

From Entropy to Epiplexity: Rethinking Information for Computationally Bounded Intelligence

Marc Finzi, Shikai Qiu, Yiding Jiang, Pavel Izmailov,
J. Zico Kolter, and Andrew Gordon Wilson (2026)

Lecture Notes

Sweta Karlekar

October 2026

Shannon Entropy

Suppose X is a coin flip with

$$P(X = \text{heads}) = \frac{1}{2}, \quad P(X = \text{tails}) = \frac{1}{2}.$$

The Shannon information of observing a particular outcome x is:

$$I(x) = -\log_2 P(X = x).$$

For example,

$$I(\text{heads}) = -\log_2\left(\frac{1}{2}\right) = 1 \text{ bit}.$$

Shannon entropy is the average surprisal:

$$H(X) = \mathbb{E}[I(X)] = -\sum_x P(x) \log_2 P(x).$$

↪ This is a lower bound on the average code length needed to communicate samples to another party.

$$H(X) = 1 \text{ bit for a fair coin}.$$

To calculate Shannon information, you need a probability model P .

What if the model is misspecified?

Let Q be the true probability distribution:

$$X \sim Q.$$

But we use P_M . Our model gives x an ideal code length $-\log_2 P_M(x)$.

The Shannon cross-entropy identity gives the expected code length under M for data sampled from Q :

$$\begin{aligned} H(Q, P_M) &= \mathbb{E}_{X \sim Q}[-\log_2 P_M(X)] \\ &= -\sum_x Q(x) \log_2 P_M(x) \\ &= -\sum_x Q(x) \log_2 Q(x) + \sum_x Q(x) [\log_2 Q(x) - \log_2 P_M(x)] \\ &= H(Q) + \sum_x Q(x) \log_2 \frac{Q(x)}{P_M(x)} \\ &= \underbrace{H(Q)}_{\text{intrinsic uncertainty}} + \underbrace{D_{\text{KL}}(Q \| P_M)}_{\text{extra bits from model mismatch}}. \end{aligned}$$

This still assumes a probability distribution. What if we only have an object x ?

Kolmogorov Complexity

x is a binary string; fix a universal programming machine U .

$$K(x) = \min_p \{|p| : U(p) = x\}.$$

$\hookrightarrow$ Length of the shortest program that prints x and stops.

Example: $x = 0101010101\dots$

$p() : \text{repeat "01" } n \text{ times}$
then we need to communicate n $K(x) = O(\log_2 n)$.

$\longrightarrow$ We were able to compress this by a lot!

Example: $x = 431780\dots$ (random noise)

$p() : \text{print "431780\dots"}$ $K(x) \approx n$.

$\longrightarrow$ We were not able to compress this.

Can we talk about incompressibility as randomness?

Martin-Löf Randomness

We have some infinite sequence; let

$$x_{1:n} = x_1 x_2 \cdots x_n.$$

A sequence is Martin-Löf random if there is some fixed constant c such that, for every n ,

$$K(x_{1:n}) \geq n - c.$$

That is, no matter how much of the sequence we examine, we can never compress that portion by more than c bits.

Going back to the previous example,

$$x = 010101\dots, \quad K(x_{1:n}) = O(\log n),$$
$$K(x_{1:n}) < n - c \quad \text{for sufficiently large } n, \text{ for any fixed } c.$$

So this fails the randomness condition.

$\hookrightarrow$ The randomness condition is equivalent to passing all effective statistical tests with suitably controlled false-positive probability.

What does randomness look like for a bounded observer?

Pseudorandomness

Kolmogorov complexity allows for unlimited running time. This is not realistic.

For pseudorandomness: a sequence is pseudorandom if observers with a specified amount of computation cannot distinguish it from genuinely random data.

We have some pseudorandom generator G :

$$G : \{0, 1\}^k \longrightarrow \{0, 1\}^{n(k)}, \quad n(k) > k,$$

with a seed S chosen uniformly from $\{0, 1\}^k$.

G is deterministic, so there are $2^{n(k)}$ possible $n(k)$ -bit strings, but at most 2^k actual outputs.

What is the Shannon entropy? Assuming G is injective,

$$\begin{aligned} H(G(S)) &= \mathbb{E}[-\log_2 P_{G(S)}(G(S))] \\ &= -\log_2(2^{-k}) \\ &= k. \end{aligned}$$

And the output has a short algorithmic description:

$$K(G(s)) \leq K(G) + k.$$

$\hookrightarrow$ Program: store the seed s , then run G .

So the output does not have $n(k)$ bits of algorithmic randomness, but it still looks random.

Let U_n be a genuinely uniform random n -bit string, and let D be a polynomial-time distinguisher which outputs 1 for “pseudorandom” and 0 for “truly random”.

Then, for every such D ,

$$\underbrace{|\Pr[D(G(S)) = 1] - \Pr[D(U_n) = 1]|}_{\text{distinguishing advantage}} \quad \text{must be negligible.}$$

$\hookrightarrow \Pr[D(G(S)) = 1]$: chance of classifying $G(S)$ as pseudorandom.

