N-gram Language Models

Rob Minneker

2026-01-20

Sources

Content derived from: J&M Ch. 3

Part 0: The Language Modeling Problem

What is a language model?

- A **language model** assigns probabilities to sequences of words
- For any sequence $w_1, w_2, \ldots, w_n$, we want to compute:

$$P(w_1, w_2, \ldots, w_n)$$

- This defines a **probability distribution over language**
- Higher probability = more "likely" or "natural" sequence

The language modeling task

Given a sequence of words...

The | students | opened | their | ?

...predict the next word (or assign probability to possible continuations)

books	laptops	minds	refrigerator
P = 0.35	P = 0.22	P = 0.15	P = 0.0001

Why model probability distributions over language?

- **Speech recognition**: Which word sequence best matches the audio?
  - "recognize speech" vs "wreck a nice beach"
- **Machine translation**: Which translation sounds most natural?
- **Text generation**: Sample from the distribution to produce text
- **Spelling/grammar correction**: Find the most probable intended sequence
- **Information retrieval**: Score document relevance

The fundamental challenge: Language is infinite

- Vocabulary V has $|V|$ words (e.g., 50,000)
- Possible bigrams: $|V|^2 = 2.5 \times 10^9$
- Possible 10-word sentences: $|V|^{10} \approx 10^{47}$

Number of atoms in the observable universe:

$\sim 10^{80}$

- We can never observe all possible sentences
- Yet we must assign probabilities to **every** possible sequence

The solution: Decompose using the chain rule

- Joint probability of a sequence can be decomposed:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1})$$

- Example: $P(\text{the cat sat}) = P(\text{the}) \times P(\text{cat} \mid \text{the}) \times P(\text{sat} \mid \text{the cat})$
- **But**: Each conditional still depends on unbounded history!
- **Key insight**: Approximate by limiting history (Markov assumption)

Part 1: Foundations of N-gram Language Models

N-gram models predict words using only the previous N-1 words as context

- An N-gram model estimates $P(w_n \mid w_{n-(N-1)}, \ldots, w_{n-1})$
- The conditional probability captures how likely the next word is given recent context
- This formulation embodies a key simplifying assumption about language

Trigrams condition on two preceding words to predict the next

- Example: $P(\text{'processing'} \mid \text{'natural'}, \text{'language'})$
- The model assumes: recent context is sufficient for prediction
- This enables efficient, statistical prediction for speech recognition, text generation, and spelling correction

The chain rule decomposes sentence probability into conditional factors

- Joint probability of a sentence $\mathbf{w} = w_1, w_2, \ldots, w_T$:

$$P(\mathbf{w}) \approx \prod_{n=1}^{T} P(w_n \mid w_{n-(N-1)}, \ldots, w_{n-1})$$

- Each factor depends only on a fixed-size context window
- This approximation makes computation tractable

Chain rule in action: Bigram factorization

$P(\text{"the cat sat"}) =$

start → first word
$P(\text{the} \mid \langle s \rangle)$ ×

first → second
$P(\text{cat} \mid \text{the})$ ×

second → third
$P(\text{sat} \mid \text{cat})$ ×

end marker
$P(\langle /s \rangle \mid \text{sat})$

Each factor uses **only one word** of context (bigram assumption)

Different values of N trade off context richness against data sparsity

- **Unigrams** ($N = 1$): $P(w_n)$ — no context at all
- **Bigrams** ($N = 2$): $P(w_n \mid w_{n-1})$ — one word of context
- **Trigrams** ($N = 3$): $P(w_n \mid w_{n-2}, w_{n-1})$ — two words of context

Visualizing context windows: What each model "sees"

Sentence: The cat sat on the **mat**

Unigram: The cat sat on the **?** *P(mat)*

Bigram: The cat sat on **the** **?** *P(mat | the)*

Trigram: The cat sat **on** **the** **?** *P(mat | on, the)*

> **Concept Check**
>
> If you have a vocabulary of 10,000 words, how many possible bigrams exist? How many trigrams? What does this imply for data requirements?

