# IN-CONTEXT LEARNING AND OCCAM'S RAZOR

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A central goal of machine learning is generalization. While the No Free Lunch Theorem states that we cannot obtain theoretical guarantees for generalization without further assumptions, in practice we observe that *simple* models which explain the training data generalize best—a principle called *Occam's razor*. Despite the need for simple models, most current approaches in machine learning only minimize the training error, and at best indirectly promote simplicity through regularization or architecture design. Here, we draw a connection between Occam's razor and in-context learning—an emergent ability of certain sequence models like Transformers to learn at inference time from past observations in a sequence. In particular, we show that the next-token prediction loss used to train in-context learners is directly equivalent to a data compression technique called prequential coding, and that minimizing this loss amounts to jointly minimizing both the training error *and* the complexity of the model that was implicitly learned from context. Our theory and the empirical experiments we use to support it not only provide a normative account of in-context learning, but also elucidate the shortcomings of current in-context learning methods, suggesting ways in which they can be improved.

## 1 INTRODUCTION

The goal of machine learning (ML) is to learn models that generalize to unseen data. Longstanding theory shows that minimizing training error alone can lead to overfitting and poor generalization (Bishop & Nasrabadi, 2006). To enable better generalization, ML follows the principle of *Occam's razor*—the best explanation is the simplest one that explains the observations (Rathmanner & Hutter, 2011; Sunehag & Hutter, 2014; Hutter, 2010). The intuition is that simple rules that explain the data cannot simply memorize observations, and must instead capture more general patterns. Consequently, learning algorithms usually trade off low training error and low model complexity with *ad hoc* approaches (e.g., via regularization and inductive biases), motivating the need for notions of complexity that can be tractably minimized directly.

Although there exist mathematical notions of model complexity such as VC dimension or Kolmogorov complexity, these quantities cannot be directly minimized, or even tractably computed for the latter. In practice, we instead learn predictors that minimize training error as well as *proxies* of the model's complexity, such as the $L_1$ norm of the parameters, or rely on inductive biases for low-complexity solutions that are implicit in the model class and learning algorithm. Defying this trend, however, pretrained large language models (LLMs) have a surprising ability to rapidly learn and generalize from small amounts of data presented in their context (or *prompt*) (Radford et al., 2019). This ability called *in-context learning* (ICL) is typically explained through the lens of *memory-based meta-learning* (e.g., Xie et al., 2022; Chan et al., 2022), a theoretical framework where sequence models are explicitly trained to learn statistical models from sequences of observations.

The main contribution of this paper is to provide theoretical arguments linking ICL to Occam's razor and a preference for simple models. Briefly, our theory frames ICL as a meta-learning algorithm whose next-token prediction objective is directly equivalent to a powerful compression method called prequential coding (Blier & Ollivier, 2018). Given the relationship between optimal compression and Kolmogorov complexity, we show that the meta-objective in ICL is to find a learner capable of jointly minimizing both training error *and* model complexity across a diverse range of tasks. Our theory, along with the empirical experiments that we use to support it, explain why ICL has proven so effective in meta-learning settings, and also explain the shortcomings of current ICL methods.

Namely, we find that current methods produce learning algorithms which are susceptible to underfitting and can fail to generalize to novel tasks, suggesting principled avenues for future research.

## 2 Occam's razor and In-context learning

In this section, we introduce a meta-learning objective that directly targets simple models, and then show that it is equivalent to the next-token prediction objective underlying ICL. We reach this result via four key steps:

1. We begin by formalizing both training error and model simplicity through the lens of Kolmogorov complexity, which deals with optimal data and model compression.
2. We then show how learning algorithms can be used to compress data through a technique called prequential coding (Blier & Ollivier, 2018), and that minimizing the resulting "prequential code length" achieved by a learning algorithm is equivalent to jointly minimizing the training error and complexity of the model it fits.
3. We then introduce the idea of finding a learning algorithm that minimizes prequential code length by formalizing a meta-learning problem that appears difficult to optimize.
4. Finally, we show that the next-token prediction objective underlying ICL *already* solves this meta-learning problem in an efficient and scalable way.

### 2.1 Kolmogorov complexity and data compression

Kolmogorov complexity (Kolmogorov, 1965; Li et al., 2008) is a notion of information quantity. Intuitively, the Kolmogorov complexity $K(x)$ of an object $x$ is the length of the shortest program (in some programming language) that outputs $x$. A related notion is the conditional Kolmogorov complexity $K(x|y)$ of the object $x$ given another object $y$, which is the length of the shortest program that takes $y$ as input and outputs $x$. Finally, the Kolmogorov complexity of encoding two objects jointly is denoted $K(x, y)$. While quite abstract, this notion of complexity has deep ties to *compression*, making it intuitive as a measure of information quantity. The smaller and more "structured" an object is—regularity, patterns, rules, etc.—the more easily it can be described by a short program, correspondingly having lower Kolmogorov complexity. Although Kolmogorov complexity is very general—objects $x$, $y$ can be datasets, programs, models—it is intractable to compute. However, it can often be tractably estimated or bounded, as we will show below.

A quantity relevant to ML is the joint Kolmogorov complexity of a dataset $D = (d_1, ..., d_n)$ and of a generative model $p(d)$, where each sample $d_i \in \mathcal{D}$ is drawn *iid*:

$$K(D, p) = K(D|p) + K(p), \tag{1}$$

where $K(p)$ refers to the complexity of the model (i.e., the length of the shortest program that outputs function $p : \mathcal{D} \to \mathbb{R}^+$). This term is intractable to compute as it requires an enumeration over all programs that output $p$, but the conditional complexity $K(D|p)$ can be easily computed. According to (Grünwald, 2007), if the dataset is sufficiently large, the optimal method for compressing a data point $d_i$ uses only $-\log_2 p(d_i)$ bits (e.g., using an arithmetic coding scheme, Witten et al., 1987), as in the case of Shannon information (Shannon, 2001). As such, we have $K(D|p) = -\sum_D \log_2 p(d_i)$ which is the negative log-likelihood of the data under model $p(d)$, a commonly used objective function in ML. It follows that models which achieve lower error under this objective better compress data. We provide further background on Kolmogorov complexity in Appendix A.

As we are interested in model optimization, we henceforth consider parameterized models $p_\theta$ with parameters $\theta$. We denote a learning algorithm by a function $T : \mathcal{P}(\mathcal{D}) \to \Theta$, where $\mathcal{P}$ denotes the power-set, which maps a dataset $D$ to a model $p_{T(D)}$. Maximum likelihood training, which is the norm in ML, is a learning algorithm $T^{ml}$ which fits a model that best compresses the training data:

$$T^{ml}(D) = \arg\min_{\theta'} -\sum_{d \in D} \log_2 p_{\theta'}(d) = \arg\min_{\theta'} K(D|p_{\theta'}). \tag{2}$$

However, Occam's razor says that we also need simple models. Thus, we consider the learning algorithm $T^{oc}$, which defines "simple" via complexity:

$$T^{oc}(D) = \arg\min_{\theta'} \left[K(D|p_{\theta'}) + K(p_{\theta'})\right]. \tag{3}$$

In reality, $T^{oc}$ is intractable since $K(p_{\theta'})$ cannot be computed. In practice, maximum log-likelihood training $T^{ml}$ is often enhanced with regularizers (e.g., $L_2$) and inductive biases (e.g., restricting the model class) to implicitly favor low-complexity models that combat overfitting and improve generalization. For instance, deep neural networks (DNNs) trained through SGD tend to be biased towards simple solutions (Blier & Ollivier, 2018; Goldblum et al., 2023). However, existing regularizers at most amount to *indirect* methods that roughly penalize model complexity $K(p_\theta)$ along with training error. No known learning algorithm (which we will often call a "learner" for brevity) directly attempts to minimize Equation (3) as $T^{oc}$ would. In what follows, we introduce learners $T_\phi$ that have learnable parameters $\phi$, estimated via meta-optimization, to approximate the ideal learner $T^{oc}$.

### 2.2 Prequential coding

While a learner $T$ that adheres to Occam's razor and solves Equation (3) would improve generalization, it is difficult to design one in practice. Even if $K(p_\theta)$ could be computed efficiently, there is the further challenge of minimizing it. We will first describe an approach to the problem of estimating $K(p_\theta)$, and then consider the optimization problem in the next section.

