CS336: Language Modeling from Scratch

Lecture 1: Overview and Tokenization

Stanford University - Spring 2025

Abstract

Abstract: This inaugural lecture establishes the foundation for CS336, introducing the course philosophy of understanding through building from scratch. We explore the motivation behind the course, address the challenges of modern AI research disconnection from underlying technology, and dive deep into tokenization fundamentals. The lecture covers the evolution from character-based to byte pair encoding (BPE), providing both theoretical understanding and practical implementation details essential for building language models efficiently.

Enhanced Summary by:
GitHub: HtmMhmd | LinkedIn: Hatem Mohamed

Contents

1 Course Motivation and Philosophy

1.1 The Research Crisis

Disconnection from Technology: The field faces a growing crisis where researchers become increasingly disconnected from underlying technology:

- **Eight years ago:** Researchers implemented and trained their own models
- **Six years ago:** Researchers downloaded models like BERT and fine-tuned them
- **Today:** Many researchers rely solely on prompting proprietary models

The Abstraction Problem: While abstraction enables broader research participation, current abstractions are "leaky":

- Programming languages and operating systems provide clear abstractions
- Language models offer unclear "string in, string out" abstractions
- Understanding the abstraction requires understanding the implementation

1.2 Fundamental Research Requirements

Deep Technical Understanding: Fundamental research requires "tearing up the stack" and co-designing different aspects:

- Data processing and curation strategies
- System architecture and optimization
- Model design and training procedures
- Hardware-software co-optimization

The Build-to-Understand Philosophy: True comprehension emerges from implementation experience, not abstract knowledge.

1.3 The Industrialization Challenge

Scale Barriers:

- GPT-4: Rumored 1.8 trillion parameters, $100 million training cost
- xAI: Building clusters with 200,000 H100s
- Industry investment: Over $500 billion across four years
- No public details on construction methods

Limited Transparency: GPT-4 technical report: "Due to competitive landscape and safety limitations, we disclose no details."

2 Scale Limitations and Learning Objectives

2.1 Small Model Limitations

Non-Representative Behavior: Small models may not reflect large-scale phenomena.

Example 1 - FLOP Distribution:

- Small models: Attention and MLP layers have comparable FLOP usage
- 175B parameter models: MLPs dominate computation
- Optimization focus shifts dramatically with scale

Example 2 - Emergent Behavior:

- Capabilities like in-context learning emerge suddenly at scale
- Small-scale experiments may incorrectly conclude models "don't work"
- Critical capabilities only appear beyond specific compute thresholds

2.2 Three Types of Knowledge

1. Mechanics (Fully Teachable):

- Transformer architecture implementation
- Model parallelism for GPU efficiency
- Training loop and optimization procedures
- Raw ingredients and technical components

2. Mindset (Cultivatable):

- Maximize hardware utilization efficiency
- Take scaling considerations seriously
- Prioritize algorithmic efficiency improvements
- Think in terms of compute-accuracy trade-offs

3. Intuitions (Partially Transferable):

- Which data decisions lead to good models
- Which architectures scale effectively
- Limited transferability from small to large scale
- Requires extensive experimentation experience

3 The Bitter Lesson and Efficiency

3.1 Misconceptions About Scale

Wrong Interpretation: "Scale is all that matters, algorithms don't matter"

Correct Interpretation: "Algorithms at scale matter" - accuracy equals efficiency multiplied by resources.

3.2 Efficiency Importance

Cost Scaling: At hundreds of millions of dollars, efficiency becomes critical - no room for waste.

Historical Evidence: OpenAI from 2020 showed 44x algorithmic efficiency improvement (2012-2019) for ImageNet training - faster than Moore's Law.

Core Question: What is the best model one can build given specific compute and data budgets?

4 Course Structure and Five Units

4.1 Unit 1: Basics

Objective: Implement complete pipeline fundamentals.

Components:

- **Tokenizer:** Byte Pair Encoding (BPE) implementation
- **Architecture:** Transformer with modern improvements
- **Training:** Cross-entropy loss, AdamW optimizer, training loop

Assignment 1: Full stack implementation with OpenWeb text perplexity leaderboard (90 minutes on H100).

4.2 Unit 2: Systems

Objective: Maximize hardware utilization efficiency.

Components:

- **Kernels:** GPU optimization using Triton
- **Parallelism:** Data and model parallelism strategies
- **Inference:** Pre-fill and decode optimization

Key Concepts: Memory hierarchy, data movement minimization, speculative decoding.