Standard notation enables reproducible model specification

- V: vocabulary set; $|V|$ is vocabulary size
- $\langle s \rangle$, $\langle / s \rangle$: special tokens for sentence boundaries
- Proper notation clarifies assumptions and enables fair comparison

The Markov assumption limits dependence to the k most recent words

- Order-k Markov assumption:

$$P(w_n \mid w_1, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-k}, \ldots, w_{n-1})$$

- This reduces unbounded context to a fixed-length window of size k
- The model "forgets" everything before the window

The Markov window “forgets” everything outside its scope

Sentence: The quick brown fox jumps over the lazy dog

k=1: ~~The~~ ~~quick~~ ~~brown~~ ~~fox~~ ~~jumps~~ ~~over~~ ~~the~~ lazy ? P(dog | lazy)

k=2: ~~The~~ ~~quick~~ ~~brown~~ ~~fox~~ ~~jumps~~ ~~over~~ the lazy ? P(dog | the, lazy)

k=4: ~~The~~ ~~quick~~ ~~brown~~ ~~fox~~ jumps over the lazy ? P(dog | jumps, over, the, lazy)

> ~~Crossed out~~ = forgotten by the model (Markov "memoryless" property)

Independence assumptions make parameter estimation feasible

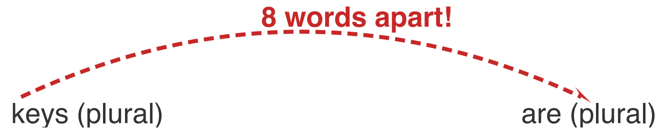
- For trigrams: $P(w_4 \mid w_1, w_2, w_3) \approx P(w_4 \mid w_2, w_3)$
- Parameters grow polynomially with k, not exponentially with sequence length
- Finite corpora can reliably estimate a tractable number of parameters

Markov models cannot capture long-range linguistic dependencies

- Subject-verb agreement across clauses requires longer context
- Anaphora resolution (pronoun references) often spans many words
- These limitations motivate RNNs and Transformers

Long-range dependencies: What trigrams miss

"The keys to the cabinet in the corner of the room are on the table."



Trigram window: sees only "room are" — no access to "keys"

Markov pioneered statistical text analysis in 1913

- A. A. Markov demonstrated word sequences exhibit capturable dependencies
- Modeled transitions: $P(w_n \mid w_{n-1})$
- Introduced the “memoryless” property: future depends only on recent past

Chomsky's formal grammars revealed hierarchical language structure

- 1956: Proposed finite-state, context-free, and context-sensitive grammars
- Emphasized recursive and hierarchical structure beyond word transitions
- N-grams capture local patterns but miss deeper syntactic structure

Modern N-gram models combine statistical dependence with practical utility

- Evolution from raw frequencies: $P(w) = \frac{\text{count}(w)}{N}$
- To conditional models: $P(w_n \mid w_{n-1}, \ldots, w_{n-(n-1)})$
- Foundation for text generation, speech recognition, and machine translation

Part 2: Estimation and Smoothing

MLE estimates probabilities by counting n-gram occurrences

- Maximum Likelihood Estimate for N-grams:

$$P_{\text{MLE}}(w_n \mid w_{n-1}, \ldots, w_1) = \frac{C(w_1, \ldots, w_{n-1}, w_n)}{C(w_1, \ldots, w_{n-1})}$$

- Simply divide n-gram count by context count
- Intuitive and unbiased on large corpora

MLE in action: Counting bigrams in a corpus

"I want to eat. I want to sleep. I need to go."

Bigram Counts

I want	**2**
want to	**2**
to eat	1
to sleep	1
to go	1
I need	1
need to	1

→

MLE Probabilities

$P(\text{want} \mid \text{I}) = {}^{2}/_{3} = \mathbf{0.67}$

$P(\text{need} \mid \text{I}) = {}^{1}/_{3} = \mathbf{0.33}$

$P(\text{to} \mid \text{want}) = {}^{2}/_{2} = \mathbf{1.00}$

MLE assigns zero probability to unseen n-grams

- If $C(\text{unseen bigram}) = 0$, then $P_{\text{MLE}} = 0$
- Zero probability means: “impossible according to the model”
- But unseen doesn’t mean impossible—just unobserved