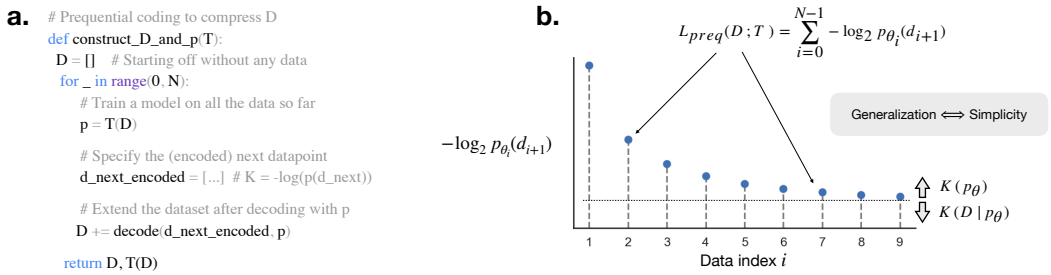


Figure 1: **Illustration of prequential coding, a method for estimating $K(D,\theta) = K(D|p_\theta) + K(p_\theta)$ using $p_\theta$'s learning algorithm $T$. a.** Pseudocode of the prequential coding program, which jointly compresses $D$ and $p_\theta$ by incrementally training a model using $T$ on increasingly more data. The primary contribution to total program length comes from specifying each next datapoint $d_{i+1}$ using the current model $p_{\theta_i}$, which takes $-\log_2 p_{\theta_i}(d_{i+1})$ bits. `decode()` is a short function that decodes an compressed object using arithmetic coding (Witten et al., 1987) **b.** A visual illustration of prequential coding. As the learner $T$ sees more data, it outputs models that assign a higher likelihood to new observations, and can thus better compress them. The total prequential code length $L_{preq}(D;T)$ is given by the area under the curve. The area underneath the curve's last point is equal to the complexity of the dataset given the final model, $K(D|p_\theta)$. Since $L_{preq}(D;T) = K(D|p_\theta) + K(p_\theta)$, the area above the curve's last point is equal to $K(p_\theta)$. Prequential coding formalizes the intuition that simple models generalize better from less data.

While $K(p_\theta)$ is difficult to measure directly, it turns out that we can estimate the joint complexity $K(D, p_\theta) = K(D|p_\theta) + K(p_\theta)$ using a compression algorithm called *prequential coding* (illustrated in Figure 1) that leverages the learner $T$ which gave $p_\theta$ (i.e., $p_\theta = T(D)$). Consider an ordering of *iid* datapoints $D = \{d_1, ..., d_N\}$, and denote $D_{1:i} = \{d_1, ..., d_i\}$. Prequential coding uses the learner $T$ to train models on increasing amounts of data. First, we train a model on just the first data point to get $p_{\theta_1} = T(d_1)$. Because the model is trained on a single datapoint, it will not be very accurate; however, it should be better than a random model that has seen no data at all. We can then use this model $p_{\theta_1}$ to compress the next (unseen) datapoint $d_2$, which takes $-\log_2 p_{\theta_1}(d_2)$ bits. At this point, we can train a new model $p_{\theta_2} = T(D_{1:2})$. Having seen more data, this model should assign a higher likelihood to a new datapoint $d_3$, which we can compress using $-\log_2 p_{\theta_2}(d_3)$ bits. This process repeats until the entire dataset has been covered. At this point, the model $p_\theta$ can be obtained simply by applying the learning algorithm to the complete dataset $p_\theta = T(D)$.

The total number of bits that it takes to jointly compress $D$ and $p_\theta$ using prequential coding is the sum of how many bits it takes to compress each datapoint using a model that was trained on all previous ones. Visually, it is the area under the *prequential coding curve* shown in Figure 1b. The length of this program is called the *prequential code length* $L_{preq}(D;T)$ (Blier & Ollivier, 2018):

$$L_{preq}(D;T) = \sum_{i=0}^{N-1} -\log_2 p_{\theta_i}(d_{i+1}) \;\geq\; K(D, p_\theta) = K(D|p_\theta) + K(p_\theta). \tag{4}$$

$L_{preq}(D;T)$ is an upper-bound on $K(D, p_\theta)$: prequential coding is *one* way to jointly compress the data and model, but it is not necessarily the optimal way. However, in Section 2.3 we will minimize this upper-bound with respect to the learner $T$, and thus minimize the joint data and model complexity $K(D, p_\theta)$.

Prequential coding relates Kolmogorov complexity to intuitions about generalization in ML: the simpler a model is, the quicker it generalizes from limited amounts of training data. Although the relationship in Equation (4) offers a promising way forward to operationalize the idealized learner $T^{oc}$, there is a problem. The prequential code length given by Equation (4) *conditions* on the choice of a learner $T$. However, prequential coding also requires us to encode the learning algorithm itself. When we take the description length of $T$ into account, the quantity $L_{preq}(D;T) + K(T)$ is an upper-bound on $K(D|p_\theta) + K(p_\theta)$ (see Appendix B). Since we will optimize for learners $T_\phi$ that minimize $L_{preq}(D;T)$, we will need to ensure that $T_\phi$ has low complexity.

### 2.3 Minimizing prequential code length through meta-learning

Consider a parameterized learner $T_\phi$ that minimizes the prequential code length $L_{preq}(D;T_\phi)$ of a dataset $D$. This objective upper-bounds the objective that the idealized learner $T^{oc}$ minimizes, but only when $K(T_\phi)$ is low. This second criteria is violated if $T_\phi$ overfits to a single dataset $D$. To forbid $T_\phi$ from memorizing a single dataset, we consider a meta-dataset $\mathscr{D} = \{D^1, ..., D^M\}$ coming from $M$ different tasks and meta-learn $T_\phi$ to minimize prequential code length on average across the meta-dataset $\mathscr{D}$. This allows us to write the following objective for the learner $T_\phi$:

$$\mathcal{L}(\mathscr{D};\phi) = \sum_{i=1}^{M} L_{preq}(D^i;T_\phi) \geq \sum_{i=1}^{M} K(D^i, p_\theta|T_\phi) \tag{5}$$

$$= \left[\sum_{i=1}^{M} K(D^i|p_\theta, T_\phi) + K(p_\theta|T_\phi)\right] \tag{6}$$

$$= \left[\sum_{i=1}^{M} K(D^i|p_\theta) + K(p_\theta|T_\phi)\right], \tag{7}$$

where $p_\theta = T_\phi(D^i)$, and the last line is obtained from noticing that all the relevant information about $D^i$ contained in $T_\phi$ is already encoded in the model $p_\theta = T_\phi(D^i)$.

By minimizing $\mathcal{L}(\mathscr{D};\phi) = \sum_{i=1}^{M} L_{preq}(D^i;T_\phi)$, we thus minimize an upper-bound on the training error $K(D^i|p_\theta)$ and model complexity given the learner $K(p_\theta|T_\phi)$ in expectation over datasets. This approach of minimizing an upper-bound on an objective is a common practice when dealing with intractable objectives, as in the case of the evidence-lower-bound (ELBO) in variational inference (Kingma, 2013). As a result, minimizing expected prequential code length in Equation (5) meta-trains a learner $T_{\phi^*} = \arg\min_\phi \mathcal{L}(\mathscr{D};\phi)$ which fits simple models that explain their training data. After obtaining $T_{\phi^*}$ through meta-training, the prequential code length of a new dataset of interest $D$ is then:

$$L_{preq}(D;T_{\phi^*}) \geq K(D, p_{\theta^*}|T_{\phi^*}) \tag{8}$$

$$= K(D|p_{\theta^*}, T_{\phi^*}) + K(p_{\theta^*}|T_{\phi^*}) \tag{9}$$

$$= K(D|p_{\theta^*}) + K(p_{\theta^*}|T_{\phi^*}). \tag{10}$$

Note that the learners $T_{\phi^*}$ and $T^{oc}$ ($= \arg\min_{\theta'} [K(D|p_{\theta'}) + K(p_{\theta'})]$) are not equivalent: $T^{oc}$ aims to minimize $K(p_{\theta'})$ directly whereas $T_{\phi^*}$ fits models that are simple *given* $T_{\phi^*}$ (i.e. low $K(p_\theta|T_\phi)$). Despite these differences, the two learners are deeply related. As a result of its meta-objective in Equation (5), the learner $T_{\phi^*}$ attempts to minimize training error across many datasets while fitting compressible models. The learner $T_{\phi^*}$ will succeed in doing this on a *novel* dataset $D$ when it *generalizes* to that novel dataset.

### 2.4 Training for ICL meta-learns a prequential code length minimizer

In practice, solving the meta-learning problem in Equation (5) involves several constraints:

1. The performance of $T_\phi(\cdot)$ must be evaluated w.r.t. a dataset's prequential code length.
2. $T_\phi(\cdot)$ must be fast to evaluate because it is iteratively called on multiple datasets.
3. To meta-optimize $\phi$, it must be easy to take gradients of $L_{preq}(\cdot; T_\phi)$ w.r.t. $\phi$.
4. $\phi$ must parameterize an expressive class of learning algorithms, capable of minimizing prequential code length on a broad distribution of tasks and generalizing to unseen ones.

