Infini-gram: Scaling n-gram Language Models to a Trillion Tokens

The return of n-grams

To Infinity and Beyond!

Infinitely long n-grams with backoff

Prompt ... conducts research at the Paul G. Allen School of Computer Science and Engineering, University of

5-gram LM $(n = 5)$

∞-gram LM ($n = 16$ for this case)

cnt(Engineering, University of) $= 274644$

$P(\,*\,|$ Engineering, University of$) =$

_California	(20896 / 274644)	8%
_Illinois	(10631 / 274644)	4%
_Michigan	(9094 / 274644)	3%
_Colorado	(6438 / 274644)	2%
_Southern	(6340 / 274644)	2%
_Washington	(6340 / 274644)	2%

...

cnt(research at the Paul G. Allen School of Computer Science and Engineering, University of) $= 0$

cnt(at the Paul G. Allen School of Computer Science and Engineering, University of) $= 10$

$P(\,*\,|$ at the Paul G. Allen School of Computer Science and Engineering, University of$) =$

_Washington	(10 / 10)	100%

5-gram table (28 TiB)

Infini-gram index (10 TiB)

N-gram

- Sequence of n adjacent symbols in order
  - 1 gram: a, b, c, d
  - 2gram: ab, cd, ee, po
  - 3gram: abc, pow, ivo, ovq

N-Gram Probability

- Probability of the next token given (n-1) context
  - Bigram: p(a | b)
  - Trigram Example: p(a | po)
- N-Gram Probability is used interchangeably with N-Gram
  - Choose the right definition according to context
- In general, n-gram probability is calculated using

$$P(a_n | a_1 a_2 \cdots a_{n-1}) = \frac{cnt(a_1 a_2 \cdots a_{n-1} a_n)}{cnt(a_1 a_2 \cdots a_{n-1})}$$

Backoff

- Sometimes n-gram sequence is rare or unseen in training data
- In this case, using less context might be helpful

If $cnt(a_1 a_2 \cdots a_{n-1} a_n) = 0$ but $cnt(a_3 \cdots a_{n-1} a_n) = 10$, we might want to calculate

$$P(a_n | a_3 \cdots a_{n-1}) = \frac{cnt(a_3 \cdots a_{n-1} a_n)}{cnt(a_3 \cdots a_{n-1})}$$

Instead of

$$P(a_n | a_1 a_2 \cdots a_{n-1}) = \frac{cnt(a_1 a_2 \cdots a_{n-1} a_n)}{cnt(a_1 a_2 \cdots a_{n-1})}$$

∞-Gram: definition

$$P_\infty(w_i \mid w_{1:i-1}) = \frac{\text{cnt}(w_{i-(n-1):i-1} w_i \mid \mathcal{D})}{\text{cnt}(w_{i-(n-1):i-1} \mid \mathcal{D})}$$

- N-grams that are extrapolated to infinity.
- Backoff when the denominator is zero
- An infini-gram is **sparse** when $P_\infty(w_i|w_{1:i-1}) = 1$ for some w_i
- An **effective n** of an infini-gram is equal to one plus the length of the prompt's longest suffix that appears in the training data.
  - Given a string, n of the longest n-gram that appeared in the corpus.