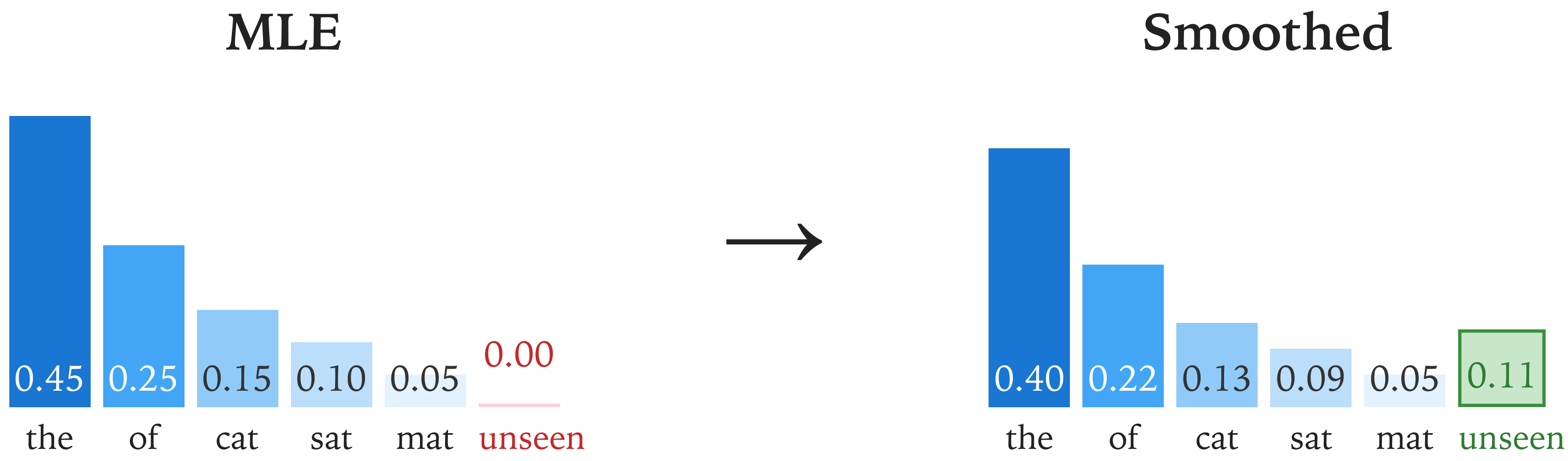
Concept Check

If your training corpus never contains “quantum computing,” what probability will an MLE bigram model assign to “quantum computing” in test data? What’s wrong with this?

Smoothing redistributes probability mass from seen to unseen events

- Core insight: “steal” small amounts from frequent n-grams
- Give that mass to rare and unseen n-grams
- Result: no probability is exactly zero

Visualizing probability redistribution



Laplace smoothing adds a constant to all counts

- Add $\alpha > 0$ to each count:

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

- Simple but can over-smooth for large vocabularies
- Often $\alpha = 1$ (add-one smoothing)

Good-Turing uses frequency of frequencies to estimate unseen mass

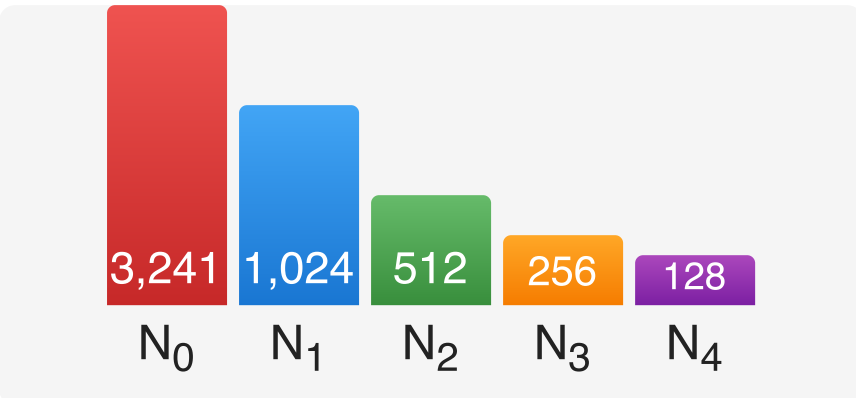
- Key insight: count how many n-grams appear exactly r times
- Adjusted count formula:

$$c^*(r) = \frac{(r+1)N_{r+1}}{N_r}$$

- N_r: number of n-grams with count r
- Uses rare events to estimate probability of unseen events

Good-Turing intuition: Rare events predict unseen events

Frequency of Frequencies (N_r)



N_r = count of n-grams appearing r times

→ Use N_1 to estimate N_0

Key Insight

Unseen (r=0): How much probability mass?

