Training LLMs over Neurally Compressed Text

Aydan Huang, Shaobo Liang

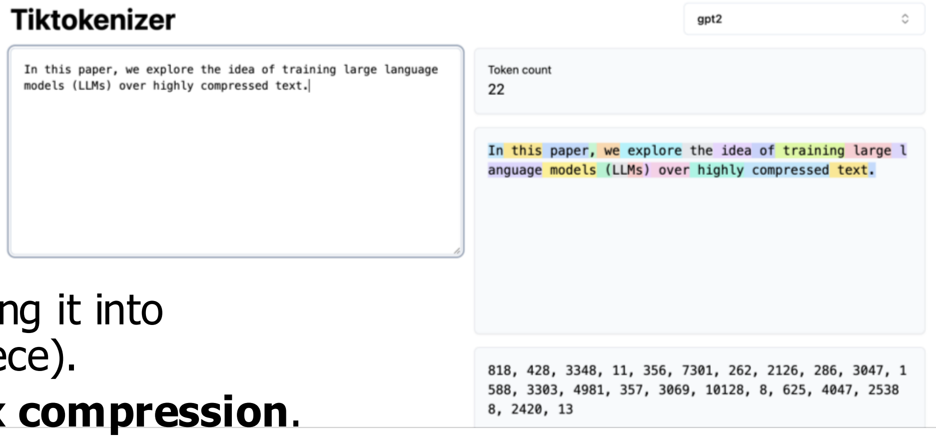
Johns Hopkins Whiting School of Engineering

Intro

- **Goal**: Improve training efficiency and handle longer contexts.
- **Why Compression?**
  - Reduce token length.
  - Maximize compute efficiency.
  - Process longer sequences within model limits.

What is compression?

- Represent text with fewer bits while retaining information
- **Types of Compression**:
  - **Lossless**: No information loss (e.g. Arithmetic Coding)
  - **Lossy**: Discards some information for higher compression rates (e.g. JPEG).
- **Information Theory Basics**:
  - **Entropy**: Minimum average bit length needed to represent symbols.
  - **Probabilistic Modeling**: Assigns shorter codes to frequent symbols, longer codes to rare ones.



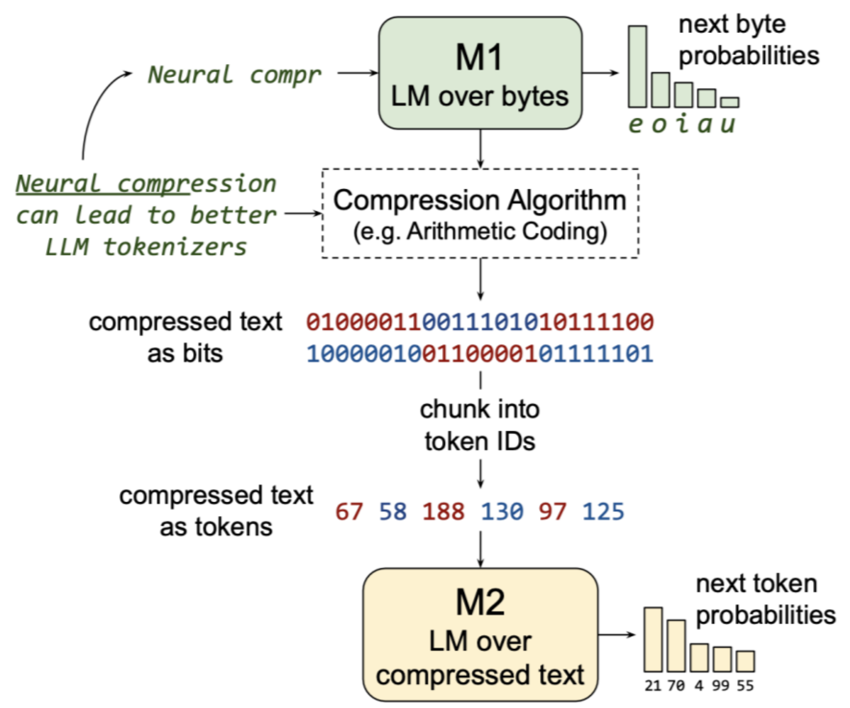
Background

- **Subword Tokens**
  - Traditional LLMs process text by breaking it into subword tokens (e.g. BPE, SentencePiece).
  - Common tokenizers achieve around **4x compression**.
- **Neurally Compressed Text**
  - Train a model to compress text by assigning probabilities to sequences.
  - LLM can achieve **12x compression** over English text
- **Question**
  - Could we compress text even further to achieve greater compression rates and improve model efficiency?

(Delétang et al., 2024)

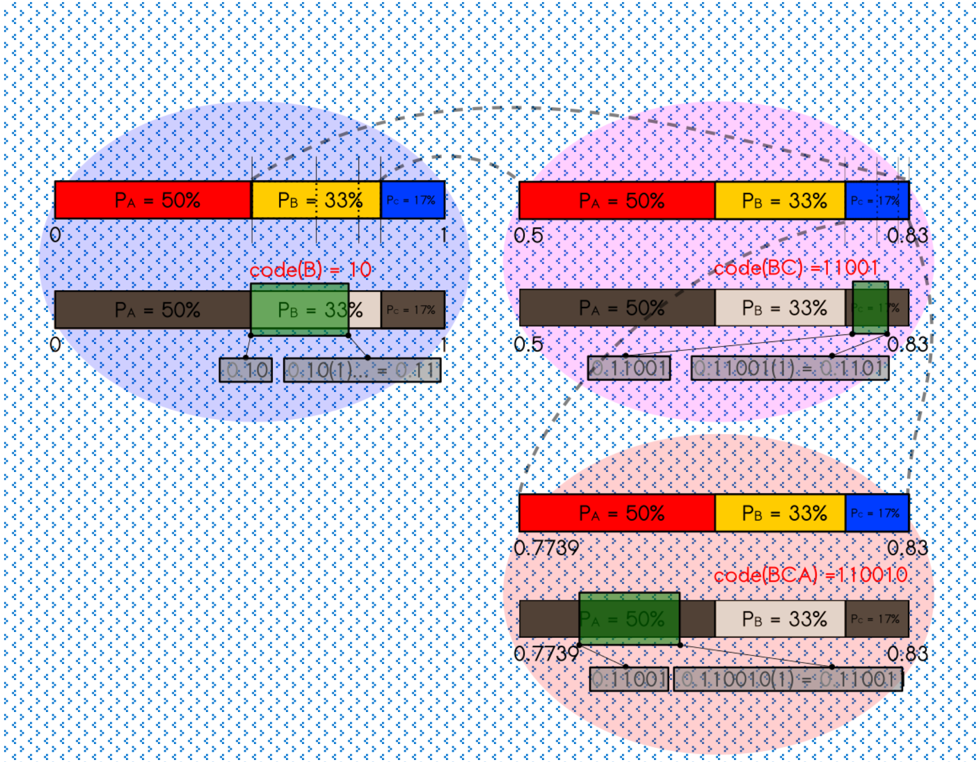
Pipeline

- Use Arithmetic Coding to reach near-optimal compression
- Pipeline:
  - **M1:** Small language model trained on **raw byte sequences**.
  - **AC:** Compresses text to a bitstream.
  - **M2:** Trains on **compressed tokens** from AC bitstream.