While this may appear daunting, it turns out that these desiderata are readily addressed by ICL in probabilistic sequence models. Such models are trained to predict the distribution over the next element in a sequence given its past context: $F(d_t|D_{1:t-1})$. Crucially, the sequence model $F$ is *both* the learner $T_\phi$ and the inner model $p_\theta$. Indeed, $\phi$ corresponds to the parameters of the sequence model $F$ (e.g. weights in a Transfomer), and $\theta = T_\phi(D_{1:t-1})$ is encoded by the activations of hidden units in the model when presented with the context $D_{1:t-1}$. Thus, the predicted distribution over the next token is given by: $F(d_t|D_{1:t-1}) = p_{T_\phi(D_{1:t-1})}(d_t)$. The model is trained to minimize the cumulative next-token prediction error: $\mathcal{L}(D;\phi) = \sum_{t=1}^{N} -\log p_{T_\phi(D_{1:t-1})}(d_t)$, which corresponds exactly to the prequential code length in Equation (4).

The dual nature of the sequence model as both the learner and the learned model offers a natural solution to the constraints above, enabling fast and differentiable evaluation of $T_\phi(\cdot)$ (2 & 3 above) with respect to cumulative next-token prediction loss (1 above). Moreover, modern sequence models can parameterize a rich class of learning algorithms, which is crucial to minimizing Equation (5) (4 above). Notably, architectures such as Transformers are known to have components which make them especially good meta-learners, such as multi-head attention (Olsson et al., 2022). It is thus no surprise that sequence models are leveraged in settings outside of the language domain (Von Oswald et al., 2023a; Bauer et al., 2023; Kirsch et al., 2022), making them general-purpose meta-learners.

This predictive formulation is quite flexible as it can be used to model data which contains sequential correlations, such as language, but can also be used to process any *iid* dataset. Indeed, consider $D = \{(x_1, y_1), ..., (x_T, y_T)\}$ and the supervised task of learning a function $y = f(x)$. In this setting, a data point is given by the pair $d_t = (x_t, y_t)$, and straightforward tokenization schemes can be used to append a novel query $x^*$ to the context $D$ such that the predicted output $\hat{y}^*$ is given by the next token in the sequence. This ICL setup is well-suited for regression-type tasks (see e.g. (see e.g., Von Oswald et al., 2023a;b)) but can be used for most supervised tasks. ICL thus turns the training of a sequence model into a meta-optimization problem over datasets—an approach also called *memory-based* meta-learning (Hochreiter et al., 2001; Santoro et al., 2016; Ortega et al., 2019). It is assumed here that $(x_t, y_t)$ are *iid*. Although pretrained LLMs that can execute tasks with instructions given via context (or prompt) (Radford et al., 2019) break this *iid* data assumption, prequential code length is well-defined over arbitrary sequences, and our theory can possibly be adapted to settings with non-stationary data. Further exploration of this topic is left for future work.

**Summary.** We showed that sequence models trained on cumulative next-token prediction losses explicitly optimize a meta-learning objective that jointly minimizes training error and model complexity. This provides a normative account of ICL in terms of Occam's razor, and explains recent experimental findings showing that LLMs are good universal compressors (Delétang et al., 2023).

## 3 Experiments

Our experiments are designed to illustrate the benefits of ICL in terms of fitting simple models that generalize on *iid* examples. In Section 3.1, we compare ICL's standard next-token prediction objective to an alternative that minimizes training error alone, rather than prequential code length. Section 3.2 then compares ICL to standard gradient-based learners that minimize training error, such as SGD. Section F.2 shows the impact of regularization on gradient-based learners from a compression perspective. In Section 3.3, we explore the impact of learner $T_\phi$'s architecture on prequential code length minimization. Section Section 3.4 explores the ability of $T_\phi$ to generalize to novel tasks. Finally, in Section 3.5 we use insights from our theory to control the data distribution seen by $T_\phi$ in order to better minimize prequential code length. Experimental details not described in the main paper (e.g., precise architectures, hyperparameters, etc.) can be found in Appendix F.

**Tasks.** In line with similar work studying ICL in a controlled setting (Mahankali et al., 2023; Garg et al., 2023; Akyürek et al., 2023), we use synthetically-generated tasks. Each task consists of a supervised learning dataset $D^i = \{(x_1, y_1), ..., (x_k, y_k)\}$, where the labels are a (potentially

stochastic) function of the input $y_j = f^i(x_j, \epsilon_j)$. ICL learners $T_\phi$ are trained on a meta-dataset $\mathscr{D} = \{D^1, ..., D^N\}$, where each $D^i$ is associated with a different ground-truth data-generating function $f^i$. We primarily study three meta-datasets: **(1) Linear regression** problems where $x \in \mathbb{R}^3$ and $y \in \mathbb{R}$. The ground-truth functions $f^i$ are noisy linear mappings $y_j = W^i x_j + b^i + \epsilon_j$, where each $\{W^i, b^i\}$ is sampled from a standard Normal distribution and $\epsilon_j$ is Gaussian noise with $\sigma^2 = 0.04$. **(2) Sinusoidal regression** problems where $x_j \in \mathbb{R}$ and functions $f^i$ are linear combinations $y_j = \sum_{l=1}^{L} \alpha^{i,l} \sin(\omega^l x_j)$. We use $L = 3$ with frequencies $\omega^l \sim U(0, 5)$ that are shared across tasks, varying only the amplitudes $\alpha_{i,l} \sim \mathcal{N}(0, 1)$. **(3) Mastermind**: a multi-label classification problem inspired by the code-breaking game *Mastermind*. Each $f^i$ is associated with an underlying discrete code (a fixed-size sequence of digits) that needs to be inferred from random guesses that return partial information. The inputs $x_j$ are random guesses for the code, and $y_j$ is a tuple of two class labels where the first specifies the number of digits in $x_j$ that are correct in terms of both position and value, and the second label specifies the number of digits that are correct in value but not necessarily position. We use randomly sampled codes of length 8 with digits varying from $1..6$.

### 3.1 Comparisons to in-context learning with a train-risk objective

We have argued that standard ICL can be seen as a meta-learning method who's meta-objective is to minimize training error and model complexity through cumulative next-token prediction (prequential code length). However, this is not the only meta-objective that one could design for ICL. In particular, we can design an alternative meta-objective that minimizes *only* training error simply by training $T_\phi$ to predict *past* datapoints in the context rather than future unseen ones. In both cases, the learner $T_\phi$ is some function that takes a context (i.e., a partial dataset) as input, and outputs a model $p_\theta$ capable of making predictions for arbitrary datapoints. For supervised learning, this can be represented as $\hat{y}_q = T_\phi((x, y)_{1:j}, x_q)$ where $(x, y)_{1:j}$ corresponds to an observed context, $x_q$ is the queried input, and the model $p_\theta$ is implicitly encoded in $T_\phi$'s weights and latent activations given the context. In standard ICL (which we will refer to as *prequential ICL*), the query $x_q$ is a novel input that does not appear in the context. In the alternative form of ICL (which we will call *train-risk ICL*), the query $x_q$ is a randomly-selected input that appeared previously in the context $x_{1:j}$. Note the similarities of train-risk ICL to standard objectives of learners that minimize training error: it processes some fixed-sized training set (here a context) and attempts to minimize the empirical risk on a subset of that very same data (here a single query that appeared in the context). While nobody uses train-risk ICL in practice, it serves as an ideal control to illustrate our theory of ICL and the generalization benefits of minimizing prequential code length as opposed to only training error. One can use an identical architecture for $T_\phi$ in both cases and train using precisely the same methodology and loss function; the only difference is which query the loss function is evaluated on.

In our experiments, we parameterize $T_\phi$ using a Transformer. For the train-risk case, a standard Transformer could simply attend to the context position that matches $x_q$ and retrieve the corresponding label. To prevent this trivial solution, we instead use a bottlenecked architecture for $T_\phi$ described in Mittal et al. (2024). In this architecture, a Transformer first summarizes the context into a low-dimensional vector $z = \texttt{Transformer}_\phi((x, y)_{1:j})$, and a separate prediction head—here a multi-layer perceptron (MLP)—subsequently outputs a prediction for the query $\hat{y}_q = \texttt{MLP}_\phi(x_q, z)$. For fair comparison, we use the same bottleneck architecture for train-risk ICL and prequential ICL in all experiments, unless otherwise stated. Figure 2a shows our comparisons between prequential ICL to train-risk ICL, where we plot the prequential coding curves for each ICL method after loss convergence on a meta-dataset. The curves are constructed at inference time by evaluating the average *iid* generalization error (i.e., unseen next-token prediction loss) on *unseen* tasks from the meta-dataset, for varying context lengths.

**Findings.** Two findings follow directly from our theory. The first is that for large context lengths, generalization error is identical for both prequential ICL and train-risk ICL. This is because with significant data, overfitting is less likely to occur, even when minimizing training error alone. The benefits of simple models are instead expected to be most prominent in *low-data* regimes where generalization is difficult, and this is precisely what we observe. Across all tasks, prequential ICL consistently outperforms train-risk ICL in terms of generalization for short context lengths, and this performance gap extends further the more difficult the task (e.g., it is small for linear regression, and larger for sinusoid regression and mastermind). We confirm that the performance gap widens with increasing task difficulty by fixing the function class and increasing the dimensionality of the inputs $x$ in Appendix C, which is expected given that harder tasks require more data for generalization.