Suffix Array

- A sorted list of all suffix start indices in a string

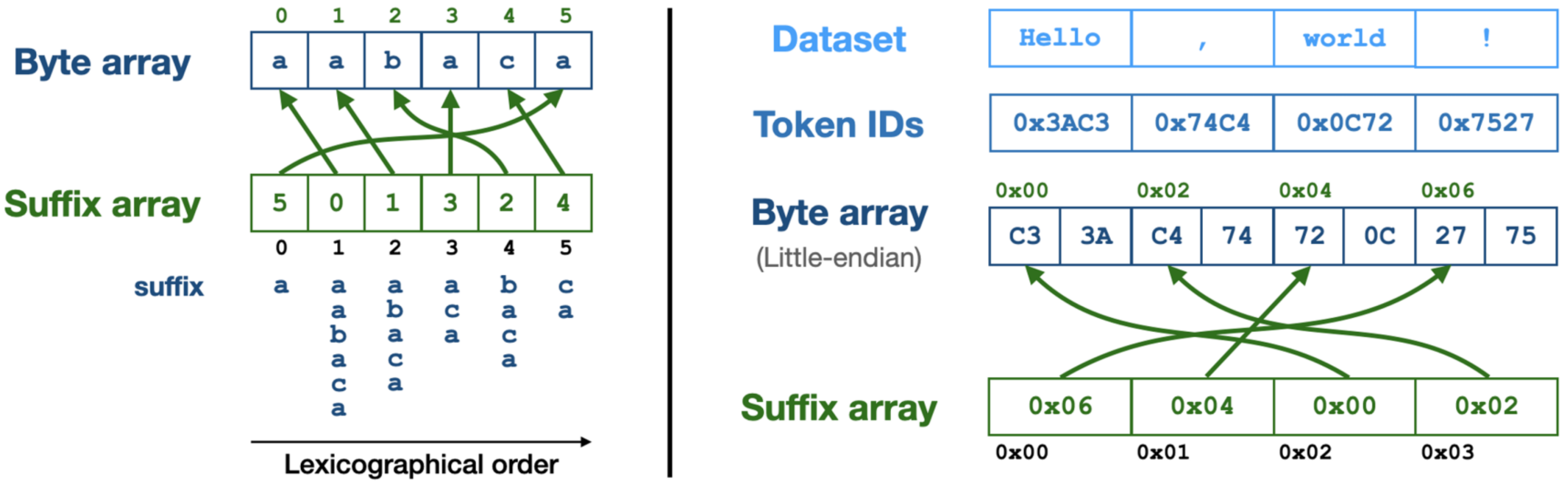


Figure 2: **Left:** the suffix array for a toy string. **Right:** illustration of the suffix array in the infini-gram index, with $N = 4$ tokens in the dataset.

Suffix Array

- Suffixes of “APPLE”
    - APPLE
    - PPLE
    - PLE
    - LE
    - E

Suffix Array

- Suffixes of “APPLE”
  - APPLE
  - PPLE
  - PLE
  - LE
  - E

Alphabetical ordering

- Ordered Suffixes of “APPLE”
  - APPLE
  - E
  - LE
  - PLE
  - PPLE

Suffix Array

- Suffixes of “APPLE”
  - APPLE
  - PPLE
  - PLE
  - LE
  - E

Alphabetical ordering

- Ordered Suffixes of “APPLE” (index)
  - APPLE [0]
  - E [4]
  - LE [3]
  - PLE [2]
  - PPLE [1]

Suffix Array: [0,4,3,2,1]

Can use binary search to find suffixes (infini-grams)

N-Gram Search

- Ordered Suffixes of “APPLE” (index)
  - APPLE [0]
  - E [4]
  - LE [3]
  - PLE [2]
  - PPLE [1]

w = APPLE.   w[j]: suffix after jth index

i = [0, 4, 3, 2, 1]

Want to find bigram that starts with P.

N-Gram Search

- Ordered Suffixes of “APPLE” (index)
    - APPLE [0]
    - E [4]
    - LE [3]
    - PLE [2]
    - PPLE [1]

w = APPLE.   w[j]: suffix after jth index

i = [0, 4, 3, 2, 1]

Want to find bigram that starts with P.

Check w[i[2]] = w[3] = LE

N-Gram Search

- Ordered Suffixes of “APPLE” (index)
  - APPLE [0]
  - E [4]
  - LE [3]
  - PLE [2]
  - PPLE [1]

w = APPLE. w[j] suffix after jth index

i = [0, 4, 3, 2, 1]

Want to find bigram that starts with P.

Check w[i[2]] -> w[3] = LE

LE < P, check w[i[4]]

N-Gram Search

- Ordered Suffixes of “APPLE” (index)
  - APPLE [0]
  - E [4]
  - LE [3]
  - PLE [2]
  - PPLE [1]

w = APPLE.   w[j] suffix after jth index

i = [0, 4, 3, 2, 1]

Want to find bigram that starts with P.