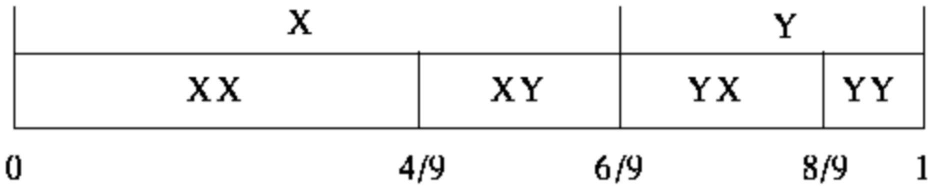
Arithmetic Coding

- Near-optimal coding method that encodes text into a bitstream.
- **Process**:
  - Divides [0, 1) into intervals based on symbol probabilities.
  - Encodes sequences by refining intervals.



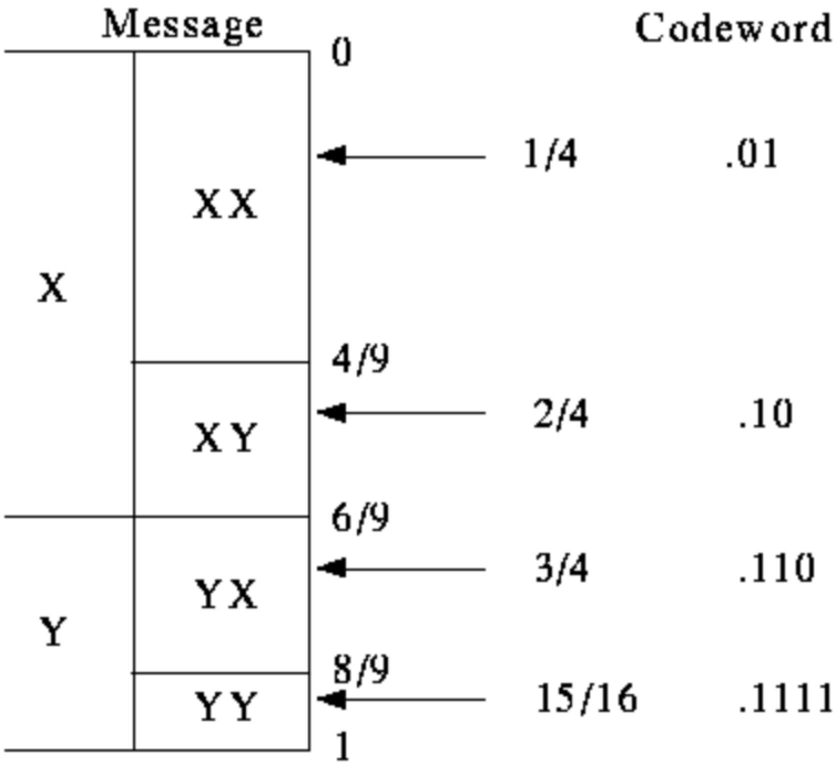
Example!

- Let A = {X, Y}
- P(X)=2/3, P(Y)=1/3
- Encoding length 2 message



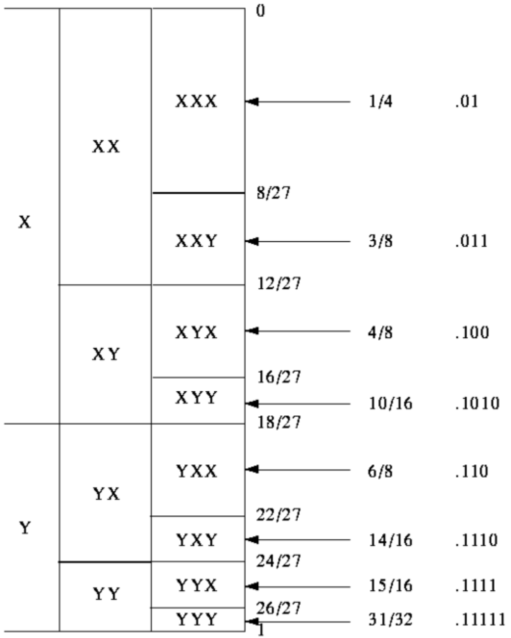
Example!

- To encode message, just send enough bits of a binary fraction that uniquely specifies the interval



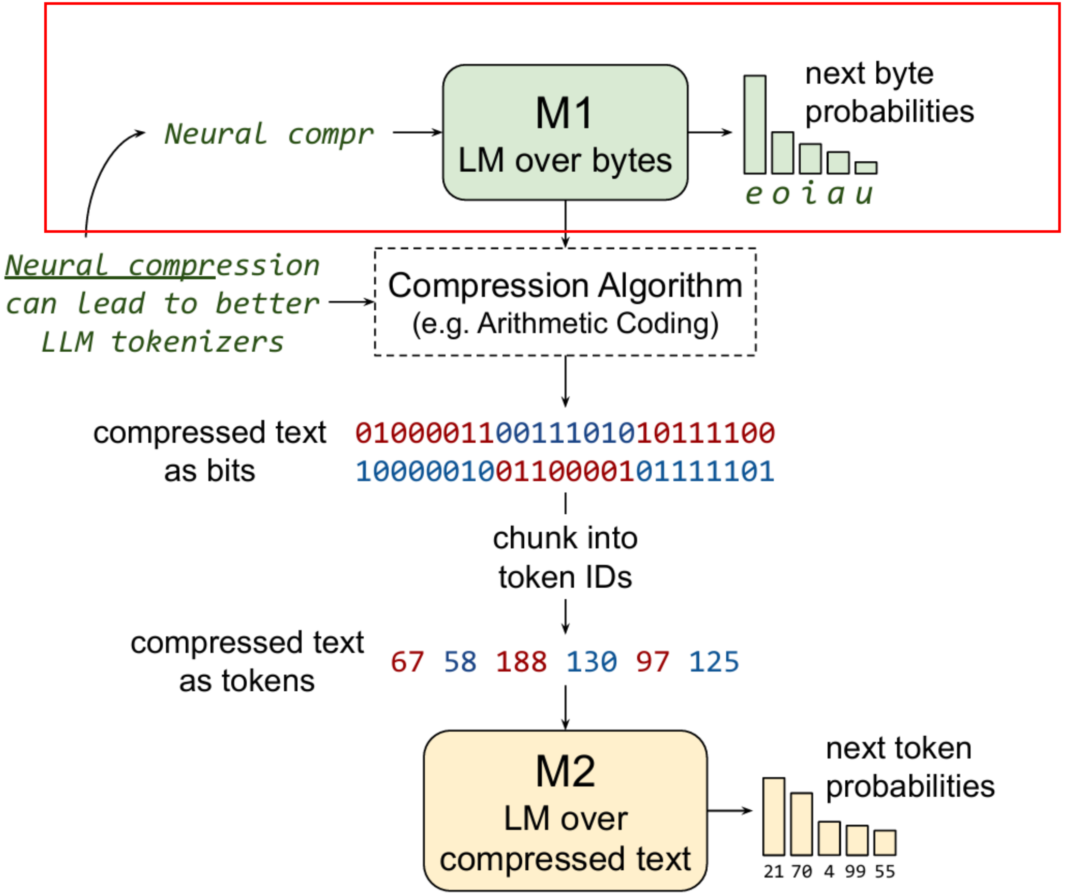
Example!