4.3 Unit 3: Scaling Laws

Objective: Predict optimal hyperparameters for larger scales.

Core Question: Given FLOP budget, what model size optimizes performance?

Chinchilla Optimal: For model size N, train on approximately 20N tokens.

Assignment 3: Training API experiments with FLOP budget constraints.

4.4 Unit 4: Data

Objective: Transform raw data into high-quality training material.

Components:

- **Evaluation:** Perplexity, MMLU, instruction following
- **Curation:** HTML-to-text, filtering, deduplication
- **Sources:** Common Crawl, Wikipedia, GitHub, StackExchange

Reality Check: Most web data is "trash" requiring extensive processing.

4.5 Unit 5: Alignment

Objective: Transform base models into useful assistants.

Components:

- **Supervised Fine-tuning (SFT):** Instruction-response pairs
- **Preference Learning:** DPO, GRPO algorithms
- **Safety:** Harmful content refusal capabilities

Key Insight: Even 1,000 examples sufficient for instruction following from good base models.

5 Tokenization Fundamentals

5.1 Definition and Requirements

Tokenization: Converting raw text (Unicode strings) into sequences of integers (tokens).

Requirements:

- Encoder: strings $\rightarrow$ tokens (integers)
- Decoder: tokens $\rightarrow$ strings (reversible)
- Fixed vocabulary size
- Efficiency optimization

Vocabulary Size: Number of possible token values (integer range).

5.2 Tokenization Examples

Spacing Convention:

- Spaces become part of tokens
- Convention: space precedes the token
- "hello" vs " hello" are different tokens
- Everything must be reversible

Number Handling: Numbers decomposed left-to-right, not semantically grouped.

Tokenization Analysis

Compression Ratio Analysis: GPT-2 tokenizer achieves 1.6 bytes per token compression ratio, balancing vocabulary size with sequence length efficiency. This metric becomes crucial for computational efficiency in attention mechanisms.

6 Tokenization Approaches

6.1 Character-Based Tokenization

Method: Map each Unicode character to its code point integer.

Examples:

- 'A' → 97
- World emoji → 127,757
- Reversible through Unicode standard

Problems:

- Huge vocabulary size (1.27M+ code points)
- Inefficient allocation - rare characters waste vocabulary slots
- Poor compression ratio (≈1.5 bytes/token)
- Frequent characters underutilized

6.2 Byte-Based Tokenization

Method: Convert Unicode strings to UTF-8 bytes, tokenize bytes.

Advantages:

- Fixed vocabulary size (256 possible byte values)
- No sparsity issues with rare characters
- Most elegant approach theoretically

Critical Problem: Compression ratio of 1.0 (one byte per token) creates extremely long sequences, leading to quadratic attention cost blowup.

6.3 Word-Based Tokenization

Method: Split text using regular expressions, assign integer per word segment.

Advantages:

- Captures adaptive compression intuition
- Semantic units preserved

- Better compression than character/byte approaches

Critical Problems:

- Unbounded vocabulary size
- Out-of-vocabulary (OOV) words require UNK tokens
- New words create evaluation problems
- Real words can be rare, causing UNK issues

7 Byte Pair Encoding (BPE)

7.1 Historical Context

Origins:

- **1994:** Phillip Gage develops BPE for data compression
- **2016:** Introduced to NLP for neural machine translation
- **2019:** GPT-2 adopts BPE for language modeling

Innovation: Train the tokenizer on raw text rather than using predefined rules.

7.2 BPE Algorithm

Core Principle: Successively merge the most common pair of adjacent tokens.

Process:

1. Convert string to byte sequence
2. Iteratively merge most frequent adjacent token pairs
3. Common sequences become single tokens
4. Rare sequences remain as multiple tokens
5. Organic vocabulary development based on frequency

Pre-processing: GPT-2 uses word-based pre-tokenization for efficiency, then applies BPE to each segment.

Technical Challenge

BPE Implementation Challenge: While conceptually simple, BPE implementation involves intricate details around frequency counting, merge operations, and handling edge cases. The algorithm's apparent simplicity masks significant implementation complexity that students often underestimate.

8 BPE Algorithm Details

8.1 Training Process

Initialization: Start with byte-level vocabulary (256 tokens).