Check w[i[2]] -> w[3] = LE

LE < P, check w[i[4]]

w[I[4]] = w[1] = PPLE

PPLE > P and is the last index, search for start index if bigram that starts with p

N-Gram Search

- Ordered Suffixes of “APPLE” (index)
  - APPLE [0]
  - E [4]
  - LE [3]
  - PLE [2]
  - PPLE [1]

w = APPLE.   w[j] suffix after jth index

i = [0, 4, 3, 2, 1]

Want to find bigram that starts with P.

Check w[i[2]] -> w[3] = LE

LE < P, check w[i[4]]

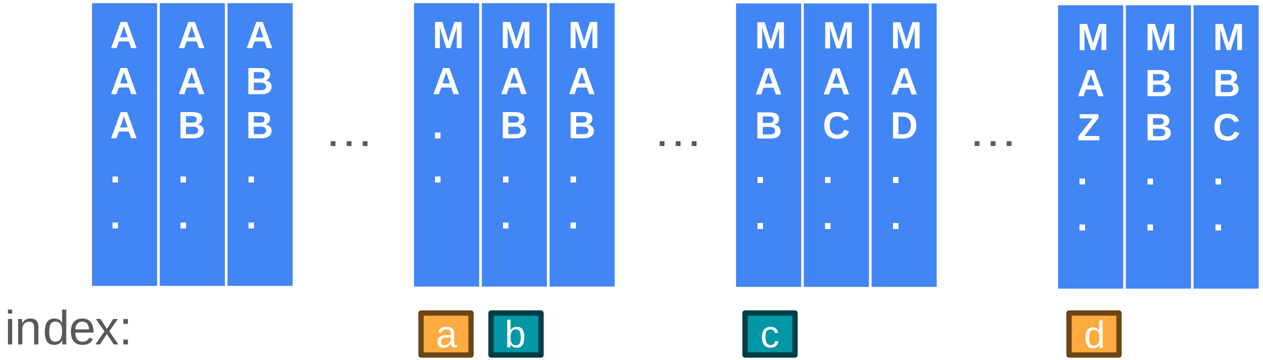
w[I[4]] = w[1] = PPLE

PPLE > P and is the last index, search for start index if bigram that starts with p

w[i[3]] = w[2] = PLE

Thus there are two bigrams that start with **P**. Namly **PP** and **PL**

Infini-grams



- Want to find $p(B \mid MA)$
- $p(B \mid MA) = \dfrac{cnt(MAB)}{cnt(MA)} = \dfrac{c - b}{d - a}$
- Values of a, b, c, d can be found by binary search

Let i_c^s denote the first index from the suffix array where the suffix starts with s. Let i_c^e denote the last index from the suffix array where the suffix array starts with s. Then

$$P_\infty(w|c) = \frac{i_{wc}^e - i_{wc}^s}{i_c^e - i_c^s}$$

- Indices can be found by binary search

Suffix Array is Efficient

- Space efficient in O(N). ... (N: size of the corpus)
    - O(N log(N)) But in reality O(N)
- Search Efficient in O(log(N))
- Next word prediction with prob > .5 is fast
    - Check the .25, .5, .75 index of the n-gram
    - ∞-grams **agree** with text if the next word prediction from suffix array has probability greater than .5 and matches text.

Suffix Array is (not) Efficient

- Full next-token distribution calculation is slow
  - O(V logN) (V is size of vocab, N is size of corpus)
- argmax (most possible next word prediction) is slow

Reference Data $(\rightarrow)$ **Query Type** $(\downarrow)$	**Pile-train** $N = 0.36\text{T}$ $S = 2$	**RPJ** $N = 1.4\text{T}$ $S = 8$	**Time Complexity** (measured by number of random disk accesses)
1. Counting an n-gram			$O(\log N)$
... $(n = 1)$	7 ms	9 ms	
... $(n = 2)$	13 ms	20 ms	
... $(n = 5)$	14 ms	19 ms	
... $(n = 10)$	13 ms	18 ms	
... $(n = 100)$	13 ms	19 ms	
... $(n = 1000)$	14 ms	19 ms	
2. Computing a token probability from n-gram LM $(n = 5)$	19 ms	30 ms	$O(\log N)$
3. Computing full next-token distribution from n-gram LM (n = 5)	31 ms	39 ms	$O(V \cdot \log N)$
4. Computing a token probability from ∞-gram LM	90 ms	135 ms	$O(\log L \cdot \log N)$
... on consecutive tokens	12 ms	20 ms	$O(\log N)$
5. Computing full next-token distribution from ∞-gram LM	88 ms	180 ms	$O((\log L + V) \cdot \log N)$

