luật tổng quát, dẫn đến hiệu năng kém trên dữ liệu mới.

**Tăng cường Dữ liệu (Data Augmentation)** là quá trình tạo ra các mẫu dữ liệu huấn luyện mới, "giả" nhưng hợp lý, từ các mẫu dữ liệu hiện có. Mục tiêu là làm tăng kích thước và sự đa dạng của bộ dữ liệu huấn luyện, giúp mô hình trở nên mạnh mẽ hơn (more robust) và có khả năng tổng quát hóa tốt hơn.

Điều quan trọng là các phép biến đổi phải **giữ nguyên nhãn** (label-preserving). Ví dụ, nếu bạn thay đổi một câu có cảm xúc "Tích cực", câu mới được tạo ra cũng phải giữ nguyên cảm xúc "Tích cực".

Chúng ta sẽ khám phá một số kỹ thuật tăng cường dữ liệu phổ biến, từ đơn giản đến phức tạp.

### 1.4.1. EDA: Các Phép toán Thay thế Đơn giản

EDA (Easy Data Augmentation - Wei & Zou, 2019) là một bộ bốn kỹ thuật đơn giản nhưng lại hiệu quả một cách ngạc nhiên.

- **Synonym Replacement (SR) - Thay thế bằng Từ đồng nghĩa:**
  - **Cơ chế:** Chọn ngẫu nhiên  $n$  từ trong câu (không phải là stop words) và thay thế mỗi từ bằng một từ đồng nghĩa của nó, thường được lấy từ một kho từ vựng như WordNet.
  - **Ví dụ:** "Bộ phim này rất tuyệt vời và hấp dẫn." → "Bộ phim này rất xuất sắc và lôi cuốn."
- **Random Insertion (RI) - Chèn ngẫu nhiên:**
  - **Cơ chế:** Tìm các từ đồng nghĩa của một vài từ ngẫu nhiên trong câu, sau đó chèn các từ đồng nghĩa đó vào các vị trí ngẫu nhiên trong câu.
  - **Ví dụ:** "Bộ phim này rất tuyệt vời." → "Bộ phim xuất sắc này rất tuyệt vời."
- **Random Swap (RS) - Hoán đổi ngẫu nhiên:**
  - **Cơ chế:** Chọn ngẫu nhiên hai từ trong câu và hoán đổi vị trí của chúng.
  - **Ví dụ:** "Bộ phim này rất tuyệt vời." → "Bộ phim tuyệt vời rất này."
- **Random Deletion (RD) - Xóa ngẫu nhiên:**
  - **Cơ chế:** Xóa ngẫu nhiên mỗi từ trong câu với một xác suất  $p$  nào đó.
  - **Ví dụ:** "Bộ phim này rất tuyệt vời và hấp dẫn." → "Bộ phim này tuyệt vời và hấp dẫn."

**Lưu ý:** Các kỹ thuật EDA có thể tạo ra các câu không tự nhiên hoặc sai ngữ pháp (đặc biệt là RS và RI). Tuy nhiên, việc đưa một ít "nhiều" này vào quá trình huấn luyện đôi khi lại giúp mô hình trở nên mạnh mẽ hơn. Chúng rất dễ triển khai và là một điểm khởi đầu tốt.

### 1.4.2. Back-Translation: Tận dụng Sức mạnh của Dịch máy

Đây là một trong những kỹ thuật tăng cường dữ liệu chất lượng cao và phổ biến nhất.

#### Trực giác cốt lõi

Nếu chúng ta dịch một câu từ ngôn ngữ A sang ngôn ngữ B, rồi lại dịch ngược kết quả từ B trở lại A, chúng ta thường sẽ nhận được một câu có cùng ý nghĩa với câu gốc, nhưng được diễn đạt bằng các từ ngữ và cấu trúc câu khác.

#### Quy trình Back-Translation

Câu gốc (Tiếng Việt)  $\xrightarrow{\text{Dịch máy (Vi} \rightarrow \text{En)}}$  Câu trung gian (Tiếng Anh)  $\xrightarrow{\text{Dịch máy (En} \rightarrow \text{Vi)}}$  Câu mới (Tiếng Việt)

#### Ví dụ 17: Minh họa Back-Translation

- **Câu gốc (Vi):** "Tôi nghĩ rằng đây là một ý tưởng cực kỳ thông minh."
- **Dịch sang Anh:** "I think that this is an extremely intelligent idea."
- **Dịch ngược về Việt:** "Tôi cho rằng đây là một ý tưởng vô cùng thông minh."

Câu mới được tạo ra có cùng nhãn với câu gốc nhưng lại là một mẫu dữ liệu huấn luyện mới, giúp mô hình học cách khai thác qua các cách diễn đạt khác nhau.

#### Ưu điểm và Nhược điểm

- **Ưu điểm:**

- Thường tạo ra các câu có chất lượng ngữ pháp và ngữ nghĩa cao hơn nhiều so với các phương pháp EDA.
- Có khả năng tạo ra sự đa dạng lớn về cả từ vựng và cấu trúc câu.
- **Nhược điểm:**
  - Yêu cầu phải có các mô hình dịch máy chất lượng cao.
  - Chi phí tính toán cao hơn vì phải gọi đến các mô hình dịch.
  - Có nguy cơ ý nghĩa bị thay đổi ("lost in translation") nếu các mô hình dịch không đủ tốt.

### 1.4.3. Tăng cường Dữ liệu dựa trên LLM

Với sự ra đời của các LLM mạnh mẽ, chúng ta có thể thực hiện các phép tăng cường dữ liệu tinh vi hơn nhiều.

#### Điển giải lại (Paraphrasing)

Đây là một phiên bản nâng cao của Back-Translation. Thay vì đi qua một ngôn ngữ khác, chúng ta có thể yêu cầu trực tiếp một LLM:

**Prompt:**

Hãy viết lại câu sau đây theo 5 cách khác nhau, nhưng vẫn giữ nguyên ý nghĩa cốt lõi.  
Câu gốc: "Sản phẩm này có chất lượng tuyệt vời so với giá tiền."

LLM có thể sinh ra các phiên bản như: "Chất lượng của sản phẩm này vượt xa mong đợi so với mức giá.", "Đây là một món hời, chất lượng rất tốt.", v.v.

#### Tạo dữ liệu theo ngữ cảnh

Chúng ta có thể yêu cầu LLM tạo ra các ví dụ phù hợp với một kịch bản cụ thể, đặc biệt hữu ích để xử lý các "trường hợp rìa"(edge cases) mà mô hình thường làm sai.

**Ví dụ 18: Tạo dữ liệu cho các trường hợp phủ định**

Giả sử phân tích lỗi cho thấy mô hình phân tích cảm xúc của chúng ta thường sai ở các câu có yếu tố phủ định. Chúng ta có thể dùng prompt:

**Prompt:**

Dưới đây là một câu có cảm xúc Tích cực. Hãy viết lại nó để biến nó thành một câu có cảm xúc Tiêu cực bằng cách sử dụng các từ ngữ phủ định hoặc mỉa mai, nhưng vẫn giữ chủ đề chính.

Câu gốc (Tích cực): "Dịch vụ khách hàng của họ rất nhanh và hiệu quả."

Câu mới (Tiêu cực):

**Đầu ra có thể có của LLM:** "Đừng mong đợi dịch vụ khách hàng của họ sẽ nhanh và hiệu quả." hoặc "Nhanh và hiệu quả' không phải là những từ tôi sẽ dùng để mô tả dịch vụ khách hàng của họ."

Cách tiếp cận này cho phép chúng ta tạo ra dữ liệu một cách có chủ đích để vá các "điểm yếu" cụ thể của mô hình.

### 1.4.4. Lựa chọn Kỹ thuật Tăng cường

- **Bắt đầu đơn giản:** Luôn bắt đầu với EDA. Nó nhanh, miễn phí và có thể mang lại những cải thiện bất ngờ.

- Khi cần chất lượng cao: Back-Translation là một lựa chọn mạnh mẽ nếu bạn có quyền truy cập vào các API dịch tốt.
- Khi cần sự kiểm soát và tinh vi: Tận dụng LLM để diễn giải lại hoặc tạo ra dữ liệu cho các trường hợp cụ thể là hướng đi hiện đại và mạnh mẽ nhất.
- Thận trọng: Không phải lúc nào tăng cường dữ liệu cũng giúp ích. Nếu các mẫu dữ liệu mới được tạo ra có chất lượng thấp hoặc làm thay đổi nhãn, chúng có thể làm hại mô hình. Luôn đánh giá hiệu năng trên một tập validation riêng biệt để kiểm tra xem việc tăng cường có thực sự hiệu quả hay không.

## 1.5. Quản lý Dữ liệu và Phiên bản (Data Version Control - DVC)

Trong phát triển phần mềm truyền thống, \*\*Git\*\* là công cụ tiêu chuẩn để quản lý phiên bản của mã nguồn. Nó cho phép chúng ta theo dõi mọi thay đổi, quay lại các phiên bản cũ, và cộng tác một cách hiệu quả.

Tuy nhiên, trong các dự án học máy, **Mã nguồn chỉ là một phần của câu chuyện**. Kết quả của một mô hình không chỉ phụ thuộc vào code, mà còn phụ thuộc rất nhiều vào:

- **Dữ liệu (Data)**: Bộ dữ liệu nào đã được sử dụng để huấn luyện?
- **Mô hình (Model)**: Trọng số của mô hình được tạo ra từ lần chạy nào?
- **Siêu tham số (Hyperparameters)**: Các thiết lập như tốc độ học, kích thước batch là gì?

Vấn đề: "Nó chạy được trên máy của tôi mà!"

Nếu không có một hệ thống quản lý phiên bản hoàn chỉnh, chúng ta sẽ rơi vào một "địa ngục tái lập"(reproducibility hell). Bạn có thể có một mô hình hoạt động rất tốt, nhưng vài tháng sau, khi dữ liệu thay đổi hoặc một đồng nghiệp muốn tái tạo lại kết quả của bạn, không ai có thể làm được vì không ai biết chính xác phiên bản dữ liệu và các thiết lập nào đã tạo ra mô hình đó.

**Git không được thiết kế để xử lý các file lớn** như các bộ dữ liệu hay các file trọng số mô hình. Việc commit các file lớn này sẽ làm cho kho Git trở nên cực kỳ cồng kềnh và chậm chạp. Đây là lúc DVC (Data Version Control) phát huy tác dụng.

### 1.5.1. DVC là gì và Triết lý hoạt động

DVC là một công cụ mã nguồn mở được thiết kế để mang lại khả năng quản lý phiên bản cho toàn bộ pipeline học máy. Nó hoạt động **bên trên Git**, không phải thay thế Git.

Triết lý của DVC

Hãy để Git làm những gì nó giỏi nhất: quản lý phiên bản của các file code nhỏ. Hãy để một bộ nhớ từ xa (**remote storage**) (như Amazon S3, Google Cloud Storage, hoặc thậm chí là Google Drive) lưu trữ các file dữ liệu và mô hình lớn. DVC sẽ đóng vai trò là "cầu nối", tạo ra các file metadata nhỏ để theo dõi phiên bản của các file lớn đó, và các file metadata này sẽ được quản lý bởi Git.

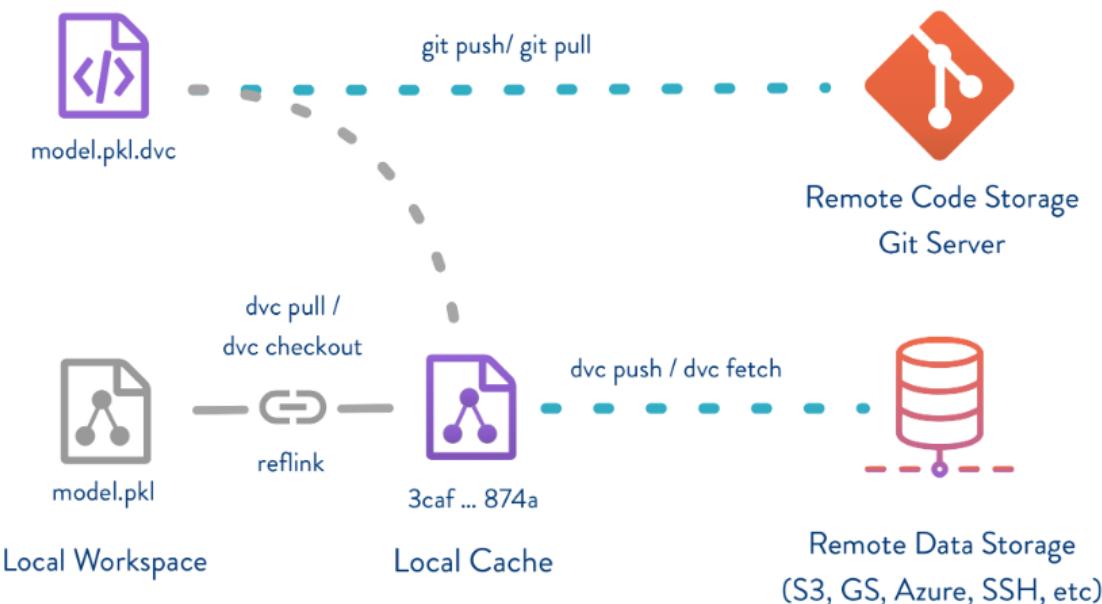
### Cơ chế hoạt động

Khi bạn muốn theo dõi một file dữ liệu lớn (ví dụ: 'data.csv') bằng DVC, các bước sau sẽ xảy ra:

1. 'dvc add data.csv':

- DVC sao chép file 'data.csv' vào một thư mục ẩn gọi là '.dvc/cache'. Nội dung của file được lưu trữ dựa trên một giá trị băm (hash) duy nhất.

- DVC tạo ra một file metadata nhỏ, có tên là ‘data.csv.dvc’. File này chỉ chứa thông tin về file gốc, quan trọng nhất là giá trị băm của nó.
  - DVC tự động thêm ‘data.csv’ vào file ‘.gitignore’ để Git sẽ bỏ qua file dữ liệu lớn này.
2. ‘git add data.csv.dvc’ và ‘git commit’:
    - Bây giờ, bạn sẽ commit cái file ‘data.csv.dvc’ nhỏ xíu này vào Git.
    - Bất kỳ ai clone kho Git của bạn sẽ có file metadata này.
  3. ‘dvc push’:
    - Lệnh này sẽ tải nội dung của file dữ liệu thực sự (từ ‘.dvc/cache’) lên bộ nhớ từ xa mà bạn đã cấu hình (ví dụ: một bucket S3).
  4. ‘dvc pull’:
    - Khi một đồng nghiệp clone kho Git, họ sẽ không có file ‘data.csv’. Nhưng họ có thể chạy lệnh ‘dvc pull’.
    - DVC sẽ đọc file ‘data.csv.dvc’, tìm ra giá trị băm cần thiết, và tải về phiên bản dữ liệu chính xác từ bộ nhớ từ xa vào máy của họ.



Hình 1.3: Luồng làm việc của DVC kết hợp với Git. Git quản lý code và các file metadata ‘.dvc’. DVC quản lý việc đồng bộ hóa các file dữ liệu/mô hình lớn với bộ nhớ từ xa.