- all possible length 3 messages to intervals



- In general, number of bits is determined by the size of the interval
  - need **–log2(p)** bits to represent **interval of size p**
- Approaches optimal encoding as message length got to infinity

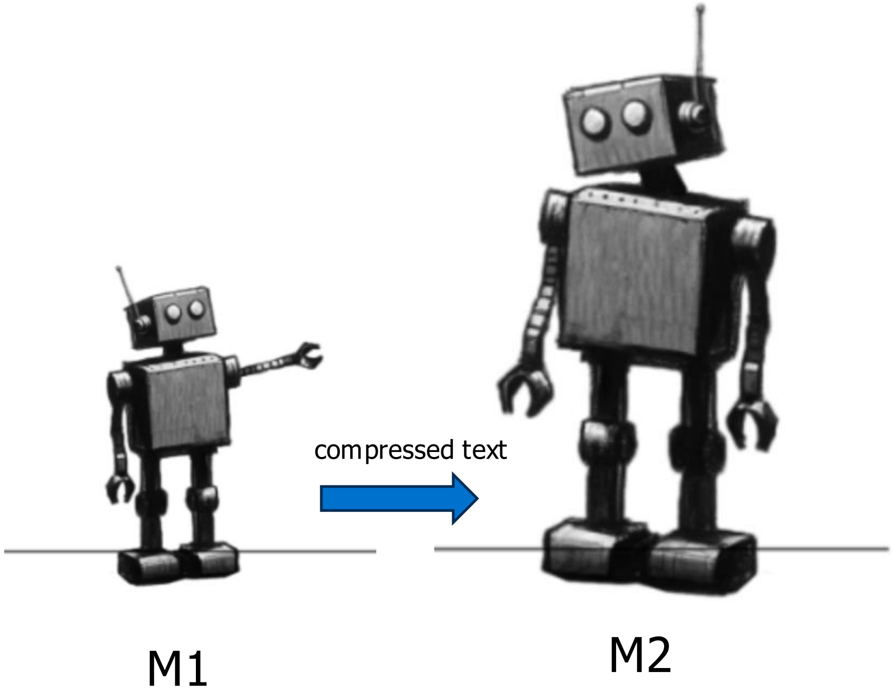
M1 Model



Neural compr
M1
LM over bytes
next byte probabilities
e o i a u
Neural compression can lead to better LLM tokenizers
Compression Algorithm
(e.g. Arithmetic Coding)
compressed text as bits
0100001100111010101111100
1000001001100001011111101
chunk into token IDs
compressed text as tokens
67 58 188 130 97 125
M2
LM over compressed text
next token probabilities
21 70 4 99 55

M1 Model

- **Purpose of M1**:
  - Predicts **probabilities** for each symbol
  - Simplifies **low-level patterns** for compression
- **Examples of Patterns**:
  - Spelling, grammar, and word frequency
- **Result**: Leaves **high-level structure** for M2 to learn.

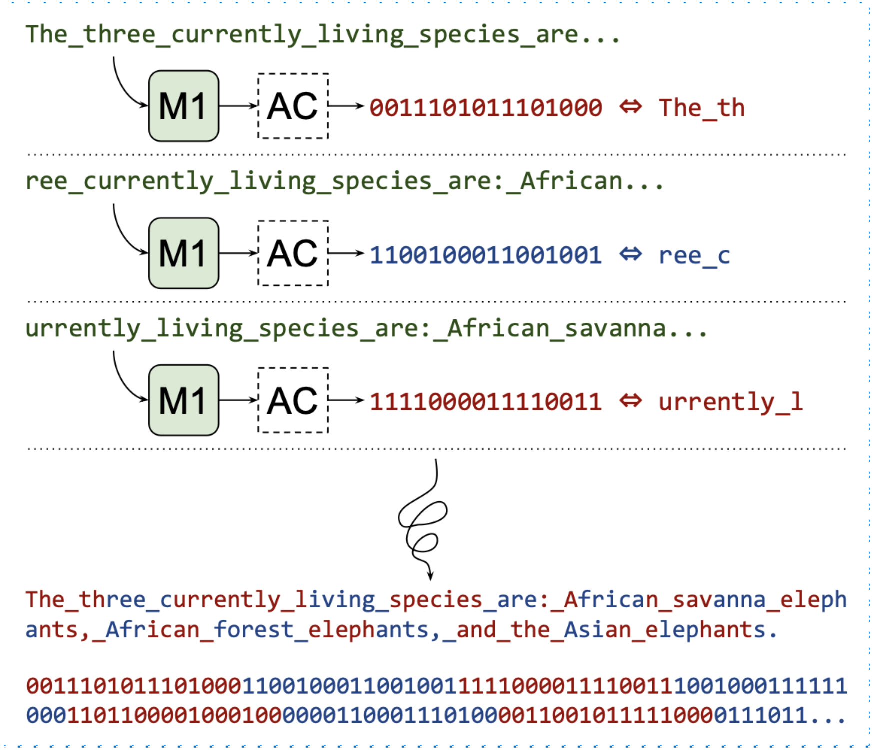


Problems with AC Compression

- **Challenges**:
  - **Random-looking output**: Hard for M2 to learn from.
  - **Dependence on M1's accuracy**: Imperfections leave learnable patterns.
  - **No stable mappings**: Context-dependent bit sequences.
  - **Long-range dependencies**: Expensive to process.
- **Impact**: AC output can be difficult for M2 to interpret effectively.