Demo

- https://huggingface.co/spaces/liujch1998/infini-gram

Train / Test Data

- Train (Reference):
  - Pile-train (380B tokens)
  - RedPajama (1.4T tokens) for some experiments
- Test:
  - Pile-val
  - Pile-test

Decontamination of the Training Data

- Filtering out repeated documents in training and test data
  - Important, because ∞-grams memorizes sparse sequences
- Document-wise filter
- 80% 13-gram overlap

REDPAJAMA			
Subset	**Total docs**	**Filtered docs**	**Ratio filtered**
arxiv	1558306	213	0.01%
book	205744	711	0.3%
c4	364868892	53195	0.01%
common_crawl	476276019	0	0%
github	28793312	614259	2%
stackexchange	29825086	40086	0.01%
wikipedia	29834171	21973	0.07%
Total	**931361530**	**730437**	**0.08%**

PILE (TRAIN)			
Subset	**Total docs**	**Filtered docs**	**Ratio filtered**
Arxiv	2377741	1089	
BookCorpus2	25355	6	
Books3	277655	99	
DM Mathematics	1918535	0	0%
Enron Emails	926132	18236	2%
EuroParl	131723	21	
FreeLaw	5069088	11821	0.2%
Github	18044218	961726	5.3%
Gutenberg (PG-19)	66981	70	0.1%
HackerNews	1571968	14	
NIH ExPorter	1777926	3739	0.2%
OpenSubtitles	632485	5754	0.9%
OpenWebText2	32333654	136914	0.4%
PhilPapers	63875	2324	0.4%
Pile-CC	52441354	19928	
PubMed Abstracts	29329202	2312	
PubMed Central	5679903	4230	0.1%
StackExchange	29529008	2072	
USPTO Backgrounds	11123325	80088	0.7%
Ubuntu IRC	20067	10	
Wikipedia (en)	16939503	45052	0.3%
YoutubeSubtitles	328030	871	0.3%
Total	**210607728**	**1296376**	**0.6%**

Results

Comparing with Human-Written Text: Setup

- Next-token prediction
- Measure token-wise agreement between the predicted token and human-written text
  - A prediction is deemed in-agreement if $p > 0.5$
  - Why agreement? Why not perplexity?
    - Because getting probabilities for every possible token is slow…
    - If the prediction is sparse and wrong, then perplexity = ∞

The cat sat on the ____

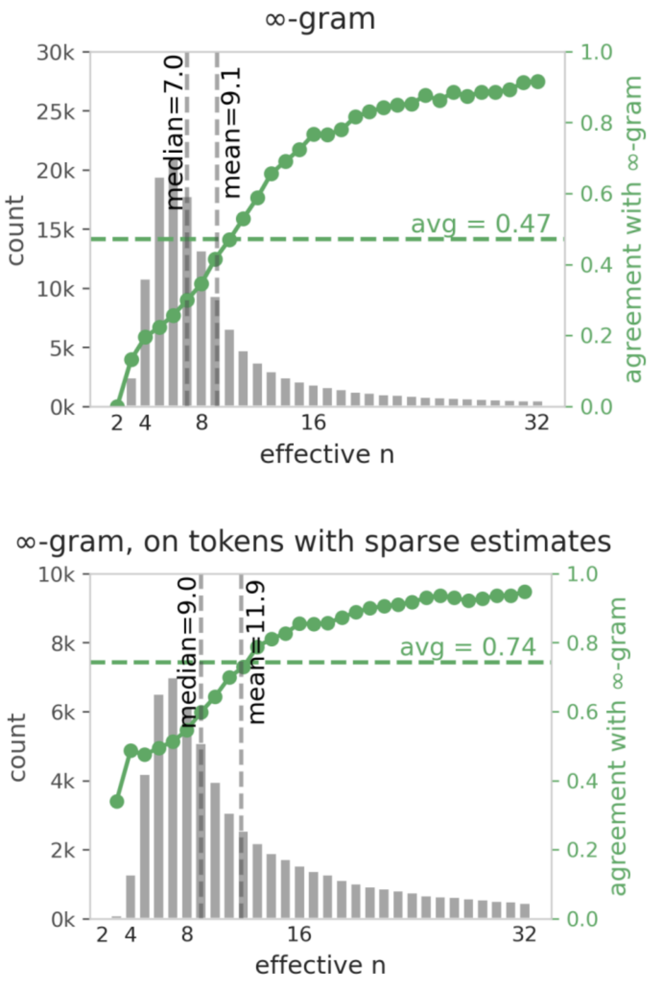
mat	(p=0.6)	✅
desk	(p=0.2)	
bed	(p=0.05)	
⋮	⋮	