### 1.5.2. DVC Pipelines: Tái lập toàn bộ Thủ nghiệm

DVC còn tiến một bước xa hơn bằng cách cho phép bạn định nghĩa và kết nối các bước trong pipeline học máy của mình.

- **Định nghĩa các Giai đoạn (Stages):** Bạn có thể tạo một file (ví dụ: ‘dvc.yaml’) để định nghĩa các giai đoạn của thử nghiệm, ví dụ: ‘preprocess’, ‘train’, ‘evaluate’.
- **Mô tả mỗi Giai đoạn:**
  - ‘cmd’: Lệnh cần chạy (ví dụ: ‘python train.py’).
  - ‘deps’ (Dependencies): Các file đầu vào của giai đoạn này (ví dụ: code ‘train.py’, dữ liệu ‘data/train.csv’).
  - ‘outs’ (Outputs): Các file được tạo ra bởi giai đoạn này (ví dụ: file trọng số ‘model.pkl’, file metric ‘metrics.json’).
- **Chạy và Tái tạo Pipeline (‘dvc repro’):**

- Lệnh ‘dvc repro’ (reproduce) sẽ tự động kiểm tra xem có bất kỳ file ‘deps’ nào đã thay đổi hay không.
- Nếu có một sự thay đổi (ví dụ, bạn thay đổi một siêu tham số trong code hoặc cập nhật dữ liệu), DVC sẽ tự động chạy lại giai đoạn đó và tất cả các giai đoạn phụ thuộc vào nó.
- Nếu không có gì thay đổi, DVC sẽ không chạy lại, giúp tiết kiệm thời gian và tài nguyên tính toán.

Bằng cách commit file ‘dvc.yaml’ và ‘dvc.lock’ (file ghi lại các giá trị băm chính xác của các deps/outs) vào Git, bạn đã ghi lại toàn bộ ”provenance”(nguồn gốc) của một thử nghiệm. Bất kỳ ai cũng có thể checkout một commit Git cụ thể và chạy ‘dvc repro’ để tái tạo lại chính xác kết quả của bạn.

### 1.5.3. Lợi ích của việc sử dụng DVC

- **Tính tái lập (Reproducibility):** Đảm bảo rằng các thử nghiệm có thể được tái tạo một cách đáng tin cậy bởi bạn trong tương lai hoặc bởi các đồng nghiệp.
- **Cộng tác Dễ dàng:** Các thành viên trong nhóm có thể chia sẻ dữ liệu và mô hình một cách liền mạch mà không làm phì phèo to kho Git.
- **Theo dõi Thủ nghiệm:** DVC cho phép bạn so sánh các kết quả giữa các nhánh Git hoặc các commit khác nhau, giúp bạn hiểu rõ ảnh hưởng của các thay đổi (dữ liệu, code, siêu tham số) đến metric của mô hình.
- **Sự linh hoạt về Lưu trữ:** Hoạt động với nhiều loại backend lưu trữ khác nhau (S3, GCS, Azure Blob, HDFS, SSH, ...).

Việc tích hợp DVC vào quy trình làm việc đòi hỏi một chút nỗ lực ban đầu, nhưng lợi ích về lâu dài về sự ổn định, tính tái lập và khả năng cộng tác là vô giá trong bất kỳ dự án học máy nghiêm túc nào.

---

## KẾT THÚC CHƯƠNG 1

*Trong chương đầu tiên của phần thực chiến này, chúng ta đã xây dựng nền móng vững chắc cho bất kỳ dự án NLP thành công nào. Chúng ta đã bắt đầu với một quy trình làm việc có hệ thống, sau đó đi sâu vào công việc chiếm phần lớn thời gian và công sức: dữ liệu. Bạn đã học các kỹ thuật để thu thập, làm sạch, gán nhãn, tăng cường, và quan trọng là quản lý phiên bản dữ liệu một cách chuyên nghiệp. Bài học cốt lõi của chương này là: một pipeline dữ liệu mạnh mẽ và có thể tái lập chính là xương sống của một hệ thống AI hiệu năng cao. Giờ đây, khi đã có trong tay một bộ dữ liệu chất lượng, chúng ta đã sẵn sàng để chuyển sang giai đoạn hấp dẫn tiếp theo: huấn luyện và đánh giá các mô hình state-of-the-art.*

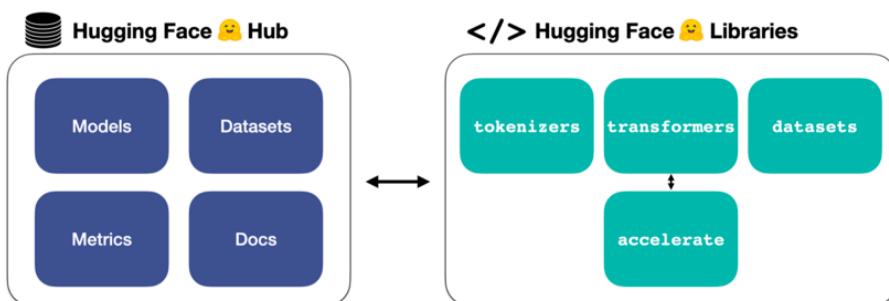
## Chương 2

# HUẤN LUYỆN VÀ ĐÁNH GIÁ MÔ HÌNH

Sau khi đã có trong tay dữ liệu đã được chuẩn bị kỹ lưỡng, chúng ta bước vào giai đoạn hấp dẫn nhất của một dự án NLP: huấn luyện mô hình. Trong kỷ nguyên hiện đại, việc này không còn có nghĩa là phải xây dựng các kiến trúc Transformer từ đầu. Thay vào đó, chúng ta sẽ tận dụng sức mạnh của cộng đồng và các công cụ mã nguồn mở để tăng tốc quá trình phát triển.

Chương này sẽ là một chuyến đi sâu vào **Hệ sinh thái Hugging Face** – một tập hợp các thư viện đã cách mạng hóa cách chúng ta làm việc với các mô hình ngôn ngữ. Bạn sẽ học cách thiết lập một môi trường làm việc hoàn chỉnh, từ việc tải về các mô hình và bộ dữ liệu, đến việc triển khai một quy trình huấn luyện và đánh giá bài bản. Chúng ta cũng sẽ khám phá các công cụ chuyên nghiệp để theo dõi thí nghiệm và thậm chí là huấn luyện các mô hình khổng lồ trên nhiều GPU. Đây là những kỹ năng thực chiến sẽ giúp bạn chuyển từ việc hiểu lý thuyết sang việc tạo ra các mô hình NLP state-of-the-art.

### 2.1. Thiết lập Môi trường với Hệ sinh thái Hugging Face



Hình 2.1: Các thư viện cốt lõi trong Hệ sinh thái Hugging Face.

Hugging Face đã phát triển từ một công ty chatbot thành trung tâm của cộng đồng NLP mã nguồn mở. Hệ sinh thái của họ cung cấp các công cụ mạnh mẽ, được thiết kế để hoạt động liền mạch với nhau, giúp đơn giản hóa và chuẩn hóa quy trình làm việc với các mô hình Transformer.

Để bắt đầu, chúng ta cần cài đặt và hiểu vai trò của bốn thư viện cốt lõi.

baselinestretch

---

```
# Cài đặt các thư viện chính bằng pip
pip install transformers datasets accelerate evaluate
```

---

Hãy cùng tìm hiểu vai trò của từng thư viện.

### 2.1.1. Transformers: Trái tim của Hệ sinh thái

- Mục đích:** Cung cấp một giao diện đơn giản và nhất quán để tải về, sử dụng và huấn luyện hàng nghìn mô hình Transformer đã được huấn luyện trước (pre-trained models) từ **Hugging Face Hub**.
- Hugging Face Hub là gì?** Là một nền tảng trực tuyến (tương tự như GitHub) nơi cộng đồng có thể chia sẻ và khám phá các mô hình, bộ dữ liệu và các bản demo. Đây là tài nguyên quan trọng nhất trong hệ sinh thái.
- Các thành phần chính:**
  - ‘pipeline’: Một API cấp cao, cực kỳ dễ sử dụng để thực hiện các tác vụ NLP phổ biến chỉ trong vài dòng code. Rất tuyệt vời cho việc thử nghiệm nhanh và tạo demo.
  - ‘AutoModel’, ‘AutoTokenizer’: Các lớp “tự động” thông minh. Bạn chỉ cần cung cấp tên của một mô hình trên Hub (ví dụ: “bert-base-uncased”), các lớp này sẽ tự động tìm, tải về và khởi tạo đúng kiến trúc mô hình và tokenizer tương ứng.
  - ‘Trainer’ API: Một API huấn luyện cấp cao, mạnh mẽ, giúp xử lý tất cả các công việc phức tạp của vòng lặp huấn luyện (training loop) như tối ưu hóa, lên lịch tốc độ học (learning rate scheduling), đánh giá, và lưu trữ mô hình.

Ví dụ 19: Sử dụng ‘pipeline’ để phân loại cảm xúc

baselinestretch

```

1  from transformers import pipeline
2
3  # Tải một mô hình đã được fine-tune cho phân tích cảm xúc
4  # Lần đầu chạy, pipeline sẽ tự động tải mô hình từ Hub
5  classifier = pipeline("sentiment-analysis")
6
7  # Sử dụng mô hình
8  result = classifier("Hugging Face is democratizing NLP!")
9  print(result)
10 # Output: [{'label': 'POSITIVE', 'score': 0.9998...}]

```

### 2.1.2. Datasets: Quản lý Dữ liệu Hiệu quả

- Mục đích:** Cung cấp một cách thức hiệu quả và tiêu chuẩn hóa để tải về, xử lý, và làm việc với các bộ dữ liệu, đặc biệt là các bộ dữ liệu lớn.
- Các tính năng chính:**
  - Truy cập hàng nghìn bộ dữ liệu:** Tương tự như ‘transformers’, thư viện ‘datasets’ cho phép bạn tải về hàng nghìn bộ dữ liệu công khai từ Hugging Face Hub chỉ bằng một dòng lệnh (‘load\_dataset’).
  - Xử lý hiệu quả bộ nhớ (Memory-mapping):** Đối với các bộ dữ liệu khổng lồ (hàng trăm GB), ‘datasets’ không tải toàn bộ vào RAM. Thay vào đó, nó sử dụng kỹ thuật memory-mapping (được hỗ trợ bởi Apache Arrow), cho phép bạn truy cập và xử lý dữ liệu ngay trên đĩa cứng một cách cực kỳ nhanh chóng.
  - API xử lý mạnh mẽ:** Cung cấp các phương thức như ‘.map()’, ‘.filter()’, ‘.shuffle()’ cho phép bạn áp dụng các hàm tiền xử lý (như tokenization) lên toàn bộ bộ dữ liệu một cách song song và hiệu quả.

## Ví dụ 20: Tải và xử lý một bộ dữ liệu với ‘datasets’

baselinestretch

```

1   from datasets import load_dataset
2   from transformers import AutoTokenizer
3
4   # Tải bộ dữ liệu GLUE, tác vụ MRPC
5   raw_datasets = load_dataset("glue", "mrpc")
6   # >>> raw_datasets
7   # DatasetDict({
8   #     train: Dataset({
9   #         features: ['sentence1', 'sentence2', 'label', 'idx'],
10  #         num_rows: 3668
11  #     })
12  #     ...
13  # })
14
15  # Tải tokenizer tương ứng với một checkpoint
16  checkpoint = "bert-base-uncased"
17  tokenizer = AutoTokenizer.from_pretrained(checkpoint)
18
19  # Viết một hàm để tokenize dữ liệu
20  def tokenize_function(example):
21      return tokenizer(example["sentence1"], example["sentence2"],
22                      truncation=True)
23
24  # Áp dụng hàm này lên toàn bộ bộ dữ liệu
25  # batched=True xử lý nhiều mẫu cùng lúc để tăng tốc
26  tokenized_datasets = raw_datasets.map(tokenize_function,
27                                      batched=True)
28  print(tokenized_datasets)

```

## 2.1.3. Accelerate: Đơn giản hóa Huấn luyện Phân tán

- Mục đích:** Cung cấp một lớp trừu tượng (abstraction layer) đơn giản để bạn có thể chạy cùng một đoạn code huấn luyện trên các môi trường phần cứng khác nhau (một GPU, nhiều GPU, TPU) mà không cần thay đổi code.
- Triết lý:** “Write your PyTorch code once, run it anywhere.”
- Cơ chế hoạt động:** ‘accelerate’ sẽ tự động xử lý các công việc phức tạp ở phía sau như:
  - Đặt các tensor và mô hình lên đúng thiết bị (device placement).
  - Bao bọc mô hình và optimizer trong các lớp phù hợp cho huấn luyện phân tán (ví dụ: ‘DistributedDataParallel’).
  - Xử lý việc thu thập (gather) các kết quả từ nhiều tiến trình khác nhau.
- Sử dụng:** Bạn chỉ cần thêm một vài dòng code ‘accelerate’ vào đầu kịch bản huấn luyện PyTorch tiêu chuẩn của mình. ‘Trainer’ API của ‘transformers’ đã tích hợp sẵn ‘accelerate’ ở bên trong.

## 2.1.4. Evaluate: Một Thư viện Metric Toàn diện

- Mục đích:** Cung cấp một cách dễ dàng và đáng tin cậy để tính toán và so sánh các metric đánh giá trong NLP và các lĩnh vực khác.
- Các tính năng chính:**

- **Thư viện Metric lớn:** Cung cấp các cách triển khai đã được kiểm chứng cho hàng chục metric phổ biến (BLEU, ROUGE, F1, Accuracy, Perplexity, ...).
- **Giao diện đơn giản:** Chỉ cần ‘evaluate.load(“tên\_metric”)’ để tải metric, sau đó dùng ‘.add\_batch()’ và ‘.compute()’ để tính toán.
- **Hỗ trợ phân tán:** Tự động xử lý việc tổng hợp các dự đoán và nhãn từ nhiều GPU khi sử dụng với ‘accelerate’.

#### Ví dụ 21: Sử dụng ‘evaluate’ để tính F1-score

baselinestretch

```

1 import evaluate
2
3 # Tải metric F1
4 f1_metric = evaluate.load("f1")
5
6 # Giả sử có các dự đoán và nhãn
7 predictions = [0, 1, 0, 1, 1]
8 references = [0, 0, 0, 1, 1]
9
10 # Tính toán
11 results = f1_metric.compute(predictions=predictions,
12                             references=references)
13 print(results)
# Output: {'f1': 0.8}

```

Bằng cách kết hợp bốn thư viện này, chúng ta có một quy trình làm việc mạnh mẽ và được tiêu chuẩn hóa, cho phép chúng ta tập trung vào phần khoa học của việc xây dựng mô hình thay vì phải lo lắng về các chi tiết kỹ thuật phức tạp.

## 2.2. Kỹ thuật Đánh giá Thực tế

Việc theo dõi các metric đánh giá trong suốt quá trình huấn luyện là cực kỳ quan trọng. Nó không chỉ cho chúng ta biết mô hình đang hoạt động tốt đến đâu, mà còn giúp phát hiện sớm các vấn đề như quá khớp (overfitting) hoặc dưới khớp (underfitting).

