# Compression Beyond iid data

Kedar Tatwawadi, EE274

# Arithmetic/rANS recap

1. Given *any* distribution $P$, achieves *optimal* compression. Thus, Arithmetic/rANS coding allows for `model` and `entropy coding` separation.

$$H(X) \leq \frac{\mathbb{E}[l(X_1^n)]}{B} \leq H(X) + \frac{\mathcal{O}(1)}{n}$$

2. Encoding, decoding is linear time and quite efficient! As we are not saving a large codebook, memory requirements are not very high

3. Can work very well with changing distribution $P$.
   i.e. Adaptive algorithms work well with Arithmetic/rANS coding

# Arithmetic/rANS recap

Given *any* distribution $P$ and a sequence $x^n$, Arithmetic has code-length.

$$L(x^n) \approx \sum_{i=1}^{n} \log_2 \frac{1}{P(x_i)}$$

# Experiment

```
$ cat sherlock.txt
    ...
      In mere size and strength it was a terrible creature which was
      lying stretched before us. It was not a pure bloodhound and it
      was not a pure mastiff; but it appeared to be a combination of
      the two—gaunt, savage, and as large as a small lioness. Even now
      in the stillness of death, the huge jaws seemed to be dripping
      with a bluish flame and the small, deep-set, cruel eyes were
      ringed with fire. I placed my hand upon the glowing muzzle, and
      as I held them up my own fingers smouldered and gleamed in the
      darkness.

      “Phosphorus,” I said.

      “A cunning preparation of it,” said Holmes, sniffing at the dead
    ...
```

Let's try and compress this 387 KB book.

# Experiment

```
>>> from core.data_block import DataBlock
>>>
>>> with open("sherlock.txt") as f:
>>>     data = f.read()
>>>
>>> print(DataBlock(data).get_entropy()*len(data)/8, "bytes")

199833 bytes
```

```
$ gzip < sherlock.txt | wc -c
134718

$ bzip2 < sherlock.txt | wc -c
99679
```

*What are we missing here?*

# What are we missing?

1. Data in real life is NOT i.i.d (independent, identially distributed)
2. Maybe the entire file doesn't have the same distribution (think about a file with text, code interspersed).

# What are we missing?

Example:

```
Th_ go__ of this cour__ is to provi__ an understa____ of how data
compression enables representing all of this information in a succinct
manner. Both theore____ and prac___al aspects of compression will be
covered. A major component of the course is learning through doing - the
students will work on a peda__ical data compression library and imple___t
specific compression tec___ques.
```

**Data is not iid**: Previous symbols give lots of information on what the next symbol can be.

# Data is not i.i.d IRL

```
In mere size and strength it was a terrible creature which was
lying stretched before us. It was not a pure bloodhound and it
was not a pure mastiff; but it appeared to be a combination of
the two—gaunt, savage, and as large as a small lioness. Even now
in the stillness of death, the huge jaws seemed to be dripping
with a bluish flame and the small, deep-set, cruel eyes were
ringed with fire. I placed my hand upon the glowing muzzle, and
as I held them up my own fingers smouldered and gleamed in the
darkness.
```

```python
from scl.utils.test_utils import (
    create_random_text_file,
    get_random_data_block,
    try_file_lossless_compression,
    try_lossless_compression,
)
import numpy as np


def get_alphabet_fixed_bitwidth(alphabet_size):
    return 1 if (alphabet_size == 1) else int(np.ceil(np.log2(alphabet_size)))


class FixedBitwidthEncoder(DataEncoder):
    """
    - Encode each symbol using a fixed number of bits
    - Encode the alphabet using a generic pickle based encoder
    """

    def __init__(self):
        self.alphabet_encoder = PickleEncoder()
        self.DATA_SIZE_BITS = 32

    def encode_block(self, data_block: DataBlock):
        """first encode the alphabet and then each data symbol using fixed number of bits"""
        # get bit width
```

# Probability recap

Recall for $U^n = (U_1, \ldots, U_n)$:

for iid

$P(U^n) = \Pi_{i=1}^n P(U_i)$

in general

$P(U^n) = \Pi_{i=1}^n P(U_i|U^{i-1}) = \Pi_{i=1}^n P(U_i|U_1, \ldots, U_{i-1})$

# Probability recap

Recall for $U^n = (U_1, \ldots, U_n)$:

for iid

$P(U^n) = \Pi_{i=1}^n P(U_i)$

in general

$P(U^n) = \Pi_{i=1}^n P(U_i|U^{i-1}) = \Pi_{i=1}^n P(U_i|U_1, \ldots, U_{i-1})$

# Probability recap

> **Definition: Stationary Process**
>
> A stationary process is a stochastic process that is time-invariant, i.e., the probability distribution doesn't change with time (here time refers to the index in the sequence). More precisely, we have
>
> $$P(U_1 = u_1, U_2 = u_2, \ldots, U_n = u_n) = P(U_{l+1} = u_1, U_{l+2} = u_2, \ldots, U_{l+n} = u_n)$$
>
> for every $n$, every shift $l$ and all $(u_1, u_2, \ldots, u_n) \in \mathcal{U}^n$.

- Mean, variance etc. do not change with $n$.
- Can still have arbitrary time dependence.

# Examples

- Fair coin toss - i.i.d (so stationary)
- Markov processes

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

```
Transition matrix
   U_{i+1} 0   1   2
U_i
0         0.5 0.5 0.0
1         0.0 0.5 0.5
2         0.5 0.0 0.5
```

# Examples

## $k$th order Markov source

> **Definition: $k$th order Markov source**
> A $k$th order Markov source is defined by the condition
>
> $$P(U_n|U_{n-1}U_{n-2}\dots) = P(U_n|U_{n-1}U_{n-2}\dots U_{n-k})$$
>
> for every $n$. In words, the conditional probability of $U_n$ given the entire past depends only on the past $k$ symbols.

Most practical stationary sources can be approximated well with a finite memory $k$th order Markov source with higher values of $k$ typically providing a better approximation (with diminishing returns).

# Examples

## $k$th order Markov source

> **Definition: $k$th order Markov source**
> A $k$th order Markov source is defined by the condition
>
> $$P(U_n|U_{n-1}U_{n-2}\dots) = P(U_n|U_{n-1}U_{n-2}\dots U_{n-k})$$
>
> for every $n$. In words, the conditional probability of $U_n$ given the entire past depends only on the past $k$ symbols.

Most practical stationary sources can be approximated well with a finite memory $k$th order Markov source with higher values of $k$ typically providing a better approximation (with diminishing returns).

# Conditional Entropy

The conditional entropy of $U$ given $V$ is defined as

$$H(U|V) \triangleq E\left[\log \frac{1}{P(U|V)}\right]$$

# Conditional Entropy

The conditional entropy of $U$ given $V$ is defined as

$$H(U|V) \triangleq E\left[\log \frac{1}{P(U|V)}\right]$$

Can also write this as

$$\begin{aligned} H(U|V) &= \sum_{u \in \mathcal{U}, v \in \mathcal{V}} P(u,v) \log \frac{1}{P(u|v)} \\ &= \sum_{v \in \mathcal{V}} P(v) \sum_{u \in \mathcal{U}} P(u|v) \log \frac{1}{P(u|v)} \\ &= \sum_{v \in \mathcal{V}} P(v) H(U|V=v) \end{aligned}$$

# Conditional Entropy

1. Conditioning reduces entropy: $H(U|V) \leq H(U)$ with equality iff $U$ and $V$ are independent.

# Conditional Entropy

1. Conditioning reduces entropy: $H(U|V) \leq H(U)$ with equality iff $U$ and $V$ are independent.
2. Chain rule of entropy:

$$H(U,V) = H(U) + H(V|U) = H(V) + H(U|V)$$

# Conditional Entropy

1. Conditioning reduces entropy: $H(U|V) \leq H(U)$ with equality iff $U$ and $V$ are independent.
2. Chain rule of entropy:

$$H(U,V) = H(U) + H(V|U) = H(V) + H(U|V)$$

3. Joint entropy vs. sum of entropies:

$$H(U,V) \leq H(U) + H(V)$$

with equality holding iff $U$ and $V$ are independent.