$\hookrightarrow \Pr[D(U_n) = 1]$: chance of classifying uniform noise as pseudorandom.

An unbounded observer *could* distinguish them:

- Enumerate every possible seed.
- Calculate every output of G .
- Check whether the observed string is on the list.

But this requires 2^k operations. For sufficiently large k , this is infeasible.

Recall that for the pseudorandom generator,

$$K(G(s)) \leq \underbrace{K(G)}_{\text{structure}} + \underbrace{k}_{\text{particular realization}}.$$

Sophistication

This tries to separate *structure* from *randomness*.

Now, instead of looking for a program that prints x , we look for a set S containing x .

1. Describe the set S : $K(S)$.
2. Identify which member of S is x : $\log_2 |S|$.

$$K(S) + \log_2 |S|.$$

We want x to be a *typical*, or effectively random, member of S :

$$\underbrace{K(x | S)}_{\substack{\text{shortest description} \\ \text{once } S \text{ is known}}} > \underbrace{\log_2 |S|}_{\substack{\text{bits to select} \\ \text{a member of } S}} - c.$$

$\hookrightarrow$ Knowing S should leave no substantial additional pattern in x , up to an allowance of c bits.

Naive sophistication:

$$\text{naiveSoph}_c(x) = \min_S \{K(S) : K(x | S) > \log_2 |S| - c\}.$$

$\longrightarrow$ The complexity of the simplest model within which x is typical.

Example: x is a random n -bit string.

$$S = \{0, 1\}^n \quad (\text{set of all } n\text{-bit strings}).$$

A program describing S : “any n -bit string is possible.” We only need to specify n .

$$\begin{aligned} K(S) &= O(\log n), \\ \log_2 |S| &= \log_2(2^n) = n, \\ K(x | S) &\approx n. \end{aligned}$$

- High total Kolmogorov complexity.
- Low sophistication.
- Almost all information is random.

Example: x is a repetitive string 010101...

$$S = \{x\}.$$

A program describing S : “repeat 01 n times,” so

$$K(S) = O(\log n), \\ |S| = 1, \quad \log_2 |S| = 0.$$

- Low total Kolmogorov complexity.
- Low sophistication.
- Low remaining randomness.

Now, what happens when we don’t have a set S , but rather a probability model?
Minimum Description Length

Suppose M defines a probability distribution P_M on n -bit strings:

$$M \mapsto P_M, \quad P_M : \{0, 1\}^n \rightarrow [0, 1], \quad \sum_x P_M(x) = 1.$$

To describe x using M , we communicate two things:

$$\underbrace{L(M)}_{\text{bits to describe model}} + \underbrace{[-\log_2 P_M(x)]}_{\text{bits to describe } x \text{ using the model}}.$$

$\hookrightarrow L(\cdot)$ denotes the length of a model description; one can also consider the uncomputable $K(\cdot)$.

Minimum description length (MDL) chooses the model with the shortest total description:

$$\text{MDL}(x) = \min_{M \in \mathcal{M}} [L(M) - \log_2 P_M(x)], \\ M^* = \arg \min_{M \in \mathcal{M}} [L(M) - \log_2 P_M(x)].$$

$\hookrightarrow$ This minimizes the *whole description*, instead of just the model.

$\longrightarrow$ The model should be simple and assign high probability to x : compression must pay for the model’s description.

Example: recall the pseudorandom generator,

$$G : \{0, 1\}^k \rightarrow \{0, 1\}^{n(k)}, \quad k < n(k), \quad x = G(s), \quad s \sim U_k.$$

Define M_G by drawing a uniform seed s' and returning $G(s')$.

$$\underbrace{L(M_G) + k}_{\text{generator model}} \quad \text{instead of} \quad \underbrace{L(M_U) + n(k)}_{\text{uniform model}}.$$

$\hookrightarrow$ The k -bit description of x assumes G is injective (as in the earlier example).

If you’re given the seed, this is easy to compute. But what if you don’t know whether x is an output of G ?

$$P_{M_G}(x) = 2^{-k} \sum_{s \in \{0, 1\}^k} \mathbf{1}\{G(s) = x\}.$$

$\longrightarrow$ Evaluating $P_{M_G}(x)$ can be intractable for large k .

And now we've arrived at the core of the paper!

- Restrict M to models that run within a computation budget.
- Then find the model that minimizes MDL:
 - Model description part is epiplexity.
 - Remaining data randomness is time-bounded entropy.

Time-Bounded MDL

We are given a computation budget $T(n)$; n is the length of the binary object being modeled.

A program P qualifies as a T -time probabilistic model only if it can perform two actions within $T(n)$ steps:

- (1) probability evaluation (2) sampling.

(1) Probability evaluation

Given a candidate string $x \in \{0, 1\}^n$, the program must calculate $P(x)$:

$$\text{Eval}_P(x) = P_P(x).$$

$\hookrightarrow$ What probability does the model assign to x ?

(2) Sampling

Given access to random bits, the program must generate a sample

$$x \sim P_P.$$

The evaluator and sampler must agree:

$$\Pr[\text{sampler outputs } x] = P_P(x).$$

Coming back to the pseudorandom generator:

Sample

Draw one seed $s \sim U_k$.