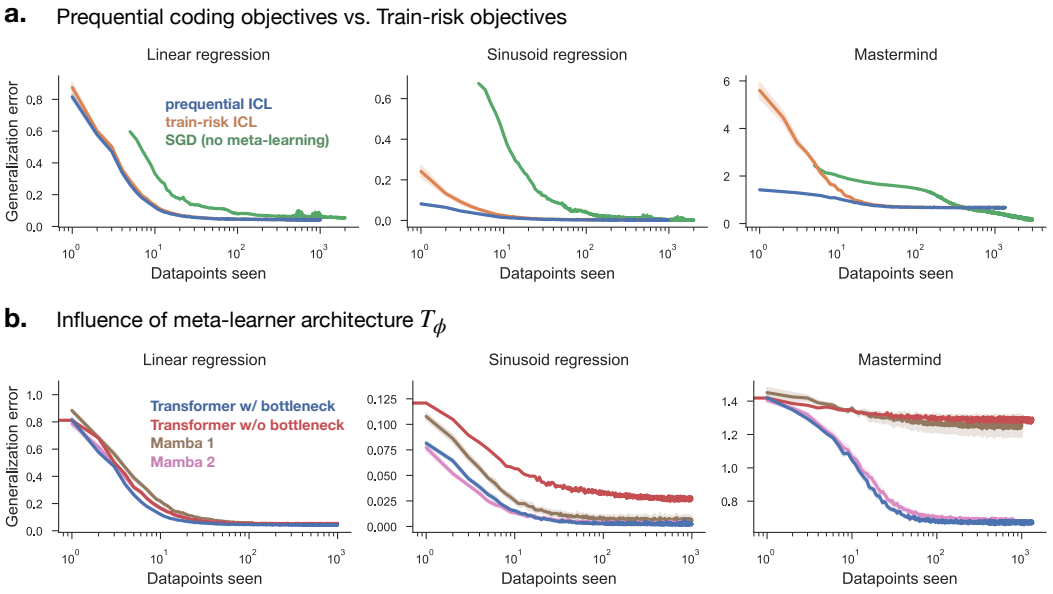


Figure 2: **Experimental results comparing different learners.** Figures show average prequential coding curves for a meta-dataset, which is the mean prediction error on unseen data (generalization error, y-axis) given observed contexts of increasing length (datapoints seen, x-axis). The area underneath these curves corresponds to prequential code length. Error is measured using MSE for linear and sinusoid regression and cross-entropy for Mastermind. Error bars show standard error across seeds (5 for ICL, 15 for SGD). **a.** ICL from next-token prediction objectives (prequential ICL, blue) yields lower prequential code lengths than ICL from past-token prediction objectives (train-risk ICL, orange), with greater effects in low-data regimes. An SGD-based learner (green) fits more complex models than prequential ICL and performs poorly in low-data regimes, but can generalize better in large-data regimes on a difficult Mastermind task due to underfitting in ICL. **b.** The architecture used to parameterize $T_\phi$ has substantial influence on ICL's ability to minimize prequential code length.

### 3.2 Comparisons to traditional gradient-based learners

We next consider whether there are empirical advantages of meta-learning a learner $T_\phi$ to minimize prequential code length through ICL, compared to using standard out-of-the-box learning algorithms. In particular, we know that traditional SGD-based learners can optimize DNN models that generalize well across a wide range of tasks, despite only explicitly minimizing training error. We consider a standard SGD-based learner that fits a randomly-initialized MLP to the training set until validation loss converges. We repeatedly sample a dataset from our meta-dataset, truncate it to a specified number of observed datapoints, apply the SGD-based learner to the truncated dataset, and evaluate the resulting model's generalization error on new datapoints.

**Findings.** Figure 2a compares this SGD-based learner to prequential (and train-risk) ICL learners. Across all tasks, the models obtained through ICL generalize better in low-data regimes as a result of directly minimizing model complexity. With enough training data, however, models obtained through the SGD-based learner generalize just as well. In fact, on the Mastermind task, SGD performs *better* in large-data regimes. This result demonstrates that even though the next-token prediction objective in ICL is well-motivated from a theoretical perspective, the degree to which that objective can successfully be minimized strongly depends on the architecture of $T_\phi$ and the methods used to train it. For instance, when $T_\phi$ is a Transformer, the expressivity of the model it implicitly fits to the context scales with the number of activations in the network ($N$), whereas the expressivity of a DNN trained through SGD scales with the number of weights ($N^2$). Furthermore, the amount of compute that $T_\phi$ uses to fit the context amounts to one forward pass of a network, whereas the amount of compute that goes into fitting a dataset using SGD can be arbitrarily large.

### 3.3 Influence of the in-context learning architecture

The previous section argued that the structure of $T_\phi$ can influence its ability to minimize prequential code length. In this section, we further illustrate this point by considering a wider breadth of neural

architectures for $T_\phi$. Since state-space models (SSMs) have recently been shown to exhibit ICL (Lu et al., 2024), we test Mamba 1 (Gu & Dao, 2023) and Mamba 2 (Dao & Gu, 2024). We also test a standard causal decoder Transformer in addition to the bottlenecked Transformer from previous sections. Prequential code length comparisons in Figure 2b show that the architecture for $T_\phi$ indeed plays a substantial role, with the bottlenecked Transformer and Mamba 2 performing best across our tasks. We find that not only the architecture but also the tokenization scheme in each of these architectures significantly affect their performance, which we discuss in Appendix E. Analyzing why this is the case is out of scope for this work; we only intend to show that having a next-token prediction objective alone does not guarantee that prequential code length can successfully be minimized in practice through ICL.

### 3.4 Large pretrained models

A core element of our theory of ICL is that $T_\phi$ is trained to minimize average prequential code length on a meta-dataset $\mathscr{D}$. There is no guarantee, however, that prequential code length will be small on a novel dataset $D$ that was unseen at training time: this depends on the generalization abilities of the learner $T_\phi$. In this section, we look at the task-generalization abilities of a large pretrained LLM (GPT-4 Achiam et al., 2023) on the Mastermind task. We do this by prompting the LLM with a description of the task and a number of in-context examples, then obtaining the logits and prediction error for a novel example. In Figure 3a, we find that despite its massive pretraining across a breadth of tasks, the LLM is unable to meaningfully minimize prequential code length on Mastermind. Not only is its prequential code length substantially higher than for a much smaller model trained on a distribution of Mastermind tasks, but it is also higher than for a naive baseline that just predicts the empirical marginal distribution over class labels in the context. These results demonstrate that even when the size of the model and meta-dataset used to train $T_\phi$ are scaled significantly, current methods for ICL can still struggle to minimize prequential code length on a novel task.

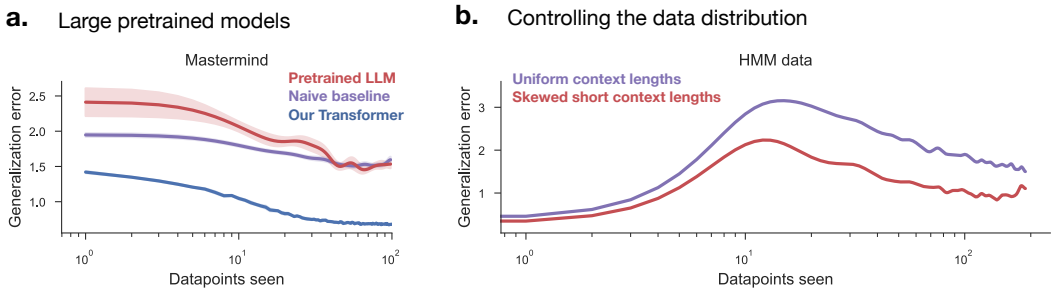


Figure 3: **Experimental results for LLM and data manipulation strategies.** Figures show average prequential coding curves for a meta-dataset, which is the mean prediction error on unseen data (generalization error, y-axis) given observed contexts of increasing length (datapoints seen, x-axis). The area underneath these curves corresponds to prequential code length. Error bars show standard error across 5 seeds. **a.** An LLM (GPT-4, red) fails to meaningfully minimize prequential code length on a novel Mastermind task, performing far worse than small ICL models trained on a distribution of Mastermind tasks (blue) and a naive baseline that predicts the marginal class distribution over the context (purple). Error is measured using cross-entropy. **b.** On a synthetic HMM dataset designed to mimic natural language, preferentially training on shorter contexts (red) yields lower prequential code lengths than training uniformly over context lengths (purple). Error is measured using reverse KL divergence between model and oracle conditioned on seen context.