# Example-1

- Fair coin toss i.i.d: `P = {H: 0.5, T: 0.5}`.
- $H(U_n|U_{n-1}, U_{n-2}, \ldots, U_1) = H(U_n)$

# Example-2 Markov source

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

- $H(U_1) = \log_2(3)$
- $H(U_2|U_1) = ??$

# Example-2 Markov source

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

- $H(U_1) = \log_2(3)$
- $H(U_2|U_1) = ??$

# Example-2 Markov source

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

- $H(U_2|U_1) = ??$

$$\begin{aligned} H(U_2|U_1) &= \\ &= H(U_2 - U_1|U_1) \\ &= H(Z_1|U_1) \\ &= H(Z_1) = 1 \end{aligned}$$

# Example-2 Markov source

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

- $H(U_3|U_2, U_1) = ?$

# Example-2 Markov source

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

- $H(U_3|U_2, U_1) = H(U_3|U_2) = H(U_2|U_1)$
- $H(U_n|U_{n-1}, U_{n-2}, \ldots, U_1) = H(U_n|U_{n-1})$

# Example-2 Markov source

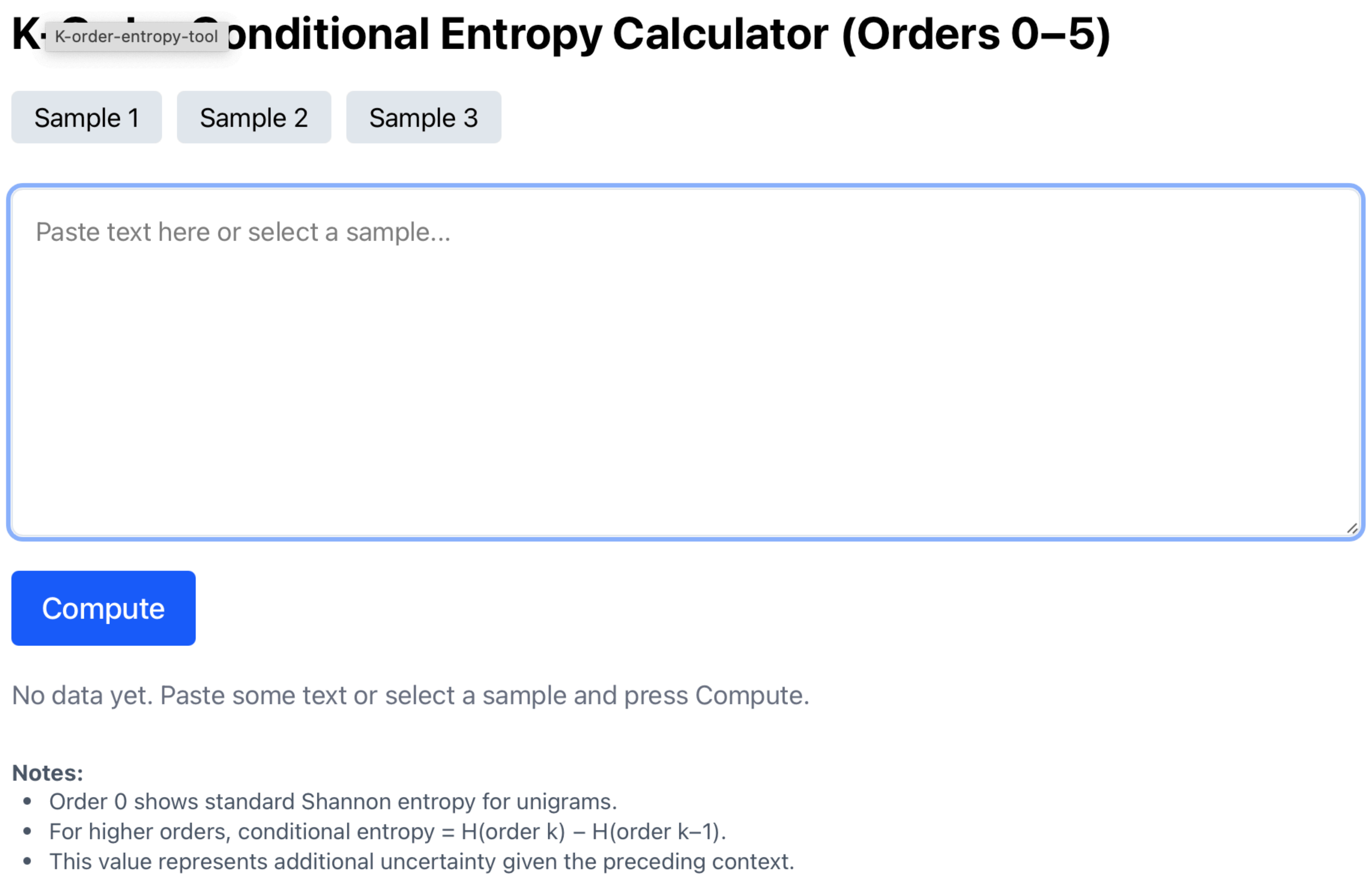
$$U_1 \sim Unif(\{0,1,2\})$$

$$U_{i+1} = (U_i + Z_i) \bmod 3$$

$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

- $H(U_3|U_2, U_1) = H(U_3|U_2) = H(U_2|U_1)$
- $H(U_n|U_{n-1}, U_{n-2}, \ldots, U_1) = H(U_n|U_{n-1})$

# chatGPT canvas tool!




https://chatgpt.com/canvas/shared/68f7c30a956481918b2d563d8cfb0587

# Entropy rate

$$H_1(\mathbf{U}) = \lim_{n\to\infty} H(U_{n+1}|U_1, U_2, \ldots, U_n)$$

$$H_2(\mathbf{U}) = \lim_{n\to\infty} \frac{H(U_1, U_2, \ldots, U_n)}{n}$$

> **C&T Thm 4.2.1**
>
> For a stationary stochastic process, the two limits above are equal. We represent the limit as $H(\mathbf{U})$ (entropy rate of the process, also denoted as $H(\mathcal{U})$).

# Entropy rate

$$H_1(\mathbf{U}) = \lim_{n\to\infty} H(U_{n+1}|U_1, U_2, \ldots, U_n)$$

*Can we guess the entropy rate?*

# Lossless Text Compression

## Prediction and Entropy of Printed English

By C. E. SHANNON

(*Manuscript Received Sept. 15, 1950*)

A new method of estimating the entropy and redundancy of a language is described. This method exploits the knowledge of the language statistics possessed by those who speak the language, and depends on experimental results in prediction of the next letter when the preceding text is known. Results of experiments in prediction are given, and some properties of an ideal predictor are developed.

# Lossless Text Compression

- Models (estimate probabilities from text):

  (a) 0th-order Markov chain (iid):

$$H(\mathcal{X}) \approx 4.76 \qquad \text{bits per letter}$$

  (b) 1st order Markov chain:

$$H(\mathcal{X}) \approx 4.03 \qquad \text{bits per letter}$$

  (c) 4th order Markov chain:

$$H(\mathcal{X}) \approx 2.8 \qquad \text{bits per letter}$$

- Estimate by asking people to guess the next letter until they get it correct. The *order* of their guesses reflects their estimate of the *order* of their conditional probabilities for the next letter. (Shannon 1952).