Equal Information Window AC

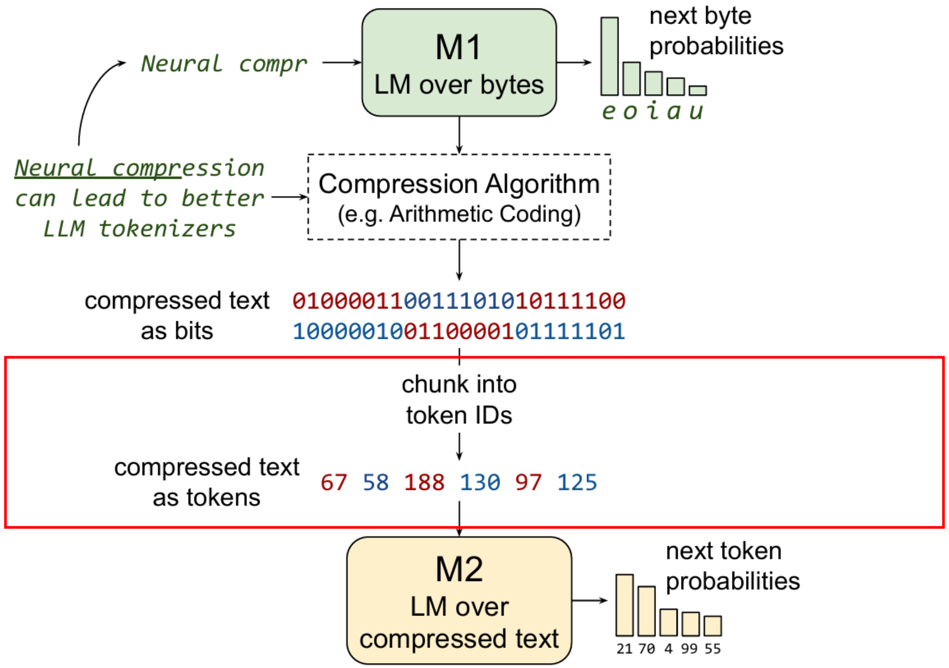
- **Equal-Info Windows**:
  - Divides text into **fixed-bit windows**.
  - Compresses each window **independently**.
  - Compression stops once a threshold (e.g. 16 bits) is reached per window
- **Reset Mechanism**:
  - Both AC algorithm and M1 model context are reset at each window
  - Ensures that each window can be independently decoded.



Benefits of Equal-Info Windows

- **Stable Mapping**:
  - Each window consistently maps a fixed number of bits to tokens
  - Reduces the context sensitivity of each token
- **Improved Learnability**:
  - M2 can learn patterns without needing to track AC state variables over long sequences
- **Efficiency Gains**:
  - Enables effective compression while maintaining learnability
  - Achieves ~**5.3x token-level compression**, > standard tokenizers

Tokenization of M2 Input

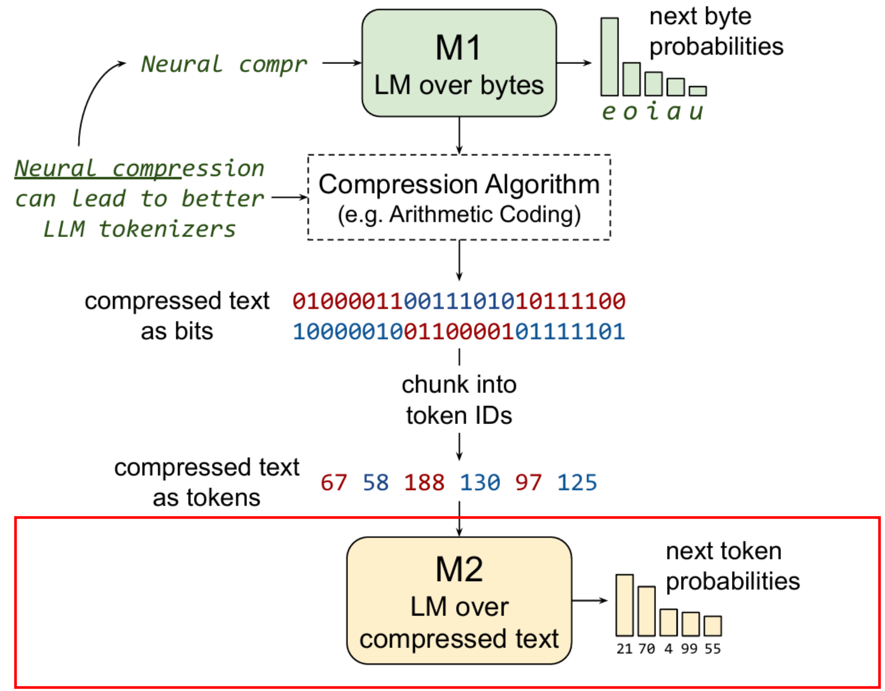


- Training over bitstream isn't ideal
  - Long sequence
  - Small vocabulary size (0 and 1)
- Group N bits into a token
  - $N \in \{8, 16\}$
  - $V \in \{256, 65536\}$
- Bigger N = higher compression ratio

Compression Ratio: Disambiguation

- Token compression ratio
  - L_{iT} / L_{oT}
  - Sequence length reduction
- Bit compression ratio
  - $L_{ib} / L_{ob} = \frac{\text{sequence length reduction}}{\text{bits / token}}$
  - E.g., SentencePiece
    - 4.28× length reduction
    - 15 bits / token (larger vocab)
    - 2.28× bit compression ratio
  - Tokenizing AC does not change bit compression ratio
- **Token compression ratio is the focus of this work**
  - Reducing number of tokens fed into M2 would reduce computational overhead

M2: Experimental Setup



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

M2: Experimental Setup

M2

Parameter Count
3m
25m
113m
403m
2b

M1

Method
Bytes
SentencePiece
AC[v=256]
StaticAC[v=256]
GZip[v=256]
EqualInfoAC[b=16, v=256]
EqualInfoAC[b=32, v=256]
EqualInfoAC[b=64, v=256]
EqualInfoAC[b=128, v=256]
AC[v=65k]
StaticAC[v=65k]
GZip[v=65k]
EqualInfoAC[b=16, v=65k]
EqualInfoAC[b=32, v=65k]
EqualInfoAC[b=64, v=65k]
EqualInfoAC[b=128, v=65k]

- 5 model sizes
  - Each on 16 different M1s

M2: Experimental Setup

Method	Compression Ratio	Tokens	Bytes
Bytes	1.0	26,188,185,600	26,188,185,600
SentencePiece	4.28	26,112,163,840	111,728,726,639
AC[v=256]	5.49	26,083,328,000	143,197,470,720
StaticAC[v=256]	1.73	26,175,078,400	45,282,885,632
GZip[v=256]	2.23	26,175,209,472	58,370,424,832
EqualInfoAC[b=16, v=256]	2.66	26,154,106,880	69,569,924,301
EqualInfoAC[b=32, v=256]	3.49	26,109,542,400	91,122,302,976
EqualInfoAC[b=64, v=256]	4.16	26,110,853,120	108,621,148,979
EqualInfoAC[b=128, v=256]	4.61	26,078,085,120	120,219,972,403
AC[v=65k]	10.98	25,952,256,000	284,955,770,880
StaticAC[v=65k]	3.46	26,133,135,360	90,420,648,346
GZip[v=65k]	4.47	26,122,649,600	116,768,243,712
EqualInfoAC[b=16, v=65k]	5.31	26,091,192,320	138,544,231,219
EqualInfoAC[b=32, v=65k]	6.97	26,049,249,280	181,563,267,482
EqualInfoAC[b=64, v=65k]	8.33	26,004,684,800	216,619,024,384
EqualInfoAC[b=128, v=65k]	9.22	25,936,527,360	239,134,782,259