I have a pet ____

dog	(p=0.3)	❌
cat	(p=0.25)	
snake	(p=0.05)	
⋮	⋮	

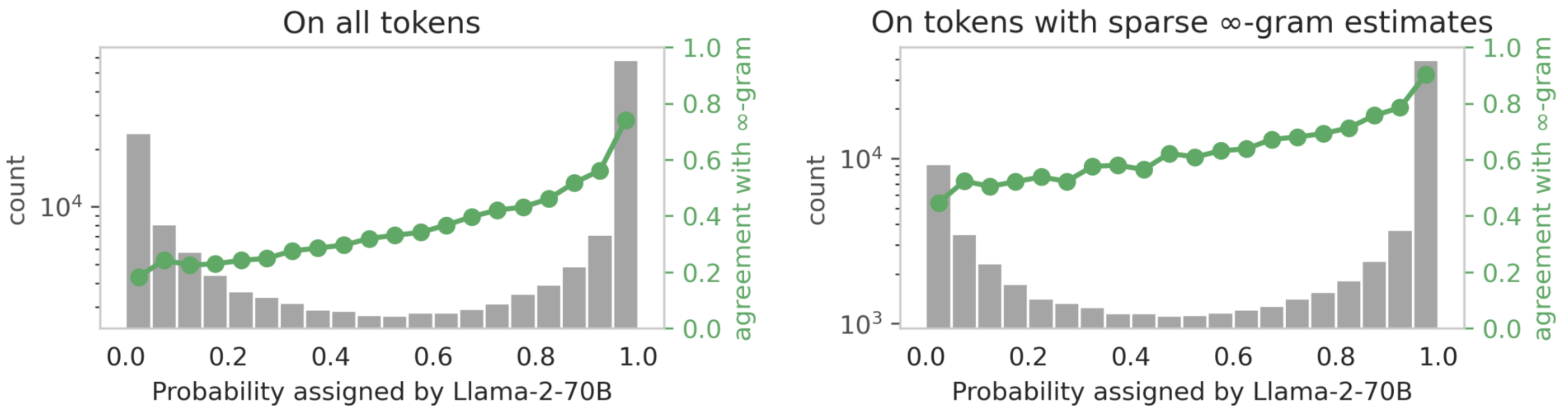
Human-Written Text: Results

- 47% overall agreement rate
- Larger effective n = higher agreement
  - Effective n: the actual length of context (+1) being used in a prediction
  - 75% agreement for n = 16
- Sparse = higher agreement
  - 75% overall sparse agreement



Human-Written Text: against neural LMs

- "∞-grams shines where neural LMs fail"
  - N-gram performance is nontrivial even for tokens in which Llama performs very poorly