‘Trainer’ API của thư viện ‘transformers’ cung cấp một cách rất tiện lợi để thực hiện việc đánh giá định kỳ trên tập validation. Chìa khóa để làm việc này là cung cấp cho nó một hàm tùy chỉnh, thường được gọi là ‘compute\_metrics’.

### 2.2.1. Hàm **compute\_metrics**: Cầu nối giữa Mô hình và Metric

- **Mục đích:** Hàm ‘compute\_metrics’ là một hàm Python mà bạn tự định nghĩa. ‘Trainer’ sẽ gọi hàm này sau mỗi epoch (hoặc sau một số bước nhất định) trong quá trình huấn luyện.
- **Đầu vào:** Hàm này nhận vào một đối tượng ‘EvalPrediction’, là một tuple chứa hai mảng NumPy: ‘predictions’ (các logits đầu ra từ mô hình) và ‘label\_ids’ (các nhãn thật từ bộ dữ liệu).
- **Đầu ra:** Hàm phải trả về một dictionary, trong đó key là tên của metric (ví dụ: “f1”) và value là giá trị của metric đó.

**Quy trình bên trong một hàm **compute\_metrics**** Một hàm ‘compute\_metrics’ điển hình sẽ thực hiện các bước sau:

1. **Tải các metric cần thiết** từ thư viện ‘evaluate’.
2. **Tiền xử lý đầu ra của mô hình:** Các ‘predictions’ thường là các logits (các số thực thô). Chúng ta cần phải chuyển đổi chúng thành các dự đoán cuối cùng (ví dụ, bằng cách lấy ‘argmax’ trên

- chiều cuối cùng) để có thể so sánh với các nhãn.
3. **Tính toán các metric:** Sử dụng các đối tượng metric đã tải để tính toán điểm số bằng cách truyền vào các dự đoán đã xử lý và các nhãn thật.
  4. **Trả về kết quả:** Trả về một dictionary chứa các điểm số đã tính được.

#### Ví dụ 22: Một hàm ‘compute\_metrics’ cho bài toán phân loại chuỗi

Đây là một ví dụ hoàn chỉnh cho một bài toán phân loại nhị phân, sử dụng F1, Precision và Recall. baselinestretch

```

1 import numpy as np
2 import evaluate
3
4 # Tải trước các metric để không phải tải lại mỗi lần gọi hàm
5 f1_metric = evaluate.load("f1")
6 precision_metric = evaluate.load("precision")
7 recall_metric = evaluate.load("recall")
8
9 def compute_metrics(eval_preds):
10     # eval_preds là một tuple (predictions, label_ids)
11     logits, labels = eval_preds
12
13     # 1. Chuyển đổi logits thành dự đoán cuối cùng
14     # Lấy chỉ số của logit lớn nhất trên chiều cuối cùng (-1)
15     predictions = np.argmax(logits, axis=-1)
16
17     # 2. Tính toán từng metric
18     f1 = f1_metric.compute(predictions=predictions,
19                           references=labels)["f1"]
20     precision =
21         precision_metric.compute(predictions=predictions,
22                                   references=labels)["precision"]
23     recall = recall_metric.compute(predictions=predictions,
24                                   references=labels)["recall"]
25
26     # 3. Trả về một dictionary
27     return {
28         "f1": f1,
29         "precision": precision,
30         "recall": recall,
31     }
32
33     # Sau đó, hàm này sẽ được truyền vào đối tượng
34     # → TrainingArguments:
35     # training_args = TrainingArguments(...,
36     #                                   evaluation_strategy="epoch")
37     # trainer = Trainer(
38     #     ...,
39     #     args=training_args,
40     #     compute_metrics=compute_metrics,
41     # )
42     # trainer.train()

```

#### 2.2.2. Đánh giá các Tác vụ Phúc tạp hơn

### Gán nhãn Chuỗi (NER) với ‘seqeval’

- Vấn đề:** Đối với các tác vụ như Nhận dạng Thực thể Tên (NER), việc đánh giá không chỉ đơn giản là so sánh từng nhãn một. Chúng ta quan tâm đến việc mô hình có nhận dạng đúng **toute bộ các thực thể** (bao gồm cả ranh giới và loại) hay không. Ví dụ, dự đoán ‘[B-PER, I-PER]’ cho “Hồ Chí Minh” là đúng, nhưng dự đoán ‘[B-PER, B-LOC]’ là sai hoàn toàn, mặc dù đúng được 1/2 nhãn.
- Giải pháp:** ‘seqeval’. Đây là một thư viện Python (cũng được tích hợp trong ‘evaluate’) chuyên để đánh giá các tác vụ gán nhãn chuỗi. Nó tính toán Precision, Recall, và F1-score ở cấp độ thực thể (entity-level) theo các định dạng chuẩn như IOB2.

#### Ví dụ 23: Hàm `compute_metrics` cho NER

baselinestretch

```

1 import numpy as np
2 import evaluate
3
4 # Tải metric seqeval
5 seqeval_metric = evaluate.load("seqeval")
6
7 # Giả sử chúng ta có mapping từ ID sang nhãn
8 label_names = ['O', 'B-PER', 'I-PER', 'B-LOC', 'I-LOC']
9
10 def compute_metrics_ner(eval_preds):
11     logits, labels = eval_preds
12     predictions = np.argmax(logits, axis=-1)
13
14     # Chuyển đổi ID trở lại thành nhãn chuỗi
15     # Bỏ qua các nhãn padding (thường có ID là -100)
16     true_labels = [
17         [label_names[l] for l in label if l != -100]
18         for label in labels
19     ]
20     true_predictions = [
21         [label_names[p] for p, l in zip(prediction, label) if l
22             != -100]
23         for prediction, label in zip(predictions, labels)
24     ]
25
26     # Tính toán metric
27     results =
28         seqeval_metric.compute(predictions=true_predictions,
29             references=true_labels)
30
31     # Trả về các metric chính
32     return {
33         "precision": results["overall_precision"],
34         "recall": results["overall_recall"],
35         "f1": results["overall_f1"],
36         "accuracy": results["overall_accuracy"],
37     }

```

**Sinh Văn bản (Tóm tắt, Dịch máy) với ROUGE và BLEU**

- Vấn đề:** Các mô hình sinh văn bản (Decoder-Only, Encoder-Decoder) tạo ra các chuỗi văn bản có độ dài thay đổi. Việc đánh giá đòi hỏi phải so sánh các chuỗi này với các chuỗi tham khảo.
- Thách thức trong ‘Trainer’:** ‘Trainer’ không thể tự động thu thập các chuỗi văn bản được sinh ra từ nhiều GPU. Hơn nữa, quá trình decode (chuyển các token ID thành văn bản) cần phải được thực hiện.
- Giải pháp:** ‘Trainer’ có một phương thức đặc biệt là ‘predict’. Khi sử dụng ‘evaluation\_strategy’ cùng với ‘predict\_with\_generate=True’ trong ‘TrainingArguments’, ‘Trainer’ sẽ tự động:
  - Chạy quá trình sinh văn bản (.generate()) trên tập validation.
  - Thu thập (gather) các kết quả từ tất cả các thiết bị.

Chúng ta chỉ cần thực hiện bước decode và tính toán metric trong hàm ‘compute\_metrics’.

Ví dụ 24: Hàm **compute\_metrics** cho Tóm tắt (sử dụng ROUGE)

baselinestretch

```

1  import numpy as np
2  import evaluate
3  from transformers import AutoTokenizer # Cần tokenizer để decode
4
5  # Giả sử tokenizer đã được định nghĩa ở ngoài
6  tokenizer = AutoTokenizer.from_pretrained("t5-small")
7  rouge_metric = evaluate.load("rouge")
8
9  def compute_metrics_summarization(eval_preds):
10    predictions, labels = eval_preds
11
12    # Decode các token ID thành văn bản
13    # Bỏ qua các token padding
14    decoded_preds = tokenizer.batch_decode(predictions,
15                                          skip_special_tokens=True)
16
17    # Thay thế các token padding (-100) trong nhãn bằng
18    # → pad_token_id
19    # để chúng cũng có thể được decode đúng cách
20    labels = np.where(labels != -100, labels,
21                      tokenizer.pad_token_id)
22    decoded_labels = tokenizer.batch_decode(labels,
23                                          skip_special_tokens=True)
24
25    # Tính ROUGE score
26    result = rouge_metric.compute(predictions=decoded_preds,
27                                  references=decoded_labels, use_stemmer=True)
28
29    # Trích xuất các điểm số chính
30    prediction_lens = [np.count_nonzero(pred !=
31                           tokenizer.pad_token_id) for pred in predictions]
32    result["gen_len"] = np.mean(prediction_lens)
33
34  return {k: round(v, 4) for k, v in result.items()}

```

Việc xây dựng một hàm ‘compute\_metrics’ đúng đắn và phù hợp với tác vụ là một bước không thể thiếu để có được một quy trình huấn luyện và đánh giá đáng tin cậy.

## 2.3. Theo dõi và Quản lý Thí nghiệm

Khi bạn bắt đầu một dự án NLP, bạn sẽ không chỉ huấn luyện mô hình một lần. Bạn sẽ thử nghiệm với:

- Các siêu tham số khác nhau (tốc độ học, kích thước batch, số epoch).
- Các mô hình nền tảng khác nhau (BERT vs. RoBERTa vs. Llama).
- Các phiên bản dữ liệu khác nhau (sau khi thêm dữ liệu tăng cường, sau khi làm sạch).
- Các kỹ thuật fine-tuning khác nhau (Full vs. LoRA).

Nếu chỉ dựa vào việc ghi chép thủ công hoặc đặt tên file một cách lộn xộn (ví dụ: ‘model\_final.pt’, ‘model\_final\_v2.pt’, ‘model\_final\_really\_final.pt’), bạn sẽ nhanh chóng lạc vào một mớ hỗn độn.

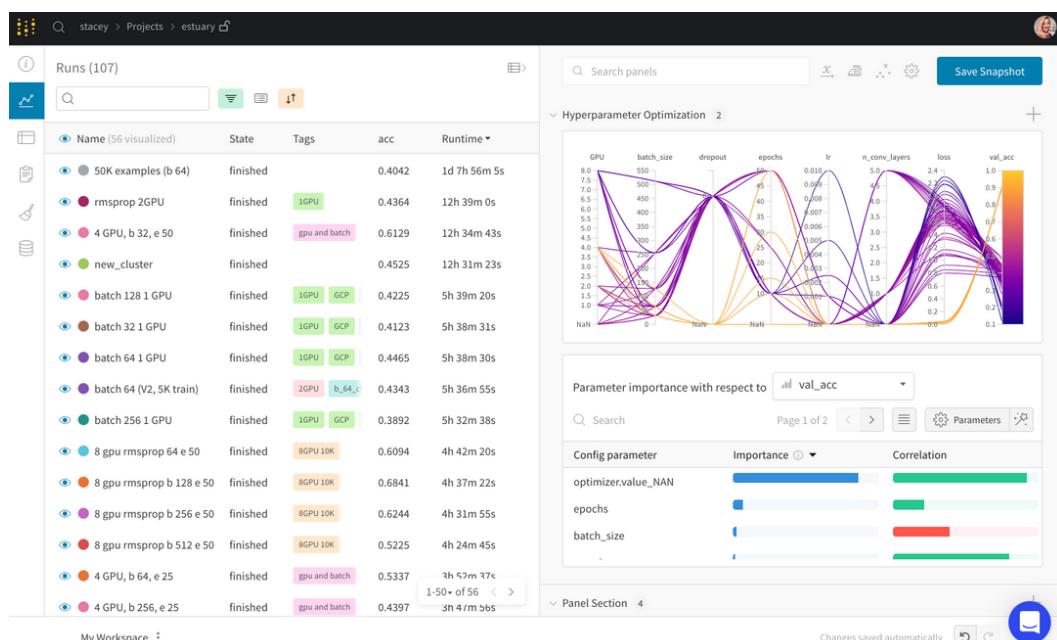
### Vấn đề: “Mô hỗn độn của Thí nghiệm”

Làm thế nào để so sánh hiệu năng giữa lần chạy tuần trước và lần chạy hôm nay? Siêu tham số nào đã tạo ra mô hình tốt nhất? Bộ dữ liệu nào đã được sử dụng cho phiên bản mô hình đang chạy trên production? Nếu không có một hệ thống theo dõi, việc trả lời những câu hỏi này là gần như không thể, làm cản trở sự tiến bộ và tính tái lập.

Các công cụ Theo dõi và Quản lý Thí nghiệm (Experiment Tracking and Management) ra đời để giải quyết vấn đề này. Chúng hoạt động như một “cuốn sổ tay phòng thí nghiệm” tự động cho các dự án học máy. Chúng ta sẽ tìm hiểu hai công cụ phổ biến nhất: Weights & Biases và MLflow.

### 2.3.1. Weights & Biases (W&B): Một Nền tảng Toàn diện và Trực quan

Weights & Biases (thường được gọi là ‘wandb’) là một nền tảng MLOps dựa trên đám mây, cung cấp một bộ công cụ cực kỳ mạnh mẽ và thân thiện với người dùng để theo dõi, trực quan hóa, và so sánh các thử nghiệm.



Hình 2.2: Một trang tổng quan (dashboard) điển hình trong Weights & Biases, cho phép so sánh nhiều lần chạy thí nghiệm một cách trực quan.

#### Các tính năng chính

- Theo dõi Tự động (Automatic Logging):

- ‘wandb’ tích hợp liền mạch với hầu hết các framework học máy phổ biến, bao gồm cả ‘Trainer’ API của Hugging Face.
- Nó có thể tự động ghi lại mọi thứ: các siêu tham số, các metric hệ thống (mức sử dụng GPU/CPU, nhiệt độ), các metric đánh giá (loss, F1, accuracy) theo từng bước, và cả các file trọng số của mô hình.
- **Trực quan hóa Dữ liệu (Powerful Visualization):**
  - Cung cấp các biểu đồ động, cho phép bạn theo dõi sự thay đổi của loss và các metric khác theo thời gian thực.
  - Dễ dàng tạo các trang tổng quan tùy chỉnh để so sánh hiệu năng của hàng chục lần chạy trên các biểu đồ khác nhau, giúp bạn nhanh chóng phát hiện ra các mẫu và các siêu tham số tốt nhất.
- **Lưu trữ và Quản lý Hiện vật (Artifacts):**
  - ‘wandb’ Artifacts là một hệ thống quản lý phiên bản cho các “hiện vật” của pipeline học máy, như các bộ dữ liệu, các mô hình đã được huấn luyện, và các bảng đánh giá.
  - Nó giúp bạn theo dõi toàn bộ “dòng dõi”(lineage) của một mô hình: nó được huấn luyện từ bộ dữ liệu nào, bởi lần chạy thí nghiệm nào.
- **Báo cáo và Cộng tác (Reports):**
  - Cho phép bạn viết các báo cáo tương tác, kết hợp văn bản, biểu đồ động, và các kết quả thí nghiệm để chia sẻ những phát hiện của mình với đồng nghiệp.

### Tích hợp với Hugging Face ‘Trainer’

Việc tích hợp ‘wandb’ vào một quy trình huấn luyện sử dụng ‘Trainer’ là cực kỳ đơn giản.