$$H(\mathcal{X}) \approx 1.3 \qquad \text{bits per letter}$$

http://reeves.ee.duke.edu/information_theory/lecture4-Entropy_Rates.pdf

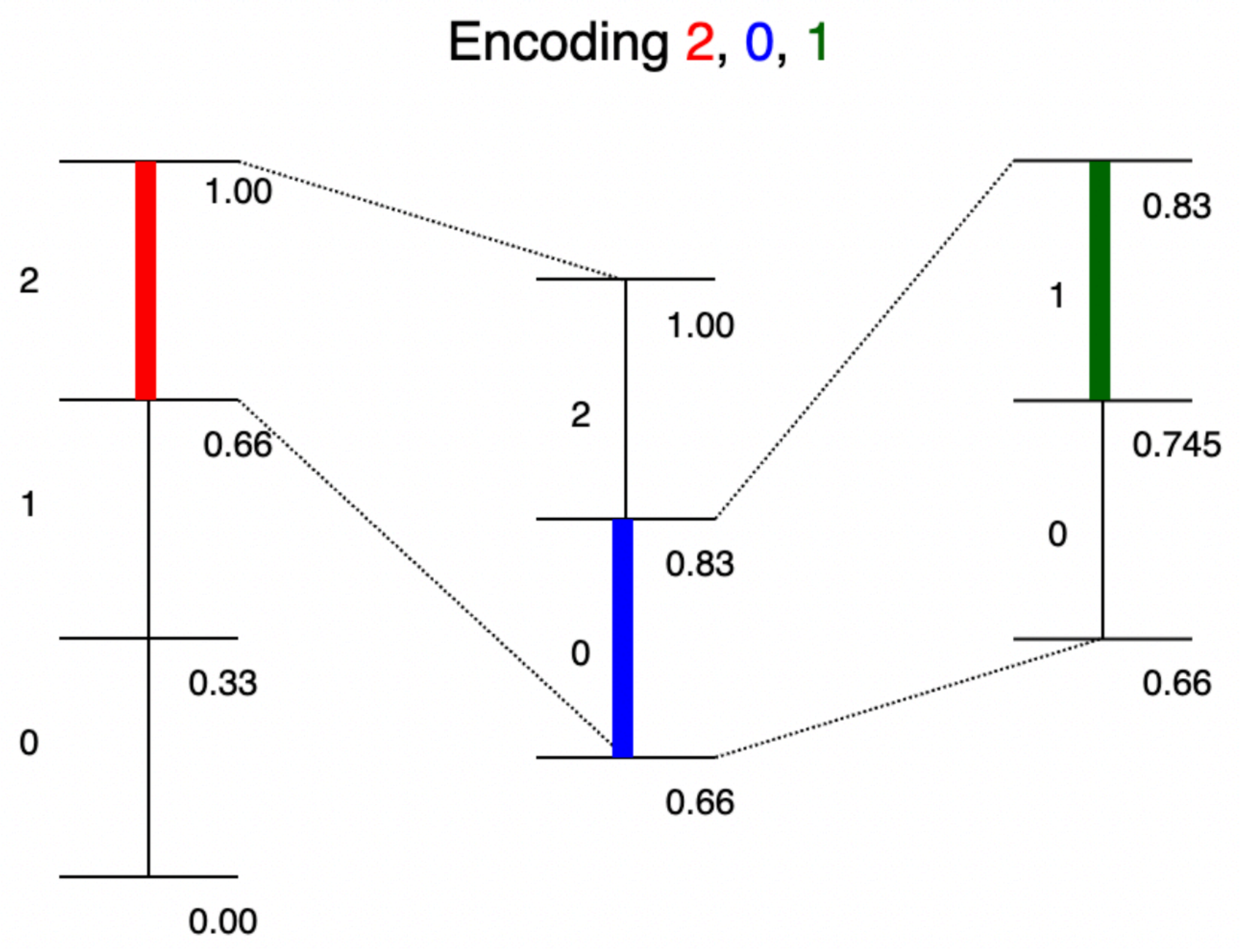
# How can we compress to reach entropy rate?

# Example-1 Markov source

$$U_1 \sim Unif(\{0,1,2\})$$
$$U_{i+1} = (U_i + Z_i) \bmod 3$$
$$Z_i \sim Ber\left(\frac{1}{2}\right)$$

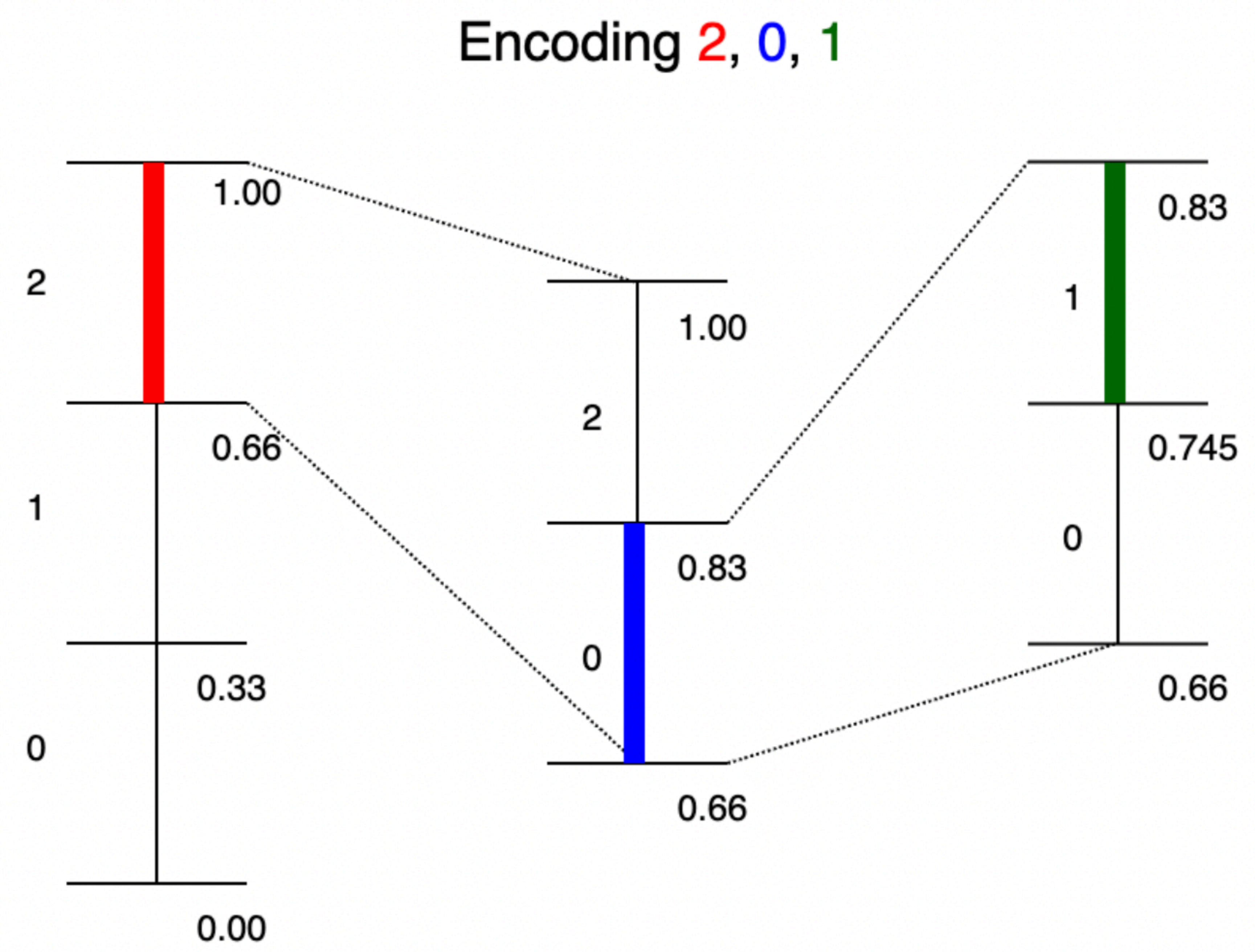
```
Transition matrix
   U_{i+1} 0   1   2
U_i
0         0.5 0.5 0.0
1         0.0 0.5 0.5
2         0.5 0.0 0.5
```