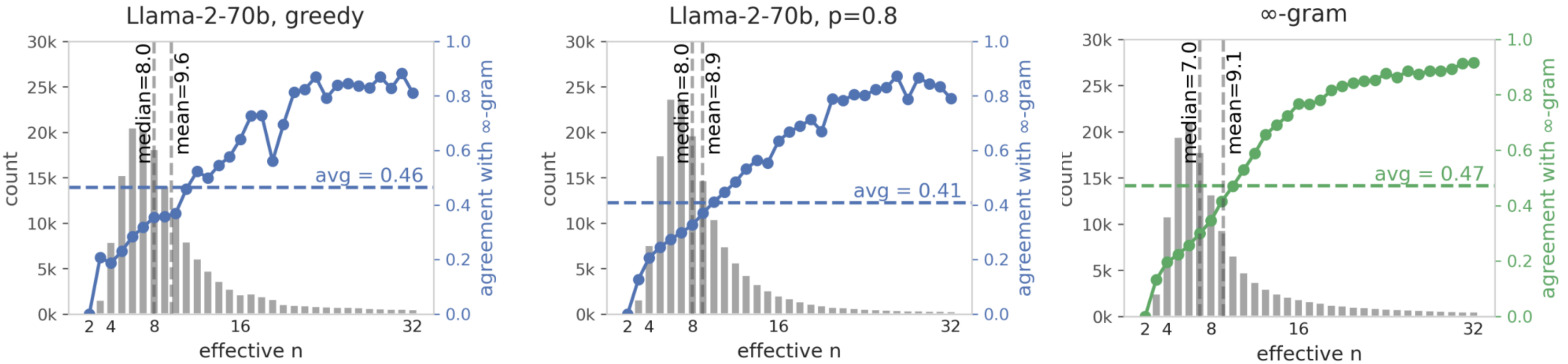


Comparing with Machine-Generated Text: Setup

- Generate a sequence with a model, then test for agreement with ∞-grams next-token prediction

Machine-Generated Text: Results

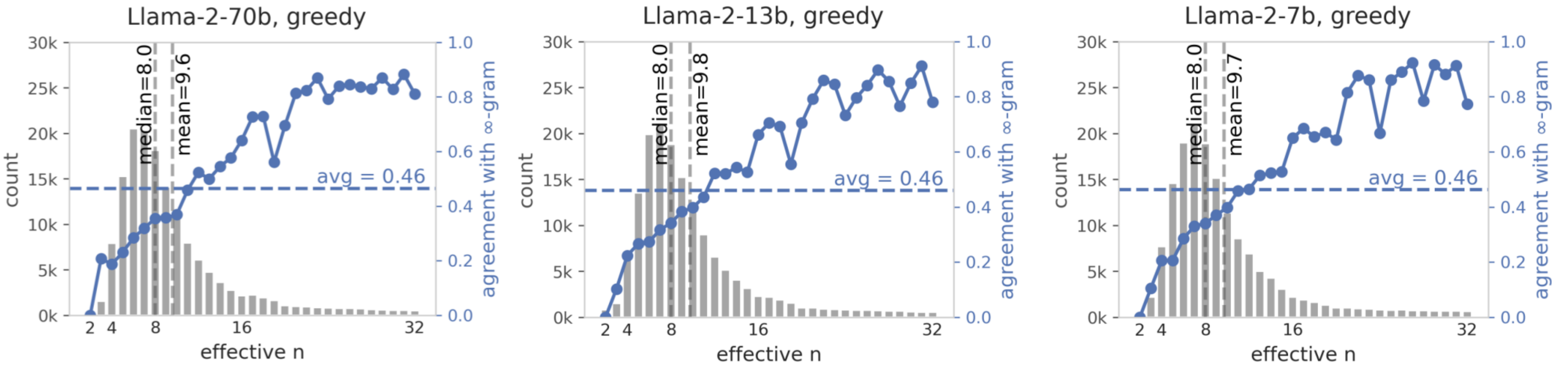
- Impact of decoding methods
  - Greedy: most agreement
  - Nucleus (top-p): most similar distribution to ∞-grams vs. human-written text



(Human)