Singletons (r=1): 1,024 n-grams seen exactly once

Intuition: N-grams seen once were recently "unseen" — they tell us about the unseen mass

Kneser-Ney smoothing considers context diversity, not just frequency

- A word that appears in many contexts is a better backoff candidate
- Formula incorporates discount D and continuation probability:

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}, w_i) - D, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{\text{continuation}}(w_i)$$

- State-of-the-art for n-gram models (Chen & Goodman, 1999)

Continuation probability: “Francisco” vs “the”

"Francisco"

Raw count: 500

Appears frequently!

But only after:

San ___

Contexts: 1

Bad backoff candidate!

VS

"the"

Raw count: 500

Also appears frequently!

Appears after:

in ___ | at ___ | to ___ | on ___ | ...

Contexts: 847

Great backoff candidate!

Kneser-Ney insight: Context diversity matters more than raw frequency for backoff

The sparsity problem motivates combining multiple models

- Higher-order n-grams capture more context but suffer from sparsity
- Trigram “flew to Seattle” may have count 0, even if bigram “to Seattle” is common
- **Key insight**: Lower-order models provide reliable fallback estimates
- Solution: Combine models of different orders via **interpolation**

Linear interpolation combines n-gram models with learned weights

- Interpolated probability is a weighted sum:

$$P_{\text{interp}}(w_n|w_{n-2}, w_{n-1}) = \lambda_1 P_1(w_n) + \lambda_2 P_2(w_n|w_{n-1}) + \lambda_3 P_3(w_n|w_{n-2}, w_{n-}$$

- Weights must sum to 1: $\sum_i \lambda_i = 1$
- Each λ_i controls how much we trust each model

Visualizing interpolation: Blending three models

Query: P(Seattle | flew, to) = ?

Unigram		Bigram		Trigram	
P(Seattle) = 0.001	$\times\lambda_1$ +	**P(Seattle|to)** = 0.02	$\times\lambda_2$ +	**P(Seattle|flew,to)** = 0.15	$\times\lambda_3$

Example weights:
$\lambda_1=0.1,\ \lambda_2=0.3,\ \lambda_3=0.6$

Result:
P = 0.0961

0.1(0.001) + 0.3(0.02) + 0.6(0.15) = 0.0001 + 0.006 + 0.09 = 0.0961

Why interpolation works: Robustness through diversity

Trigram alone

✗ Sparse: many zero counts

✗ Unreliable for rare contexts

✓ Rich context when available

Unigram alone

✓ Dense: no zero counts

✓ Always has an estimate

✗ Ignores all context

Interpolated

✓ Uses context when available

✓ Falls back gracefully

✓ Never gives zero probability

Learning the interpolation weights

- Weights λ_i are **hyperparameters** that must be tuned
- Use a **held-out development set** (separate from training and test)
- Optimize weights to maximize likelihood on held-out data:

$$\hat{\lambda} = \arg\max_{\lambda} \prod_{w \in \text{dev}} P_{\text{interp}}(w|\text{context})$$

- Expectation-Maximization (EM) algorithm finds optimal λ iteratively

Interpolation vs. Backoff: Two strategies for combining models

Approach	Strategy	When to use lower-order
Interpolation	Always mix all orders	Every prediction
Backoff	Use highest order available	Only when higher-order count = 0

- Interpolation: $P = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$
- Backoff: Use P_3 if count > 0, else P_2, else P_1
- Kneser-Ney uses a sophisticated form of backoff with discounting

Concept Check

If $\lambda_1 = 0.1$, $\lambda_2 = 0.3$, $\lambda_3 = 0.6$, and you encounter a context never seen in training, which model contributes most to the prediction? What if the trigram has a reliable estimate?