# Example-1 Markov source




**Question:** Can you explain the general idea?

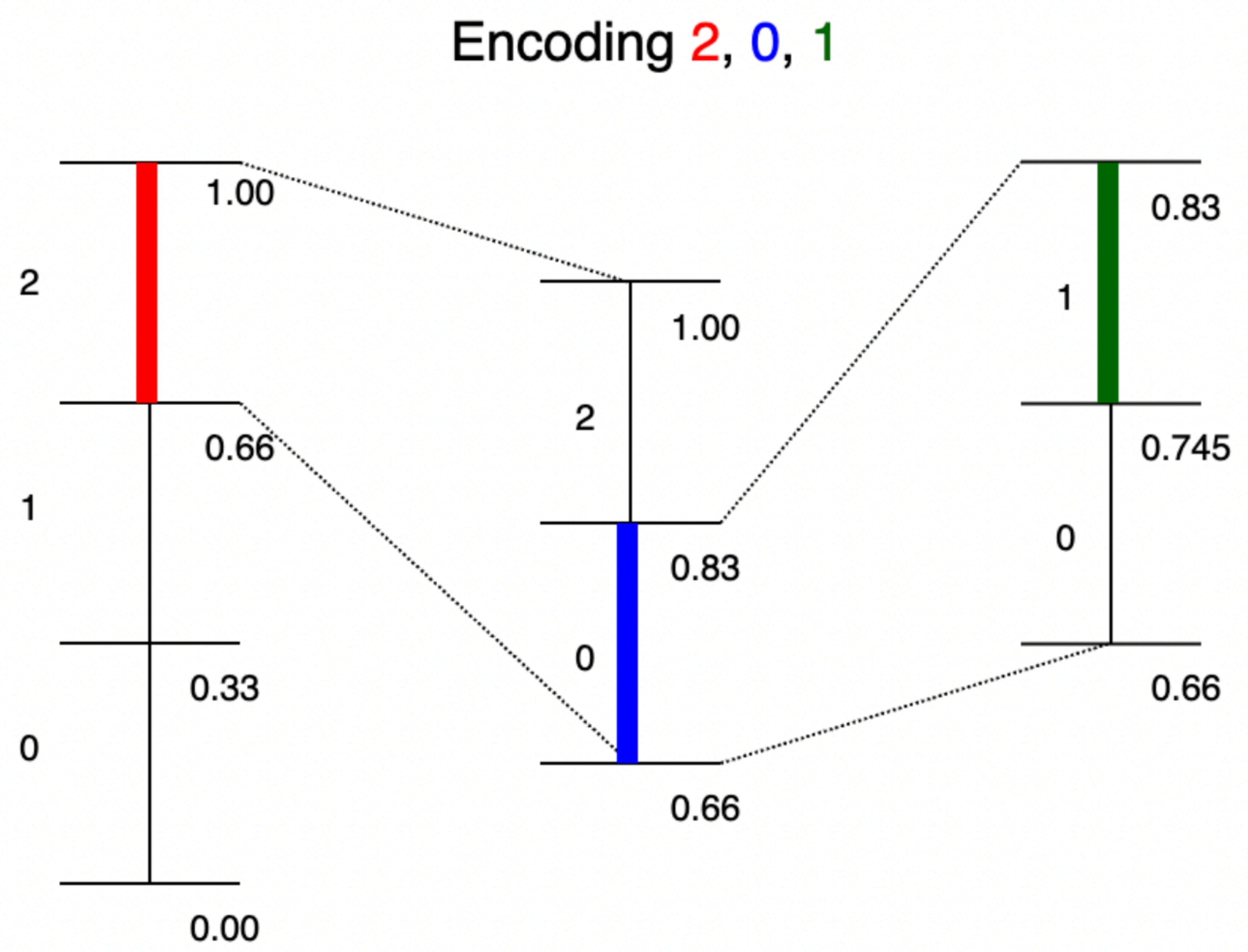
# Example-1 Markov source




**Question:** Can you explain the general idea?

**Answer:** At every step, split interval by $P(-|u_{i-1})$ [more generally by $P(-|\text{entire past})$].

# Example-1 Markov source




*What is the avg code length?*

**Question:** Can you explain the general idea?

**Answer:** At every step, split interval by $P(-|u_{i-1})$ [more generally by $P(-|\text{entire past})$].

# Example-1 Markov source | Recap

Given *any* distribution $P$ and a sequence $x^n$, Arithmetic has code-length.

$$L(x^n) \approx \sum_{i=1}^{n} \log_2 \frac{1}{P(x_i)}$$

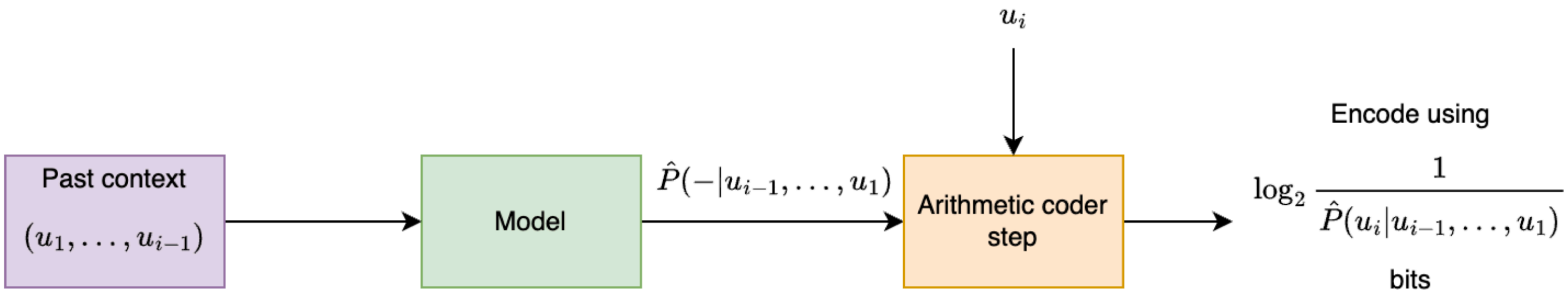
# Example-1 Markov source

Length of interval after encoding $u_1, u_2, u_3, \ldots, u_n = P(u_1)P(u_2|u_1)\ldots P(u_n|u_{n-1})$

Expected bits per symbol

$$
\begin{aligned}
&= \frac{1}{n} E\left[\log_2 \frac{1}{P(U_1)}\right] + \frac{1}{n}\sum_{i=2}^{n} E\left[\log_2 \frac{1}{P(U_i|U_{i-1})}\right] \\
&= \frac{1}{n} H(U_1) + \frac{n-1}{n} H(U_2|U_1) \\
&\sim H(U_2|U_1)
\end{aligned}
$$

# Context-based Arithmetic coding

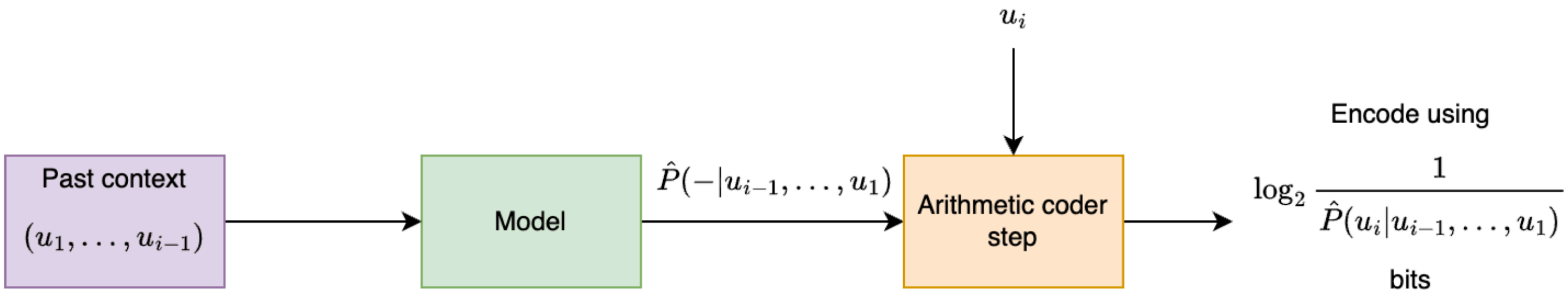



Total bits for encoding:

$$\sum_{i=1}^{n} \log_2 \frac{1}{\hat{P}(u_i|u_1, \ldots, u_{i-1})}$$

**Question:** How would the decoding work?

# Context-based Arithmetic coding



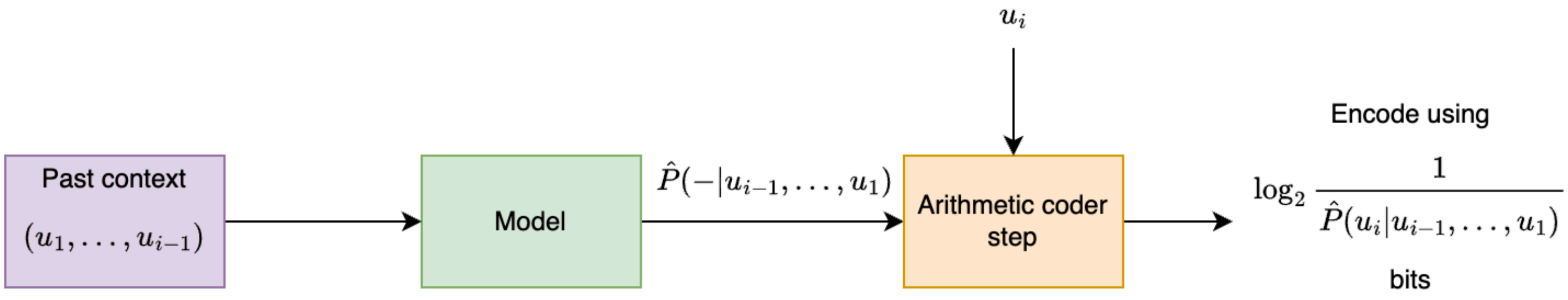

Total bits for encoding:

$$\sum_{i=1}^{n} \log_2 \frac{1}{\hat{P}(u_i|u_1, \ldots, u_{i-1})}$$

**Question:** How would the decoding work?

**Answer:** Decoder uses same model, at step $i$ it has access to $u_1, \ldots, u_{i-1}$ already decoded and so can generate the $\hat{P}$ for the arithmetic coding step!