1. Cài đặt thư viện: ‘pip install wandb’
2. Đăng nhập: Chạy ‘wandb login’ trong terminal và dán API key của bạn.
3. Kích hoạt trong ‘TrainingArguments’:

#### Ví dụ 25: Huấn luyện với WandB

baselinestretch

```

1  # 1. Đặt biến môi trường để kích hoạt wandb
2  import os
3  os.environ["WANDB_PROJECT"] = "my-first-nlp-project" # Đặt tên
   → dự án
4
5  # 2. Bật báo cáo trong TrainingArguments
6  training_args = TrainingArguments(
7      output_dir="./results",
8      num_train_epochs=3,
9      # ... các tham số khác
10     report_to="wandb", # Bật tính năng báo cáo đến wandb
11     run_name="bert-finetune-run-1", # (Tùy chọn) Đặt tên cho lần
   → chạy này
12 )
13
14 # 3. Huấn luyện như bình thường
15 trainer = Trainer(...)
16 trainer.train()

```

Chỉ với những thay đổi nhỏ này, ‘Trainer’ sẽ tự động gửi tất cả thông tin liên quan lên trang tổng quan ‘wandb’ của bạn.

### 2.3.2. MLflow: Một Lựa chọn Mã nguồn mở và Linh hoạt

MLflow là một nền tảng mã nguồn mở được tạo ra bởi Databricks để quản lý vòng đời MLOps. Nó có cách tiếp cận theo module và có thể được tự host (self-hosted), mang lại sự linh hoạt và kiểm soát cao hơn.

#### Các thành phần chính

MLflow bao gồm bốn thành phần chính:

- MLflow Tracking:** Thành phần cốt lõi, tương tự như ‘wandb’. Nó cung cấp một API và giao diện người dùng để ghi lại và truy vấn các tham số, mã nguồn, metric, và các hiện vật của các lần chạy thí nghiệm.
- MLflow Projects:** Cung cấp một định dạng chuẩn để đóng gói mã nguồn của bạn, giúp nó có thể được tái tạo và chạy một cách đáng tin cậy trên các nền tảng khác nhau.
- MLflow Models:** Cung cấp một định dạng chuẩn để đóng gói các mô hình, giúp việc triển khai chúng trên nhiều nền tảng phục vụ (serving platforms) trở nên dễ dàng hơn.
- MLflow Model Registry:** Một kho lưu trữ tập trung để quản lý vòng đời của các mô hình, bao gồm quản lý phiên bản, chuyển đổi giai đoạn (staging, production, archived), và chú thích.

#### So sánh W&B và MLflow

Weights & Biases vs. MLflow	
Weights & Biases (W&B)	MLflow
<b>Mô hình:</b> Chủ yếu là một dịch vụ SaaS (Software-as-a-Service) được quản lý, nhưng cũng có các tùy chọn tự host.	<b>Mô hình:</b> Hoàn toàn mã nguồn mở, thường được tự host.
<b>Điểm mạnh:</b> <ul style="list-style-type: none"> <li>Giao diện người dùng cực kỳ bóng bẩy, trực quan và tương tác cao.</li> <li>Tích hợp tự động sâu và liền mạch.</li> <li>Các tính năng cộng tác và báo cáo rất mạnh mẽ.</li> <li>Thiết lập rất nhanh chóng.</li> </ul>	<b>Điểm mạnh:</b> <ul style="list-style-type: none"> <li>Hoàn toàn miễn phí và mã nguồn mở.</li> <li>Kiểm soát hoàn toàn dữ liệu và cơ sở hạ tầng.</li> <li>Tích hợp chặt chẽ với hệ sinh thái Databricks/Spark.</li> <li>Có Model Registry mạnh mẽ cho quản lý vòng đời.</li> </ul>
<b>Điểm yếu:</b> <ul style="list-style-type: none"> <li>Phiên bản miễn phí có giới hạn. Phiên bản trả phí có thể tốn kém.</li> <li>Dữ liệu được lưu trữ trên máy chủ của W&amp;B (trừ khi tự host).</li> </ul>	<b>Điểm yếu:</b> <ul style="list-style-type: none"> <li>Giao diện người dùng kém trực quan và ít tính năng hơn.</li> <li>Yêu cầu nhiều công sức hơn để thiết lập và bảo trì.</li> </ul>

#### Lựa chọn nào cho bạn?

- Đối với các nhà nghiên cứu, cá nhân, hoặc các đội nhóm nhỏ muốn bắt đầu nhanh chóng và có một trải nghiệm người dùng tốt nhất, W&B thường là lựa chọn hàng đầu.
- Đối với các doanh nghiệp lớn có yêu cầu nghiêm ngặt về bảo mật dữ liệu, muốn kiểm soát hoàn toàn cơ sở hạ tầng, hoặc đã đầu tư sâu vào hệ sinh thái Databricks, MLflow là một lựa chọn rất phù hợp.

Việc áp dụng một công cụ theo dõi thí nghiệm ngay từ đầu là một trong những khoản đầu tư tốt nhất bạn có thể làm cho sự thành công và tính bền vững của dự án NLP của mình.

## 2.4. Huấn luyện Phân tán (Distributed Training)

Khi quy mô của các mô hình ngôn ngữ tăng từ hàng trăm triệu lên hàng tỷ, thậm chí hàng nghìn tỷ tham số, việc huấn luyện chúng trên một GPU duy nhất trở nên bất khả thi. Ngay cả những GPU mạnh nhất cũng không đủ bộ nhớ (VRAM) để chứa toàn bộ trọng số, gradient, và trạng thái của optimizer.

**Huấn luyện Phân tán** là một tập hợp các kỹ thuật cho phép chúng ta chia nhỏ gánh nặng tính toán và bộ nhớ của quá trình huấn luyện ra **nhiều GPU** (trên một hoặc nhiều máy).

### 2.4.1. Các Mô hình Song song (Parallelism Paradigms)

Có nhiều cách khác nhau để "chia nhỏ" quá trình huấn luyện. Ba chiến lược chính là:

#### 1. Data Parallelism (DP) - Song song Dữ liệu:

- **Ý tưởng:** Đây là chiến lược phổ biến và đơn giản nhất. Mỗi GPU sẽ giữ một bản sao đầy đủ của mô hình. Dữ liệu huấn luyện (batch) sẽ được chia nhỏ ra, và mỗi GPU sẽ xử lý một phần của batch đó.
- **Quy trình:** Các GPU tính toán gradient một cách độc lập trên phần dữ liệu của mình, sau đó chúng "giao tiếp" với nhau để đồng bộ hóa và tính trung bình các gradient trước khi cập nhật trọng số.
- **Hạn chế:** Không giải quyết được vấn đề bộ nhớ. Nếu mô hình quá lớn để vừa trên một GPU, DP sẽ không hoạt động.

#### 2. Tensor Parallelism (TP) - Song song Tensor:

- **Ý tưởng:** Thay vì sao chép mô hình, chúng ta chia nhỏ **chính các ma trận trọng số** của mô hình ra nhiều GPU.
- **Quy trình:** Ví dụ, một phép nhân ma trận lớn có thể được chia thành nhiều phép nhân ma trận nhỏ hơn được thực hiện song song trên các GPU khác nhau. Các GPU cần giao tiếp với nhau rất nhiều trong quá trình truyền thẳng và truyền ngược.
- **Lợi ích:** Cho phép huấn luyện các mô hình lớn hơn bộ nhớ của một GPU.

#### 3. Pipeline Parallelism (PP) - Song song Đường ống:

- **Ý tưởng:** Chia các **lớp (layers)** của mô hình ra nhiều GPU.
- **Quy trình:** GPU 1 sẽ xử lý các lớp đầu tiên (ví dụ: 0-11), sau đó chuyển kết quả đầu ra cho GPU 2 để xử lý các lớp tiếp theo (ví dụ: 12-23), và cứ thế tiếp tục. Nó hoạt động giống như một dây chuyền lắp ráp.
- **Hạn chế:** Có thể gây ra "bong bóng" (bubbles), tức là thời gian chờ khi các GPU ở đầu đường ống phải đợi các GPU ở cuối hoàn thành công việc.

Trong thực tế, các hệ thống huấn luyện LLM quy mô lớn thường kết hợp cả ba loại song song này (cùng với các kỹ thuật khác) để đạt được hiệu quả cao nhất.

### 2.4.2. Sử dụng ‘accelerate’ và ‘DeepSpeed’ để Huấn luyện Mô hình lớn

Việc triển khai các chiến lược song song này từ đầu là cực kỳ phức tạp. May mắn thay, chúng ta có các thư viện mạnh mẽ giúp tự động hóa quá trình này.

#### Accelerate: Giao diện Tối giản

Như đã giới thiệu ở mục 2.1.3, ‘accelerate’ cung cấp một giao diện đơn giản để chạy code PyTorch trên nhiều môi trường phần cứng.

##### • Cách hoạt động:

1. Bạn viết một kịch bản huấn luyện PyTorch tiêu chuẩn.
2. Chạy lệnh ‘accelerate config’ trong terminal để thiết lập môi trường của bạn (ví dụ: “sử dụng 2 GPU trên máy này”, “sử dụng precision fp16”).
3. Chạy kịch bản của bạn bằng lệnh ‘accelerate launch your\_script.py’.

- Tích hợp với ‘Trainer’: ‘Trainer’ của Hugging Face đã tích hợp sẵn ‘accelerate’. Khi bạn chạy một kịch bản ‘Trainer’ với ‘accelerate launch’, nó sẽ tự động áp dụng chiến lược Data Parallelism (DP) trên các GPU có sẵn.
- ‘accelerate’ là lựa chọn tuyệt vời để dễ dàng song song hóa dữ liệu. Nhưng khi mô hình trở nên quá lớn, chúng ta cần một công cụ mạnh mẽ hơn.

### DeepSpeed: Cỗ máy Tối ưu hóa Toàn diện

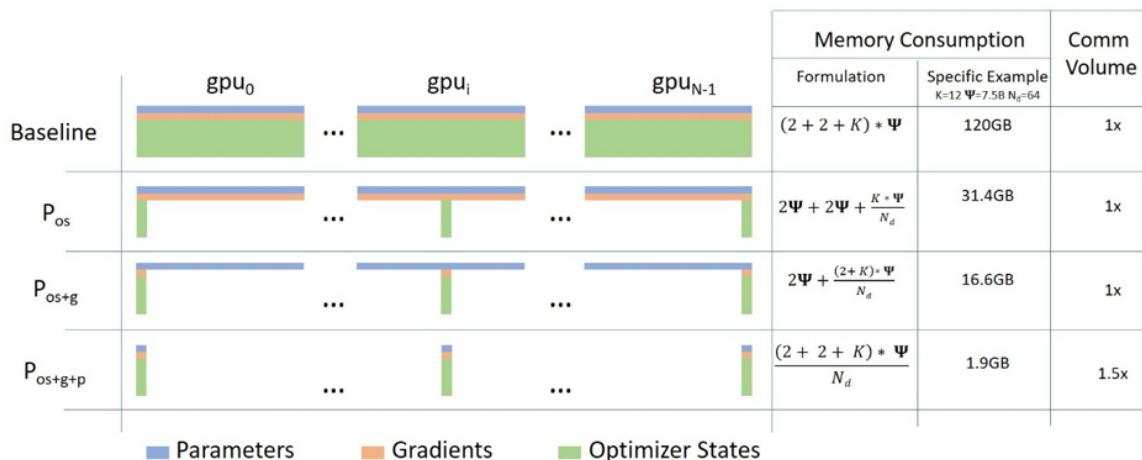
DeepSpeed là một thư viện mã nguồn mở từ Microsoft, cung cấp một bộ công cụ tối ưu hóa cực kỳ mạnh mẽ để huấn luyện các mô hình khổng lồ. Nó không chỉ là một chiến lược song song, mà là một hệ thống toàn diện.

#### ZeRO: Tối ưu hóa Bộ nhớ Cách mạng

Phát kiến cốt lõi của DeepSpeed là ZeRO (Zero Redundancy Optimizer). ZeRO nhận ra rằng trong Data Parallelism thông thường, mỗi GPU đều lưu trữ một cách thừa thãi các bản sao của **trọng số mô hình, gradient, và trạng thái optimizer**. ZeRO loại bỏ sự thừa thãi này bằng cách **phân mảnh (partitioning)** các thành phần này ra tất cả các GPU.

#### Các Giai đoạn của ZeRO

- ZeRO-1:** Phân mảnh **trạng thái optimizer**. Mỗi GPU chỉ giữ một phần của trạng thái optimizer. Giảm bộ nhớ đáng kể.
- ZeRO-2:** Phân mảnh cả **trạng thái optimizer và gradient**. Mỗi GPU chỉ tính và lưu trữ một phần của gradient. Giảm bộ nhớ nhiều hơn nữa.
- ZeRO-3:** Phân mảnh tất cả: **trạng thái optimizer, gradient, và cả trọng số mô hình**. Mỗi GPU chỉ giữ một lát cắt (slice) của mô hình. Trong quá trình truyền thẳng/ngược, các GPU sẽ giao tiếp với nhau để tập hợp lại các lớp cần thiết ngay khi cần. Đây là giai đoạn mạnh mẽ nhất, cho phép huấn luyện các mô hình nghìn tỷ tham số.



Hình 2.3: So sánh việc sử dụng bộ nhớ trong Data Parallelism tiêu chuẩn và các giai đoạn của ZeRO. ZeRO giảm đáng kể sự thừa thãi bằng cách phân mảnh các thành phần của quá trình huấn luyện.

**Tích hợp DeepSpeed với ‘Trainer’** Sức mạnh của hệ sinh thái Hugging Face là việc tích hợp các công cụ phức tạp này trở nên rất đơn giản.

- Tạo file cấu hình DeepSpeed:** Tạo một file JSON (ví dụ: ‘ds\_config.json’) để định nghĩa các thiết lập bạn muốn sử dụng (ví dụ: bật ZeRO-2, sử dụng precision fp16, thiết lập optimizer).
- Truyền vào ‘TrainingArguments’:**

## Cấu hình DeepSpeed

baselinestretch

```

1      training_args = TrainingArguments(
2          output_dir=".//results",
3          # ...
4          deepspeed="path/to/your/ds_config.json", # Chỉ
    ↳ định file config
5      )

```

3. **Khởi chạy:** Sử dụng ‘deepspeed your\_script.py’ hoặc ‘accelerate launch –use\_deepspeed your\_script.py’ để chạy kịch bản.

‘Trainer’ và ‘accelerate’ sẽ tự động xử lý tất cả các chi tiết phức tạp của việc khởi tạo và chạy DeepSpeed.

Ngoài ZeRO, DeepSpeed còn cung cấp nhiều tính năng tối ưu hóa khác như **Offloading** (chuyển bớt các tham số không dùng đến từ VRAM sang CPU RAM hoặc NVMe) và các kernel hiệu năng cao. Bằng cách kết hợp các công cụ như ‘accelerate’ và ‘DeepSpeed’, việc huấn luyện các mô hình ngôn ngữ lớn không còn là đặc quyền của một vài phòng thí nghiệm AI lớn, mà đã trở nên dễ tiếp cận hơn với cộng đồng nghiên cứu và các doanh nghiệp.