### 3.5 Improving ICL by controlling the data distribution

In addition to improving architectures used for $T_\phi$ or scaling the diversity of tasks on which it is trained, a complementary approach is to manipulate the distribution of data presented in-context at training time. This approach can be especially useful in non-*iid* settings; for instance Chan et al. (2022) found that in order for ICL to emerge in an image classification setting, the distribution over classes needed to be "bursty", or Zipfian. In this section, we consider a simple manipulation of the data distribution that is inspired by our theory, with a particular focus on improving ICL

in language-like data modalities relevant to LLMs. In prequential coding, model complexity is related to the speed of convergence in generalization error as context length increases. We might therefore be able to further bias ICL towards simple models by sampling *short* contexts, such that downstream prediction errors on larger context lengths (after which the prequential coding curve has already converged) do not disproportionately dominate the loss.

We attempt this on synthetically-generated data from Hidden Markov Models (HMMs) that were designed to mimic the statistical properties of natural language in a simplified and controlled setting (see Appendix F for details). Briefly, we generate a family of HMMs parameterized by compositional latent attributes and train a Transformer to predict the next observation in a sequence. The model is evaluated on unseen HMMs with novel compositions of latents. Our results, presented in Figure 3b, show that this data-manipulation strategy is effective. Generalization error is lower when preferentially training on short context lengths, with the gap narrowing the more tokens are seen during training as shown in Figure F.3. Surprisingly, biasing the data distribution in this way not only decreases generalization error for short context lengths, but also for long ones. In general, these results show how our theory can lead to practical improvements for ICL, where we look at prequential coding curves and compression ability to guide method design.

## 4 Related work

**Sequence modeling and compression.** The idea that probabilistic models can be used to efficiently compress data is a topic widely studied in machine learning across different modalities and settings (Ollivier, 2015; Delétang et al., 2023; Blier & Ollivier, 2018; Veness et al., 2014), specifically in sequence modeling (Goyal et al., 2018; Valmeekam et al., 2023; Delétang et al., 2023) due to its close similarities to prequential coding (Blier & Ollivier, 2018). In this area, the generic sequence modeling capabilities of certain foundation models are crucial for defining effective "universal" compressors. While Goyal et al. (2018) and Valmeekam et al. (2023) claim that learned sequence models can outperform simple compressors like JPEG or gzip, they overlook model complexity in their analysis, adhering strictly to Shannon's notion of compression. In contrast, more recent studies from Delétang et al. (2023) and Bornschein et al. (2022) opted for the Kolmogorov approach, incorporating model size to account for model complexity. Delétang et al. (2023), in particular, add nuance to the claimed advantages of foundation models due to the substantial memory allocation required to store their weights. Our theory builds on these works by relating compression and sequence modeling to the approach of meta-learning across tasks using ICL, which we show yields simple models that adhere to Occam's razor.

**In-context learning as Bayes-optimal prediction.** One of the dominant perspectives of ICL and related meta-learning approaches is that they yield Bayes-optimal learners (Ortega et al., 2019; Mikulik et al., 2020; Müller et al., 2021; Hollmann et al., 2022; Binz et al., 2023; Wang et al., 2024), in the sense that they learn a prior distribution over tasks during training, and then compute a posterior given data presented in-context at inference time. This posterior can then be used to make predictions with minimum Bayes' risk. Various studies have tested this in controlled settings with tractable posteriors (Xie et al., 2022; Panwar et al., 2024; Genewein et al., 2023; Mittal et al., 2023). Xie et al. (2022) assume a *concept* latent that parameterizes the generation of dependent samples through a Hidden Markov Model (HMM) and provide formal conditions for ICL to effectively approximate the Bayes-optimal predictor on the prompt, specifically, requiring the pretraining distribution to be structured similarly to a HMM. In a supervised fashion, Akyürek et al. (2023) construct sequence of labeled examples $(x, f(x))$ and shows that under uncertainty, ICL behaves as the Bayes-optimal predictor on noisy linear regression. Additionally, they argue that with limited capacity, ICL does not necessarily match the Bayes predictor but can meta-learn other learning algorithms, such as gradient-based algorithms on linear models and closed-form ridge regressors (Panwar et al., 2024). Grau-Moya et al. (2024) induce a prior for model simplicity in ICL by generating tasks from short programs run on Universal Turning Machines. Finally, (Raventós et al., 2024) find that under a sufficiently diverse set of pretraining tasks, ICL does *not* yield Bayes-optimal predictors, but instead infers a more uniform prior. While the Bayesian perspective of ICL is very useful and complementary to the Kolmogorov one that we have proposed, we argue in Appendix D that the Kolmogorov perspective generalizes the Bayesian one and more easily accounts for diverse findings in ICL (e.g., cases where ICL does not yield Bayes-optimal predictors).

**In-context learning as a direct meta-learned optimizer.** Elaborating on the possibility that ICL emulates non-Bayesian learning algorithms, Von Oswald et al. (2023a) show that $k$-layer linear Transformers with a specific weight parameterization can mimic $k$ steps of gradient descent for a least squares loss. Ahn et al. (2023) provide a theoretical foundation for these observations, provably showing that the optimization of the parameters of a linear Transformer under certain assumptions about the data distribution effectively implements this learning algorithm. Concurrent studies by Zhang et al. (2023) and Mahankali et al. (2023) report similar findings, albeit under slightly different assumptions regarding weight initialization or data generation processes. Beyond the scope of linear regression, Kirsch et al. (2022) explore this phenomenon on augmented natural data (MNIST, CIFAR10) and provide insightful empirical conditions for the emergence of ICL as a general-purpose learning algorithm. Other works empirically show that Transformers can learn more complex function classes in-context, such as sinusoidal regression (Von Oswald et al., 2023a), decision trees (Garg et al., 2023), and RASP-programmable functions (Zhou et al., 2023). While prior works such as these attest to the powerful meta-learning capabilities of ICL, our work differs in that it identifies the precise meta-*objective* as an implementation of Occam's razor.

## 5 Discussion and Future Work

In this work, we introduced novel theoretical arguments linking ICL and the next-token prediction objective to Occam's razor. Our theory provides a normative account of the strong generalization abilities of in-context learners at inference time, especially in low-data regimes when compared to traditional optimizers. These theoretical insights were supported by a number of empirical experiments, some of which also identified shortcomings of current methods for ICL that should be addressed in future work.

One such shortcoming is that models learned through current ICL methods can underfit data presented in-context, and that this can hamper generalization in large-data regimes on difficult tasks. We also found that the degree of underfitting was highly dependent on the architecture used to parameterize the in-context learner (i.e., the sequence model), and that commonly used architectures such as causal decoder Transformers can often underperform—a finding corroborated by Ding et al. (2024). In light of this, we hypothesize that ICL can be improved through the design of novel sequence model architectures that explicitly target prequential code length. For example, current methods learn in-context through a single forward pass of a sequence model with fixed layer depth. In contrast, DNNs can be trained using gradient-based methods until training loss converges, which can take weeks and substantial compute. One improvement to ICL might therefore be to augment current sequence model architectures with "layers" that use built-in optimization primitives with variable compute budgets, as was done in Von Oswald et al. (2023b). Another promising approach is to combine ICL and SGD through a "mixture of learners" that reaps their complementary benefits. ICL is sample-efficient and generalizes well in low-data regimes, while SGD-based methods that optimize the weights of a DNN excel on difficult tasks when significant training data is available. Recent work by Bornschein et al. (2024) explored a simple method for combining both learners by presenting a smaller number of *recent* tokens in-context to a sequence model for ICL, while at the same time using a large number of earlier tokens to fine-tune the weights of the sequence model using gradient methods, finding significant performance gains.

Another challenge of ICL that follows directly from our theory is that the in-context learner must generalize to novel tasks and datasets. While we found that task generalization was successful over narrow task distributions (e.g. a distribution of linear regression tasks), we also found that task generalization was more difficult in open-ended cases, in which even a large pretrained LLM was unable to learn in-context on a novel task that was easily solved by a small MLP trained using SGD. One possible path forward is to have many domain-specific in-context learners that each specialize in compressing data from a given task distribution. Another option is to learn *simple learners* that are more likely to generalize to novel tasks, which could be achieved through inductive biases, regularization, or, intriguingly, through an additional meta-layer of ICL at the task level that would minimize the Kolmogorov complexity of the learner itself (and not only the model it fits).

Finally, our work only provides a theoretical framework for ICL on *iid* data. Relaxing these *iid* assumptions opens up two avenues for future work: connecting ICL to generalization on out-of-distribution samples, and studying the effect of nonstationary data presented in context, as is the case in language and the HMM experiment presented here.

## REFERENCES


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning, 2023. URL https://arxiv.org/abs/2306.00297.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=0g0X4H8yN4I.

Jakob Bauer, Kate Baumli, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, et al. Human-timescale adaptation in an open-ended task space. In *International Conference on Machine Learning*, pp. 1887–1935. PMLR, 2023.