# Context-based Arithmetic coding

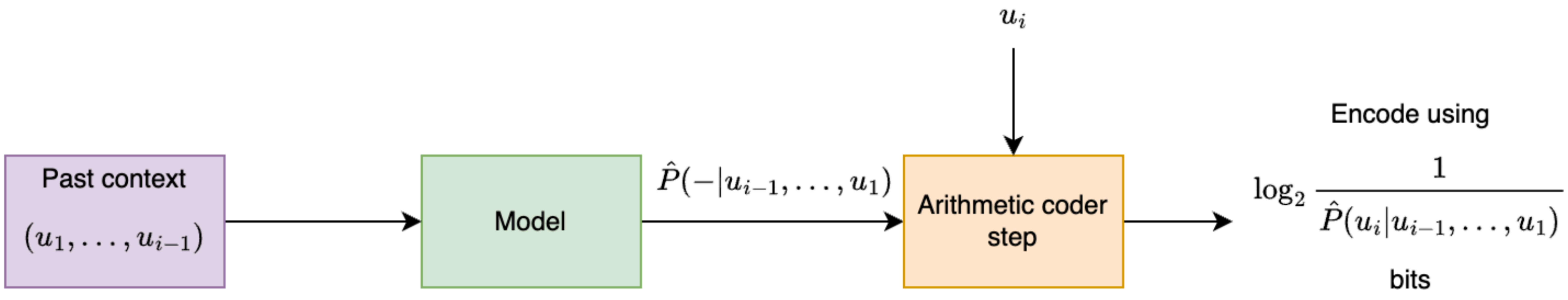



Total bits for encoding:

$$\sum_{i=1}^{n} \log_2 \frac{1}{\hat{P}(u_i|u_1, \ldots, u_{i-1})}$$

***Key-idea-1**: Better prediction model -> implies better compression!*

# Context-based Arithmetic coding




Total bits for encoding:

$$\sum_{i=1}^{n} \log_2 \frac{1}{\hat{P}(u_i|u_1, \ldots, u_{i-1})}$$

***Key-idea***: *Better prediction model -> implies better compression!*

# Context-based Arithmetic coding

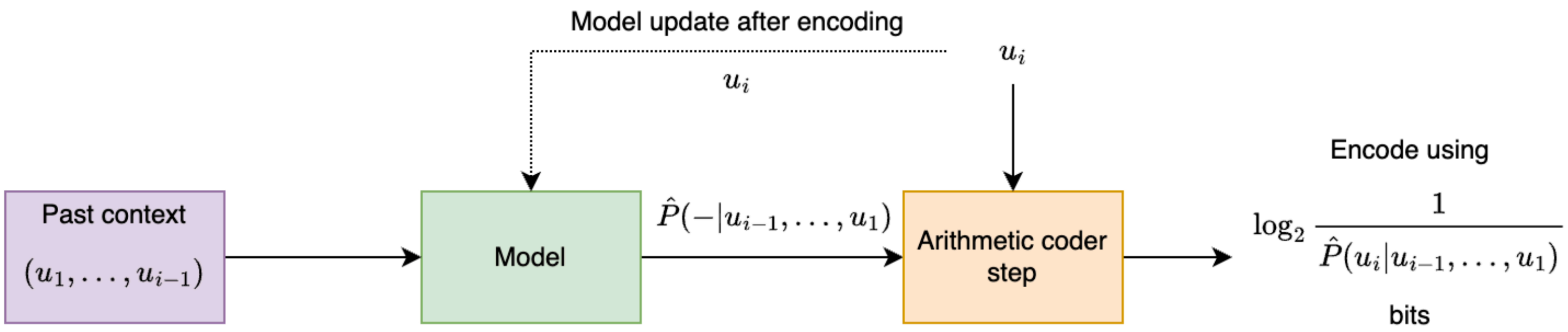
## Two-pass approach

✅ learn model from entire data, leading to potentially better compression
✅ more suited for parallelization
❌ need to store model in compressed file
❌ need two passes over data, not suitable for streaming
❌ might not work well with changing statistics

## Adaptive approach

✅ no need to store the model
✅ suitable for streaming
❌ adaptively learning model leads to inefficiency for initial samples
✅ works pretty well in practice!

# Adaptive Arithmetic coding

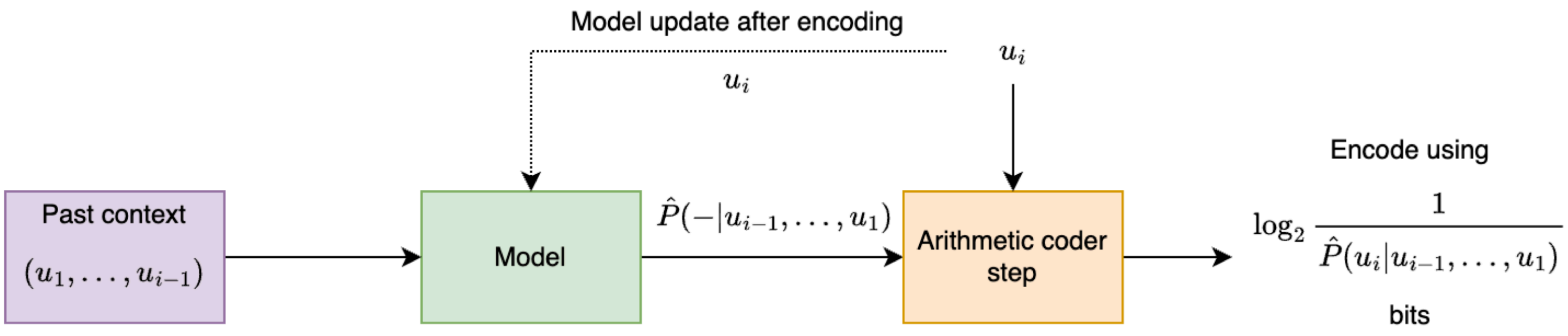



⚠️ Important for encoder and decoder to share exactly the same model state at every step (including at initialization).

⚠️ Don't go about updating model with $u_i$ before you perform the encoding for $u_i$.

⚠️ Try not to provide $0$ probability to any symbol.

# Adaptive Arithmetic coding




⚠️ Important for encoder and decoder to share exactly the same model state at every step (including at initialization).

⚠️ Don't go about updating model with $u_i$ before you perform the encoding for $u_i$.

⚠️ Try not to provide $0$ probability to any symbol.

# Adaptive Arithmetic coding

## $k$th order adaptive arithmetic coding

- Start with a frequency of 1 for each symbol in the $(k+1)$th order alphabet (to avoid zero probabilities)
- As you see symbols, update the frequency counts
- At each step you have a probability distribution over the alphabet induced by the counts

# Example: 1st order adaptive arithmetic coding

Data: 101011

Initial frequencies/counts:

$$c(0,0) = 1$$
$$c(0,1) = 1$$
$$c(1,0) = 1$$
$$c(1,1) = 1$$

Assume past is padded with 0s

# Example: 1st order adaptive arithmetic coding

Data: **1**01011

Current symbol: 1

Previous symbol: 0 (padding)

Predicted probability: $P(1|0) = \frac{c(0,1)}{c(0,0)+c(0,1)} = \frac{1}{2}$

Counts:

$c(0,0) = 1$
$c(0,1) = 1 \to 2$
$c(1,0) = 1$
$c(1,1) = 1$

## Example: 1st order adaptive arithmetic coding

Data: 1**0**1011

Current symbol: 0

Previous symbol: 1

Predicted probability: $P(0|1) = \frac{c(1,0)}{c(1,0)+c(1,1)} = \frac{1}{2}$

Counts:

$c(0,0) = 1$

$c(0,1) = 2$

$c(1,0) = 1 \to 2$

$c(1,1) = 1$

## Observations

- Over time we learn the empirical distribution of the data
- Initially start off with uniform distribution - can change prior to enforce some prior knowledge [both encoder and decoder need to know!]
- You can do this for $k = 0$ (iid data with unknown distribution)!