## KẾT THÚC CHƯƠNG 2

*Chương 2 đã trang bị cho bạn một bộ công cụ thực chiến hoàn chỉnh, từ việc làm quen với hệ sinh thái Hugging Face, xây dựng các quy trình đánh giá đáng tin cậy, đến việc quản lý các thí nghiệm một cách khoa học và huấn luyện các mô hình trên quy mô lớn. Với những kỹ năng này, bạn đã sẵn sàng để đổi mới với những thách thức phức tạp hơn trong việc triển khai và vận hành các hệ thống NLP trong môi trường production, chủ đề của chương tiếp theo.*

## CHƯƠNG 2. HUẤN LUYỆN VÀ ĐÁNH GIÁ MÔ HÌNH

## Chương 3

# TỐI UƯU HÓA, TRIỂN KHAI VÀ VẬN HÀNH (MLOps)

Hành trình của một mô hình học máy không kết thúc ở epoch huấn luyện cuối cùng hay ở con số 95% F1-score trên tập kiểm tra. Giá trị thực sự của nó chỉ được hiện thực hóa khi nó được đưa vào một sản phẩm, phục vụ hàng nghìn, thậm chí hàng triệu người dùng, và hoạt động một cách đáng tin cậy 24/7. Chào mừng bạn đến với thế giới của MLOps – Vận hành Học máy.

Trong chương này, chúng ta sẽ vượt ra khỏi khuôn khổ của sổ tay Jupyter và đối mặt với những thách thức của thế giới thực. Làm thế nào để phục vụ một mô hình 70 tỷ tham số với độ trễ thấp? Làm thế nào để đóng gói một ứng dụng AI phức tạp để nó có thể chạy ở bất cứ đâu? Làm thế nào để giám sát "sức khỏe" của mô hình sau khi triển khai và biết khi nào cần huấn luyện lại nó?

Chúng ta sẽ khám phá các thành phần cơ sở hạ tầng quan trọng như Cơ sở dữ liệu Vector, các thư viện tối ưu hóa mô hình, các framework phục vụ chuyên dụng, và cuối cùng là phác thảo một vòng đời MLOps hoàn chỉnh cho LLM. Đây là những kỹ năng sẽ biến bạn từ một nhà khoa học dữ liệu thành một kỹ sư học máy toàn diện.

### 3.1. Cơ sở dữ liệu Vector (Vector Databases)

#### 3.1.1. Khái niệm và vai trò trong NLP hiện đại

Trong các chương trước, chúng ta đã thấy rằng "embedding" là ngôn ngữ của NLP hiện đại. Mọi thứ, từ một từ, một câu, đến một tài liệu hoàn chỉnh, đều có thể được biểu diễn bằng một vector dày đặc. Các hệ thống như RAG (mục 6.1) và Bộ nhớ Dài hạn cho Tác tử (mục 6.4.3) đều dựa trên một thao tác cốt lõi: \*\*\*"Cho một vector truy vấn, hãy tìm các vector tương tự nhất trong một tập hợp hàng triệu, thậm chí hàng tỷ vector khác."\*\*\*

**Vấn đề của Tìm kiếm Brute-force** Cách tiếp cận ngây thơ nhất là tính toán độ tương đồng (ví dụ: cosine similarity) giữa vector truy vấn và *tất cả* các vector trong cơ sở dữ liệu, sau đó sắp xếp và lấy ra top-K kết quả. Phương pháp này có độ phức tạp  $O(N \cdot d)$  (với  $N$  là số vector,  $d$  là số chiều). Với  $N$  lớn, nó trở nên quá chậm để có thể sử dụng trong thực tế.

**Giải pháp: Approximate Nearest Neighbor (ANN) Search** Cơ sở dữ liệu Vector không thực hiện tìm kiếm chính xác. Thay vào đó, chúng sử dụng các thuật toán **Tìm kiếm Láng giềng Gần nhất Xấp xi** (Approximate Nearest Neighbor - ANN).

#### Triết lý của ANN

Thay vì tìm kiếm các láng giềng gần nhất **chính xác tuyệt đối** với chi phí cao, chúng ta có thể tìm thấy các láng giềng **gần như chính xác** (ví dụ: 99% đúng) với tốc độ nhanh hơn hàng nghìn lần. Sự đánh đổi nhỏ về độ chính xác này là hoàn toàn chấp nhận được trong hầu hết các ứng dụng NLP.

Các thuật toán ANN như HNSW (Hierarchical Navigable Small World) xây dựng các cấu trúc dữ liệu giống như đồ thị thông minh trong giai đoạn lập chỉ mục, cho phép việc tìm kiếm diễn ra cực kỳ nhanh chóng bằng cách điều hướng qua đồ thị thay vì quét toàn bộ.

Một Cơ sở dữ liệu Vector là một hệ thống được tối ưu hóa để lưu trữ, lập chỉ mục, và truy vấn các vector embedding quy mô lớn bằng các thuật toán ANN.

### 3.1.2. Các công cụ phổ biến

- Faiss (Facebook AI Similarity Search):
  - **Bản chất:** Là một thư viện (library), không phải một cơ sở dữ liệu hoàn chỉnh. Nó cung cấp các thuật toán ANN cốt lõi, hiệu năng cực cao, được viết bằng C++ và có giao diện Python.
  - **Khi nào dùng?** Khi bạn cần hiệu năng tìm kiếm vector cao nhất có thể và sẵn sàng tự xây dựng các lớp dịch vụ (API, quản lý dữ liệu) xung quanh nó. Rất phổ biến trong môi trường nghiên cứu và các hệ thống tự xây dựng.
- ChromaDB:
  - **Bản chất:** Một cơ sở dữ liệu vector mã nguồn mở, tập trung vào sự đơn giản và dễ sử dụng. Nó có thể chạy trực tiếp trong ứng dụng Python của bạn ("in-process") mà không cần một máy chủ riêng.
  - **Khi nào dùng?** Rất tuyệt vời cho việc phát triển nhanh, tạo mẫu (prototyping), và các ứng dụng quy mô nhỏ. Được mệnh danh là "SQLite của các vector".
- Milvus / Weaviate:
  - **Bản chất:** Là các cơ sở dữ liệu vector mã nguồn mở, đầy đủ tính năng, được thiết kế cho các hệ thống production quy mô lớn. Chúng hoạt động như các dịch vụ độc lập.
  - **Tính năng nâng cao:** Hỗ trợ lọc metadata (ví dụ: "tìm các vector tương tự nhưng chỉ trong các tài liệu được tạo sau ngày X"), khả năng mở rộng (scaling), và độ sẵn sàng cao (high availability).
  - **Khi nào dùng?** Khi xây dựng các ứng dụng RAG hoặc tìm kiếm ngữ nghĩa cho production.
- Pinecone:
  - **Bản chất:** Một dịch vụ cơ sở dữ liệu vector được quản lý hoàn toàn (fully-managed) trên đám mây (SaaS).
  - **Lợi ích:** Bạn không cần phải lo lắng về việc thiết lập, bảo trì, hay mở rộng cơ sở hạ tầng. Nó cung cấp một API đơn giản để sử dụng và được tối ưu hóa cho hiệu năng cao và độ trễ thấp.
  - **Khi nào dùng?** Khi bạn muốn có một giải pháp vector search mạnh mẽ mà không cần quản lý cơ sở hạ tầng.

## 3.2. Tối ưu hóa Mô hình với các Thư viện Sẵn có

Như đã thảo luận trong Phần 1 (mục 6.2), các kỹ thuật như Lượng tử hóa và PEFT là rất quan trọng. Trong phần này, chúng ta sẽ xem xét các thư viện cụ thể giúp bạn áp dụng các kỹ thuật này một cách dễ dàng.

### bitsandbytes

- **Mục đích chính: Lượng tử hóa (Quantization).**
- **Tính năng nổi bật:** Cung cấp các kernel CUDA được tối ưu hóa cao để thực hiện các phép toán trên các định dạng số có độ chính xác thấp. Nó là thư viện đứng sau thành công của QLoRA, cho phép lượng tử hóa 8-bit và 4-bit (bao gồm cả định dạng NF4) một cách hiệu quả.
- **Cách sử dụng (với 'transformers'):** Việc lượng tử hóa một mô hình khi tải về trở nên cực kỳ đơn giản.

## Ví dụ 26: Tải mô hình với lượng tử hóa 4-bit

baselinestretch

```

1  from transformers import AutoModelForCausalLM
2  import torch
3
4  model_id = "meta-llama/Llama-2-7b-hf"
5
6  # Tải mô hình với lượng tử hóa 4-bit
7  model = AutoModelForCausalLM.from_pretrained(
8      model_id,
9      load_in_4bit=True,
10     device_map="auto", # Tự động phân bổ các lớp lên
11     ↪ GPU/CPU
12 )

```

'bitsandbytes' giúp giảm đáng kể yêu cầu VRAM, là một công cụ không thể thiếu để làm việc với các LLM lớn trên phần cứng hạn chế.

## PEFT (Parameter-Efficient Fine-Tuning)

- **Mục đích chính:** Triển khai các kỹ thuật PEFT.
- **Tính năng nổi bật:** Là một thư viện của Hugging Face, cung cấp các cách triển khai tiêu chuẩn cho nhiều phương pháp PEFT, nổi bật nhất là LoRA (và do đó là QLoRA), Prompt-Tuning, Prefix-Tuning, và Adapter-tuning.
- **Cách sử dụng (LoRA với 'transformers'):**

## Ví dụ 27: Sử dụng LoRA với PEFT

baselinestretch

```

1  from peft import LoraConfig, get_peft_model
2  from transformers import AutoModelForCausalLM
3
4  # Tải mô hình gốc
5  model = AutoModelForCausalLM.from_pretrained("meta-llama/L_
6    ↪ lama-2-7b-hf")
7
8  # Định nghĩa cấu hình LoRA
9  config = LoraConfig(
10     r=8, # Thứ hạng (rank)
11     lora_alpha=32,
12     target_modules=["q_proj", "v_proj"], # Áp dụng LoRA vào
13     ↪ lớp nào
14     lora_dropout=0.05,
15     task_type="CAUSAL_LM",
16 )
17
18  # Bọc mô hình gốc bằng PEFT
19  peft_model = get_peft_model(model, config)
20
21  # Bây giờ, khi huấn luyện `peft_model`, chỉ các tham số
22  ↪ LoRA
23  # mới được cập nhật.

```

```

21 peft_model.print_trainable_parameters()
22 # Output: trainable params: 4,194,304 || all params:
→ 6,742,609,920 || trainable%: 0.0622

```

### ONNX (Open Neural Network Exchange)

- **Mục đích chính:** Khả chuyển (Portability) và Tăng tốc Suy luận (Inference Acceleration).
- **Bản chất:** ONNX không phải là một thư viện tối ưu hóa trực tiếp, mà là một định dạng file trung gian, tiêu chuẩn mở. Bạn có thể "export" một mô hình từ một framework (như PyTorch) sang định dạng ONNX.
- **Lợi ích:**
  - **Khả chuyển:** Một mô hình ONNX có thể được chạy trên nhiều nền tảng phần cứng và phần mềm khác nhau (web, mobile, server) bằng cách sử dụng một ONNX Runtime.
  - **Tăng tốc:** Các ONNX Runtime (ví dụ: ONNX Runtime của Microsoft) thường được tối ưu hóa cao độ. Chúng có thể áp dụng các kỹ thuật như "graph fusion" (gộp nhiều phép toán nhỏ thành một phép toán lớn) và lượng tử hóa để chạy mô hình nhanh hơn đáng kể so với môi trường PyTorch gốc.

## 3.3. Đóng gói và Phục vụ Mô hình

Sau khi đã có một mô hình đã được tối ưu hóa, làm thế nào để biến nó thành một dịch vụ mà các ứng dụng khác có thể sử dụng?

### 3.3.1. Xây dựng API với FastAPI

- **Vấn đề:** Chúng ta cần một cách để "gói" mô hình của mình lại và cung cấp một giao diện web (API) để nhận đầu vào và trả về dự đoán.
- **FastAPI là gì?** Là một framework web Python hiện đại, hiệu năng cao, được thiết kế đặc biệt để xây dựng các API.
- **Tại sao lại phô biến cho ML?**
  - **Hiệu năng cao:** Dựa trên Starlette (cho phần web) và Pydantic (cho phần xác thực dữ liệu), FastAPI có hiệu năng tương đương với NodeJS và Go.
  - **Hỗ trợ bất đồng bộ (Async):** Rất quan trọng để xử lý nhiều yêu cầu cùng lúc một cách hiệu quả.
  - **Tự động tạo tài liệu (Docs):** Tự động tạo ra các trang tài liệu API tương tác (Swagger UI và ReDoc), giúp việc kiểm thử và tích hợp trở nên cực kỳ dễ dàng.

#### Ví dụ 28: Một API phân loại văn bản đơn giản với FastAPI

baselinestretch

```

1 # File: main.py
2 from fastapi import FastAPI
3 from pydantic import BaseModel
4 from transformers import pipeline
5
6 # Khởi tạo ứng dụng FastAPI
7 app = FastAPI()
8
9 # Tải mô hình (nên thực hiện một lần khi khởi động)
10 classifier = pipeline("sentiment-analysis")
11

```

```

12 # Định nghĩa cấu trúc dữ liệu đầu vào
13 class TextInput(BaseModel):
14     text: str
15
16 # Định nghĩa API endpoint
17 @app.post("/predict/")
18 def predict_sentiment(item: TextInput):
19     # Lấy văn bản từ request
20     text_to_analyze = item.text
21     # Thực hiện dự đoán
22     result = classifier(text_to_analyze)
23     # Trả về kết quả
24     return result[0]
25
26 # Để chạy: uvicorn main:app --reload

```

### 3.3.2. Container hóa với Docker

- Vấn đề:** Ứng dụng FastAPI của bạn chạy tốt trên máy của bạn, nhưng khi triển khai lên một máy chủ khác, nó có thể bị lỗi do thiếu thư viện, sai phiên bản Python, hoặc các vấn đề môi trường khác.
- Docker là gì?** Là một nền tảng cho phép bạn "đóng gói" ứng dụng của mình và tất cả các thành phần phụ thuộc của nó (thư viện, file hệ thống, biến môi trường) vào một đơn vị tiêu chuẩn, có thể di động, gọi là **container**.
- Lợi ích:** "Build once, run anywhere." Một container sẽ chạy giống hệt nhau trên máy tính cá nhân, máy chủ nội bộ, hay trên đám mây. Nó giải quyết triệt để "nó chạy được trên máy của tôi mà!".

Một Dockerfile đơn giản cho ứng dụng FastAPI

```

baselinestretch

# File: Dockerfile
# 1. Sử dụng một image Python chính thức làm nền
FROM python:3.9-slim
# 2. Đặt thư mục làm việc bên trong container
WORKDIR /app
# 3. Sao chép file requirements vào trước để tận dụng cache
COPY requirements.txt .
# 4. Cài đặt các thư viện phụ thuộc
RUN pip install --no-cache-dir -r requirements.txt
# 5. Sao chép toàn bộ mã nguồn của ứng dụng vào
COPY . .
# 6. Lệnh sẽ được chạy khi container khởi động
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]

```

Với file này, bạn có thể xây dựng một image ('docker build') và chạy ứng dụng của mình ('docker

run') ở bất cứ đâu có Docker.