M1

- 5 model sizes
  - Each on 16 different M1s
- Compute-controlled
  - Each model sees 26.2 billion tokens
  - Total amount of text differ because of different compression ratios

Johns Hopkins Whiting School of Engineering

M2: Experimental Setup

Method	Compression Ratio	Tokens	Bytes
Bytes	1.0	26,188,185,600	26,188,185,600
SentencePiece	4.28	26,112,163,840	111,728,726,639
AC[v=256]	5.49	26,083,328,000	143,197,470,720
StaticAC[v=256]	1.73	26,175,078,400	45,282,885,632
GZip[v=256]	2.23	26,175,209,472	58,370,424,832
EqualInfoAC[b=16, v=256]	2.66	26,154,106,880	69,569,924,301
EqualInfoAC[b=32, v=256]	3.49	26,109,542,400	91,122,302,976
EqualInfoAC[b=64, v=256]	4.16	26,110,853,120	108,621,148,979
EqualInfoAC[b=128, v=256]	4.61	26,078,085,120	120,219,972,403
AC[v=65k]	10.98	25,952,256,000	284,955,770,880
StaticAC[v=65k]	3.46	26,133,135,360	90,420,648,346
GZip[v=65k]	4.47	26,122,649,600	116,768,243,712
EqualInfoAC[b=16, v=65k]	5.31	26,091,192,320	138,544,231,219
EqualInfoAC[b=32, v=65k]	6.97	26,049,249,280	181,563,267,482
EqualInfoAC[b=64, v=65k]	8.33	26,004,684,800	216,619,024,384
EqualInfoAC[b=128, v=65k]	9.22	25,936,527,360	239,134,782,259

M1

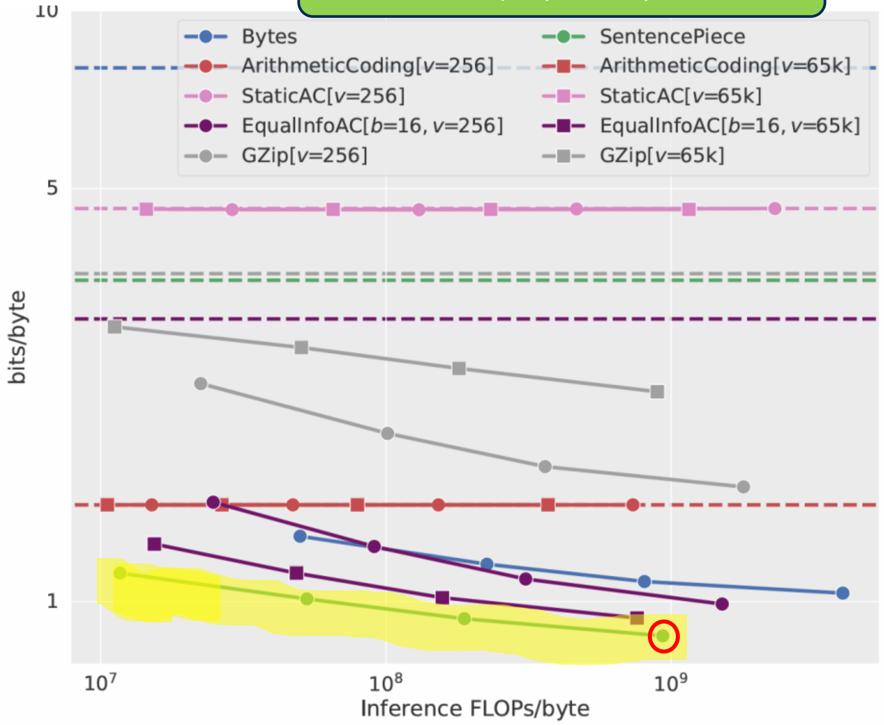
- 5 model sizes
  - Each on 16 different M1s
- Compute-controlled
  - Each model sees 26.2 billion tokens
  - Total amount of text differ because of different compression ratios
- Baselines: Byte and Sentencepiece
- Datasets: C4

Evaluation Metrics

- Models cannot be compared on per-token metrics
  - Shorter tokens will generally achieve higher likelihood and lower perplexity (because of smaller sample space)
- **[bits/byte] = $(L_{oT}/L_{iT}) * \ell / \ln(2)$**
  - ℓ: negative log likelihood (Cross Entropy)
  - L_{oT}/L_{iT}: Inverse of compression ratio
- Scaled cross-entropy
  - (Not perplexity)

Grounding baselines in familiar context

Solid Lines: trained M2 models
Dashed Lines: equal probability to all tokens



- SentencePiece 2b (best model):
  - bits/byte = 0.87
  - compression ratio = 4.28
  - Cross Entropy

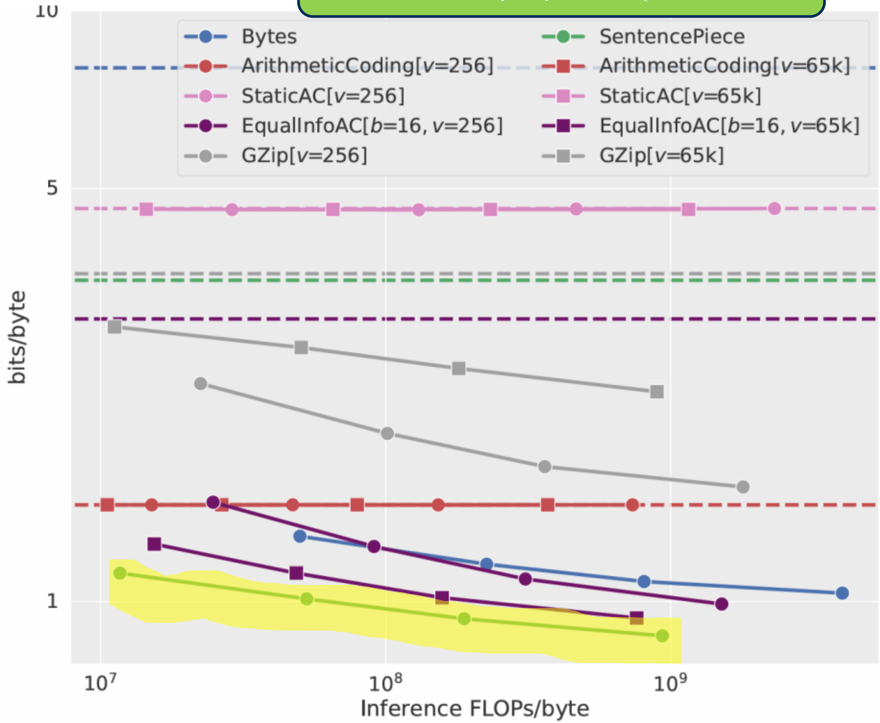
    = bits/byte * compression ratio * ln(2)

    = 0.87 * 4.28 * 0.6932

    = 2.5812
  - **Perplexity = 5.9835**
- Other LMs on C4 validation:
  - **Llama-2-7b-hf: ppl = 6.63**
  - **Mistral-7B: ppl = 6.94**

