



# CS 394E: Contemporary Topics in LLMs

## Lecture-2: N-gram Language Models

Marco Canini

Professor of Computer Science, KAUST

Fall 2026

# Announcements

- Sign up for the course Campuswire channel

# Recap: Language Models

- *The water at KAUST South Beach is so beautifully ...*
- **Language Model (LM)** → predicting language (e.g., words)
  - More formally, an LM assigns a probability to each possible next word
  - Can also assign a probability to an entire sentence
  - E.g., an LLM could tell which sequence is more likely to appear in a text
    - *all of a sudden I notice three guys standing on the sidewalk*
    - *on guys all I of notice sidewalk three a sudden standing the*

# Recap: Why Next Word Prediction?

- **Large LMs** learn an enormous amount about language solely from being trained to predict next words
- **Enables**
  - Language understanding and generation
  - Task completion and automation
  - Reasoning and world knowledge
  - Creativity and expression

## Can LLMs Generate Novel Research Ideas?

A Large-Scale Human Study with 100+ NLP Researchers

Chenglei Si, Diyi Yang, Tatsunori Hashimoto

Stanford University

{clsi, diyi, thashim}@stanford.edu

### Abstract

Recent advancements in large language models (LLMs) have sparked optimism about their potential to accelerate scientific discovery, with a growing number of works proposing research agents that autonomously generate and validate new ideas. Despite this, no evaluations have shown that LLM systems can take the very first step of producing novel, expert-level ideas, let alone perform the entire research process. We address this by establishing an experimental design that evaluates research idea generation while controlling for confounders and performs the first head-to-head comparison between expert NLP researchers and an LLM ideation agent.

By recruiting over 100 NLP researchers to write novel ideas and blind reviews of both LLM and human ideas, we obtain the first statistically significant conclusion on current LLM capabilities for research ideation: we find LLM-generated ideas are judged as more novel ( $p < 0.05$ ) than human expert ideas while being judged slightly weaker on feasibility. Studying our agent baselines closely, we identify open problems in building and evaluating research agents, including failures of LLM self-evaluation and their lack of diversity in generation. Finally, we conclude by discussing the implications of our findings and the need for more rigorous evaluations.

an end-to-end study design which recruits researchers to execute these ideas into full projects, enabling us to study whether these novelty and feasibility judgements result in meaningful differences in research outcome. <sup>1</sup>

# Agenda

- N-gram Models
- Training and Testing
- Limitations
- Tokenization

# Key Idea in Language Models: Context

**“You shall know a word by the company it keeps”**

# Firth, 1957



J. R. Firth

# Language Models → Joint Probability Distributions

- They aim to find the **joint probability** of all the words occurring in a particular order

$$P(W) = P(w_1, w_2, \dots, w_n)$$

- For a sentence like "**The cat sat on the mat**", a language model wants to estimate:

$$P(\text{"The", "cat", "sat", "on", "the", "mat"})$$

# How to compute this joint probability distribution?

$$P(W) = P(w_1, w_2, \dots, w_n)$$

# Computing Directly

- Find probability for **every possible sequence** of  $n$  words in the vocabulary
- For example, for 3-word sentences, you would need probabilities for:
  - ("the", "cat", "sat")
  - ("a", "dog", "barked")
  - ("the", "banana", "is")
  - ... and **every possible triplet** in the vocabulary.
- Do you think it would be **feasible** to compute this directly?

# Some Fun Facts about the English Language

- English vocabulary is huge\*
  - Oxford English Dictionary documents ~500k words & Phrases
  - Broader estimates (including technical/scientific terms) can exceed 1M+
- Fun Question: What is the longest sentence in English?
  - One famously extreme example is Jonathan Coe's *The Rotters' Club* (2001), which includes a **13,955-word** sentence, often cited as one of the longest in English literature

\*<https://www.oed.com/information/about-the-oed/>

# Combinatorial Explosion

- If your vocabulary has  $V$  words and you're modelling sequences of length  $n$ , the number of possible sequences is:  $V^n$
- For a modest vocabulary of 50,000 words:
  - 2-word sequences:  $50,000^2 = 2.5 \times 10^9$
  - 5-word sequences:  $50,000^5 \approx 3.1 \times 10^{23}$
  - 10-word sequences: ... astronomical
- You would need to store and estimate **a probability for each sequence** which quickly becomes infeasible in terms of both data and memory