### 3.4. Các Framework Phục vụ Chuyên dụng cho Production

FastAPI và Docker là rất tốt, nhưng khi phục vụ các LLM quy mô lớn với yêu cầu về thông lượng (throughput) cao và độ trễ (latency) thấp, chúng ta cần các công cụ chuyên dụng hơn.

**Vấn đề của việc Phục vụ LLM** Một trong những thách thức lớn nhất là quản lý bộ nhớ của KV Cache. Trong quá trình sinh văn bản tự hồi quy, các giá trị Key và Value của các token đã được sinh ra sẽ được lưu lại (cache) để không phải tính toán lại ở các bước sau. KV Cache này chiếm một lượng VRAM khổng lồ và việc quản lý nó cho các yêu cầu theo batch một cách hiệu quả là rất khó.

#### vLLM: Tối ưu hóa cho Thông lượng

- **Mục đích chính:** Một thư viện phục vụ LLM cực kỳ nhanh, được tối ưu hóa cho thông lượng cao.
- **Phát kiến cốt lõi:** PagedAttention. Lấy cảm hứng từ cơ chế phân trang (paging) trong các hệ điều hành, PagedAttention quản lý KV Cache trong các "trang"(pages) bộ nhớ không liền kề.
- **Lợi ích của PagedAttention:**
  - **Giảm lãng phí bộ nhớ:** Loại bỏ gán như hoàn toàn sự phân mảnh bộ nhớ bên trong và lãng phí do phải cấp phát bộ nhớ dự phòng.
  - **Batching hiệu quả hơn:** Cho phép các thuật toán batching thông minh hơn, có thể xử lý nhiều chuỗi hơn cùng lúc.
- **Kết quả:** vLLM có thể đạt được thông lượng cao hơn nhiều lần (lên đến 24x) so với các cách triển khai tiêu chuẩn (như của Hugging Face) mà không cần thay đổi mô hình.

#### BentoML: Chuẩn hóa Vòng đời Phục vụ

- **Mục đích chính:** Cung cấp một framework toàn diện để đóng gói, quản lý, và triển khai các dịch vụ AI một cách tiêu chuẩn hóa và có thể tái sử dụng.
- **Triết lý:** Tách biệt logic nghiệp vụ (business logic) khỏi logic phục vụ mô hình (model serving logic).
- **Các thành phần:**
  - **Bentos:** Một định dạng đóng gói tiêu chuẩn ("build target") chứa mã nguồn, các mô hình, các file phụ thuộc, và cấu hình. Một "Bento" có thể được build thành một container Docker hoặc triển khai lên các nền tảng serverless.
  - **Runners:** Một lớp trừu tượng để chạy các mô hình, có khả năng tối ưu hóa cho các loại phần cứng khác nhau.
  - **Adapters:** Xử lý việc chuyển đổi giữa dữ liệu thô (HTTP requests) và các định dạng mà mô hình có thể hiểu (tensor, DataFrame).
- **Khi nào dùng?** Khi bạn cần xây dựng các dịch vụ AI phức tạp, có thể bao gồm nhiều mô hình, và muốn có một quy trình tiêu chuẩn để quản lý và triển khai chúng.

#### NVIDIA Triton Inference Server

- **Mục đích chính:** Cung cấp một máy chủ suy luận (inference server) hiệu năng cao, được tối ưu hóa cho phần cứng NVIDIA.
- **Điểm mạnh:**
  - **Đa framework:** Hỗ trợ phục vụ các mô hình từ nhiều framework khác nhau (TensorFlow, PyTorch, ONNX, TensorRT) trong cùng một máy chủ.
  - **Dynamic Batching:** Tự động gộp các yêu cầu riêng lẻ đến theo thời gian thực thành các batch để tối đa hóa hiệu suất của GPU.

- **Tối ưu hóa cho GPU:** Tận dụng tối đa các tính năng của GPU NVIDIA.
- **Khi nào dùng?** Trong các môi trường production có yêu cầu rất cao về hiệu năng và độ trễ, và sử dụng chủ yếu là GPU NVIDIA.

### 3.5. Vòng đời MLOps cho LLM

Triển khai mô hình không phải là điểm kết thúc. Để một ứng dụng AI thành công trong dài hạn, nó cần phải được giám sát, bảo trì, và cải tiến liên tục. Đây là vòng đời MLOps.

#### 3.5.1. Giám sát mô hình (Model Monitoring)

Sau khi triển khai, chúng ta cần theo dõi hai loại vấn đề chính:

##### 1. Giám sát Hệ thống (System Monitoring)

- **Mục tiêu:** Đảm bảo dịch vụ API đang hoạt động ổn định.
- **Các metric cần theo dõi:**
  - **Độ trễ (Latency):** Thời gian để xử lý một yêu cầu.
  - **Thông lượng (Throughput):** Số lượng yêu cầu được xử lý mỗi giây.
  - **Tỷ lệ lỗi (Error Rate):** Tỷ lệ các yêu cầu trả về lỗi (ví dụ: HTTP 500).
  - **Sử dụng tài nguyên:** Mức sử dụng CPU, GPU, RAM.

##### 2. Giám sát Chất lượng Mô hình (Model Quality Monitoring)

- **Mục tiêu:** Đảm bảo mô hình vẫn đang đưa ra các dự đoán tốt trong thế giới thực.
- **Thách thức:** Chúng ta thường không có "nhân thật" ngay lập tức để so sánh.
- **Phát hiện Trôi dạt Dữ liệu (Data Drift):**
  - **Vấn đề:** Phân phối của dữ liệu trong môi trường production (live data) bắt đầu khác biệt so với phân phối của dữ liệu đã được dùng để huấn luyện mô hình. Ví dụ, một mô hình phân tích cảm xúc được huấn luyện trên các bình luận về phim ảnh có thể hoạt động kém đi khi người dùng bắt đầu bình luận về các sự kiện chính trị.
  - **Cách phát hiện:** So sánh các thuộc tính thống kê giữa dữ liệu live và dữ liệu huấn luyện. Đối với văn bản, chúng ta có thể so sánh phân phối của các vector embedding. Nếu embedding của dữ liệu live bắt đầu trôi dạt ra xa khỏi embedding của dữ liệu huấn luyện, đó là một dấu hiệu mạnh mẽ của data drift.
- **Phát hiện Trôi dạt Khái niệm (Concept Drift):**
  - **Vấn đề:** Mỗi quan hệ giữa đầu vào và đầu ra thay đổi. Ví dụ, ý nghĩa của từ "corona" đã thay đổi hoàn toàn sau năm 2020.
  - **Cách phát hiện:** Thường khó phát hiện hơn và yêu cầu phải có một luồng phản hồi từ người dùng hoặc gán nhãn lại một phần dữ liệu live.

#### 3.5.2. Quy trình Tái huấn luyện và Cập nhật Mô hình

Khi việc giám sát cho thấy hiệu năng của mô hình đang suy giảm, chúng ta cần một quy trình để cập nhật nó.

- **Thu thập Dữ liệu mới và Phản hồi:** Thiết lập một "vòng lặp phản hồi" (feedback loop) để thu thập dữ liệu mới từ môi trường production, đặc biệt là các trường hợp mà mô hình dự đoán sai.
- **Gán nhãn lại (Re-labeling):** Gán nhãn cho bộ dữ liệu mới này.
- **Tái huấn luyện (Re-training):** Huấn luyện lại mô hình trên một bộ dữ liệu kết hợp giữa dữ liệu cũ và dữ liệu mới.
- **Đánh giá và So sánh:** Đánh giá mô hình mới trên một tập kiểm tra được giữ riêng. Chỉ triển khai mô hình mới nếu nó thực sự hoạt động tốt hơn mô hình cũ.

- **Triển khai theo chiến lược an toàn:**

- **Shadow Deployment:** Triển khai mô hình mới song song với mô hình cũ, chỉ để ghi lại dự đoán của nó mà không ảnh hưởng đến người dùng, nhằm so sánh hiệu năng.
- **Canary Release / A/B Testing:** Dần dần chuyển một phần nhỏ traffic của người dùng sang mô hình mới và theo dõi chặt chẽ các metric kinh doanh trước khi chuyển toàn bộ.

Vòng đời MLOps này biến việc phát triển mô hình từ một dự án một lần thành một quy trình liên tục, đảm bảo rằng ứng dụng AI của bạn luôn được cải tiến và thích ứng với thế giới đang thay đổi.

---

## KẾT THÚC CHƯƠNG 3

*Chương 3 đã đưa chúng ta vào hành trình từ một mô hình đã được huấn luyện đến một dịch vụ AI hoàn chỉnh, hoạt động trong môi trường production. Chúng ta đã học cách tối ưu hóa, đóng gói, phục vụ và giám sát các mô hình ngôn ngữ lớn. Việc trang bị các kỹ năng MLOps này là cực kỳ quan trọng, đảm bảo rằng các sản phẩm AI không chỉ thông minh mà còn mạnh mẽ, có khả năng mở rộng và bền vững. Giờ đây, bạn đã sẵn sàng để kết hợp tất cả các kiến thức đã học và áp dụng chúng vào việc xây dựng các ứng dụng cụ thể trong chương cuối cùng của giáo trình.*

## Chương 4

# CÔNG THỨC XÂY DỰNG CÁC ỨNG DỤNG PHỐ BIỂN (RECIPES)

Chào mừng bạn đến với chương cuối cùng, chương thực hành và tổng hợp nhất của giáo trình. Sau khi đã đi qua một hành trình dài từ lý thuyết nền tảng đến các công cụ MLOps chuyên nghiệp, đây là lúc chúng ta xắn tay áo lên và thực sự "nấu ăn".

Chương này được thiết kế như một cuốn sách công thức, nơi mỗi mục là một hướng dẫn chi tiết, từng bước để xây dựng một ứng dụng NLP hiện đại và phổ biến. Từ việc xây dựng một công cụ tìm kiếm ngữ nghĩa, một hệ thống RAG thông minh, cho đến việc fine-tune các LLM cho các tác vụ chuyên biệt và tạo ra các Tác tử AI tự hành – mỗi công thức sẽ là một dự án nhỏ, kết tinh lại những kiến thức bạn đã học.

Hãy coi đây là phòng thí nghiệm của bạn, nơi các khái niệm trừu tượng biến thành các sản phẩm cụ thể. Sau khi hoàn thành chương này, bạn sẽ không chỉ "biết" về NLP, mà sẽ thực sự "làm" được NLP.

### 4.1. Xây dựng Hệ thống Tìm kiếm Ngữ nghĩa

**Mục tiêu:** Xây dựng một hệ thống đơn giản có khả năng tìm kiếm các câu hoặc đoạn văn bản tương đồng về mặt ý nghĩa, thay vì chỉ khớp từ khóa. Đây là khối xây dựng cơ bản cho RAG và nhiều ứng dụng khác.

**Thành phần chính:**

- **sentence-transformers:** Thư viện dựa trên `transformers`, cung cấp các mô hình embedding chất lượng cao cho câu/đoạn văn.
- **chromadb:** Cơ sở dữ liệu vector mã nguồn mở, cho phép lưu trữ và tìm kiếm embedding hiệu quả.

**Các bước thực hiện:**

1. Chuẩn bị dữ liệu: Danh sách các câu/đoạn văn bản để lập chỉ mục.
2. Tải mô hình embedding: Chọn và tải một mô hình từ `sentence-transformers`.
3. Lập chỉ mục (Indexing): Chuyển đổi văn bản thành vector embedding và lưu vào ChromaDB.
4. Truy vấn (Querying): Nhúng câu truy vấn và tìm kiếm các vector tương tự trong ChromaDB.

**Mã nguồn:**

```
pip install sentence-transformers chromadb
```

**Ví dụ 29:** Xây dựng hệ thống tìm kiếm ngữ nghĩa với Sentence-Transformers và ChromaDB

```
baselinestretch
```

```
1 from sentence_transformers import SentenceTransformer  
2 import chromadb
```

```

3
4 # Bước 1: Chuẩn bị dữ liệu và tải mô hình
5 model = SentenceTransformer('all-MiniLM-L6-v2')
6
7 documents = [
8     "Trái Đất quay quanh Mặt Trời.",
9     "Hệ Mặt Trời có tám hành tinh.",
10    "Mặt Trăng là vệ tinh tự nhiên của Trái Đất.",
11    "Python là một ngôn ngữ lập trình phổ biến.",
12    "Học sâu là một nhánh của học máy."
13 ]
14
15 # Bước 2: Lập chỉ mục (Embed và Store)
16 client = chromadb.Client()
17 collection = client.create_collection("semantic_search_demo")
18
19 embeddings = model.encode(documents)
20 collection.add(
21     embeddings=embeddings.tolist(),
22     documents=documents,
23     ids=[f"doc_{i}" for i in range(len(documents))])
24
25 print("Đã lập chỉ mục xong!")
26
27 # Bước 3: Truy vấn
28 query = "Các thiên thể trong vũ trụ"
29 print(f"\nTruy vấn: '{query}'")
30
31 query_embedding = model.encode(query).tolist()
32 results = collection.query(
33     query_embeddings=[query_embedding],
34     n_results=3
35 )
36
37 print("Các kết quả tìm kiếm tương tự nhất:")
38 for doc in results['documents'][0]:
39     print(f"- {doc}")

```

**Kết quả mong đợi:** Hệ thống sẽ trả về 3 câu đầu tiên về thiên văn học vì chúng tương đồng về mặt ngữ nghĩa với truy vấn, mặc dù không có nhiều từ khóa trùng lặp. Nó sẽ bỏ qua 2 câu về lập trình và học máy.

## 4.2. Xây dựng Hệ thống Hỏi-Đáp trên Tài liệu (RAG)

**Mục tiêu:** Xây dựng một hệ thống Hỏi-Đáp có thể trả lời các câu hỏi dựa trên nội dung của một tài liệu cụ thể. Phần này xây dựng trực tiếp dựa trên phần tìm kiếm ngữ nghĩa trước đó.

Thành phần chính:

- **LangChain / LlamaIndex:** Các framework cấp cao giúp đơn giản hóa việc xây dựng ứng dụng dựa trên LLM, đặc biệt là RAG. Ở đây dùng LangChain.
- **Retriever:** Thành phần tìm kiếm ngữ nghĩa (giống phần trước).
- **LLM (Generator):** Một mô hình ngôn ngữ lớn (ví dụ gpt-3.5-turbo qua API hoặc mô hình mã nguồn mở) để sinh câu trả lời cuối cùng.

Các bước thực hiện:

1. **Tải và Phân mảnh Dữ liệu:** Tải tài liệu dài và chia thành các *chunk* nhỏ.

2. Thiết lập Vector Store: Dùng mô hình embedding và cơ sở dữ liệu vector để lập chỉ mục các *chunk*.
3. Thiết lập LLM: Khởi tạo LLM sẽ sử dụng.
4. Xây dựng Chuỗi RAG (RAG Chain): Kết nối Retriever và LLM qua một `PromptTemplate`.
5. Hỏi đáp: Đặt câu hỏi và nhận câu trả lời.