Adjusted count matrices reveal smoothing's effect on probabilities

Raw Counts C

	the	cat	sat	dog
the	0	42	3	18
cat	1	0	25	0
sat	5	0	0	0
dog	0	0	12	0

9 zeros → P=0

Smoothed C* (+1)

	the	cat	sat	dog
the	1	43	4	19
cat	2	1	26	1
sat	6	1	1	1
dog	1	1	13	1

No zeros → all P>0

Difference C*–C

	the	cat	sat	dog
the	+1	+1	+1	+1
cat	+1	+1	+1	+1
sat	+1	+1	+1	+1
dog	+1	+1	+1	+1

Uniform +1 (Laplace)

Visualization helps diagnose smoothing behavior

MLE P(w|the) → (+1) → **Smoothed P(w|the)**

the	cat	sat	dog
0	.67	.05	.29

P("the the") = 0 → impossible!

the	cat	sat	dog
.01	.64	.06	.28

All events now possible

.67 → .64
frequent loses 3%

0 → .01
zero gains mass

Σ = 1.0
still a valid distribution

Part 3: Scaling to Large Corpora

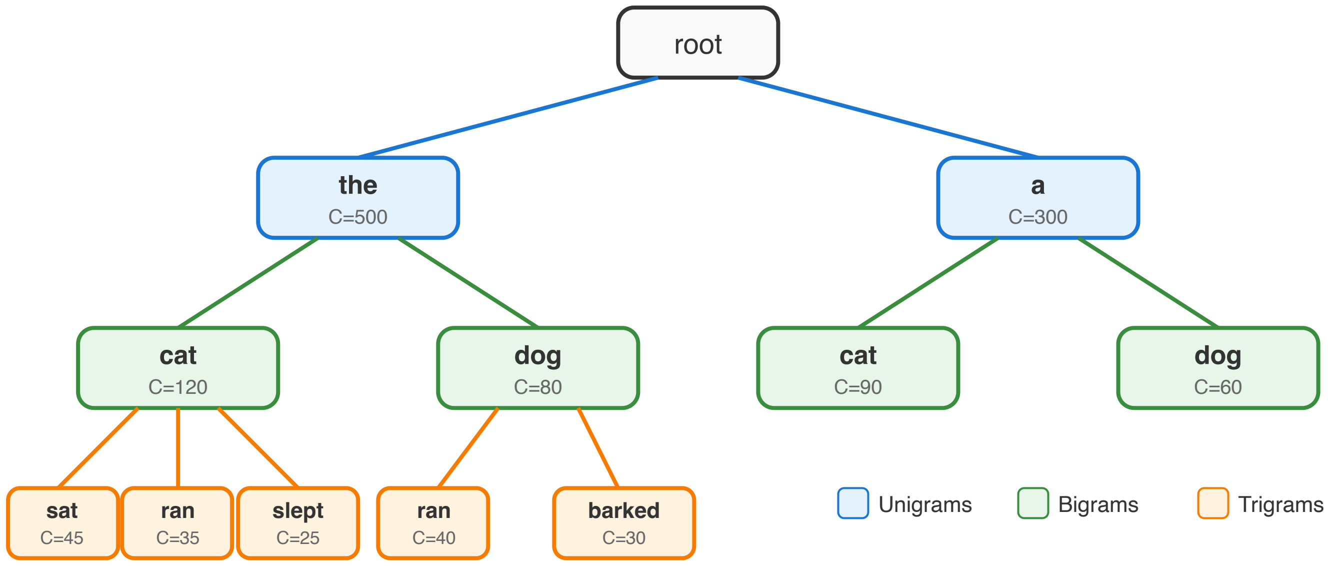
Explicit n-gram tables become infeasible at scale

- Memory grows as $O(|V|^n)$ for n-gram tables
- A trigram model with 100K vocabulary: 10^{15} possible entries
- Specialized data structures are essential

Tries share common prefixes to reduce memory

- Nodes represent shared prefixes
- Lookup for n-gram $w_1, \ldots, w_n$ is $O(n)$
- Example: "the cat sat" and "the cat ran" share "the cat" branch