# In addition ...

- To estimate the probability of each sequence accurately, you need to **observe it enough times** in the real world
- But natural language is **incredibly diverse**
- Even with billions of words of training data, most long sequences might occur **once or never**

# Let's try decomposing the joint distribution

$$P(W) = P(w_1, w_2, \dots, w_n)$$

# Chain Rule Decomposition

$$P(W) = P(w_1, w_2, \dots, w_n)$$

- Using the **chain rule of probability**, this can be decomposed as:

$$P(W) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \dots \cdot P(w_n \mid w_1, \dots, w_{n-1})$$

Recall:  $P(w_2 \mid w_1) = P(w_1, w_2) / P(w_1)$

Rewriting:  $P(w_1, w_2) = P(w_1) \cdot P(w_2 \mid w_1)$

# Chain Rule Decomposition

$P(\text{"The", "cat", "sat", "on", "the", "mat"})$



$P(\text{The}) \cdot P(\text{cat} \mid \text{The}) \cdot P(\text{sat} \mid \text{The cat}) \cdot$   
 $P(\text{on} \mid \text{The cat sat}) \cdot P(\text{the} \mid \text{The cat sat on}) \cdot P(\text{mat} \mid \text{The cat sat on the})$

But does using chain rule completely solve the problem?

# Key Challenge

- We don't know any way to compute the exact probability of a word given a **long sequence of preceding words**
- We can't just estimate by counting the number of times every word occurs following every long string in some corpus, because language is **creative** and **any particular context might have never occurred before**

# N-gram Models: Intuition

Instead of computing the probability of a word given its entire context, we can **approximate the context by just the last few words**

## Example

$$P(\text{mat} \mid \underbrace{\text{The cat sat on the}}_{\text{Context}})$$

# N-gram Models



Andrei Markov

- **N-gram models** are based on the **Markov assumption** that the probability of a word depends only on the N-1 preceding words

$$P(w_n \mid w_1, \dots, w_{n-1}) \approx P(w_n \mid w_{n-N+1}, \dots, w_{n-1})$$

- The probability is estimated using simple **counting**:

$$P(w_n \mid w_{n-N+1}, \dots, w_{n-1}) = \frac{\text{count}(w_{n-N+1}, \dots, w_n)}{\text{count}(w_{n-N+1}, \dots, w_{n-1})}$$

# N-gram

- An **N-gram** is a *sequence* of **N words**

Corpus: I am happy because am learning

Unigrams: { I , am , happy , because , learning }

Bigrams: { I am , am happy , happy because ... }

Trigrams: { I am happy , am happy because, ... }

# N-gram Models

**1-gram**

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

**2-gram**

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

**3-gram**

$$P(w_i \mid w_1, \dots, w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1})$$

**4-gram**

$$P(w_i \mid w_1, \dots, w_{i-1}) \approx P(w_i \mid w_{i-3}, w_{i-2}, w_{i-1})$$

# An Example of a Bigram Model



Dr. Seuss

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and bread </s>

$$\begin{aligned} P(I | <s>) &= \frac{2}{3} = .67 & P(\text{Sam} | <s>) &= \frac{1}{3} = .33 & P(\text{am} | I) &= \frac{2}{3} = .67 \\ P(</s> | \text{Sam}) &= \frac{1}{2} = .5 & P(\text{Sam} | \text{am}) &= \frac{1}{2} = .5 & P(\text{do} | I) &= \frac{1}{3} = .33 \end{aligned}$$

# Considering a Richer Corpus

can you tell me about any good cantonese restaurants close by  
mid priced thai food is what i'm looking for  
tell me about chez panisse  
can you give me a listing of the kinds of food that are available  
i'm looking for a good place to eat breakfast  
when is caffe venezia open during the day