Mã nguồn:

Ví dụ 30: RAG cơ bản với LangChain + Chroma

baselinestretch

```

1  # Bước 1: Cài đặt
2  #pip install langchain langchain-openai langchain-community
   ↪ sentence-transformers chromadb
3  import os
4  # os.environ["OPENAI_API_KEY"] = "sk-..."
5
6  from langchain_community.document_loaders import WebBaseLoader
7  from langchain_community.vectorstores import Chroma
8  from langchain_community.embeddings import
   ↪ SentenceTransformerEmbeddings
9  from langchain_text_splitters import
   ↪ RecursiveCharacterTextSplitter
10 from langchain_openai import ChatOpenAI
11 from langchain.prompts import PromptTemplate
12 from langchain.schema.runnable import RunnablePassthrough
13 from langchain.schema.output_parser import StrOutputParser
14
15 # Bước 2: Tải và phân mảnh dữ liệu
16 loader = WebBaseLoader(web_path="https://en.wikipedia.org/wiki/_"
   ↪ Large_language_model")
17 docs = loader.load()
18 text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
   ↪ chunk_overlap=50)
19 splits = text_splitter.split_documents(docs)
20
21 # Bước 3: Thiết lập Vector Store (Retriever)
22 embedding_function =
   ↪ SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")
23 vectorstore = Chroma.from_documents(documents=splits,
   ↪ embedding=embedding_function)
24 retriever = vectorstore.as_retriever()
25
26 # Bước 4: Thiết lập LLM và Prompt Template
27 llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
28
29 template = """Sử dụng các đoạn ngữ cảnh sau đây để trả lời câu
   ↪ hỏi.
30 Nếu bạn không biết câu trả lời, hãy nói rằng bạn không biết.
31
32 Ngữ cảnh: {context}
33
34 Câu hỏi: {question}
35
36 Câu trả lời hữu ích:"""
37 rag_prompt = PromptTemplate.from_template(template)

```

```

38
39 # Bước 55: Xây dựng Chuỗi RAG
40 rag_chain = (
41     {"context": retriever, "question": RunnablePassthrough()}
42     | rag_prompt
43     | llm
44     | StrOutputParser()
45 )
46
47 # Bước 6: Hỏi đáp
48 query = "What is the Transformer architecture?"
49 answer = rag_chain.invoke(query)
50
51 print(f"Câu hỏi: {query}\n")
52 print(f"Câu trả lời: {answer}")

```

Kết quả mong đợi: Mô hình trả lời đúng về kiến trúc Transformer dựa trên thông tin truy xuất từ Wikipedia, thay vì chỉ dựa trên kiến thức nội tại.

### 4.3. Recipe 3: Fine-tuning một LLM cho Tác vụ Phân loại

**Mục tiêu:** Fine-tune một LLM lớn (ví dụ: Llama 2 7B) cho một tác vụ phân loại văn bản một cách hiệu quả về bộ nhớ bằng cách sử dụng QLoRA.

**Thành phần chính:**

- ‘transformers’: Sử dụng ‘AutoModelForSequenceClassification’ và ‘Trainer’ API.
- ‘datasets’: Để tải và xử lý bộ dữ liệu.
- ‘peft’: Để áp dụng cấu hình LoRA.
- ‘bitsandbytes’: Để lượng tử hóa mô hình sang 4-bit.

**Các bước thực hiện:**

1. **Tải dữ liệu và tokenizer:** Tải một bộ dữ liệu phân loại và tokenizer tương ứng.
2. **Tiền xử lý dữ liệu:** Tokenize văn bản.
3. **Tải mô hình cơ sở đã được lượng tử hóa:** Tải LLM với tùy chọn ‘load\_in\_4bit=True’.
4. **Áp dụng cấu hình PEFT (LoRA):** Bọc mô hình cơ sở bằng một ‘PeftModel’.
5. **Thiết lập ‘Trainer’:** Định nghĩa ‘TrainingArguments’ và ‘Trainer’, truyền vào hàm ‘compute\_metrics’.
6. **Huấn luyện:** Bắt đầu quá trình fine-tuning.

**Mã nguồn:**

#### Ví dụ 31: Fine-tuning một LLM cho Tác vụ Phân loại

baselinestretch

```

1 # Bước 1: Cài đặt
2 # pip install transformers datasets accelerate evaluate peft
   ↪ bitsandbytes
3 import torch
4 from datasets import load_dataset
5 from transformers import (
6     AutoModelForSequenceClassification,
7     AutoTokenizer,
8     TrainingArguments,
9     Trainer,
10    BitsAndBytesConfig
11 )

```

```

12  from peft import LoraConfig, get_peft_model
13  import numpy as np
14  import evaluate
15
16  # Bước 2: Tải dữ liệu và tokenizer
17  dataset = load_dataset("yelp_review_full")
18  model_id = "meta-llama/Llama-2-7b-hf" # Cần quyền truy cập
19  tokenizer = AutoTokenizer.from_pretrained(model_id)
20  tokenizer.pad_token = tokenizer.eos_token # Llama không có pad
21    ↵ token
22
23  def tokenize_function(examples):
24      return tokenizer(examples["text"], padding="max_length",
25                       ↵ truncation=True)
26
27  tokenized_datasets = dataset.map(tokenize_function,
28      ↵ batched=True)
29  # Giảm kích thước để chạy demo nhanh
30  small_train_dataset = tokenized_datasets["train"].shuffle(seed=_
31    ↵ 42).select(range(1000))
32  small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42_
33    ↵ ).select(range(1000))
34
35  # Bước 3: Tải mô hình cơ sở đã lượng tử hóa
36  bnb_config = BitsAndBytesConfig(
37      load_in_4bit=True,
38      bnb_4bit_quant_type="nf4",
39      bnb_4bit_compute_dtype=torch.float16,
40  )
41  model = AutoModelForSequenceClassification.from_pretrained(
42      model_id,
43      num_labels=5, # Yelp review có 5 sao
44      quantization_config=bnb_config,
45      device_map="auto"
46  )
47  model.config.pad_token_id = model.config.eos_token_id
48
49  # Bước 4: Áp dụng cấu hình PEFT (LoRA)
50  lora_config = LoraConfig(r=8, lora_alpha=16, lora_dropout=0.05,
51    ↵ task_type="SEQ_CLS")
52  peft_model = get_peft_model(model, lora_config)
53  peft_model.print_trainable_parameters()
54
55  # Bước 5: Thiết lập Trainer
56  metric = evaluate.load("accuracy")
57  def compute_metrics(eval_preds):
58      logits, labels = eval_preds
59      predictions = np.argmax(logits, axis=-1)
60      return metric.compute(predictions=predictions,
61                           ↵ references=labels)
62
63  training_args = TrainingArguments(
64      output_dir="llama-7b-lora-classification",
65      learning_rate=2e-4,
66      per_device_train_batch_size=4,
67      per_device_eval_batch_size=4,

```

```

61     num_train_epochs=1,
62     evaluation_strategy="epoch",
63     save_strategy="epoch",
64     load_best_model_at_end=True,
65 )
66
67     trainer = Trainer(
68         model=peft_model,
69         args=training_args,
70         train_dataset=small_train_dataset,
71         eval_dataset=small_eval_dataset,
72         compute_metrics=compute_metrics,
73         tokenizer=tokenizer,
74     )
75
76 # Bước 6: Huấn luyện
77 trainer.train()

```

**Kết quả mong đợi:** Một mô hình LLM lớn được fine-tune hiệu quả trên một GPU tiêu dùng. Chỉ có các trọng số LoRA (vài MB) được lưu lại, thay vì toàn bộ mô hình (vài chục GB).

#### 4.4. Recipe 4: Fine-tuning cho Tác vụ Nhận dạng Thực thể (NER)

**Mục tiêu:** Fine-tune một mô hình Encoder-Only (ví dụ: RoBERTa) cho tác vụ NER. Recipe này nhắm mạnh vào việc tiền xử lý dữ liệu phức tạp hơn cho các tác vụ gán nhãn chuỗi.

**Thành phần chính:**

- **transformers:** Sử dụng AutoModelForTokenClassification và Trainer.
- **datasets:** Tải bộ dữ liệu NER.
- **evaluate (với seqeval):** Để tính toán F1-score ở cấp độ thực thể.

**Các bước thực hiện:**

1. **Tải dữ liệu và tokenizer:** Tải bộ dữ liệu CoNLL-2003.
2. **Tiền xử lý và Căn chỉnh Nhãn:** Đây là bước phức tạp. Khi một từ bị tách thành nhiều subword bởi tokenizer, chúng ta cần đảm bảo các nhãn được căn chỉnh một cách chính xác.
3. **Tải mô hình:** Tải một mô hình AutoModelForTokenClassification.
4. **Thiết lập Trainer:** Sử dụng hàm compute\_metrics với seqeval.
5. **Huấn luyện:** Bắt đầu quá trình fine-tuning.

**Mã nguồn (tập trung vào bước tiền xử lý):**

##### Ví dụ 32: Fine-tuning cho Token Classification (NER)

baselinestretch

```

1 # ... (cài đặt và import tương tự các recipe trước) ...
2 from datasets import load_dataset
3 from transformers import AutoTokenizer,
4     AutoModelForTokenClassification, Trainer, TrainingArguments,
5     DataCollatorForTokenClassification
6 import evaluate
7 import numpy as np
8
# Bước 1: Tải dữ liệu
raw_datasets = load_dataset("conll2003")

```

```

9   label_names =
10  ↪ raw_datasets["train"].features["ner_tags"].feature.names
11
12  model_checkpoint = "roberta-base"
13  tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
14
15  # Bước 2: Hàm tiền xử lý và căn chỉnh nhãn
16  def tokenize_and_align_labels(examples):
17      tokenized_inputs = tokenizer(examples["tokens"],
18                                    ↪ truncation=True, is_split_into_words=True)
19      labels = []
20      for i, label in enumerate(examples["ner_tags"]):
21          word_ids = tokenized_inputs.word_ids(batch_index=i)
22          previous_word_idx = None
23          label_ids = []
24          for word_idx in word_ids:
25              if word_idx is None: # Special tokens like [CLS],
26                  ↪ [SEP]
27                  label_ids.append(-100)
28              elif word_idx != previous_word_idx: # First token of
29                  ↪ a new word
30                  label_ids.append(label[word_idx])
31              else: # Subsequent tokens of the same word
32                  label_ids.append(-100) # Chỉ gán nhãn cho
33                  ↪ subword đầu tiên
34                  previous_word_idx = word_idx
35          labels.append(label_ids)
36      tokenized_inputs["labels"] = labels
37      return tokenized_inputs
38
39
40  tokenized_datasets = raw_datasets.map(tokenize_and_align_labels,
41                                         ↪ batched=True)
42
43  # Bước 3 & 4 & 5: Tải mô hình, thiết lập Trainer và Huấn luyện
44  data_collator =
45      ↪ DataCollatorForTokenClassification(tokenizer=tokenizer)
46  seqeval = evaluate.load("seqeval")
47
48  def compute_metrics(p):
49      predictions, labels = p
50      predictions = np.argmax(predictions, axis=2)
51      # (Phân decode và tính toán tương tự ví dụ seqeval ở Chương
52      ↪ 2.2)
53      # ...
54      return {"precision": ..., "recall": ..., "f1": ...} # Kết
55      ↪ quả từ seqeval
56
57  model = AutoModelForTokenClassification.from_pretrained(
58      model_checkpoint, num_labels=len(label_names),
59  )
60
61  args = TrainingArguments("roberta-ner",
62                          ↪ evaluation_strategy="epoch", learning_rate=2e-5)
63  trainer = Trainer(
64      model, args,
65      train_dataset=tokenized_datasets["train"],

```

```

55     eval_dataset=tokenized_datasets["validation"],
56     data_collator=data_collator,
57     tokenizer=tokenizer,
58     compute_metrics=compute_metrics
59   )
60   trainer.train()

```

**Kết quả mong đợi:** Một mô hình được fine-tune chuyên biệt cho việc trích xuất các thực thể như Tên người, Tổ chức, Địa điểm từ văn bản.

## 4.5. Recipe 5: Xây dựng một Agent đơn giản với khả năng sử dụng công cụ

**Mục tiêu:** Xây dựng một Tác tử (Agent) có thể trả lời các câu hỏi về các sự kiện gần đây hoặc các chủ đề chuyên sâu bằng cách tự động quyết định khi nào cần tìm kiếm thông tin trên Wikipedia.

**Thành phần chính:**

- ‘LangChain’: Cung cấp các công cụ để xây dựng các agent.
- LLM: Bộ não ra quyết định của agent.
- Công cụ (Tool): Một API hoặc thư viện để tương tác với thế giới bên ngoài (trong trường hợp này là Wikipedia).

**Các bước thực hiện:**

1. Thiết lập Môi trường: Cài đặt thư viện và API keys.
2. Định nghĩa Công cụ: Tạo một hoặc nhiều công cụ mà agent có thể sử dụng.
3. Khởi tạo LLM và Agent: Chọn một LLM và sử dụng các hàm của LangChain để “trang bị” công cụ cho nó.
4. Thực thi Agent: Đưa ra một câu hỏi và quan sát vòng lặp ‘Thought-Action-Observation’.

**Mã nguồn:**

### Ví dụ 33: Xây dựng Agent Tra cứu Wikipedia

baselinestretch

```

1  # Bước 1: Cài đặt
2  # pip install langchain langchain-openai wikipedia
3  import os
4  # os.environ["OPENAI_API_KEY"] = "sk-..."
5
6  from langchain_openai import ChatOpenAI
7  from langchain.agents import tool, AgentExecutor,
8      create_react_agent
9  from langchain import hub # Để tải prompt ReAct có sẵn
10
11 # Bước 2: Định nghĩa Công cụ
12 @tool
13 def search_wikipedia(query: str) -> str:
14     """
15         Searches Wikipedia to find information about a given query.
16         Use this tool for questions about current events, facts, or
17             specific entities.
18     """
19     from wikipedia import summary
20     try:

```

```

19     return summary(query, sentences=2)
20 except Exception as e:
21     return f"Error searching Wikipedia: {e}"
22
23 tools = [search_wikipedia]
24
25 # Bước 3: Khởi tạo LLM và Agent
26 llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
27
28 # Tải một prompt ReAct đã được thiết kế sẵn từ LangChain Hub
29 prompt = hub.pull("hwchase17/react")
30
31 # Tạo agent, kết hợp llm, tools, và prompt
32 agent = create_react_agent(llm, tools, prompt)
33
34 # Tạo executor để chạy agent
35 agent_executor = AgentExecutor(agent=agent, tools=tools,
36                                verbose=True)
37
38 # Bước 4: Thực thi Agent
39 query = "What is the main idea behind the LLaVA model?"
40 result = agent_executor.invoke({"input": query})
41
42 print("\nFinal Answer:")
43 print(result["output"])

```

Kết quả mong đợi ('verbose=True'): Bạn sẽ thấy log của agent, thể hiện rõ các bước suy nghĩ của nó:

```

> Entering new AgentExecutor chain...
Thought: The user is asking about the LLaVA model. I don't have detailed,
up-to-the-minute information on specific models in my internal knowledge.
I should use the Wikipedia search tool to get a summary.
Action: search_wikipedia
Action Input: "LLaVA model"
Observation: LLaVA (Large Language and Vision Assistant) is a large
multimodal model (LMM) created by researchers from the University of
Wisconsin-Madison, Microsoft Research, and Columbia University. It combines a
vision encoder with a large language model (LLM) to enable multimodal
chat capabilities, demonstrating impressive performance on various benchmarks.
Thought: I now have the main idea of the LLaVA model. It combines a vision
encoder and an LLM for multimodal chat. I can now formulate the final answer.
Action: finish
Final Answer: The main idea behind the LLaVA model is to combine a vision
encoder with a large language model (LLM) to create a powerful multimodal
assistant capable of understanding and discussing images in a conversational
manner.

> Finished chain.