Results: Performance

Solid Lines: trained M2 models
Dashed Lines: equal probability to all tokens



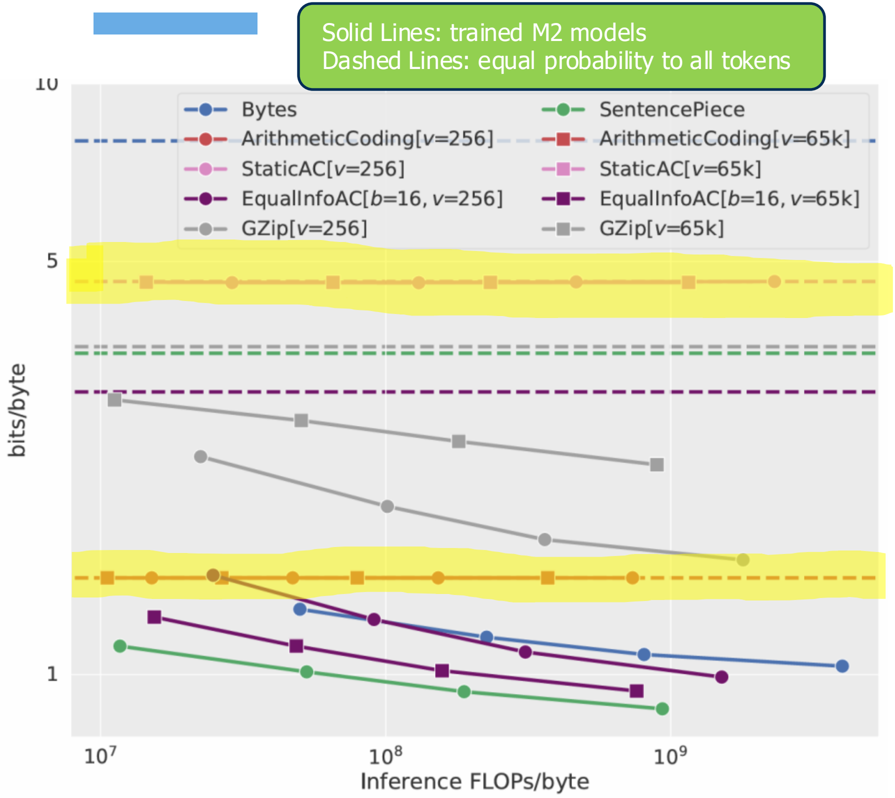
- SentencePiece is the overall strongest

Compressors are Language Models

Method	Uniform bits/byte	Unigram bits/byte	Δ
Bytes	8.000	4.602	3.398
SentencePiece	3.497	2.443	1.054
AC[v=256]	1.457	1.457	0.000
StaticAC[v=256]	4.624	4.624	0.000
EqualInfoAC[b=16, v=256]	3.008	2.976	0.032
EqualInfoAC[b=32, v=256]	2.292	2.285	0.007
EqualInfoAC[b=64, v=256]	1.923	1.921	0.002
EqualInfoAC[b=128, v=256]	1.735	1.735	0.000
GZip[v=256]	3.587	3.586	0.001

- Trivial M2s shows non-trivial performance on some M1s
- Almost 0 gain from unigram modeling compression output
  - Uniform distribution over tokenized vocabulary
  - M2 can only learn contextual information from these M1s

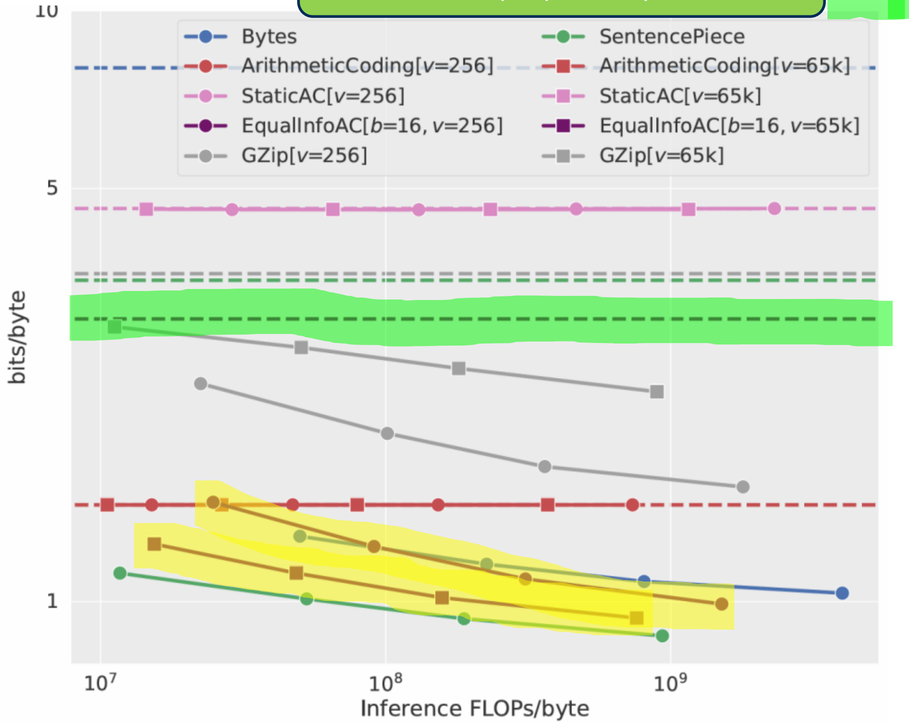
Results: Performance



- SentencePiece is the overall strongest
- M2 fails to learn on AC and StaticAC
  - M2 ends up with the same performance as M1