Compute $x = G(s)$.

Return x .

Probability evaluation

Examine every seed to compute $P_{M_G}(x)$.

$\hookrightarrow$ Sampling is fast; probability evaluation can be infeasible.

We now define MDL using only the programs that meet the time bound.

Let $X \sim Q$ be the random variable generating n -bit data:

$$X \in \{0, 1\}^n, \quad P \in \underbrace{\mathcal{P}_T}_{\text{set of feasible programs}}.$$

$$P^* = \arg \min_{P \in \mathcal{P}_T} \{|P| + \mathbb{E}_{x \sim Q}[-\log_2 P(x)]\}.$$

Choose the feasible program with the smallest expected two-part description.

Time-bounded epiplexity

$$S_T(X) := |P^*|.$$

$\hookrightarrow$ Description length of the best feasible model.

Time-bounded entropy

$$H_T(X) := \mathbb{E}_{x \sim Q}[-\log_2 P_{P^*}(x)].$$

$\hookrightarrow$ Bits that remain unpredictable by the best feasible model.

$$\text{MDL}_T(X) = \underbrace{S_T(X)}_{\text{epiplexity}} + \underbrace{H_T(X)}_{\text{time-bounded entropy}}.$$

How does time-bounded entropy compare to Shannon entropy?

Shannon entropy uses the true distribution Q :

$$H(X) = \mathbb{E}_{x \sim Q}[-\log_2 Q(x)].$$

Time-bounded entropy uses the optimal feasible model:

$$H_T(X) = \mathbb{E}_{x \sim Q}[-\log_2 P_{P^*}(x)].$$

This is the cross-entropy between Q and P_{P^*} .

$$\begin{aligned}
H_T(X) &= H(Q, P_{\mathbf{P}^*}) \\
&= - \sum_x Q(x) \log_2 P_{\mathbf{P}^*}(x) \\
&= - \sum_x Q(x) \log_2 Q(x) + \sum_x Q(x) [\log_2 Q(x) - \log_2 P_{\mathbf{P}^*}(x)] \\
&= H(Q) + \sum_x Q(x) \log_2 \frac{Q(x)}{P_{\mathbf{P}^*}(x)} \\
&= \underbrace{H(Q)}_{\text{intrinsic randomness}} + \underbrace{D_{\text{KL}}(Q \| P_{\mathbf{P}^*})}_{\text{extra bits from model mismatch}}.
\end{aligned}$$

Since KL divergence is non-negative, we get a lower bound:

$$\boxed{H_T(X) \geq H(Q) = H(X).}$$

↪ Using the best feasible model cannot beat the average code length under the true distribution.

What about an upper bound?

We can just give up on compression entirely and transmit the n bits:

$$X \in \{0, 1\}^n, \quad P_{\text{unif}}(x) = 2^{-n} \quad \forall x \in \{0, 1\}^n.$$

Every possible string has code length

$$-\log_2 P_{\text{unif}}(x) = -\log_2(2^{-n}) = n.$$

Model: probability evaluation

Eval(x): return 2^{-n} .

Model: sampling

Generate n independent fair bits.

$$|P_{\text{unif}}| = c = O(1),$$

$$|P_{\text{unif}}| + \mathbb{E}_{X \sim Q}[-\log_2 P_{\text{unif}}(X)] = c + n.$$

Recall that $\mathbf{P}^*$ is selected by minimizing over every feasible time- T model, so

$$\boxed{S_T(X) + H_T(X) \leq n + O(1).}$$

↪ Upper bound: the uniform model must fit within $T(n)$; the string length n is supplied to the program.

Conditional epiplexity

So far, our model describes X on its own. What if we already know X and only need to describe Y ? Let $(X, Y) \sim Q$. We now use a conditional probabilistic model P :

$$P \mapsto P_P(y \mid x), \quad \sum_y P_P(y \mid x) = 1 \quad \text{for every } x.$$

Given x , the program must still do both operations within T steps:

$$\begin{aligned} \text{probability evaluation: } (x, y) &\mapsto P_P(y \mid x), \\ \text{sampling: } (x, u) &\mapsto y \sim P_P(\cdot \mid x). \end{aligned}$$

We write $\mathcal{P}_T^X$ for the set of feasible *conditional* programs.

We choose the model that gives the shortest expected two-part description of Y , given X :

$$P_{Y|X}^* = \arg \min_{P \in \mathcal{P}_T^X} \left\{ \underbrace{|P|}_{\text{model bits}} + \underbrace{\mathbb{E}_{(X,Y) \sim Q}[-\log_2 P_P(Y \mid X)]}_{\text{remaining bits for } Y} \right\}.$$

We define the **conditional epiplexity** and **time-bounded conditional entropy** as

$$\begin{aligned} S_T(Y \mid X) &:= |P_{Y|X}^*|, \\ H_T(Y \mid X) &:= \mathbb{E}_{(X,Y) \sim Q}[-\log_2 P_{P_{Y|X}^*}(Y \mid X)], \\ \underbrace{\text{MDL}_T(Y \mid X)}_{\text{total description}} &= \underbrace{S_T(Y \mid X)}_{\text{conditional epiplexity}} + \underbrace{H_T(Y \mid X)}_{\text{conditional entropy}}. \end{aligned}$$

$\hookrightarrow$ The program is described once; X is available as input and does not need to be encoded again.