Trie structure: Shared prefixes reduce storage



"the cat sat" and "the cat ran" share nodes for "the" and "the → cat"

Hash tables offer O(1) lookup but different tradeoffs

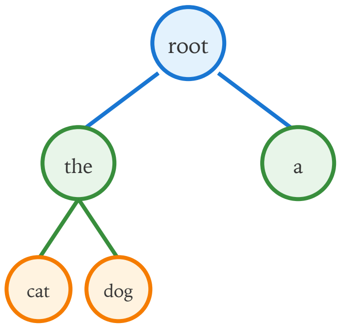
- Map n-grams directly to counts: $(\text{'the'}, \text{'cat'}) \rightarrow 42$
- $O(1)$ average lookup time
- Memory depends on collision handling and load factor

Structure choice depends on access patterns and constraints

Structure	Memory	Lookup	Best For
Trie	High (prefix sharing)	$O(n)$	Prefix queries, smoothing
Hash Table	Depends on load	$O(1)$ avg	Flat access, fixed n

Visual comparison: Trie vs Hash Table

Trie (Prefix Tree)



Prefix sharing saves memory

Prefix queries
Smoothing support

Hash Table

`h("the cat")`	→	42
`h("the dog")`	→	28
`h("a cat")`	→	15
`h("a dog")`	→	12

Flat structure: O(1) direct lookup

Fast lookup
Simple implementation

Concept Check

When would you prefer a trie over a hash table for n-gram storage? When would hash tables be better?

Infini-gram models remove the fixed-n constraint entirely

- Traditional models fix n and vocabulary V
- Infini-gram: context length k can be arbitrarily large
- Vocabulary grows dynamically with the data stream

Streaming algorithms enable trillion-token scale

- Online updates: as each token w_t arrives, update all relevant statistics
- Approximate counting (e.g., count-min sketch) bounds memory
- Adaptive pruning removes rare contexts

Infini-gram captures variable-length dependencies

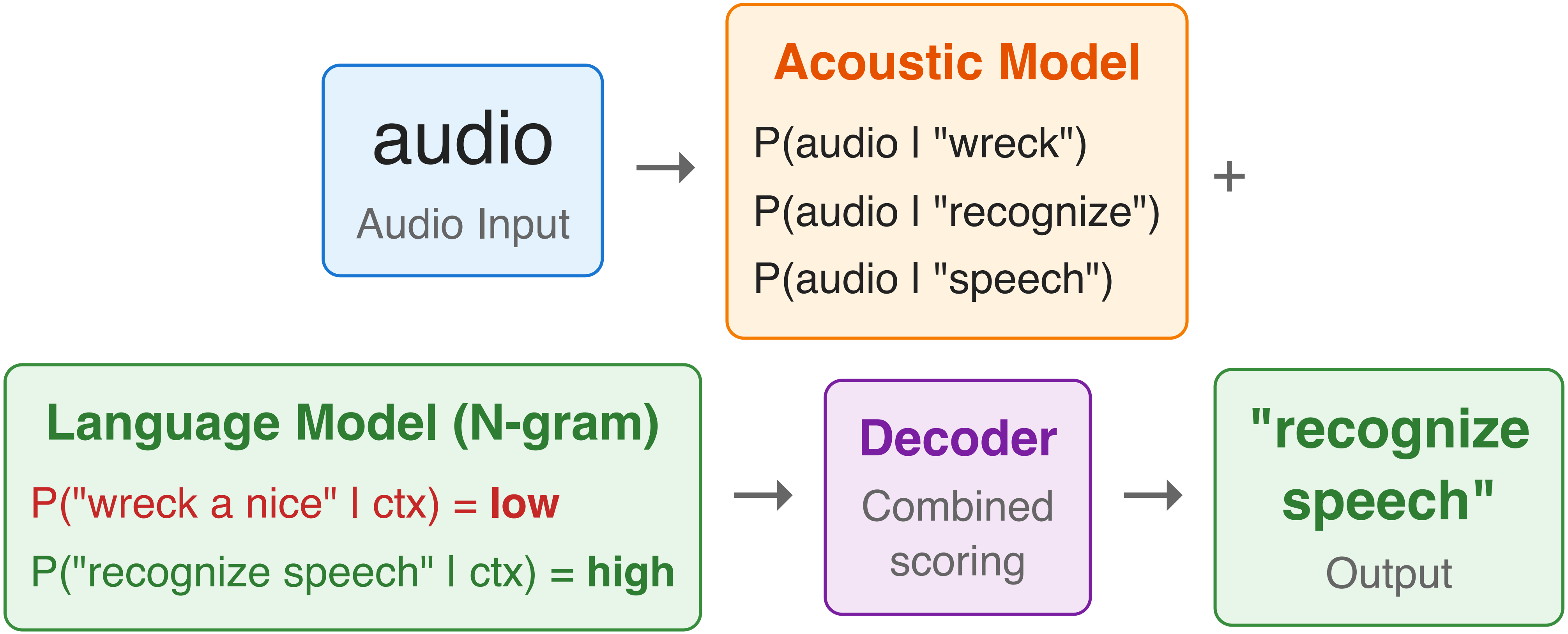
- Better fit for phenomena with variable context needs
- Code, dialogue, and poetry benefit from flexible context
- Empirical results show gains on rare events and long-context prediction