Marcel Binz, Ishita Dasgupta, Akshay K Jagadish, Matthew Botvinick, Jane X Wang, and Eric Schulz. Meta-learned models of cognition. *Behavioral and Brain Sciences*, pp. 1–38, 2023.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Léonard Blier and Yann Ollivier. The description length of deep learning models. *Advances in Neural Information Processing Systems*, 31, 2018.

Jorg Bornschein, Yazhe Li, and Marcus Hutter. Sequential learning of neural networks for prequential mdl. *arXiv preprint arXiv:2210.07931*, 2022.

Jorg Bornschein, Yazhe Li, and Amal Rannen-Triki. Transformers for supervised online continual learning. *arXiv preprint arXiv:2403.01554*, 2024.

Gregory J Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, et al. Language modeling is compression. *arXiv preprint arXiv:2309.10668*, 2023.

D. Deutsch. *The Beginning of Infinity: Explanations that Transform the World*. Always learning. Penguin Books, 2012. ISBN 9780140278163. URL https://books.google.ca/books?id=WFZl7YvsiuIC.

Nan Ding, Tomer Levinboim, Jialin Wu, Sebastian Goodman, and Radu Soricut. CausalLM is not optimal for in-context learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=guRNebwZBb.

Lance Fortnow. Kolmogorov complexity. In *Aspects of Complexity, Minicourses in Algorithmics, Complexity, and Computational Algebra, NZMRI Mathematics Summer Meeting, Kaikoura, New Zealand*, pp. 73–86, 2000.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes, 2023. URL https://arxiv.org/abs/2208.01066.

Tim Genewein, Grégoire Delétang, Anian Ruoss, Li Kevin Wenliang, Elliot Catt, Vincent Dutordoir, Jordi Grau-Moya, Laurent Orseau, Marcus Hutter, and Joel Veness. Memory-based meta-learning on non-stationary distributions. In *International conference on machine learning*, pp. 11173–11195. PMLR, 2023.

Micah Goldblum, Marc Finzi, Keefer Rowan, and Andrew Gordon Wilson. The no free lunch theorem, kolmogorov complexity, and the role of inductive biases in machine learning. *arXiv preprint arXiv:2304.05366*, 2023.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Deepzip: Lossless data compression using recurrent neural networks, 2018. URL https://arxiv.org/abs/1811.08162.

Jordi Grau-Moya, Tim Genewein, Marcus Hutter, Laurent Orseau, Grégoire Delétang, Elliot Catt, Anian Ruoss, Li Kevin Wenliang, Christopher Mattern, Matthew Aitchison, et al. Learning universal predictors. *arXiv preprint arXiv:2401.14953*, 2024.

Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

Peter D Grünwald and Paul MB Vitányi. Kolmogorov complexity and information theory. with an interpretation in terms of questions and answers. *Journal of Logic, Language and Information*, 12:497–529, 2003.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*, pp. 87–94. Springer, 2001.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

Marcus Hutter. A complete theory of everything (will be subjective). *Algorithms*, 3(4):329–350, 2010. ISSN 1999-4893. doi: 10.3390/a3040329. URL http://arxiv.org/abs/0912.5434.

Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458*, 2022.

Andrei N Kolmogorov. Three approaches to the quantitative definition of information'. *Problems of information transmission*, 1(1):1–7, 1965.

Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Arvind Mahankali, Tatsunori B. Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention, 2023. URL https://arxiv.org/abs/2307.03576.

Vladimir Mikulik, Grégoire Delétang, Tom McGrath, Tim Genewein, Miljan Martic, Shane Legg, and Pedro Ortega. Meta-trained agents implement bayes-optimal agents. *Advances in neural information processing systems*, 33:18691–18703, 2020.

Sarthak Mittal, Niels Leif Bracher, Guillaume Lajoie, Priyank Jaini, and Marcus A Brubaker. Exploring exchangeable dataset amortization for bayesian posterior inference. In *ICML 2023 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*, 2023.

Sarthak Mittal, Eric Elmoznino, Leo Gagnon, Sangnie Bhardwaj, Dhanya Sridhar, and Guillaume Lajoie. Does learning the right latent variables necessarily improve in-context learning?, May 2024. URL http://arxiv.org/abs/2405.19162. arXiv:2405.19162 [cs].

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

Yann Ollivier. Auto-encoders: reconstruction versus compression, 2015. URL https://arxiv.org/abs/1403.7752.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

Pedro A Ortega, Jane X Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alex Pritzel, Pablo Sprechmann, et al. Meta-learning of sequential strategies. *arXiv preprint arXiv:1905.03030*, 2019.

Madhur Panwar, Kabir Ahuja, and Navin Goyal. In-context learning through the bayesian prism, 2024. URL https://arxiv.org/abs/2306.04891.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011. ISSN 1099-4300. doi: 10.3390/e13061076. URL http://arxiv.org/abs/1105.5721.

Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. *Advances in Neural Information Processing Systems*, 36, 2024.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850. PMLR, 2016.

Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1): 1–22, 1964.

Peter Sunehag and Marcus Hutter. Intelligence as inference or forcing Occam on the world. In *Proc. 7th Conf. on Artificial General Intelligence (AGI'14)*, volume 8598 of *LNAI*, pp. 186–195, Quebec City, Canada, 2014. Springer. ISBN 978-3-319-09273-7. doi: 10.1007/978-3-319-09274-4_18.

Chandra Shekhara Kaushik Valmeekam, Krishna Narayanan, Dileep Kalathil, Jean-Francois Chamberland, and Srinivas Shakkottai. Llmzip: Lossless text compression using large language models, 2023. URL https://arxiv.org/abs/2306.04050.

Joel Veness, Marc G. Bellemare, Marcus Hutter, Alvin Chua, and Guillaume Desjardins. Compress and control, 2014. URL https://arxiv.org/abs/1411.5326.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023a.

Johannes Von Oswald, Eyvind Niklasson, Maximilian Schlegel, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, Razvan Pascanu, et al. Uncovering mesa-optimization algorithms in transformers. *arXiv preprint arXiv:2309.05858*, 2023b.

Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, and William Yang Wang. Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning, 2024. URL https://arxiv.org/abs/2301.11916.

Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference, 2022. URL https://arxiv.org/abs/2111.02080.

Ruiqi Zhang, Spencer Frei, and Peter L. Bartlett. Trained transformers learn linear models in-context, 2023. URL https://arxiv.org/abs/2306.09927.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What Algorithms can Transformers Learn? A Study in Length Generalization, October 2023. URL http://arxiv.org/abs/2310.16028. arXiv:2310.16028 [cs, stat].

# APPENDIX A BACKGROUND ON KOLMOGOROV COMPLEXITY

Kolmogorov complexity was independently developed in the 1960s by Kolmogorov (1965), Solomonoff (1964), and Chaitin (1966), and defines a notion of "information quantity".

Intuitively, the Kolmogorov complexity of an object is the length of the shortest program (in some programming language) that outputs that object. Specifically, given some finite string $x$, $K(x)$ is the length $l(r)$ (in bits) of the shortest binary program $r$ that prints $x$ and halts. Let $U$ be a universal Turing machine that executes these programs. The Kolmogorov complexity of $x$ is then:

$$K(x) = \min_r \{l(r) : U(r) = x, r \in \{0,1\}^*\}, \tag{11}$$

where $\{0,1\}^*$ denotes the space of finite binary strings. A related notion is the conditional Kolmogorov complexity of a string $x$ given another string $y$, which is the length of the shortest program that takes $y$ as input and outputs $x$:

$$K(x|y) = \min_r \{l(r) : U(r(y)) = z, r \in \{0,1\}^*\}, \tag{12}$$

where $r(y)$ denotes a program taking $y$ as input. Finally, we can also define a "joint" Kolmogorov complexity $K(x, y)$, which denotes the length of the shortest program that jointly outputs both $x$ and $y$. Surprisingly, joint Kolmogorov complexity is related to conditional Kolmogorov complexity (up to an additive logarithmic term, which we will ignore) by the Symmetry of Information theorem (Li et al., 2008):

$$K(x, y) = K(y|x) + K(x) = K(x|y) + K(y). \tag{13}$$

Kolmogorov complexity has many intuitive properties that make it attractive as a measure of information quantity, and although it is less common than notions from Shannon information theory (Shannon, 2001), it is strictly more general (as we will show later below). The smaller and the more "structure" an object has—regularity, patterns, rules, etc.—the more easily it can be described by a short program and the lower its Kolmogorov complexity. Kolmogorov complexity therefore is deeply rooted in the idea of compression. For instance, a sequence with repeating patterns or a dataset that spans a low-dimensional subspace can be significantly compressed relative to its original size, and this results in low Kolmogorov complexity. In contrast, a random string devoid of any structure cannot be compressed at all and must in effect be "hard-coded", making its Kolmogorov complexity equal to its original size in bits.