Some simple examples

Copying: suppose $Y = X$. A program can just return the input:

$$P_{\text{copy}}(y \mid x) = \mathbf{1}\{y = x\}, \quad -\log_2 P_{\text{copy}}(Y \mid X) = -\log_2 1 = 0.$$

$\longrightarrow$ A constant-size program achieves zero conditional log-loss, so $S_T(Y \mid X), H_T(Y \mid X) = O(1)$.

Deterministic rule: suppose $Y = f(X)$, where f is simple and runs within T . For example, $Y = X_1 \oplus \dots \oplus X_n$.

$$P_f(y \mid x) = \mathbf{1}\{y = f(x)\}, \quad |P_f| = O(1), \quad \mathbb{E}[-\log_2 P_f(Y \mid X)] = 0.$$

$\hookrightarrow$ The rule is stored in the model; knowing X leaves no uncertainty about Y .

Random labels: suppose $Y \in \{0, 1\}^m$ is uniform and independent of X .

$$\begin{aligned} Y &\sim U_m, \quad Y \perp X, \quad P_{\text{unif}}(y \mid x) = 2^{-m}, \\ -\log_2 P_{\text{unif}}(Y \mid X) &= m. \end{aligned}$$

Since $H(Y \mid X) = m$ and the uniform conditional model costs only $O(1)$ bits,

$$\boxed{S_T(Y \mid X) = O(1), \quad m \leq H_T(Y \mid X) \leq m + O(1).}$$

Image classification: X is an image and Y is its label.

$$X = \text{image}, \quad Y \in \{1, \dots, C\}, \quad P_{\mathcal{P}}(y \mid x).$$

Our model only needs to describe the *label given the image*:

$$\text{MDL}_T(Y \mid X) = \underbrace{|\mathcal{P}_{Y|X}^*|}_{\text{model bits}} + \underbrace{\mathbb{E}[-\log_2 P_{\mathcal{P}_{Y|X}^*}(Y \mid X)]}_{\text{remaining label bits}}.$$

→ We are not trying to model the distribution of pixels; only the mapping from images to labels.

Does conditional time-bounded entropy inherit the order-independence of Shannon entropy?

Recall Shannon's chain rule

Suppose the true joint distribution is $Q(X, Y)$. We can factor it in either order:

$$Q(x, y) = Q(x)Q(y \mid x) = Q(y)Q(x \mid y).$$

Taking $-\log_2$ and then the expectation gives

$$\begin{aligned} H(X, Y) &= \mathbb{E}[-\log_2 Q(X, Y)] \\ &= \mathbb{E}[-\log_2 Q(X) - \log_2 Q(Y \mid X)] \\ &= H(X) + H(Y \mid X), \end{aligned}$$

so similarly,

$$\boxed{H(X) + H(Y \mid X) = H(X, Y) = H(Y) + H(X \mid Y)}.$$

↔ The expected bits are the same whether we describe X then Y , or Y then X .

But with a time bound, the two conditional models may require *very different computations*:

$$H_T(X) + H_T(Y \mid X) \stackrel{?}{=} H_T(Y) + H_T(X \mid Y).$$

We can show that this equality fails using a one-way permutation.

An example where order matters

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a **one-way permutation**: computing $f(x)$ is fast, but computing $f^{-1}(y)$ is infeasible in polynomial time.

$$X \sim U_n, \quad Y = f(X), \quad \underbrace{X \xrightarrow{f \text{ (fast)}} Y}_{\text{forward}} \quad \underbrace{Y \xrightarrow{f^{-1} \text{ (hard)}} X}_{\text{reverse}}.$$

Since f is a permutation, Y is also uniform. Under Shannon entropy:

$$H(X) = H(Y) = n, \quad H(Y \mid X) = H(X \mid Y) = 0.$$

$$\underbrace{H(X) + H(Y \mid X)}_{n+0} = \underbrace{H(Y) + H(X \mid Y)}_{n+0} = n.$$

Forward: given x , computing $y = f(x)$ is feasible.

$$P_f(y | x) = \mathbf{1}\{y = f(x)\}, \quad \underbrace{\mathbb{E}[-\log_2 P_f(Y | X)]}_{\text{conditional log-loss}} = 0.$$

A constant-size program implements this model, including evaluation and sampling:

$$\boxed{S_{\text{poly}}(Y | X) = O(1), \quad H_{\text{poly}}(Y | X) = O(1).}$$

$\hookrightarrow$ Zero loss is achievable by a feasible model; the MDL minimizer has total length $O(1)$.

Reverse: the exact conditional probability is also easy to *evaluate*:

$$Q(x | y) = \mathbf{1}\{f(x) = y\}.$$

But a sampler from this distribution must output $x = f^{-1}(y)$.

$$\underbrace{\text{Eval}(x, y) = \mathbf{1}\{f(x) = y\}}_{\text{fast}} \quad \underbrace{\text{Sample}(y) = f^{-1}(y)}_{\text{infeasible}}.$$

$\longrightarrow$ The exact reverse conditional model is *not* feasible: evaluation alone is not enough.