Part 4: Applications

Speech recognition combines acoustic and language model scores

- Decoder searches for most probable word sequence
- N-gram model provides $P(w_n \mid w_{n-1}, \ldots)$ for candidate words
- Acoustic model provides $P(\text{audio} \mid \text{words})$

Speech decoder: Language model guides acoustic search



Key insight: "Wreck a nice beach" sounds like "recognize speech" — the LM breaks the tie!

Smartphone keyboards use n-grams for next-word prediction

- Fast computation: lookup, not neural inference
- Suggest top-k words by $P(w \mid \text{context})$
- Enables low-latency, battery-efficient prediction

Concept Check

Your phone suggests "you" after "thank." Why might it not suggest "quantum"? What does this reveal about n-gram predictions?

N-gram models score translation fluency in statistical MT

- For target sentence $y = (y_1, \ldots, y_T)$:

$$P(y) = \prod_{t=1}^{T} P(y_t \mid y_{t-n+1}, \ldots, y_{t-1})$$

- Higher probability = more fluent target language
- Combined with alignment model for full translation score

SMT systems balance fidelity and fluency

- Joint scoring:

$$\text{Score} = \lambda_1 \log P_{\text{align}}(y \mid x) + \lambda_2 \log P_{\text{LM}}(y)$$

- P_{align}: how well does translation match source?
- P_{LM}: how fluent is the target sentence?

SMT scoring: Balancing fidelity and fluency

Source (German): *"Das Buch ist auf dem Tisch"*

Candidate	P_{align}	P_{LM}	Score
Candidate A: "The book is on the table"	**0.85**	**0.92**	**-2.1**
Candidate B: "Book the is table on the"	**0.90**	**0.001**	**-8.7**

P_{align}: Words match source?

P_{LM}: Sounds like English?

N-gram locality limits global coherence in translation

- Markov assumption restricts context to $n - 1$ tokens
- Subject-verb agreement across clauses may not be enforced
- Neural models (Transformers) capture longer dependencies

AAC systems can be personalized with user-specific n-gram models

- Train on user's own text to capture idiolect and preferences
- Support non-standard language: code-switching, creative expression
- Personalization enables authentic self-expression

Data sparsity is severe for personalized, non-standard models

- User-specific corpora are small
- Non-standard forms may never appear in training
- Smoothing and vocabulary updates are essential

Part 5: Evaluation and Limitations

Perplexity measures a model's average uncertainty per word

- For sequence $w_1, \ldots, w_N$:

$$\text{Perplexity}(M) = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P_M(w_i|w_{1:i-1})}$$

- Lower perplexity = higher average probability = better model
- Equivalent to the geometric mean of inverse probabilities

Perplexity intuition: "Effective vocabulary size"

Perplexity = 100

word1 | word2 | word3 | ... | word100

Model equally unsure among **100 choices**

VS

Perplexity = 10

the | a | to | ... | word10

Model narrows to **~10 likely choices**

Key insight: Perplexity = "branching factor" = average number of equally likely next words