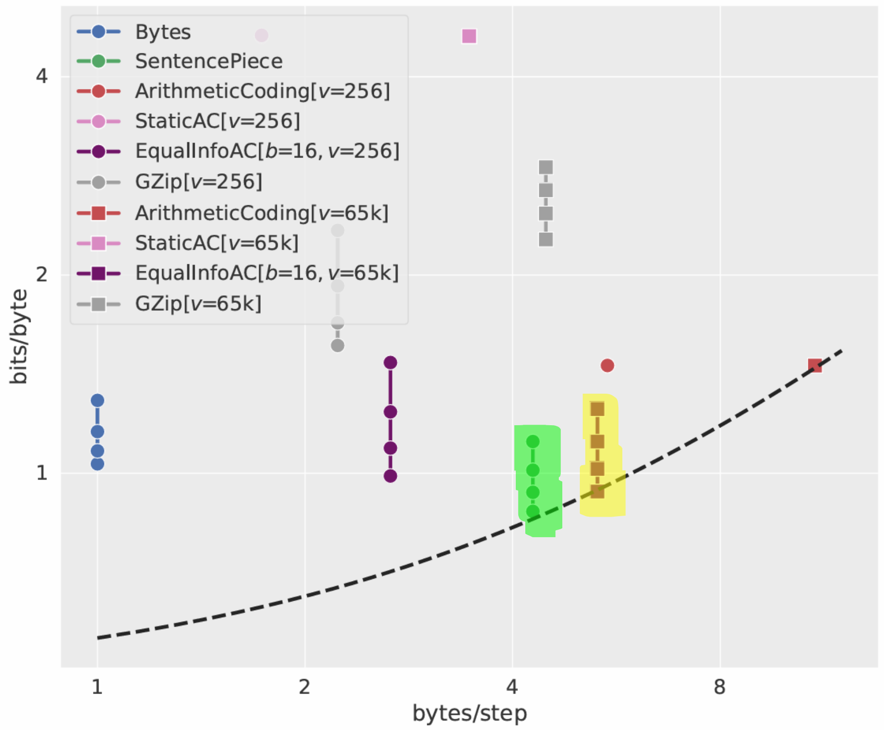
Results: Performance

Solid Lines: trained M2 models
Dashed Lines: equal probability to all tokens



- SentencePiece is the overall strongest
- M2 fails to learn on AC and StaticAC
  - M2 ends up with the same performance as M1
- Equal-Info Window AC are learnable
  - Bigger tokens improves performance and cost
  - Can learn from uniform-unigram vocabulary—shows capacity to model longer contexts

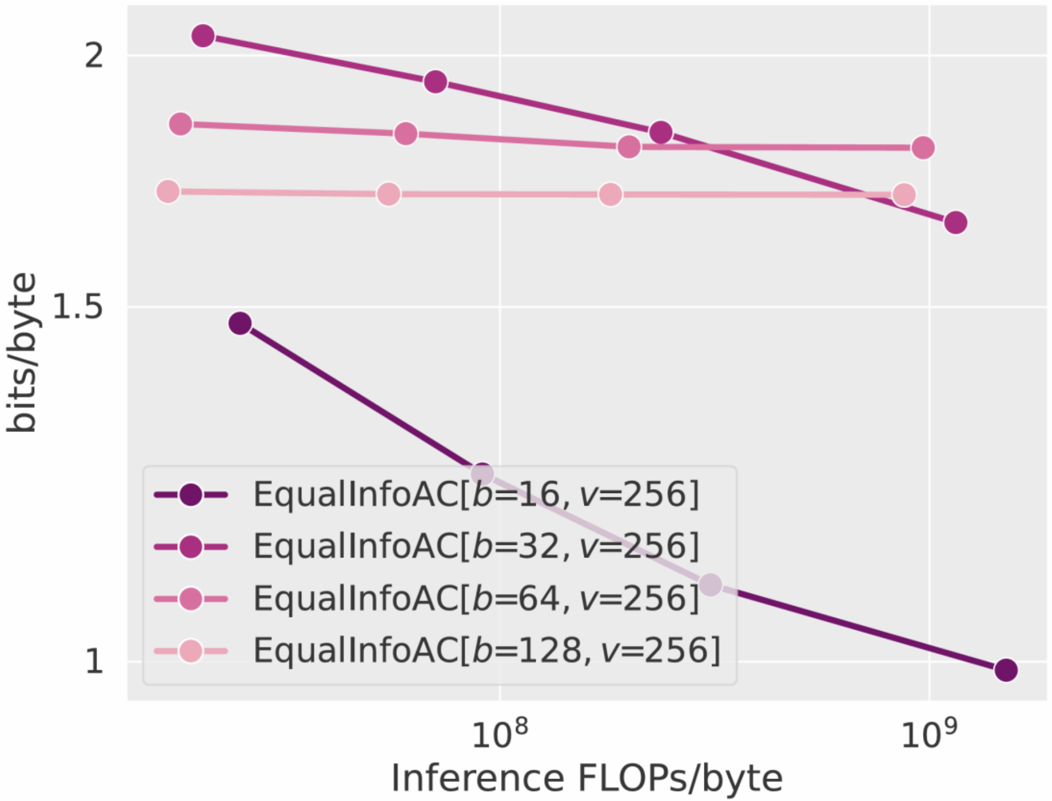
Advantage: Higher Bytes/Step



Dashed Line: Pareto frontier for tradeoff between bits/byte and bytes/step

- EqualInfoAC outperforms SentencePiece in terms of bytes/step
  - EqualInfoAC[b=16, v=65k]:
    - 5.31 bytes/step
  - SentencePiece:
    - 4.28 bytes/step
- Takes fewer steps to generate the same amount of text
- Potentially reduces generation latency
  - (Recall speculative decoding)

Shorter Windows are Better



Analysis: Qualitative Difference

Input Text	The three currently living species are: African savanna elephants, African forest elephants, and the Asian elephants.
SentencePiece Tokens	[The] [three] [currently] [living] [species] [are] [:] [African] [ ] [s] [a] [v] [anna] [elephant] [s] [,] [African] [forest] [elephant] [s] [,] [and] [the] [Asian] [elephant] [s] [.]
EqualInfoAC [b=16, v=65k] Tokens	[The th] [ree c] [urrently l] [iving] [species] [are] [: A] [frica] [n sav] [anna] [ele] [pha] [nts,] [Afr] [ican] [forest] [eleph] [ants,] [and the] [Asi] [an e] [lep] [hant] [s.]

EqualInfoAC is less stable

- Stability: same text→token mapping

Input Text	The three currently living species are: African savanna elephants, African forest elephants, and the Asian elephants.
SentencePiece Tokens	[The] [three] [currently] [living] [species] [are] [:] [African] [ ] [s] [a] [v] [anna] [elephant] [s] [,] [African] [forest] [elephant] [s] [,] [and] [the] [Asian] [elephant] [s] [.]
EqualInfoAC [b=16, v=65k] Tokens	[The th] [ree c] [urrently l] [iving] [species] [are] [: A] [frica] [n sav] [anna] [ele] [pha] [nts,] [Afr] [ican] [forest] [eleph] [ants,] [and the] [Asi] [an e] [lep] [hant] [s.]

EqualInfoAC is less semantic

- Semantic: tokens should align with meaningful linguistic units
  - SentencePiece tokens aligns with words morphemes