How much loss must a polynomial-time conditional model incur? Let $\mathbf{P}^* = \mathbf{P}_{X|Y}^*$ and define

$$\phi(y) := P_{\mathbf{P}^*}(f^{-1}(y) | y).$$

$\hookrightarrow \phi(y)$ is the chance that one sample from the model gives the correct inverse of y .

If we use the model's sampler to try to invert f , its success probability is

$$\begin{aligned} \varepsilon(n) &:= \Pr_{X \sim U_{n,u}}[\text{Sample}_{\mathbf{P}^*}(f(X), u) = X] \\ &= \mathbb{E}_Y[\phi(Y)]. \end{aligned}$$

So by Jensen's inequality,

$$\begin{aligned} H_{\text{poly}}(X | Y) &= \mathbb{E}_Y[-\log_2 \phi(Y)] \\ &\geq -\log_2 \mathbb{E}_Y[\phi(Y)] \\ &= \log_2 \frac{1}{\varepsilon(n)}. \end{aligned}$$

By one-wayness, $\varepsilon(n)$ is negligible:

$$\varepsilon(n) < n^{-c} \text{ eventually for every } c > 0 \quad \implies \quad \boxed{H_{\text{poly}}(X | Y) = \omega(\log n).}$$

Finally, $H_{\text{poly}}(X) = n + O(1)$ and $H_{\text{poly}}(Y) = n + O(1)$, so

$$\begin{aligned} H_{\text{poly}}(X) + H_{\text{poly}}(Y | X) &= n + O(1), \\ H_{\text{poly}}(Y) + H_{\text{poly}}(X | Y) &\geq n + \omega(\log n). \end{aligned}$$

$\longrightarrow$ Same joint data, different prediction order $\longrightarrow$ different time-bounded entropy.

We have defined epiplexity using the shortest feasible program. How do we estimate its description length in practice?

$\longrightarrow$ **Prequential coding**

Prequential coding

We have defined epiplexity using the description length of a feasible model:

$$S_T(X) = |\mathbf{P}^*|.$$

How do we estimate the number of bits needed to describe a *trained neural network*?

For example, a network with 10^9 parameters stored as 32-bit numbers takes about 4 GB.

But training it on 10 examples does not mean it learned 4 GB of information *from those examples*.

Instead, how many bits of training data must we send to reproduce the trained model?

Recall the Kolmogorov complexity of the trained model P_M :

$$K(P_M) = \text{length of its shortest program description.}$$

Suppose $Z_{:M} = (Z_0, \dots, Z_{M-1})$ is the training data and P_M is the final model. The information-decomposition idea is

$$K(P_M) \approx K(Z_{:M}, P_M) - K(Z_{:M} | P_M).$$

$\hookrightarrow$ By symmetry of information, joint minus conditional describes the model (up to lower-order terms).

Since K is uncomputable, we estimate these terms using code lengths L :

$$|\mathbf{P}_{\text{preq}}| \approx \underbrace{L(Z_{:M}, P_M)}_{\text{data and final model}} - \underbrace{L(Z_{:M} | P_M)}_{\text{data given final model}}.$$

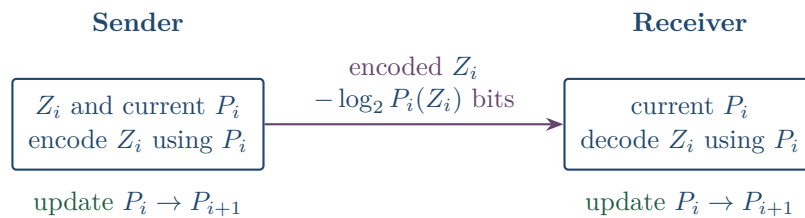
$\hookrightarrow$ This is a heuristic estimate; we have not yet constructed a stand-alone code for P_M .

1. Describe both the training data and the model

We begin with a model P_0 , then train on one example at a time:

$$P_0 \xrightarrow{Z_0} P_1 \xrightarrow{Z_1} P_2 \cdots \xrightarrow{Z_{M-1}} P_M.$$

Sender and receiver share the initialization and deterministic training procedure.



For each Z_i , the receiver *first* decodes it, *then* trains on it. Thus both sides recover the same $Z_{:M}$ and final model P_M .

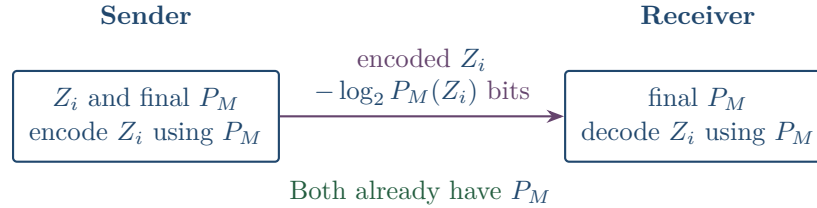
$$L(Z_{:M}, P_M) \approx \sum_{i=0}^{M-1} \underbrace{\log_2 \frac{1}{P_i(Z_i)}}_{\text{bits using the current model}}$$

$\hookrightarrow$ The predictions $P_i(Z_i)$ are made *before* training on Z_i ; fixed setup costs are omitted.