Deriving perplexity: From probability to uncertainty

- Start with the probability of the test set $W = w_1, w_2, \ldots, w_N$:

$$P(W) = P(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} P(w_i \mid w_1, \ldots, w_{i-1})$$

- Problem: This number gets astronomically small!
- Solution: Work in log space and normalize by length

From log probability to cross-entropy

- **Log probability** of the test set:

$$\log P(W) = \sum_{i=1}^{N} \log P(w_i \mid w_{1:i-1})$$

- **Per-word log probability** (normalized):

$$\frac{1}{N} \sum_{i=1}^{N} \log P(w_i \mid w_{1:i-1})$$

Cross-entropy formula

- **Cross-entropy** H (using log base 2):

$$H(W) = -\frac{1}{N} \sum_{i=1}^{N} \log_2 P(w_i \mid w_{1:i-1})$$

From cross-entropy to perplexity

- **Perplexity** is 2 raised to the cross-entropy:

$$\text{PP}(W) = 2^{H(W)} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_{1:i-1})}$$

- Equivalently, the **inverse geometric mean** of probabilities:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i \mid w_{1:i-1})}}$$

- Why perplexity instead of cross-entropy?
  - More interpretable: "effective branching factor"
  - A perplexity of 100 = choosing uniformly among 100 options

Perplexity as branching factor: A visual metaphor

Predicting each word = choosing a branch

Intrinsic vs. Extrinsic evaluation

Intrinsic Evaluation

What: Evaluate the model itself

Metric: Perplexity on held-out data

Pros: Fast, task-independent, reproducible

Cons: May not correlate with task; sensitive to vocab

Extrinsic Evaluation

What: Evaluate on downstream task

Metric: Task accuracy (WER, BLEU, etc.)

Pros: Measures what we care about; actionable

Cons: Expensive; confounds LM with other factors

Best practice: Use intrinsic for rapid development; validate with extrinsic before deployment

The perplexity-task performance gap

- Lower perplexity **usually** helps, but not always!

When perplexity correlates well

- Similar domain (train $\approx$ test)
- Same vocabulary
- Task relies heavily on fluency

When it may not correlate

- Domain mismatch
- Task needs specific knowledge
- Other system components dominate

- **Rule of thumb**: A 10-20% perplexity reduction often yields measurable task gains
- Always validate with extrinsic evaluation before deployment!

OOV words can cause perplexity to explode to infinity

- If $P_M(w_i) = 0$, perplexity is undefined (infinite)
- Smoothing prevents this by ensuring all words have nonzero probability
- Fair comparison requires identical vocabulary and OOV handling

Perplexity benchmarks reveal model quality differences

Model	Penn Treebank Perplexity
Trigram	$\sim$140
LSTM	$\sim$80
Transformer	$\sim$20

- Lower is better, but domain and vocabulary must match
- Perplexity doesn't capture all aspects of quality

Concept Check

A model has perplexity 50 on news text but perplexity 200 on social media. What might explain this? Is the model "bad"?

Context windows fundamentally limit what n-grams can capture

- Trigrams see only 2 words of history
- Even Transformers have finite context windows (512, 2048, …)
- Long-range dependencies (agreement, coreference) may be missed

Training data biases propagate into model predictions

- Underrepresented dialects (e.g. regional varieties) get worse predictions
- Models amplify demographic, topical, and stylistic imbalances
- More data doesn't automatically fix bias—diversity matters

Language models can perpetuate and amplify societal biases

- Web-scale data encodes human biases
- Underrepresentation leads to systematic errors for marginalized groups
- Fairness means comparable performance across language varieties

Mitigation requires explicit attention to fairness

- Data augmentation with diverse language varieties
- Dialect-sensitive evaluation benchmarks
- Model interpretability to understand failure modes
- Algorithmic transparency for deployment decisions

Summary: N-gram models remain foundational despite limitations

- **Core idea**: Predict next word from fixed context window
- **Estimation**: MLE + smoothing to handle unseen events
- **Scaling**: Tries, hashing, streaming for large corpora
- **Applications**: Speech, translation, AAC, text prediction
- **Evaluation**: Perplexity, but watch for bias and fairness gaps