While powerful, Kolmogorov complexity has certain limitations. First and foremost, Kolmogorov is intractable to compute exactly because it requires a brute force search over an exponentially large space of possible programs. It is therefore often of conceptual rather than practical value, although it can nevertheless be upper-bounded using more efficient compression strategies. Second, Kolmogorov complexity depends on the programming language of choice. For instance, if a programming language has a built-in primitive for the object being encoded, Kolmogorov complexity is trivially small. This concern, however, is often overblown: given any two Turing-complete programming languages, the difference in Kolmogorov complexity that they assign to an object is upper-bounded by a constant that is independent of the object itself, because any Turing-complete programming language can simulate another (Grünwald & Vitányi, 2003; Fortnow, 2000). In practice, we can simply consider "reasonable" Turing-complete programming languages that don't contain arbitrary object-specific primitives, in which case this simulation constant will be relatively small and the particular programming language of choice will have little effect. Finally, Kolmogorov complexity is only defined for discrete objects because no terminating program can output a continuous number with infinite precision. This concern is also less consequential in practice, because we can always represent continuous objects using finite (e.g., floating-point) precision.

**Important properties for machine learning** In ML, we are often concerned with datasets and probabilistic models. Kolmogorov complexity relates to these two concepts in several interesting

ways. First, we can ask about the Kolmogorov complexity of a finite dataset $X = (x_1, ..., x_n)$ where each sample is drawn *iid* from a distribution $p(x)$. It turns out that if we have access to the true distribution $p(x)$, optimal algorithms such as arithmetic coding (Witten et al., 1987) can encode each sample using only $\log_2 p(x_i)$ bits. Intuitively, this is because samples that occur more frequently can be encoded using shorter codes in order to achieve an overall better compression. We thus have that:

$$K(X|p) = -\sum_{i=1}^{n} \log_2 p(x_i). \tag{14}$$

If instead of access to the true distribution $p(x)$ we only have a probabilistic model of the data $p_\theta(x)$, we have that:

$$K(X|p) \leq K(X|p_\theta) \leq -\sum_{i=1}^{n} \log_2 p_\theta(x_i), \tag{15}$$

where we have equality on the LHS when $p_\theta = p$ and equality on the RHS when the cost of improving $p_\theta$ (in bits of written code) would be greater than the benefits from more accurate modeling. In practice, if $p_\theta$ is close to $p$, we can say that $K(X|p_\theta) \approx -\sum_{i=1}^{n} \log_2 p_\theta(x_i)$.

This insight is significant. Notice that $-\sum_{i=1}^{n} \log_2 p_\theta(x_i)$ is the negative log-likelihood of the data under the model, which is a common loss function used in ML. This tells us that models with lower error better compress their data, and directly relates Kolmogorov complexity to optimization in ML. However, what if we do not have a model? What is the Kolmogorov complexity of the data itself? Intuitively, if the dataset is sufficiently large, the optimal method for encoding it should be to first specify a model and then encode the data using that model as in Equation (15). Specifically, using identities in Fortnow (2000), we have:

$$K(X) \leq K(X|p_\theta) + K(p_\theta). \tag{16}$$

This encoding scheme on the RHS is referred to as a 2-part code (Grünwald, 2007). For large datasets, we have equality when the model's description length and error are jointly minimized, which occurs when the model $p_\theta(x)$ is equivalent to the true distribution $p(x)$:

$$K(X) = \arg\min_{p_\theta} \left[K(X|p_\theta) + K(p_\theta)\right] = \arg\min_{p_\theta} \left[-\sum_{i=1}^{n} \log_2 p_\theta(x_i) + K(p_\theta)\right] \tag{17}$$

$$= K(X|p) + K(p) = -\sum_{i=1}^{n} \log_2 p(x_i) + K(p). \tag{18}$$

Again, we can draw important connections to ML. Equation (16) says that the Kolmogorov complexity of a dataset is upper-bounded by the a model's error and complexity. In addition, Equations (17) and (18) tell us that the simplest model that explains the data is most likely to be the true one, which draws a theoretical link between compression, maximum likelihood training, model complexity, and generalization (Goldblum et al., 2023).

**Relation to Shannon information** In Shannon information theory (Shannon, 2001), the notion of information quantity is entropy. Given a random variable $X \sim p(x)$, entropy is defined as: $H(X) = \mathbb{E}_{x\sim p(x)} - \log_2(p(x))$. Notice that the $-\log_2(p(x))$ inside the expectation is equal the quantity inside the sum of Equation (14), which specified the minimum number of bits needed to encode a sample from a dataset given the distribution that sample was drawn from. This is no accident: entropy can be seen as the average number of bits needed to compress events from a distribution using an optimal encoding scheme when the distribution $p(x)$ is known. If we simply sum these bits

for a finite number of samples instead of taking an expectation, we get exactly $K(X|p)$ as defined in Equation (14).

As we have seen, though, the assumption about a known distribution $p(x)$, need not be made in the Kolmogorov complexity framework. In this sense, Kolmogorov complexity is a strict generalization of Shannon information theory: $K(X)$ as defined in Equation (18) is equivalent to summed entropy plus the complexity of the distribution $p(x)$, which is unknown and needs to be encoded. In the Shannon framework, it is difficult to derive a meaningful notion for the information quantity in the distribution $p(x)$ because it is an individual object—a function, in particular—and Shannon information is only defined for random variables (Grünwald & Vitányi, 2003). A second drawback of Shannon information is that entropy is a measure of statistical determinability of states; information is fully determined by the probability distribution on states and unrelated to the representation, structure, or content of the individual states themselves (Grünwald & Vitányi, 2003). For this current work, we require a notion of complexity that can account for representations and functions, making Kolmogorov complexity better suited to the task.

## Appendix B Prequential coding and compression without a known learning algorithm

When introducing the relationship between prequential coding and optimal compression in Equation (4), we mentioned that a key assumption is that the learning algorithm $T$ is known. In reality, then, we have that:

$$
\begin{aligned}
K(D|p_\theta) + K(p_\theta) &= K(D, p_\theta) && (19)\\
&\leq K(D, p_\theta, T) && (20)\\
&= K(D, p_\theta|T) + K(T) && (21)\\
&\leq L_{preq}(D; T) + K(T) && (22)\\
\implies L_{preq}(D; T) + K(T) &\geq K(D|p_\theta) + K(p_\theta), && (23)
\end{aligned}
$$

where the first inequality on line Equation (20) appears because compressing additional objects can only take more bits, and the second inequality on line Equation (22) comes from the fact that prequential coding is not necessarily the optimal way to compress a dataset and model given a learning algorithm. If the learning algorithm is a short program like SGD, however, then $K(T) \approx 0$ and $L_{preq}(D; T)$ is an upper-bound of $K(D|p_\theta) + K(p_\theta)$. For simple learning algorithms, then, Equation (4) holds.

## Appendix C Effect of task difficulty on prequential code length

In Section 3.1 Figure 2a, we found that a meta-learned in-context learner trained to minimize prequential code length (prequential ICL) was better able to generalize than one that only minimized training error (train-risk ICL). We further noted that the gap in generalization error between these two learners was greater in low-data regimes, and that the gap extended further as a function of task difficulty (i.e., more in-context data was required to close the gap going from linear regression, to sinusoid regression, to Mastermind). This result is predicted by our theory relating ICL to Occam's razor. A complex task requires the algorithm to learn more complex functions to successfully minimize train risk. However, learning more complex functions with very limited data leads to overfitting, which is the basis for our hypothesis that as task complexity increases, simple predictors learned by minimizing prequential code length enjoy a bigger advantage over predictors learned by minimizing train risk.

To investigate the effect of task difficulty more systematically in this section, we fix the underlying meta-dataset (sinusoid regression tasks) and vary the dimensionality of the input data $dim(x)$. We plot our results in Figure C.1, showing the difference in generalization error between train-risk ICL learners and prequential ICL learners. As expected, ask task difficulty increases, this generalization gap extends further, and the train-risk learners must observe more data in-context in order to close it.

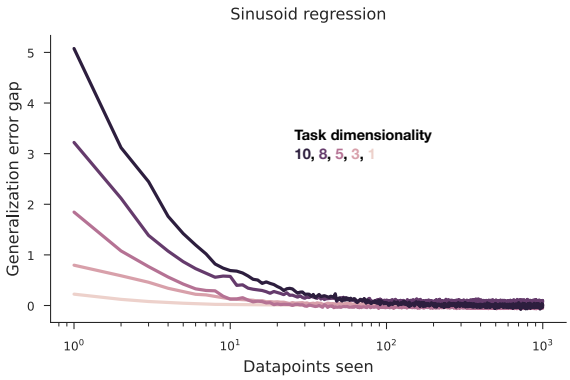


Figure C.1: **Comparison of gap between prequential ICL and train-risk ICL as a function of task difficulty.** Figure shows the difference in average prequential coding curves (i.e., generalization error for train-risk ICL $-$ generalization error for prequential ICL) for sinusoid regression tasks of increasing input dimensionality. Error is measured using MSE. Error bars show standard error across 5 seeds. For all task dimensionalities, the performance gap is positive: ICL from next-token prediction objectives (prequential ICL) yields lower prequential code lengths than ICL from past-token prediction objectives (train-risk ICL), with greater effects in low-data regimes. This gap in generalization error increases with task dimensionality, demonstrating that learners which minimize prequential code length generalize better in virtue of fitting simpler models, and that these simpler models are most important when generalization is difficult (i.e., when the task difficulty is too great for the amount of training data observed).