# $k$th order adaptive arithmetic coding (AAC)

```python
def freqs_current(self):
    """Calculate the current freqs. We use the past k symbols to pick out
    the corresponding frequencies for the (k+1)th.
    """
        freqs_given_context = np.ravel(self.freqs_kplus1_tuple[tuple(self.past_k)])
```

```python
def update_model(self, s):
    """function to update the probability model. This basically involves update the count
    for the most recently seen (k+1) tuple.

    Args:
        s (Symbol): the next symbol
    """
    # updates the model based on the new symbol
    # index self.freqs_kplus1_tuple using (past_k, s) [need to map s to index]
    self.freqs_kplus1_tuple[(*self.past_k, s)] += 1

    self.past_k = self.past_k[1:] + [s]
```

# $k$th order adaptive arithmetic coding (AAC)

On `sherlock.txt`:

```
>>> with open("sherlock.txt") as f:
>>>     data = f.read()
>>>
>>> data_block = DataBlock(data)
>>> alphabet = list(data_block.get_alphabet())
>>> aec_params = AECParams()
>>> encoder = ArithmeticEncoder(aec_params, AdaptiveOrderKFreqModel(alphabet, k, aec_params.MAX_ALLOWED_TOTAL_FREQ))
>>> encoded_bitarray = encoder.encode_block(data_block)
>>> print(len(encoded_bitarray)//8) # convert to bytes
```

# $k$th order adaptive arithmetic coding

| Compressor | compressed bits/byte |
|---|---|
| $0$th order | 4.26 |
| $1$st order | 3.34 |
| $2$nd order | **2.87** |
| $3$rd order | 3.10 |
| gzip | 2.78 |
| bzip2 | **2.05** |

# $k$th order adaptive arithmetic coding (AAC)

## Limitations

- slow, memory complexity grows exponentially in $k$
- counts become very sparse for large $k$, leading to worse performance
- unable to exploit similarities in prediction for *similar* contexts

Some of these can be overcome with smarter modeling as discussed later.

**Note:** Despite their performance limitations, context based models are still employed as the entropy coding stage after suitably preprocessing the data (LZ, BWT, etc.).

# Context based arithmetic coding in practice

Entropy coding

HEVC uses a single entropy-coding engine, which is based on Context Adaptive Binary Arithmetic Coding (CABAC) [CABAC], whereas H.264 uses two distinct entropy coding engines. CABAC in HEVC shares many similarities with CABAC of H.264, but contains several improvements. Those include improvements in coding efficiency and lowered implementation complexity, especially for parallel architectures.

Entropy coding

Similar to HEVC, VVC uses a single entropy-coding engine, which is based on context adaptive binary arithmetic coding [CABAC] but with the support of multi-window sizes. The window sizes can be initialized differently for different context models. Due to such a design, it has more efficient adaptation speed and better coding efficiency. A joint chroma residual coding scheme is applied to further exploit the correlation between the residuals of two color components. In VVC, different residual coding schemes are applied for regular transform coefficients and residual samples generated using transform-skip mode.

## What if we did a two-pass approach?

| order | adaptive | empirical conditional entropy |
| --- | --- | --- |
| $0$th order | 4.26 | 4.26 |
| $1$st order | 3.34 | 3.27 |
| $2$nd order | 2.87 | 2.44 |
| $3$rd order | 3.10 | 1.86 |

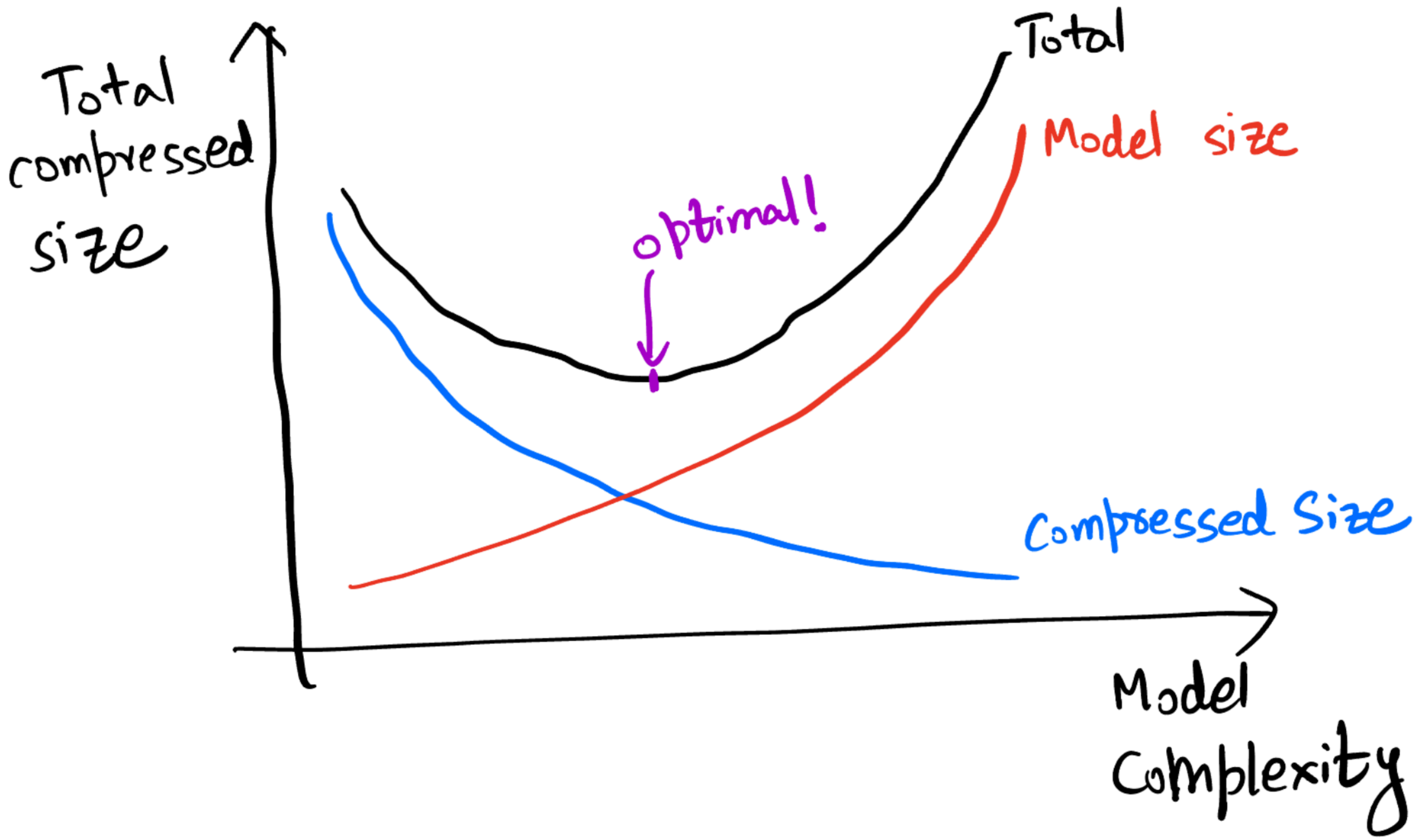
Why is there an increasing gap between adaptive coding performance and empirical entropy as we increase the order?

## Cost of storing the model!

- As the order increases, knowing the empirical distribution becomes closer to just storing the data itself in the model.
- At the extreme, you just have a single $|data\_size|$ long context and the model is just the data itself!
- We need to account for the cost of storing the model.
- In practice, adaptive models are often preferred due to their simplicity and not requiring two passes over the data.