From Berkeley Restaurant Project

# Raw Bigram Counts

- Out of 9332 sentences

|         | i  | want | to  | eat | chinese | food | lunch | spend |
|---------|----|------|-----|-----|---------|------|-------|-------|
| i       | 5  | 827  | 0   | 9   | 0       | 0    | 0     | 2     |
| want    | 2  | 0    | 608 | 1   | 6       | 6    | 5     | 1     |
| to      | 2  | 0    | 4   | 686 | 2       | 0    | 6     | 211   |
| eat     | 0  | 0    | 2   | 0   | 16      | 2    | 42    | 0     |
| chinese | 1  | 0    | 0   | 0   | 0       | 82   | 1     | 0     |
| food    | 15 | 0    | 15  | 0   | 1       | 4    | 0     | 0     |
| lunch   | 2  | 0    | 0   | 0   | 0       | 1    | 0     | 0     |
| spend   | 1  | 0    | 1   | 0   | 0       | 0    | 0     | 0     |

# Raw Bigram Counts

- Normalize by unigrams:

| i    | want | to   | eat | chinese | food | lunch | spend |
|------|------|------|-----|---------|------|-------|-------|
| 2533 | 927  | 2417 | 746 | 158     | 1093 | 341   | 278   |

- Result:

|         | i       | want | to     | eat    | chinese | food   | lunch  | spend   |
|---------|---------|------|--------|--------|---------|--------|--------|---------|
| i       | 0.002   | 0.33 | 0      | 0.0036 | 0       | 0      | 0      | 0.00079 |
| want    | 0.0022  | 0    | 0.66   | 0.0011 | 0.0065  | 0.0065 | 0.0054 | 0.0011  |
| to      | 0.00083 | 0    | 0.0017 | 0.28   | 0.00083 | 0      | 0.0025 | 0.087   |
| eat     | 0       | 0    | 0.0027 | 0      | 0.021   | 0.0027 | 0.056  | 0       |
| chinese | 0.0063  | 0    | 0      | 0      | 0       | 0.52   | 0.0063 | 0       |
| food    | 0.014   | 0    | 0.014  | 0      | 0.00092 | 0.0037 | 0      | 0       |
| lunch   | 0.0059  | 0    | 0      | 0      | 0       | 0.0029 | 0      | 0       |
| spend   | 0.0036  | 0    | 0.0036 | 0      | 0       | 0      | 0      | 0       |

# Training and Test Sets

- **Training Set:** For simple n-gram language models it's the corpus from which we get the counts that we normalize into the probabilities of the n-gram language model
- **Development set or validation set:** We do all our testing on this dataset until the very end, and then we test on the test set once to see how good our model is
- **Test Set:** is a different, held-out set of data, not overlapping with the training set, that we use to evaluate the model

# Evaluation

- **Extrinsic Evaluation:** Measure the application or task performance for which the Language Model is used
- **Intrinsic Evaluation:** Measures the quality of a language model independent of any application

# Perplexity

- If you are comparing two LMs, the better LM would be the one which assigns a **higher probability** to the **test set**
- The perplexity (PP or PPL) of a language model on a test set is the inverse probability of the test set (one over the probability of the test set), normalized by the number of words  $N$

$$\text{perplexity}(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

- Lower the perplexity of a model on the data, the better the model

# Perplexity of different N-gram Models

- Unigram, bigram, trigram models trained on **38 million words** from Wall Street Journal newspaper\*

|            | Unigram | Bigram | Trigram |
|------------|---------|--------|---------|
| Perplexity |         |        |         |

\* *Speech and Language Processing* by Dan Jurafsky and James H. Martin

# Limitations of N-Gram Models?

# Limitations of N-gram Models

- **Limited context:** N-gram models only consider a fixed number of preceding words, typically up to 5
  - This restricts their ability to capture **long-range dependencies** in language
- **Curse of dimensionality:** As N increases, the number of possible N-grams grows exponentially
  - Increased data, storage and computational requirements
- **Sparsity:** Many possible N-grams may not appear in the training data
  - **Unseen N-grams get assigned zero probability**, which can cause issues in applications like language generation and speech recognition without smoothing/backoff
- **Lack of semantic abstraction/generalization:** N-grams treat contexts as discrete symbols, so they cannot capture similarity

# Neural Language Models

- Bengio et al., 2003 pioneered **neural language models**, where the probability distribution  $P(w_n | w_1, \dots, w_{n-1})$  is approximated by a neural network:
  - NLMs can capture **long-range dependencies** and **semantic relationships** better than traditional statistical models