Iterative Merging:

1. Count frequency of all adjacent token pairs
2. Identify most frequent pair
3. Create new token representing this pair
4. Replace all instances with new token
5. Add to vocabulary and merge rules
6. Repeat until desired vocabulary size

Frequency-Based Selection: Most common sequences earn dedicated tokens, optimizing for data compression.

8.2 Encoding Process

Text-to-Tokens:

1. Pre-tokenize text into segments
2. Convert each segment to bytes
3. Apply learned merge rules in training order
4. Result: sequence of token integers

Deterministic Process: Same text always produces same tokens given fixed merge rules.

8.3 Decoding Process

Tokens-to-Text:

1. Map each token to its byte representation
2. Concatenate all bytes
3. Convert byte sequence to UTF-8 string

Reversibility: Lossless round-trip conversion essential for model training.

9 Implementation Considerations

9.1 Practical Challenges

Edge Cases:

- Empty strings and single characters
- Unicode normalization consistency
- Handling of control characters
- Byte sequence validation

Efficiency Concerns:

- Frequency counting optimization
- Memory usage during training
- Fast lookup structures for encoding
- Parallel processing considerations

9.2 Vocabulary Size Selection

Trade-offs:

- **Larger vocabulary:** Better compression, shorter sequences
- **Smaller vocabulary:** More generalizable, less memory
- **Typical sizes:** 32K-100K tokens for modern models

Considerations:

- Model parameter scaling with vocabulary size
- Attention computational complexity
- Out-of-vocabulary robustness
- Cross-lingual representation quality

10 Modern Tokenization Advances

10.1 Current Limitations

Known Issues:

- Inconsistent treatment of spaces
- Poor handling of mathematical notation
- Language-specific biases in vocabularies

- Suboptimal compression for some domains

Research Directions:

- Tokenizer-free approaches using raw bytes
- Adaptive tokenization during training
- Multimodal tokenization for text+image
- Learned tokenization end-to-end with models

10.2 Alternative Approaches

Promising Methods:

- **Byte-level models:** New architectures handling raw bytes efficiently
- **SentencePiece:** Unigram language model approach
- **WordPiece:** Likelihood-based vocabulary construction
- **Learned tokenizers:** End-to-end optimization with model training

Current Status: BPE remains dominant for frontier models despite active research in alternatives.

11 Practical Implementation Tips

11.1 Development Strategy

Incremental Implementation:

1. Start with simple byte conversion
2. Implement frequency counting correctly
3. Add merge rule application
4. Test with small examples extensively
5. Scale to real datasets gradually

Testing Approach:

- Verify round-trip conversion on diverse inputs
- Compare against reference implementations
- Test edge cases and Unicode handling
- Benchmark performance on large texts

11.2 Common Pitfalls

Implementation Errors:

- Incorrect frequency counting for overlapping pairs
- Wrong merge rule ordering during encoding
- Unicode encoding/decoding inconsistencies
- Off-by-one errors in vocabulary indexing

Performance Issues:

- Inefficient data structures for counting
- Repeated string operations without optimization
- Memory leaks during vocabulary construction
- Lack of early termination conditions