# Minimum Description Length (MDL) principle

Minimize sum of model size and compressed size given model



# Lossless Text Compression - Hutter prize

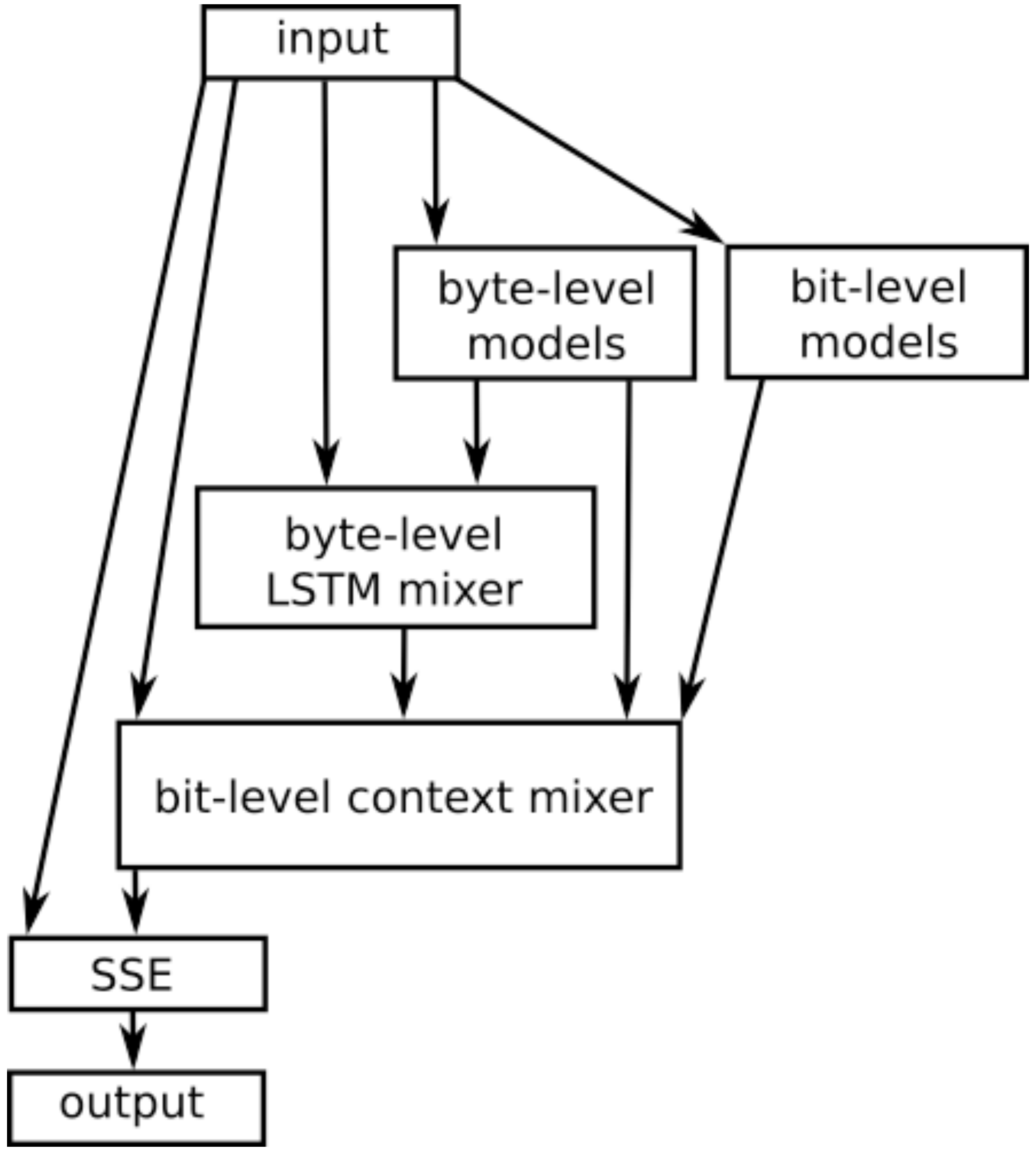
**500'000€ Prize for Compressing Human Knowledge**

**(widely known as the Hutter Prize. Total payout so far: 29'945€)**

**Compress the 1GB file enwik9 to less than the current record of about 110MB**

| Author (enwik9) | Date | Decompressor | Total Size | Compr.Factor\|RAM\|time | % \| Award | Sponsor |
|---|---|---|---|---|---|---|
| You? | 202? | ? | <109'685'197 | >9.12 \| <10GB \| <50h | >1%\| >5'000€ | Marcus Hutter |
| Kaido Orav & Byron Knoll | 3.Sep 2024 | fx2-cmix | 110'793'128 | 9.03 \| 95% \| 99% | 1.59%\| 7'950€ | Marcus Hutter |
| Kaido Orav | 2.Feb 2024 | fx-cmix | 112'578'322 | 8.88 \| 8.9GB \| ~50h | 1.38%\| 6'911€ | Marcus Hutter |
| Saurabh Kumar | 16.Jul 2023 | fast cmix | 114'156'155 | 8.76 \| 8.4GB \| 43h | 1.04%\| 5'187€ | Marcus Hutter |
| Artemiy Margaritov | 31.May 2021 | starlit ... | 115'352'938 | 8.67 \| 10GB \| ~50h | 1.1% \| 9000€ | Marcus Hutter |
| Alexander Rhatushnyak | 4.Jul 2019 | phda9v1.8 ... | 116'673'681 | 8.58 \| 6.3GB \| ~23h | -- \| pre-prize | - |

# Lossless Text Compression - CMIX



# Lossless Text Compression

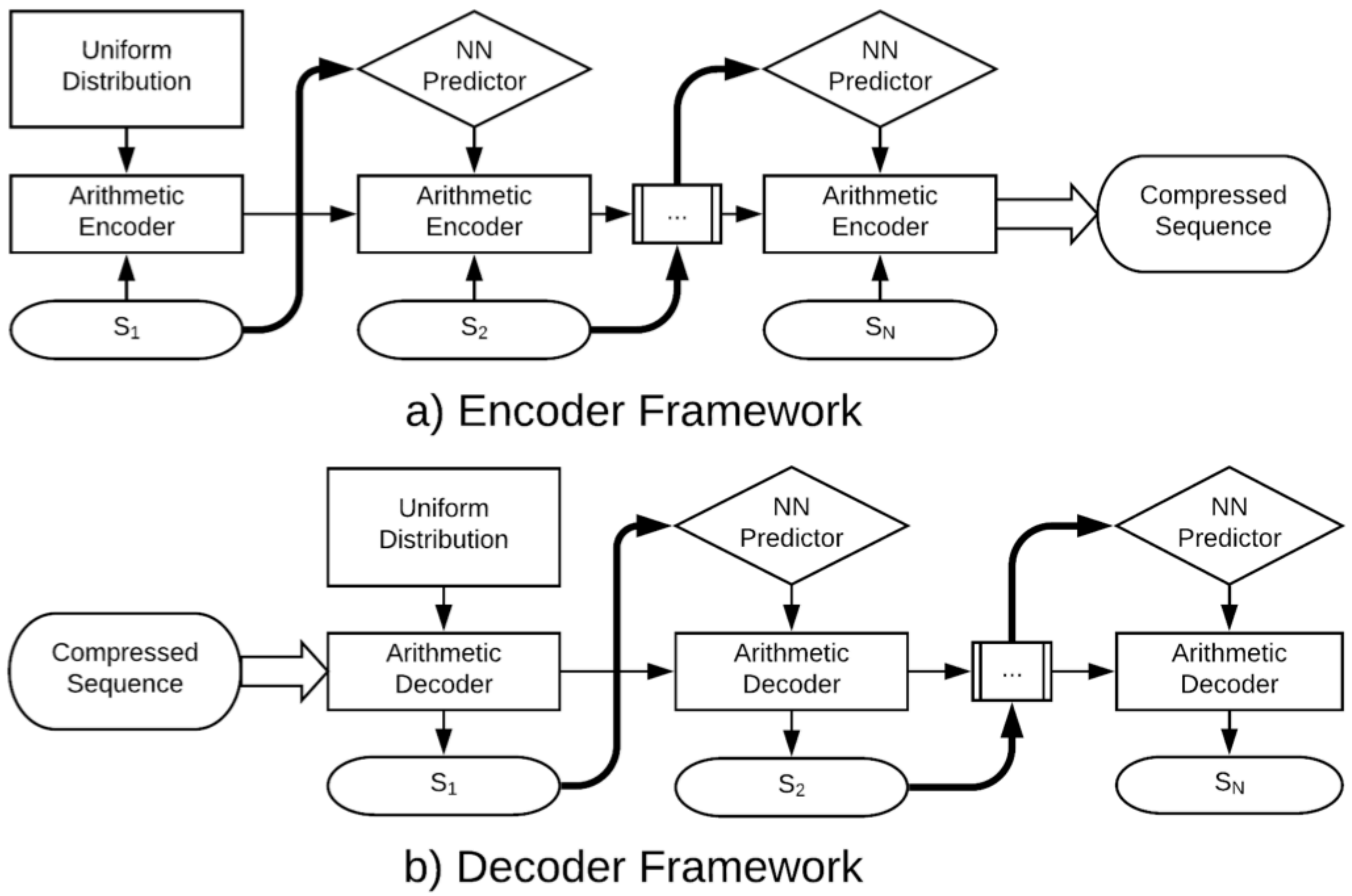
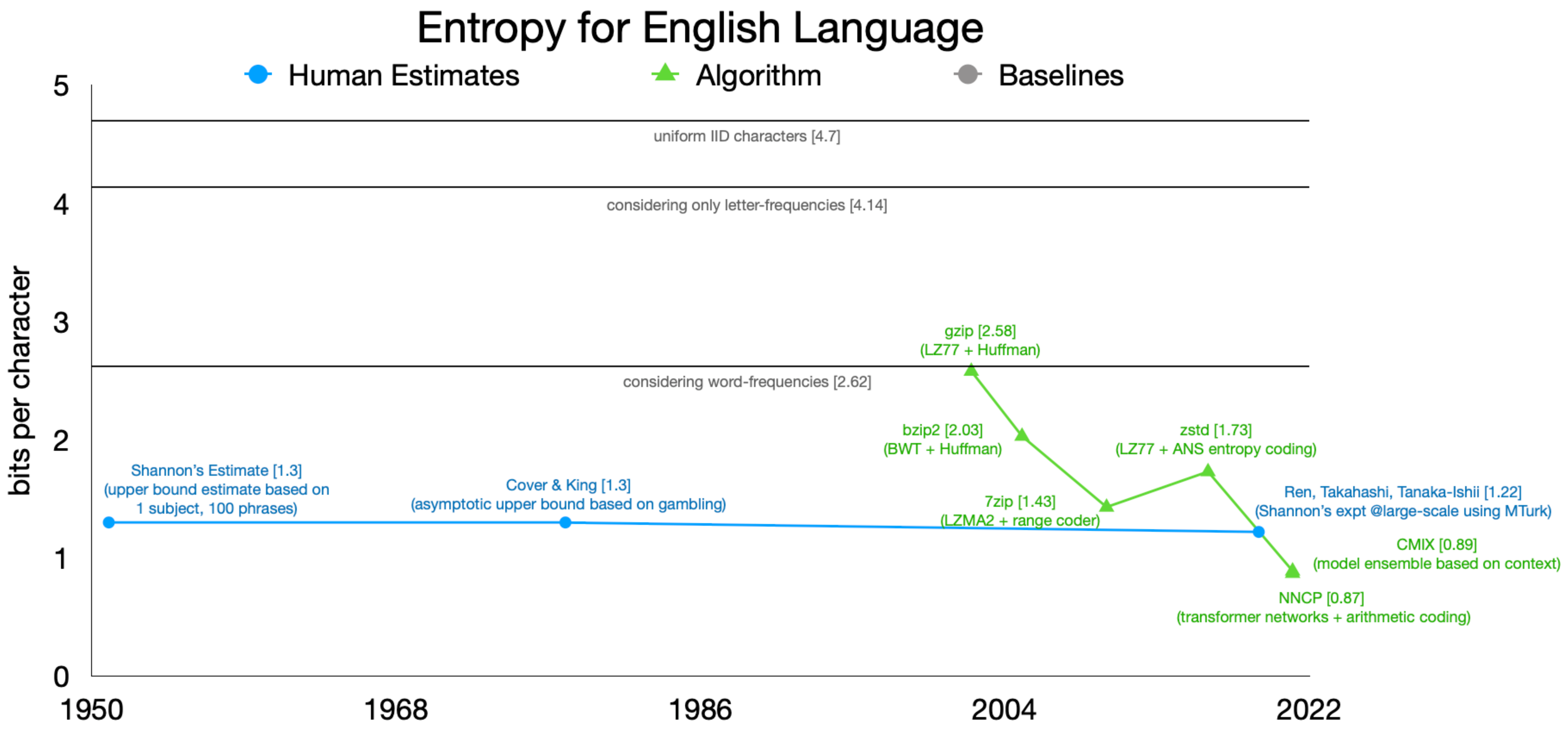