```

## 4.6. Recipe 6: Trích xuất Thông tin có Cấu trúc từ Văn bản

**Mục tiêu:** Trích xuất thông tin từ một đoạn văn bản tự do và trả về dưới dạng một đối tượng JSON có cấu trúc được định nghĩa trước.

**Thành phần chính:**

- **LLM hỗ trợ Function Calling:** Các mô hình của OpenAI (GPT-3.5/4) hoặc các mô hình mã nguồn mở hỗ trợ chế độ này.
- **Định nghĩa Cấu trúc (Schema):** Một định nghĩa rõ ràng về cấu trúc JSON mong muốn, bao gồm tên các trường, kiểu dữ liệu, và mô tả.

**Các bước thực hiện:**

1. **Định nghĩa "Công cụ":** Mô tả cấu trúc dữ liệu bạn muốn trích xuất dưới dạng một "function signature".
2. **Gọi LLM:** Gửi văn bản đầu vào và định nghĩa công cụ đến API của LLM.
3. **Phân tích Kết quả:** LLM sẽ trả về một đối tượng JSON tuân thủ theo schema bạn đã định nghĩa.

**Mã nguồn:**

Ví dụ 34: Trích xuất Thông tin với OpenAI Function Calling

baselinestretch

```

1  # Bước 1: Cài đặt
2  # pip install openai
3  import os
4  import json
5  from openai import OpenAI
6
7  # os.environ["OPENAI_API_KEY"] = "sk-..."
8  client = OpenAI()
9
10 # Bước 2: Định nghĩa "Công cụ" (cấu trúc JSON mong muốn)
11 extraction_tool = {
12     "type": "function",
13     "function": {
14         "name": "extract_user_info",
15         "description": "Extracts user information from a given
16             ↪ text.",
17         "parameters": {
18             "type": "object",
19             "properties": {
20                 "name": {
21                     "type": "string",
22                     "description": "The full name of the user."
23                 },
24                 "email": {
25                     "type": "string",
26                     "description": "The email address of the
27                         ↪ user."
28                 },
29                 "order_id": {
30                     "type": "string",
31                     "description": "The ID of the user's order."
32                 }
33             },
34             "required": ["name", "email", "order_id"]
35         }
36     }
37 }
```

```

33         }
34     }
35 }
36
37 # Văn bản đầu vào
38 text_input = """
39 Chào bạn, tôi là Trần Văn An, email của tôi là
40     ↪ an.tran@example.com.
41 Tôi muốn hỏi về tình trạng của đơn hàng #A123-456. Cảm ơn!
42 """
43
44 # Bước 3: Gọi LLM với công cụ đã định nghĩa
45 response = client.chat.completions.create(
46     model="gpt-4-turbo",
47     messages=[
48         {"role": "system", "content": "You are a helpful
49             ↪ assistant that extracts structured information."},
50         {"role": "user", "content": text_input}
51     ],
52     tools=[extraction_tool],
53     tool_choice={"type": "function", "function": {"name":
54         "extract_user_info"} } # Bước mô hình phải dùng công cụ
55             ↪ này
56 )
57
58 # Bước 4: Phân tích kết quả
59 tool_call = response.choices[0].message.tool_calls[0]
60 if tool_call.function.name == "extract_user_info":
61     arguments = json.loads(tool_call.function.arguments)
62     print("Thông tin đã được trích xuất:")
63     print(json.dumps(arguments, indent=2, ensure_ascii=False))
64

```

Kết quả mong đợi: baselinestretch

---

```

Thông tin đã được trích xuất:
{
    "name": "Trần Văn An",
    "email": "an.tran@example.com",
    "order_id": "A123-456"
}

```

---

Lưu ý: Với các mô hình không hỗ trợ Function Calling, có thể đạt được kết quả tương tự bằng cách sử dụng các thư viện như ‘Instructor’ kết hợp với ‘Pydantic’ để định nghĩa schema và thêm các hướng dẫn chi tiết vào prompt.

## KẾT THÚC CHƯƠNG 4

*Chương 4 đã là một chuyến du hành thực tiễn qua các ứng dụng phổ biến nhất của NLP hiện đại. Bằng cách làm theo các “công thức” chi tiết, bạn đã học được cách kết hợp các thư viện, kiến trúc và kỹ thuật khác nhau để xây dựng các hệ thống hoàn chỉnh. Đây là bước cuối cùng trong việc chuyển hóa kiến thức lý thuyết thành năng lực thực hành. Giờ đây, bạn không chỉ hiểu về NLP, mà còn có đủ công cụ và sự tự tin để bắt đầu xây dựng các giải pháp của riêng mình.*

## KẾT THÚC PHẦN 2

---

Phần 2 đã đưa bạn vào một hành trình thực chiến, biến những lý thuyết phức tạp thành các kỹ năng và sản phẩm cụ thể. Từ những bước đầu tiên trong việc xử lý dữ liệu đến việc huấn luyện và triển khai các mô hình ngôn ngữ lớn, bạn đã được trang bị một bộ công cụ toàn diện để giải quyết các bài toán NLP trong thế giới thực. Phần này không chỉ dạy bạn cách sử dụng các thư viện, mà còn rèn luyện một tư duy MLOps bền vững. Với kiến thức từ cả hai phần, bạn đã có đủ hành trang để tự tin bước đi trên con đường của một chuyên gia NLP.

---

### Tóm lược nội dung Phần 2:

- Chương 1 – Quy trình làm việc và Công cụ Xử lý Dữ liệu:** Phác thảo vòng đời dự án NLP tinh gọn, các kỹ thuật thu thập dữ liệu (APIs, Web Scraping), làm sạch và gán nhãn (Pandas, Polars, Label Studio, Snorkel, LLM-based), tăng cường dữ liệu, và quản lý phiên bản với DVC.
- Chương 2 – Huấn luyện và Đánh giá Mô hình:** Làm chủ hệ sinh thái Hugging Face ('transformers', 'datasets', 'accelerate', 'evaluate'), xây dựng các hàm đánh giá thực tế ('compute\_metrics'), theo dõi thí nghiệm (Weights & Biases, MLflow), và huấn luyện phân tán với DeepSpeed.
- Chương 3 – Tối ưu hóa, Triển khai và Vận hành (MLOps):** Tìm hiểu về Cơ sở dữ liệu Vector, các thư viện tối ưu hóa ('bitsandbytes', 'peft', ONNX), đóng gói và phục vụ mô hình (FastAPI, Docker), các framework chuyên dụng (vLLM, BentoML), và vòng đời MLOps hoàn chỉnh (giám sát, tái huấn luyện).
- Chương 4 – Công thức Xây dựng các Ứng dụng Phổ biến (Recipes):** Hướng dẫn từng bước để xây dựng các ứng dụng thực tế: Tìm kiếm Ngữ nghĩa, Hệ thống RAG, Fine-tuning cho Phân loại và NER, xây dựng Tác tử đơn giản, và Trích xuất Thông tin có Cấu trúc.

# Tài liệu tham khảo

- [1] Emily Alsentzer, John R Murphy, William Boag, Wei-Hung Weng, Di Jindi, Tristan Naumann, and Matthew McDermott. Publicly available clinical bert embeddings. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 72–78, 2019.
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.
- [3] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- [4] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 3615–3620, 2019.
- [5] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8003–8014, 2020.
- [6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. In *Transactions of the Association for Computational Linguistics*, volume 5, pages 135–146, 2017.
- [8] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, 2016.
- [9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in neural information processing systems* 33, pages 1877–1901, 2020.
- [10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pires, Quoc Le, Yong-wook Lee, Oleksii Kuchaiev, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.
- [12] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

- [13] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in neural information processing systems 30*, 2017.
- [14] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, , et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.
- [15] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.
- [16] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [17] Marta R Costa-jussà, James Tran, Artem Sokolov, Machel Dewan, et al. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*, 2022.
- [18] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems 35*, pages 16344–16359, 2022.
- [19] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.
- [20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.
- [21] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, 2018.
- [22] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *The Journal of Machine Learning Research*, volume 22, pages 1–39, 2021.
- [23] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box attacks against rnns: The power of gradient-free methods. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 117–130, 2018.
- [24] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, 2021.
- [25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems 27*, 2014.
- [27] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- [28] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2022.

- [29] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.
- [30] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.
- [31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [32] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. John Wiley & Sons, 2013.
- [33] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.
- [34] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, 2018.
- [35] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [36] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra S Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- [37] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7930–7937, 2020.
- [38] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. In *Transactions of the Association for Computational Linguistics*, volume 8, pages 64–77, 2020.
- [39] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.
- [40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [41] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [42] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- [43] Taku Kudo and John Richardson. 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*, pages 66–71, 2018.

- [44] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning (ICML 2001)*, 2001.
- [45] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.
- [46] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chang Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. volume 36, pages 1234–1240, 2020.
- [47] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.
- [48] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems 27*, 2014.
- [49] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
- [50] Raymond Li, Loubna Allal, Yangtian Zi, Leslie Tow, Mirac Briesch, Yumo Gu, Carl Akiki, Zhaowei Mou, Manar Naila, Denis Kocetkov, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.
- [51] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.
- [52] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. In *arXiv preprint arXiv:1907.11692*, 2019.
- [53] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, 2015.
- [54] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.
- [55] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamel Seddah, and Benoît Sagot. Camembert: a tasty french language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7213–7224, 2020.
- [56] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the 1st International Conference on Learning Representations (ICLR)*, 2013.
- [57] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems 26*, 2013.

- [58] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2786–2792, 2016.
- [59] Dat Quoc Nguyen and Anh Tuan Nguyen. Phobert: Pre-trained language models for vietnamese. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1037–1042, 2020.
- [60] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [61] OpenAI. Gpt-4o. <https://openai.com/index/hello-gpt-4o/>, 2024. Accessed: 2024-05-15.
- [62] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35*, pages 27730–27744, 2022.
- [63] Bowen Peng, Jeffrey Quesnelle, Angsheng Fan, and Ayush Aneja. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.
- [64] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [65] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, 2018.
- [66] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2022.
- [67] Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2401–2410, 2020.
- [68] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [69] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.
- [70] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [71] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.
- [72] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. In *The Journal of Machine Learning Research*, volume 21, pages 1–67, 2020.

- [73] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992, 2019.
- [74] Sascha Rothe, Shashi Narayan, and Ali Seifert. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280, 2020.
- [75] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xian Tan, Guillaume Lample, Côme Grand, Thomas Lavril, Marie-Anne Lachaux, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- [76] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [77] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019*, 2019.
- [78] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2009.
- [79] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.
- [80] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [81] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.
- [82] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- [83] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- [84] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems 33*, pages 16857–16867, 2020.
- [85] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.
- [86] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [87] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and John Schulman. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems 33*, pages 3008–3021, 2020.

- [88] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunbo Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [89] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems 27*, 2014.
- [90] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Stanford Center for Research on Foundation Models. Stanford alpaca: An instruction-following llama model. [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca), 2023.
- [91] Yi Tay, Mostafa Dehghani, Vinh Q Tran, Siyuan Hsieh, et al. Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022.
- [92] The BLOC team. Ntk-aware scaled rope allows llama models to have extended (8k) context length without any fine-tuning. [https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware\\_scaled\\_rope\\_allows\\_llama\\_models\\_to\\_have/](https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/), 2023.
- [93] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems 30*, 2017.
- [95] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.
- [96] Liang Wang, Nan Yang, Fnu Fariha, Hao Shen, and Haolan Liu. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.
- [97] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems 33*, pages 5776–5788, 2020.
- [98] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [99] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.
- [100] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.
- [101] Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972.
- [102] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. In *arXiv preprint arXiv:1609.08144*, 2016.

- [103] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.
- [104] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems* 32, 2019.
- [105] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Sha, Silvio Savarese, and Anima Anandkumar. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.
- [106] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems* 33, pages 17283–17297, 2020.
- [107] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.
- [108] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. Dialogpt: Large-scale generative pre-training for conversational response generation. *arXiv preprint arXiv:1911.00536*, 2019.

# Lời kết

---

*Thế là ta đã đi hết một vòng roller-coaster mang tên Xử lý Ngôn ngữ Tự nhiên. Nếu bạn còn sống để đọc đến đây, xin chúc mừng: bạn đã vượt qua một đống công thức rối rắm, những cái tên paper khó nhớ, và cả mấy đoạn văn nghe như ChatGPT mơ ngủ. Giờ thì đi ngủ đi, bạn đã đi được xa lắm rồi.*

---

## Con đường phía trước: Từ ngu đến đồ ngu

**Phần 1 là móng nhà, không phải bảo tàng.** Đừng coi “Bách khoa Toàn thư về Lý thuyết” là thứ đọc xong rồi xếp xó. Nó chính là cái nền để bạn dựng mọi mô hình sau này. Khi có một paper mới ra với tên nghe như thần chú, quay lại phần này: bạn sẽ thấy bản chất nó vẫn xoay quanh Attention, học biểu diễn, và mấy định luật xác suất quen thuộc. Muốn làm “pro” thật sự thì không chỉ biết xài tool, mà còn phải hiểu vì sao nó chạy được. (Và vâng, debug một mô hình đôi khi cũng giống như đọc lại sách giáo trình vậy.)

**Phần 2 là phòng lab, không phải sách nấu ăn.** “Cẩm nang Thực chiến” đưa công thức cho bạn, nhưng công thức thì sinh ra là để phá. Hãy nghịch: đổi mô hình, đổi dữ liệu, phá optimizer, thậm chí làm sai cũng được. Mỗi lần máy báo lỗi đó lòm, hãy coi đó là một mentor thầm lặng. Không giáo trình nào dạy nhanh bằng cách tự fix một đống bug đâu cưng.

**Cuối cùng, đừng quên trách nhiệm.** NLP không chỉ là code chạy đúng hay loss xuống đẹp. Những mô hình bạn build hôm nay có thể ảnh hưởng đến cách con người giao tiếp, tiếp nhận thông tin, thậm chí là... cãi nhau trên mạng xã hội (có lần mình code một tool cho LLMs tự động chửi nhau trên facebook xong bị khóa mõm rồi :>). Thế nên, ngoài chuyên optimize cho nhanh, hãy luôn nghĩ đến bias, fairness, và impact. Làm kỹ sư NLP vừa vui, vừa “nguy hiểm” – nêu trách nhiệm cũng phải đi kèm.

## Lời kết

---

*Cảm ơn bạn đã đồng hành tới cuối chặng đường này. Cuốn sách thì đóng lại, nhưng bug thì vẫn còn nhiều, và hành trình học của bạn mới chỉ bắt đầu thôi.  
Chúc bạn vừa code giỏi, vừa ngủ đủ giấc!*