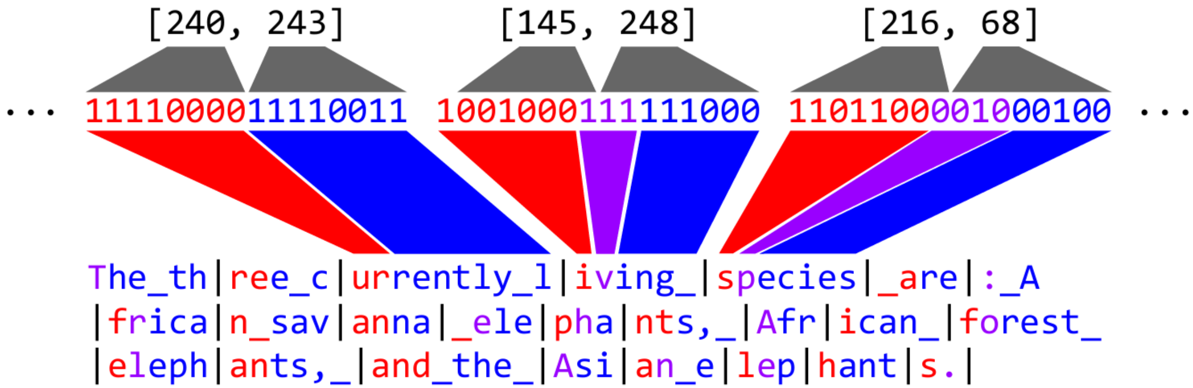
Input Text	The three currently living species are: African savanna elephants, African forest elephants, and the Asian elephants.
SentencePiece Tokens	[The] [three] [currently] [living] [species] [are] [:] [African] [ ] [s] [a] [v] [anna] [elephant] [s] [,] [African] [forest] [elephant] [s] [,] [and] [the] [Asian] [elephant] [s] [.]
EqualInfoAC [b=16, v=65k] Tokens	[The th] [ree c] [urrently l] [iving] [species] [are] [: A] [frica] [n sav] [anna] [ele] [pha] [nts,] [Afr] [ican] [forest] [eleph] [ants,] [and the] [Asi] [an e] [lep] [hant] [s.]

Johns Hopkins Whiting School of Engineering

Token to text stability

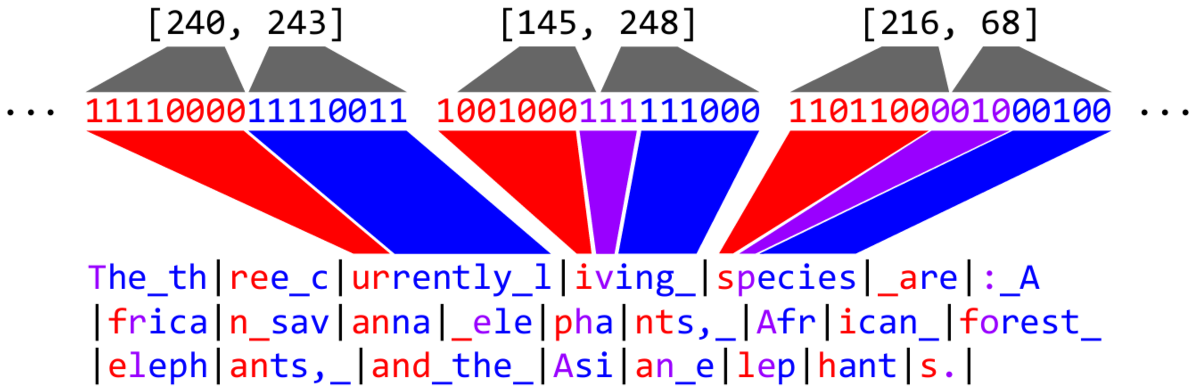
Token	Window Position	Window Text
151	1	[lew] / [lea] / [led] / [len] / [less] / [led] / [les] / [lew]
	2	[thoug] / [ust] / [this] / [etti] / [npo] / [thoug] / [un] / [imag]
185	1	[ord a] / [or k] / [ord] / [or f] / [or al] / [or a] / [ore i] / [ora]
	2	[ery] / [s may] / [cian] / [onte] / [h de] / [cri] / [opp] / [ides]

- token→text is stable when token size and window size match (b=16, v=65k)
  - Same token always map to the same output text



Token to text stability

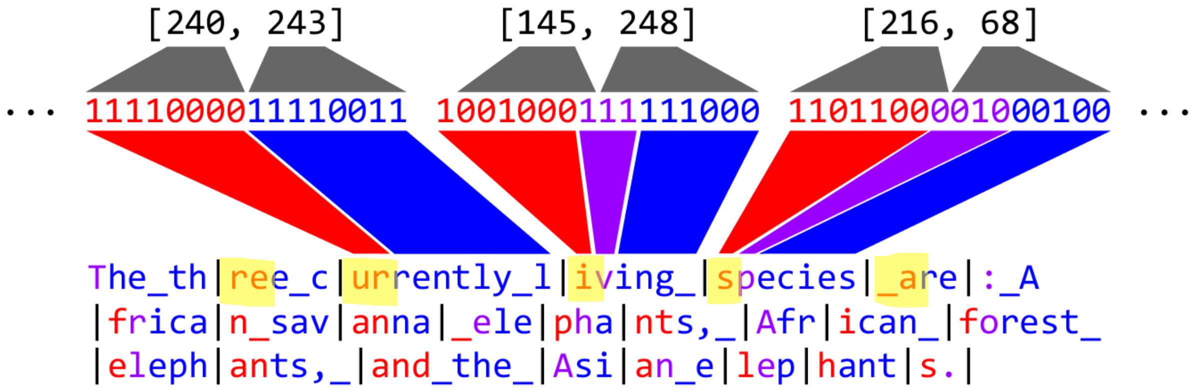
Token	Window Position	Window Text
151	1	[lew] / [lea] / [led] / [len] / [less] / [led] / [les] / [lew]
	2	[thoug] / [ust] / [this] / [etti] / [npo] / [thoug] / [un] / [imag]
185	1	[ord a] / [or k] / [ord] / [or f] / [or al] / [or a] / [ore i] / [ora]
	2	[ery] / [s may] / [cian] / [onte] / [h de] / [cri] / [opp] / [ides]



- token→text is stable when token size and window size match (b=16, v=65k)
  - Same token always map to the same output text
- When there are multiple tokens per window, the text from the first token is more consistent

Token to text stability

Token	Window Position	Window Text
151	1	[lew] / [lea] / [led] / [len] / [less] / [led] / [les] / [lew]
	2	[thoug] / [ust] / [this] / [etti] / [npo] / [thoug] / [un] / [imag]
185	1	[ord a] / [or k] / [ord] / [or f] / [or al] / [or a] / [ore i] / [ora]
	2	[ery] / [s may] / [cian] / [onte] / [h de] / [cri] / [opp] / [ides]



- token→text is stable when token size and window size match (b=16, v=65k)
  - Same token always map to the same output text
- When there are multiple tokens per window, the text from the first token is more consistent
- Window-initial characters are not well-compressed
  - Because M1 resets for every window

Takeaways

- Nothing better than subword tokenization (yet)
- Tokenizers with simple compression algorithms produces unlearnable sequences
- EqualInfoAC works (sort of)
  - Advantage: higher compression rate, may reduce inference latency
- Optimistic for future work
  - Even higher compression rates
  - Equal information per token may help with compute allocation
  - "Compression will give models a more direct view of the underlying raw text, thus helping with spelling and pronunciation tasks"
- Expect somewhat less stability in text↔token mapping

Questions?

- What is the best approach for designing a tokenizer?
  - Should we prioritize linguistic properties?
  - Is a purely mathematical approach enough?
- Are there any other advantage of tokenization through data compression algorithms?
- Do you think we need better tokenizers?
  - What are the problems of subword tokenizers?