Figure 1: Encoder-Decoder Framework.

- Context-based arithmetic coding

# Lossless Text Compression [REPLACE]



# Lossless Text Compression

Sample prediction from Llama-2 3B param model

```
predict_next_token("Thank", top_k=5)
```

```
[0]: Token: you, Probability: 88.8%
[1]: Token: You, Probability: 4.2%
[2]: Token: fully, Probability: 1.8%
[3]: Token: good, Probability: 1.1%
[4]: Token: God, Probability: 1.0%
```

```
predict_next_token("I bought one banana and two apples. Total number of fruits is ", top_k=5)
```

```
[0]: Token: 3, Probability: 75.1%
[1]: Token: 4, Probability: 6.3%
[2]: Token: 5, Probability: 5.8%
[3]: Token: 1, Probability: 4.2%
[4]: Token: 2, Probability: 2.7%
```

***LLMs** are great predictors. Can we use them for compression?*

# Lossless Text Compression

```
compress("Twinkle Twinkle little sun",
             model_path="TheBloke/Llama-2-7B-GPTQ",
             max_context_size=5, verbose=True)
```

```
Next token: Tw        , Probability assigned by LLM: 0.00%, num_bits: 15.36 bits
Next token: ink       , Probability assigned by LLM: 2.74%, num_bits: 5.19 bits
Next token: le        , Probability assigned by LLM: 31.59%, num_bits: 1.66 bits
Next token: Tw        , Probability assigned by LLM: 28.61%, num_bits: 1.81 bits
Next token: ink       , Probability assigned by LLM: 98.05%, num_bits: 0.03 bits
Next token: le        , Probability assigned by LLM: 99.27%, num_bits: 0.01 bits
Next token: little    , Probability assigned by LLM: 1.40%, num_bits: 6.15 bits
Next token: sun       , Probability assigned by LLM: 0.02%, num_bits: 12.48 bits
model: TheBloke/Llama-2-7B-GPTQ, max_context_size: 5, total_size(bits): 42.7, bits_per_char: 1.64
```

Sample compression result from Llama-2 3B param model

# Lossless Text Compression

| Dataset | codec | bits/char |
|---|---|---|
| 2023 short stories | Huffman coding | 4.1 |
| 2023 short stories | 2nd order Arithmetic coding | 2.77 |
| 2023 short stories | gzip | 2.43 |
| 2023 short stories | bzip2 | 1.9 |
| 2023 short stories | Llama-2[3B, 512ctx] | 1.13 |
| 2023 short stories | Llama-2[13B, 512ctx] | 0.87 |
| 2023 short stories | Llama-2[70B, 512ctx] | 0.68 |

Better probability modeling
=> better compression

# Lossless Text Compression

| Dataset | codec | bits/char |
|---|---|---|
| 2023 short stories | Huffman coding | 4.1 |
| 2023 short stories | 2nd order Arithmetic coding | 2.77 |
| 2023 short stories | gzip | 2.43 |
| 2023 short stories | bzip2 | 1.9 |
| 2023 short stories | Llama-2[3B, 512ctx] | 1.13 |
| 2023 short stories | Llama-2[13B, 512ctx] | 0.87 |
| 2023 short stories | Llama-2[70B, 512ctx] | 0.68 |

| Dataset | codec | compressed bits/char |
|---|---|---|
| sherlock-novels | gzip | 2.31 |
| sherlock-novels | bzip2 | 2.03 |
| sherlock-novels | Llama-13B-(512ctx) | 0.2 |

Mis-leading/incorrect!

# State of Lossless Text Compression

| Data Type | Codec | Compression | Speed | Comments |
|---|---|---|---|---|
| **English Text** | Gzip | 2.3 bits/char | 100,000,000 char/sec | |
| **English Text** | Llama-70B, +arithmetic coding | **0.7 bits/char** | 10 char/sec | End-end text codec |
| **English Text** | Shannon Limit | 1.3 - 0.6 bits/ char | N/A | Fundamental limit on compression |

# Compression vs prediction

## Compression and prediction

Cross-entropy loss for prediction (classes $\mathcal{C}$, predicted probabilities $\hat{P}$, ground truth class: $y$):

$$\sum_{c \in \mathcal{C}} \mathbf{1}_{y_i = c} \log_2 \frac{1}{\hat{P}(c | y_1, \ldots, y_{i-1})}$$

Loss incurred when ground truth is $y_i$ is $\log_2 \frac{1}{\hat{P}(y_i | y_1, \ldots, y_{i-1})}$

Exactly matches the number of bits used for encoding with arithmetic coding!

# Compression vs prediction

## Compression and prediction

Cross-entropy loss for prediction (classes $\mathcal{C}$, predicted probabilities $\hat{P}$, ground truth class: $y$):

$$\sum_{c \in \mathcal{C}} \mathbf{1}_{y_i = c} \log_2 \frac{1}{\hat{P}(c | y_1, \ldots, y_{i-1})}$$

Loss incurred when ground truth is $y_i$ is $\log_2 \frac{1}{\hat{P}(y_i | y_1, \ldots, y_{i-1})}$

Exactly matches the number of bits used for encoding with arithmetic coding!

# Thank you!