Prequential coding: isolating the model bits

2. Describe the training data given the final model

Now suppose the receiver already knows the final model P_M . We only need to send $Z_{1:M}$; neither side has to train to decode it.



Each example is encoded with the probabilities assigned by the *final* model:

$$L(Z_{1:M} | P_M) \approx \sum_{i=0}^{M-1} \underbrace{\log_2 \frac{1}{P_M(Z_i)}}_{\text{bits using the final model}}$$

Now subtract the two descriptions to estimate how much information was learned:

$$\begin{aligned} |P_{\text{preq}}| &\approx L(Z_{1:M}, P_M) - L(Z_{1:M} | P_M) \\ &\approx \sum_{i=0}^{M-1} \left[\log_2 \frac{1}{P_i(Z_i)} - \log_2 \frac{1}{P_M(Z_i)} \right] \\ &= \sum_{i=0}^{M-1} [-\log_2 P_i(Z_i) + \log_2 P_M(Z_i)] \\ &= \sum_{i=0}^{M-1} \log_2 \frac{P_M(Z_i)}{P_i(Z_i)} \quad \text{ratio: } \frac{\text{final model probability}}{\text{original prediction probability}} \end{aligned}$$

A small example

Suppose the current model assigns Z_i probability $1/8$; after training, the final model assigns $1/2$:

$$P_i(Z_i) = \frac{1}{8}, \quad P_M(Z_i) = \frac{1}{2}.$$

Then the two code lengths are

$$\begin{aligned} \text{current model:} & \quad -\log_2(1/8) = 3 \text{ bits,} \\ \text{final model:} & \quad -\log_2(1/2) = 1 \text{ bit.} \end{aligned}$$

$$\log_2 \frac{P_M(Z_i)}{P_i(Z_i)} = \log_2 \frac{1/2}{1/8} = \log_2 4 = 2 \text{ bits.}$$

→ The final model describes this example using 2 fewer bits; the gap contributes 2 bits to the estimate.

Prequential coding and the training loss

How do we calculate these quantities during neural-network training?

We already record the **loss before updating** on each training example:

$$\ell_i^{\text{preq}} := -\log_2 P_i(Z_i).$$

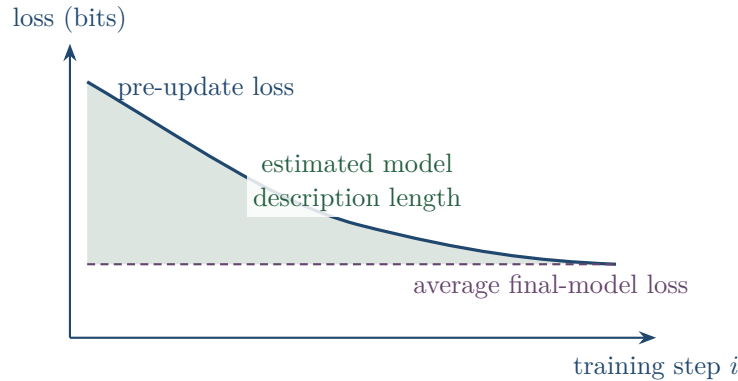
After training, evaluate the *same* examples using the final model:

$$\ell_i^{\text{final}} := -\log_2 P_M(Z_i).$$

Then the prequential estimate is simply the difference between these two loss sums:

$$|\mathbf{P}_{\text{preq}}| \approx \underbrace{\sum_{i=0}^{M-1} \ell_i^{\text{preq}}}_{\text{loss during training}} - \underbrace{\sum_{i=0}^{M-1} \ell_i^{\text{final}}}_{\text{loss under final model}}.$$

For i.i.d. data, this is the *area between the averaged pre-update loss curve and the final loss*:



↪ Since P_i has not yet seen Z_i , pre-update loss estimates the loss on unseen data.

A network learns more *structure* when its loss decreases substantially over training.

- Pure noise: loss hardly decreases → small area.
- Simple pattern: loss drops quickly → small area.
- Hard but learnable pattern: sustained improvement → larger area.

We can now estimate the two parts of time-bounded MDL for a separate test dataset $X \sim Q$:

$$\underbrace{\widehat{\text{MDL}}_T(X)}_{\text{estimated total bits}} \approx \underbrace{|\mathbf{P}_{\text{preq}}|}_{\text{estimated epiplexity}} + \underbrace{\mathbb{E}_{X \sim Q}[-\log_2 P_M(X)]}_{\text{remaining data bits}}.$$

↪ Estimate the last expectation with held-out test loss; do not confuse X with training data $Z_{:M}$.

But can subtracting two code lengths really produce a valid code for the *model alone*?

$$L(Z_{:M}, P_M) \geq K(Z_{:M}, P_M), \quad L(Z_{:M} \mid P_M) \geq K(Z_{:M} \mid P_M)$$

Subtracting two upper bounds does not give an upper bound on their difference.