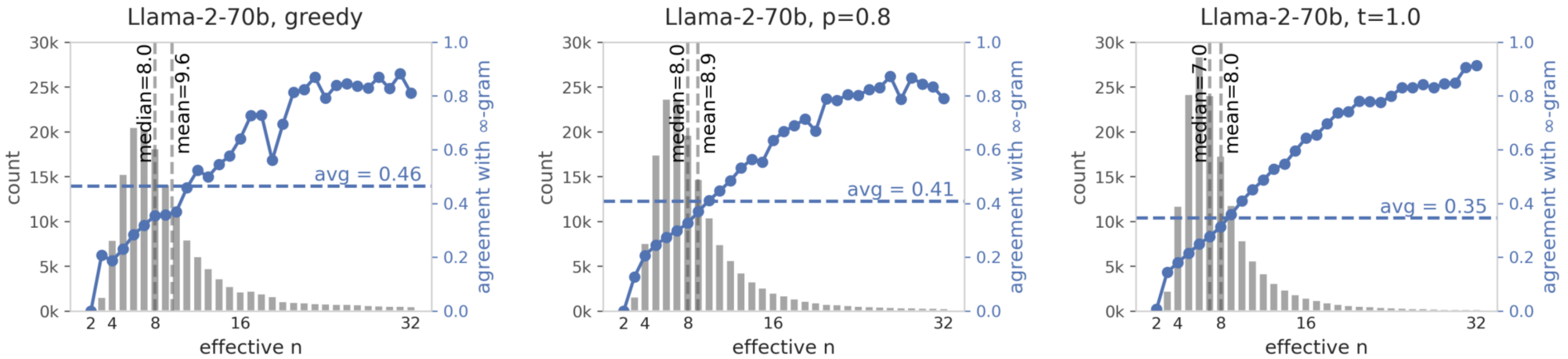
Machine-Generated Text: Results

- Impact of model size
  - Claim: increasing model size increases agreement level and slightly increases effective-n
  - What does effective-n mean?
    - Higher effective-n = the generator is more likely to copy verbatim from the training data (if the training and reference data overlap)



Machine-Generated Text: Results

- Curious phenomenon:
  - For greedy decoding, agreement level fluctuates as effective-n increases
  - Not for nucleus or temperature sampling
  - Suspected reason: have something to do with positional embeddings



Can this help LLMs?

- Interpolate the probability of the infini-grams and neural network
  - Different lambda values for sparse infinigrams
  - Lambda values optimized on Pile-val

$$
\begin{cases}
P(y \mid x) = \lambda_1 P_\infty(y \mid x) + (1-\lambda_1) P_{\text{neural}}(y \mid x) & \boxed{\text{if } P_\infty(y_i|x) = 1 \quad \text{(sparse)}} \\
P(y \mid x) = \lambda_2 P_\infty(y \mid x) + (1-\lambda_2) P_{\text{neural}}(y \mid x) & \boxed{\text{if } 0 < P_\infty(y_i|x) < 1 \quad \text{(non-sparse)}}
\end{cases}
$$

Interpolating with neural LMs: Results

- Significant improvements on perplexity
  - 11% to 42%

Neural LM	Size	Reference Data	Validation			Test		
			Neural	+ ∞-gram		Neural	+ ∞-gram	
GPT-2	117M	Pile-train	22.82	**13.71**	(42%)	22.86	**13.58**	(42%)
GPT-2	345M	Pile-train	16.45	**11.22**	(34%)	16.69	**11.18**	(35%)
GPT-2	774M	Pile-train	15.35	**10.39**	(35%)	15.40	**10.33**	(35%)
GPT-2	1.6B	Pile-train	14.42	**9.93**	(33%)	14.61	**9.93**	(34%)
GPT-Neo	125M	Pile-train	13.50	**10.76**	(22%)	14.08	**10.79**	(25%)
GPT-Neo	1.3B	Pile-train	8.29	**7.31**	(13%)	8.61	**7.36**	(16%)
GPT-Neo	2.7B	Pile-train	7.46	**6.69**	(12%)	7.77	**6.76**	(15%)
GPT-J	6.7B	Pile-train	6.25	**5.75**	(10%)	6.51	**5.85**	(12%)
Llama-2	7B	Pile-train	5.69	**5.05**	(14%)	5.83	**5.06**	(16%)
Llama-2	13B	Pile-train	5.30	**4.75**	(13%)	5.43	**4.76**	(15%)
Llama-2	70B	Pile-train	4.59	**4.21**	(11%)	4.65	**4.20**	(12%)
Llama-2	7B	Pile-train + RedPajama	5.69	**4.66**	(22%)	5.83	**4.66**	(24%)
Llama-2	13B	Pile-train + RedPajama	5.30	**4.41**	(21%)	5.43	**4.42**	(23%)
Llama-2	70B	Pile-train + RedPajama	4.59	**3.96**	(18%)	4.65	**3.95**	(19%)