Journal of Machine Learning Research 3 (2003) 1137–1155

Submitted 4/02; Published 2/03

## A Neural Probabilistic Language Model

**Yoshua Bengio**  
**Réjean Ducharme**  
**Pascal Vincent**  
**Christian Jauvin**

*Département d'Informatique et Recherche Opérationnelle  
Centre de Recherche Mathématiques  
Université de Montréal, Montréal, Québec, Canada*

BENGIOY@IRO.UMONTREAL.CA  
DUCHARME@IRO.UMONTREAL.CA  
VINCENTP@IRO.UMONTREAL.CA  
JAUVINC@IRO.UMONTREAL.CA



A Neural probabilistic language model

Y Bengio, R Ducharme, P Vincent  
Journal of Machine Learning Research 3, 1137-1155

11543

2003

# Summary

- Language Models aim to find the **joint probability**

$$P(W) = P(w_1, w_2, \dots, w_n)$$



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



**N-gram**  $P(w_n | w_1, \dots, w_{n-1}) \approx P(w_n | w_{n-N+1}, \dots, w_{n-1})$

In **neural language models**, the probability distribution  $P(w_n | w_1, \dots, w_{n-1})$  is approximated by a neural network

# Next ...

- Discuss important building blocks before we dive into neural language models
  - Tokenization and Vector Embeddings

# Modeling Tokens not Words

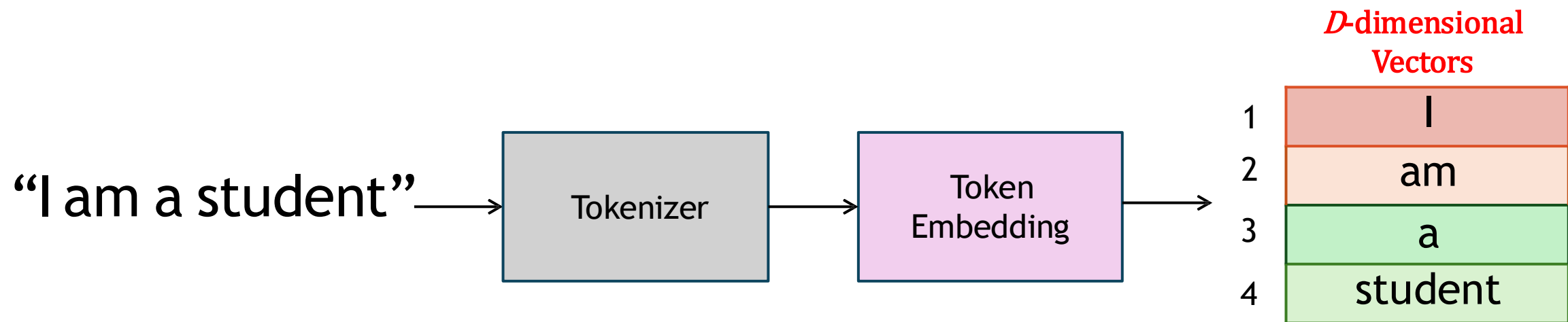
$$P(W) = P(w_1, w_2, \dots, w_n) \quad w_1 \text{ to } w_n \text{ represent tokens}$$