→ Prequential coding: explicitly describe the model with a known runtime.

Requential coding

Recall our estimate from prequential coding:

$$|\mathcal{P}_{\text{preq}}| \approx L(Z_{:M}, P_M) - L(Z_{:M} \mid P_M).$$

But both code lengths only give *upper bounds* on the corresponding Kolmogorov complexities:

$$\begin{aligned} K(Z_{:M}, P_M) &\leq L(Z_{:M}, P_M), \\ K(Z_{:M} \mid P_M) &\leq L(Z_{:M} \mid P_M). \end{aligned}$$

Subtracting two upper bounds does not give us an explicit code for P_M .

Instead of sending the *original* training data, can we send a different training sequence that produces a useful model?

We have two models at each training step i :

$$\begin{aligned} P_i^t &: \text{teacher, trained using real data,} \\ P_i^s &: \text{student, which we want to transmit.} \end{aligned}$$

The student learns from *synthetic* examples drawn from the teacher:

$$\tilde{Z}_i \sim P_i^t, \quad P_{i+1}^s = \text{Update}(P_i^s, \tilde{Z}_i).$$

$\hookrightarrow$ Our goal is to reconstruct the final student P_M^s , not the teacher or exact real data.

What does the sender transmit?

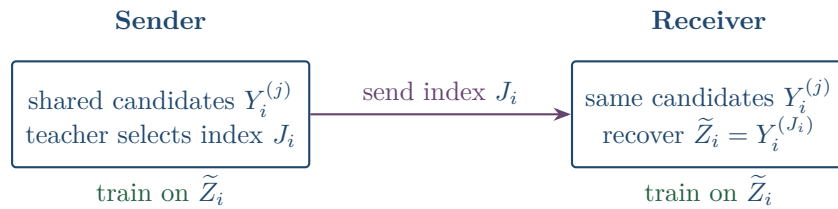
Sender and receiver share the initial student, training rule, and a random seed. At step i , both sides can generate the *same* candidate examples:

$$Y_i^{(1)}, Y_i^{(2)}, \dots \stackrel{\text{i.i.d.}}{\sim} P_i^s.$$

The teacher favors candidates with larger probability ratio

$$\underbrace{w_i(y)}_{\text{relative weight}} = \frac{P_i^t(y)}{P_i^s(y)}.$$

$\hookrightarrow$ More weight when the teacher assigns y higher probability than the student does.



A suitable *randomized* selection rule makes the selected example satisfy

$$\tilde{Z}_i = Y_i^{(J_i)} \sim P_i^t.$$

$\longrightarrow$ We only transmit the selected index; the receiver never needs the teacher.

Requential coding: a Gaussian example

Suppose the student proposes values near 0, but the teacher wants values near 1:

$$P^s = \mathcal{N}(0, 1), \quad P^t = \mathcal{N}(1, 1).$$

Their probability densities are

$$p_s(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}, \quad p_t(y) = \frac{1}{\sqrt{2\pi}} e^{-(y-1)^2/2}.$$

The teacher reweights samples proposed by the student:

$$\begin{aligned} w(y) &= \frac{p_t(y)}{p_s(y)} \\ &= \frac{e^{-(y-1)^2/2}}{e^{-y^2/2}} \\ &= \exp\left(\frac{y^2 - (y-1)^2}{2}\right) \\ &= \boxed{e^{y-1/2}}. \end{aligned}$$

↪ Larger candidate values receive more weight because the teacher is shifted to the right.

For intuition, imagine these four shared proposals:

$$(y^{(1)}, y^{(2)}, y^{(3)}, y^{(4)}) = (-1, 0, 1, 2).$$

Then

$$(w_1, w_2, w_3, w_4) = (e^{-3/2}, e^{-1/2}, e^{1/2}, e^{3/2}) \approx (0.22, 0.61, 1.65, 4.48).$$

An illustrative *importance-resampling* rule assigns probabilities proportional to these weights:

$$\Pr(J = j \mid y^{(1:4)}) = \frac{w_j}{\sum_{k=1}^4 w_k} \approx (0.03, 0.09, 0.24, 0.64).$$

If $J = 4$, the sender communicates “4” and both sides recover the proposal $y^{(4)} = 2$.

↪ This four-candidate example only illustrates reweighting; exact relative entropy coding uses a more careful randomized selection rule.

How many bits does this cost?

The leading communication cost is the *KL divergence* between teacher and student:

$$\begin{aligned} D_{\text{KL}}(P^t \parallel P^s) &= \mathbb{E}_{Y \sim P^t} \left[\log_2 \frac{p_t(Y)}{p_s(Y)} \right] \\ &= \mathbb{E}_{Y \sim \mathcal{N}(1, 1)} \left[\log_2(e^{Y-1/2}) \right] \\ &= \frac{1}{\ln 2} \mathbb{E} \left[Y - \frac{1}{2} \right] \\ &= \frac{1 - 1/2}{\ln 2} = \boxed{\frac{1}{2 \ln 2} \approx 0.72 \text{ bits.}} \end{aligned}$$

→ Close distributions → small KL → little new information must be communicated.