Interpolating with neural LMs: Results

- Smaller LMs = more improvement (for models in the same family)
  - Does not hold across families, some models are already trained on Pile

Neural LM	Size	Reference Data	Validation			Test		
			Neural	+ ∞-gram		Neural	+ ∞-gram	
GPT-2	117M	Pile-train	22.82	**13.71**	(42%)	22.86	**13.58**	(42%)
GPT-2	345M	Pile-train	16.45	**11.22**	(34%)	16.69	**11.18**	(35%)
GPT-2	774M	Pile-train	15.35	**10.39**	(35%)	15.40	**10.33**	(35%)
GPT-2	1.6B	Pile-train	14.42	**9.93**	(33%)	14.61	**9.93**	(34%)
GPT-Neo	125M	Pile-train	13.50	**10.76**	(22%)	14.08	**10.79**	(25%)
GPT-Neo	1.3B	Pile-train	8.29	**7.31**	(13%)	8.61	**7.36**	(16%)
GPT-Neo	2.7B	Pile-train	7.46	**6.69**	(12%)	7.77	**6.76**	(15%)
GPT-J	6.7B	Pile-train	6.25	**5.75**	(10%)	6.51	**5.85**	(12%)
Llama-2	7B	Pile-train	5.69	**5.05**	(14%)	5.83	**5.06**	(16%)
Llama-2	13B	Pile-train	5.30	**4.75**	(13%)	5.43	**4.76**	(15%)
Llama-2	70B	Pile-train	4.59	**4.21**	(11%)	4.65	**4.20**	(12%)
Llama-2	7B	Pile-train + RedPajama	5.69	**4.66**	(22%)	5.83	**4.66**	(24%)
Llama-2	13B	Pile-train + RedPajama	5.30	**4.41**	(21%)	5.43	**4.42**	(23%)
Llama-2	70B	Pile-train + RedPajama	4.59	**3.96**	(18%)	4.65	**3.95**	(19%)

Interpolating with neural LMs: Results

- More reference context helps

Neural LM	Size	Reference Data	Validation			Test		
			Neural	+ ∞-gram		Neural	+ ∞-gram	
GPT-2	117M	Pile-train	22.82	**13.71**	(42%)	22.86	**13.58**	(42%)
GPT-2	345M	Pile-train	16.45	**11.22**	(34%)	16.69	**11.18**	(35%)
GPT-2	774M	Pile-train	15.35	**10.39**	(35%)	15.40	**10.33**	(35%)
GPT-2	1.6B	Pile-train	14.42	**9.93**	(33%)	14.61	**9.93**	(34%)
GPT-Neo	125M	Pile-train	13.50	**10.76**	(22%)	14.08	**10.79**	(25%)
GPT-Neo	1.3B	Pile-train	8.29	**7.31**	(13%)	8.61	**7.36**	(16%)
GPT-Neo	2.7B	Pile-train	7.46	**6.69**	(12%)	7.77	**6.76**	(15%)
GPT-J	6.7B	Pile-train	6.25	**5.75**	(10%)	6.51	**5.85**	(12%)
Llama-2	7B	Pile-train	5.69	**5.05**	(14%)	5.83	**5.06**	(16%)
Llama-2	13B	Pile-train	5.30	**4.75**	(13%)	5.43	**4.76**	(15%)
Llama-2	70B	Pile-train	4.59	**4.21**	(11%)	4.65	**4.20**	(12%)
Llama-2	7B	Pile-train + RedPajama	5.69	**4.66**	(22%)	5.83	**4.66**	(24%)
Llama-2	13B	Pile-train + RedPajama	5.30	**4.41**	(21%)	5.43	**4.42**	(23%)
Llama-2	70B	Pile-train + RedPajama	4.59	**3.96**	(18%)	4.65	**3.95**	(19%)

Text generation

- Might harm generation, because the infinigram model may predict completely irrelevant tokens and make the model digress.

Questions

- Isn’t this just memorization?
  - (Isn’t neural LMs also just memorization of probabilistic distributions of a language?)
- Loss on infinigrams alone?
  - We can use backoff + smoothing to estimate perplexity.