12 Course Overview Mindmap and Vision

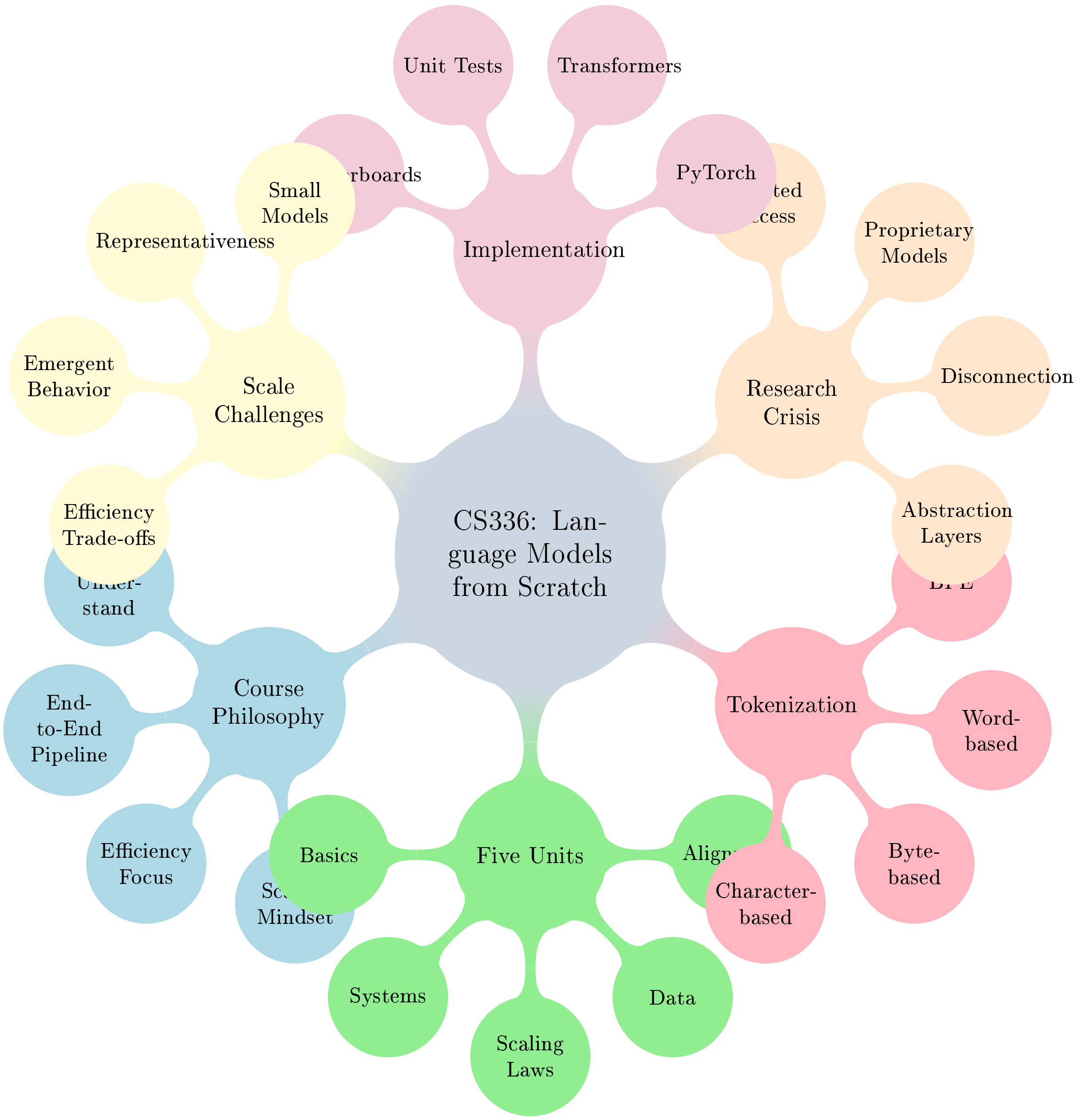

Figure 1: Comprehensive mindmap of CS336 Lecture 1, illustrating the course philosophy, structure, tokenization fundamentals, research challenges, implementation approach, and scaling considerations.

12.1 Mindmap Description

The CS336 course mindmap captures the interconnected nature of building language models from first principles:

Central Hub - CS336 Course: The comprehensive journey from raw implementation to functional language models, emphasizing hands-on learning and deep technical understanding.

Course Philosophy Branch (Blue): The foundational approach emphasizing that true understanding comes through building, requiring end-to-end pipeline comprehension, efficiency optimization, and adoption of a scaling mindset that considers computational resources as the primary constraint.

Five Units Branch (Green): The structured curriculum progression through Basics (tokenization, transformers, training), Systems (kernels, parallelism, inference), Scaling Laws (optimal compute allocation), Data (curation, evaluation), and Alignment (instruction following, safety).

Tokenization Branch (Red): The fundamental text-to-integer conversion process, evolving from naive character-based approaches through byte-based methods to sophisticated word-based tokenization, culminating in Byte Pair Encoding (BPE) that balances compression efficiency with vocabulary manageability.

Research Crisis Branch (Orange): The motivation for the course, addressing how increasing abstraction layers have disconnected researchers from underlying technology, with proprietary models limiting access to implementation details essential for fundamental research advancement.

Implementation Branch (Purple): The practical execution framework using PyTorch foundations while building transformers from scratch, validated through comprehensive unit tests and competitive leaderboards that encourage optimization and best practices.

Scale Challenges Branch (Yellow): The inherent limitations of working with small models, including questions about representativeness, missing emergent behaviors, and the continuous tension between efficiency optimization and capability development.

Course Philosophy

The mindmap reveals that CS336 is not merely a technical course but a comprehensive response to the industrialization of AI research. By building from scratch, students gain the deep understanding necessary for fundamental research and efficient system design in an era where compute efficiency determines research feasibility.