- **Tokens** represent words, word parts, and special characters

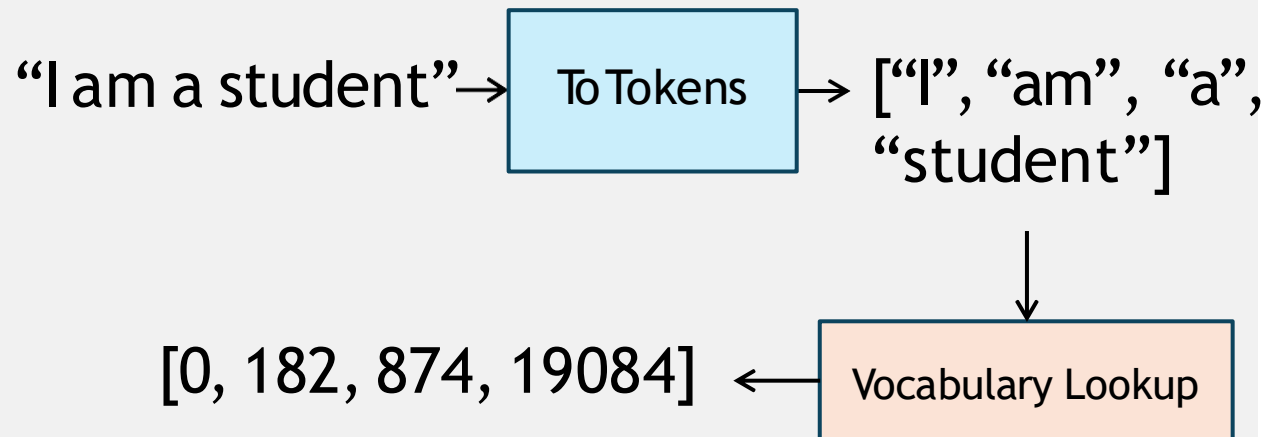
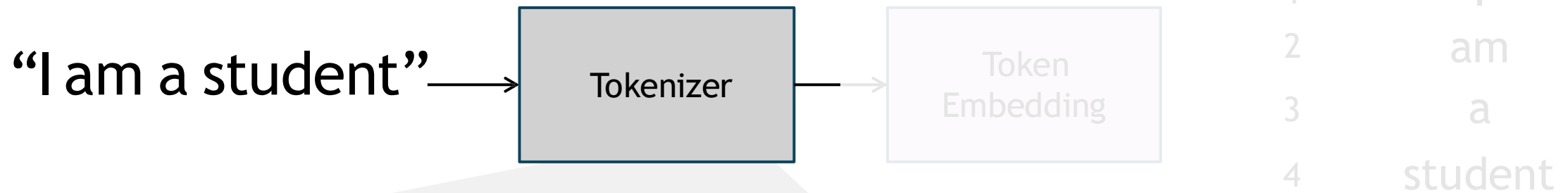
“The smallest tokenizer!” →

[“The”, “ small”, “est”, “ token”, “izer”, “!”]

# Language to Tokens to Embeddings



# Language to Tokens



The tokenizer breaks the text into tokens, basic elements of processing

Vocabulary is the collection of all tokens

# Core Properties of Efficient Tokenizers

- **Vocabulary Size Management**

- **Optimal Balance**: Neither too small (losing semantic richness) nor too large (computational overhead)
  - Ex: Llama 3 has roughly **128k tokens** (Llama 2 had **32k**)
  - Ex (if vocabulary is too small): Should mother-**in-law** and father-**in-law** be completely different?
  - Problematic in morphologically rich languages, where words have many different forms depending on grammatical features such as tense and gender (e.g., **أنتَ**, **أنتِ**, **أنتم** )

- **Out-of-Vocabulary (OOV) or Unknown Tokens**

- Any word that isn't in the vocab is usually replaced by a single symbol like **<unk>**
  - Example: if COVID wasn't in the training vocab, "I had COVID" → "I had <unk>"
- **Robustness**: Must handle unseen words gracefully

- **Language Independence**

- **Multilingual Support**: Work across different writing systems

```
text.split(' ')
```

# A Strawman Tokenizer

- What do you think about this tokenizer? How effective would it be?
- **Pros:** Easy to implement
- **Cons:**
  - Isn't effective for languages, where **sentences are written without spaces** between words:  
Chinese: 我今天去了商店。  
我 (I) + 今天 (today) + 去了 (went) + 商店 (store)
  - Tradeoff between parameters size and unknown chances
    - Smaller vocab → fewer parameters to learn, easier to generate, more OOV
    - Larger vocab → more parameters to learn, harder to generate, less OOV
  - Therefore, splitting by spaces to identify words is **quite problematic**

# Character-level Tokenization

T h e m o s t e a g e r i s O r e g ...

- Each **letter and punctuation** is a token
- **Pros:**
  - Very small vocabulary (except for some languages, e.g., Chinese)
  - No Out-of-Vocabulary token
- **Cons:**
  - Tokens do not represent semantic meaning

# Morpheme-level Tokenization

- Meaningful units of words: **root** + **affix**
  - For example: *I've, careful, cats*

A **morpheme** is the smallest **meaning-bearing unit** of a language

- unlikeliest has 3 morphemes **un-**, **likely**, and **-est**
- **Pros:**
  - Smaller than a word and captures semantic meaning
  - Lower OOV than word-based tokenization
- **Cons:**
  - Hard to segment in certain languages

# Next Lecture

- Continue discussion on tokenization
- Vector embeddings