Sequential coding: model description length

Why does KL determine the communication cost? For a sample $\tilde{Z}_i \sim P_i^t$, compare its description under the student and teacher:

$$\begin{aligned} & \underbrace{-\log_2 P_i^s(\tilde{Z}_i)}_{\text{student's code length}} - \underbrace{[-\log_2 P_i^t(\tilde{Z}_i)]}_{\text{teacher's code length}} \\ &= \log_2 \frac{P_i^t(\tilde{Z}_i)}{P_i^s(\tilde{Z}_i)}. \end{aligned}$$

Taking the expectation over teacher samples:

$$\mathbb{E}_{\tilde{Z}_i \sim P_i^t} \left[\log_2 \frac{P_i^t(\tilde{Z}_i)}{P_i^s(\tilde{Z}_i)} \right] = \boxed{D_{\text{KL}}(P_i^t \| P_i^s)}.$$

$\hookrightarrow$ Relative entropy coding turns this quantity into an achievable code for the selected index.

Write $D_i := D_{\text{KL}}(P_i^t \| P_i^s)$, in bits. The expected number of bits to send the selection at step i is bounded by

$$\mathbb{E}[\ell(J_i)] \leq D_i + \log_2(1 + D_i) + 4.$$

$\hookrightarrow$ The $\log_2(1 + D_i) + 4$ terms are coding overhead; D_i is the leading term.

The receiver repeats this for $i = 0, \dots, M-1$ and reconstructs P_M^s :

$$\begin{aligned} |\mathbf{P}_{\text{req}}| &\lesssim \sum_{i=0}^{M-1} [D_i + \log_2(1 + D_i) + 4] + O(1) \\ &\approx \boxed{\sum_{i=0}^{M-1} D_{\text{KL}}(P_i^t \| P_i^s)}. \end{aligned}$$

$\hookrightarrow$ The approximation neglects overhead when it is small compared with the total code length.

This is an explicit code for the student model: send the index sequence, replay training, and recover P_M^s .

$\longrightarrow$ The original real examples and the teacher's parameters are never transmitted.

We can use the reconstructed student in the same two-part MDL objective:

$$\underbrace{\widehat{\text{MDL}}_T(X)}_{\text{total description}} \approx \underbrace{|\mathbf{P}_{\text{req}}|}_{\text{student model bits}} + \underbrace{\mathbb{E}_{X \sim Q}[-\log_2 P_M^s(X)]}_{\text{remaining data bits}}.$$

$\hookrightarrow$ Evaluate the final student on held-out data; include its reconstruction time in the budget T .

The teacher can change over time. Choosing P_i^t close to P_i^s keeps each KL small, while still teaching the student.

How are prequential and requential coding related?

Recall the prequential estimate from the previous section:

$$\begin{aligned} |\mathbf{P}_{\text{preq}}| &\approx \sum_{i=0}^{M-1} \left[\log_2 \frac{1}{P_i(Z_i)} - \log_2 \frac{1}{P_M(Z_i)} \right] \\ &= \sum_{i=0}^{M-1} \log_2 \frac{P_M(Z_i)}{P_i(Z_i)}. \end{aligned}$$

$\hookrightarrow$ Final model probability / original prediction probability.

Suppose, as an idealization, that each Z_i is sampled from the final model P_M . Then for one training step,

$$\begin{aligned} \mathbb{E}_{Z_i \sim P_M} \left[\log_2 \frac{P_M(Z_i)}{P_i(Z_i)} \right] \\ &= \sum_z P_M(z) \log_2 \frac{P_M(z)}{P_i(z)} \\ &= \boxed{D_{\text{KL}}(P_M \| P_i)}. \end{aligned}$$

So under this idealization, the expected prequential estimate becomes

$$\mathbb{E}[|\mathbf{P}_{\text{preq}}|] \approx \sum_{i=0}^{M-1} D_{\text{KL}}(P_M \| P_i).$$

But the leading requential code length is

$$|\mathbf{P}_{\text{req}}| \approx \sum_{i=0}^{M-1} D_{\text{KL}}(P_i^{\text{t}} \| P_i^{\text{s}}).$$

The two have the same form if we choose

$$\boxed{P_i^{\text{t}} = P_M \text{ (fixed teacher)}, \quad P_i^{\text{s}} = P_i.}$$

$\hookrightarrow$ This comparison assumes the training examples behave like samples from P_M and student training follows similar dynamics.

Prequential coding looks like requential coding with a fixed final-model teacher.

In practice, the requential teacher can change with i and stay close to the student:

$$P_0^{\text{t}} \longrightarrow P_1^{\text{t}} \longrightarrow \cdots, \quad P_i^{\text{s}} \text{ follows it.}$$

$\longrightarrow$ Smaller teacher–student gaps can give shorter model descriptions than a fixed teacher.

Prequential coding is convenient: estimate it from an ordinary training loss curve. Requential coding requires synthetic training, but constructs a model code with a known decoding procedure.

Now that we can estimate the model and data bits: what happens as compute and dataset size increase?