## Appendix D Advantages over the Bayesian perspective

The Bayes-optimal prediction perspective of ICL and meta-learning says that by meta-training on some set of tasks $\mathscr{D}$, the learner infers some prior over latent task variables—or, equivalently, a prior over models—$p(p_\theta|\mathscr{D})$. On some novel task $D$, the learner then infers a posterior over models that both explain the training data (i.e., assign it a high likelihood) and are consistent with the prior: $p_{\mathscr{D}}(p_\theta|D) = p(D|p_\theta)p(p_\theta|\mathscr{D})/Z$, where $Z$ is a normalizing constant. According to the theory, subsequent predictions are then done through implicit Bayesian averaging under this posterior model distribution.

Crucial differences in our theory are that $\mathscr{D}$ does not need to be drawn from a well-defined distribution over tasks for us to reason about the meta-learning problem—the Kolmogorov framework does not require this—and $K(p_\theta|T_\phi)$ is not *literally* a prior probability distribution over models given $\mathscr{D}$—it only implicitly defines a prior based on the meta-learned $T_\phi$. As a result, our theory generalizes the Bayesian perspective.

To see why these generalizations provide value, consider where the prior in the Bayesian framework $p(p_\theta|\mathscr{D})$ comes from. This prior is not defined explicitly in the ICL framework; instead, it is implicitly defined based on $\mathscr{D}$, the implicit *initial* prior $p(p_\theta)$, and the implicit inference machinery that approximates $p(p_\theta|\mathscr{D}) = p(\mathscr{D}|p_\theta)p(\theta)/Z$. All of these implicit components make any meaningful analysis difficult, since it is difficult to characterize them. However, these implicit components are all intrinsic properties of the meta-learning algorithm (the meta-learner's architecture, the meta-objective, etc.), which we *do* have *explicit* control over. Our theory only makes reference to this meta-learner $T_\phi$ and the description length of models under it $K(p_\theta|T_\phi)$, rather than to objects that are only implicitly defined (and never known). As such, we argue that our theory is more amenable to analysis and provides more explanatory value.

For example, in the Kolmogorov framework that we have proposed, it is easy to see how ICL might in some cases generalize to a novel dataset $D$ that is entirely out-of-domain with respect to $\mathscr{D}$. Perhaps, for instance, the tasks have compositional structure and $T_\phi$ has some inductive biases for compositional generalization. In contrast, it is far more difficult to find a good explanation for such a phenomenon in the Bayesian framework. The explanation would have to be in terms of some implicit initial prior $p(p_\theta)$ (which we never defined) and the subsequent prior $p(p_\theta|\mathscr{D})$ that

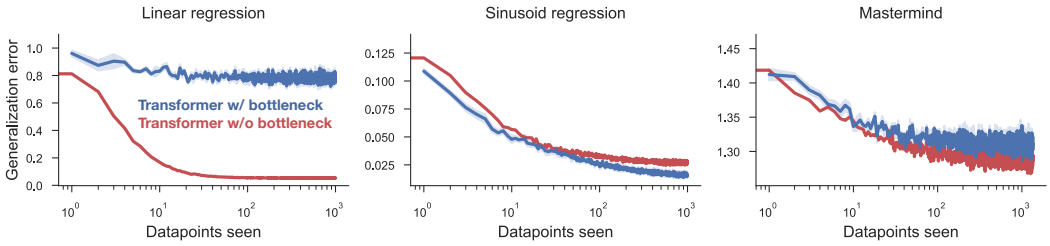


Figure E.1: **Experimental results comparing different Transformer architectures under separate tokenization of $x$ and $y$ values.** Figures compare the performance of Transformers with and without bottleneck under separate $x$ and $y$ tokenization for each of the tasks under consideration. Under this identical tokenization scheme, we find that the performance of the Transformer with bottleneck is much worse compared to the Transformer without bottleneck in 2 out of the 3 tasks. These results indicate that separate tokenization of $x$ and $y$ values is an inferior tokenization scheme which hinders learning compared to single tokenization of $x$ and $y$ values via concatenation.

it induced. Proponents of the Bayesian framework would thus have to say "ahh, generalization here must have been possible because $p(p_\theta)$ had the right kind of structure". However, this same rationale could be used to explain *any* outcome (positive or negative), and therefore is a bad scientific explanation (Deutsch, 2012).

Another problem with the Bayesian perspective is that its predictions do not always hold in practice. Notably, Raventós et al. (2024) found that when the diversity in pretraining tasks is sufficiently large, solutions emerge that are *not* consistent with a Bayes-optimal predictor that uses the pretraining task distribution as its prior. Instead, the solution is consistent with a much broader prior, which allows the learner to adapt to novel tasks that are outside of the pretraining task distribution. Our theory, in contrast, permits explanations for this phenomenon. For instance, perhaps that model used to parameterize $T_\phi$ had insufficient capacity to encode a diverse (and potentially complex) prior over tasks, and instead learned a simpler approximation with more broad coverage over a larger space of tasks.

## Appendix E Influence of tokenization scheme on the performance of Transformers

In section Section 3.3 Figure 3b, we compared different architectures and their influence on ICL's ability to minimize the prequential code length. Among the Transformer architectures, the one with a bottleneck performs significantly better compared to the Transformer without the bottleneck on each of the tasks. However, in this experiment, the tokenization schemes (described in detail in Section F.1) used for each of these Transformer architectures were different. For the Transformer with a bottleneck, $x$ and $y$ values were concatenated to represent a single token, while in the case of Transformer without a bottleneck, we used separate tokens and added positional embeddings. Separate tokens are required for a standard Transformer without bottleneck so that the query token doesn't contain the output $y$ we are trying to predict. This is not required for the Transformer with a bottleneck, however, because the prediction head used in that model only takes as input the latent representation of the task (i.e. the bottleneck) and the query $x$ to predict $y$.

In order to make a fair comparison between the performance of these two architectures and to understand the influence of the tokenization scheme, we compare the performance of both these architectures using an identical tokenization scheme with separate tokens for $x$ and $y$ values, which is a common modeling choice for Transformers in prior work (Garg et al. (2023), Von Oswald et al. (2023a)). Using a causal mask allows the model to make predictions for all $x$ tokens in parallel without attending to their corresponding $y$ values. However, this choice makes learning difficult, as seen in our comparison plot in Figure E.1, where previously the Transformer with a bottleneck outperformed the one without a bottleneck, but now performs worse when $x$ and $y$ are separately tokenized.

# APPENDIX F EXPERIMENT DETAILS

In this section, we provide additional experimental details, including a comprehensive overview of the model architectures and hyperparameters used during training.

## F.1 META-LEARNER ARCHITECTURES

We considered different architectures which exhibit ICL to study and compare their ability to minimize prequential code length (Section 3.3). Each architecture described here parameterizes the meta-learner $T_\phi$.

**Transformer with bottleneck.** We use a standard causal decoder-only Transformer with 4 layers, 4 attention heads, 256 latent dimensions and a feed-forward network with 512 dimensions. Additionally, it has linear projection that bottlenecks the Transformer to 128 dimension. A 5-layer MLP with RELU activations and 256 latent dimensions is used as a separate prediction head.

The Transformer takes a dataset $D$ as input in the format $[x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n]$ (where $x_i$ and $y_i$ are concatenated and each $[\cdot]$ is a token) and computes $T_\phi(D_{1:t-1})$ for each context size starting from 1 to $n-1$. The computation of $T_\phi(D_{1:t-1})$ is based on the encoding of the $t$-th token, which attends only to tokens that appear to the left of $[x_t, y_t]$ and itself. Information leakage from future tokens is prevented using a causal mask. After computing $T_\phi(D_{1:t-1})$, we concatenate it with $x_t$ (i.e., $[T_\phi(D_{1:t-1}), x_t]$) and pass this combined input to an MLP prediction head to predict the next $y$-token.

**Transformer without bottleneck** We use a standard decoder-only Transformer with 4 layers, 4 attention heads, 256 latent dimensions and a feed-forward network with 512 dimensions. Also, in contrast to the previous architecture we don't use a separate prediction head.

This Transformer takes a dataset $D$ as input in the format $[x_1], [y_1], [x_2], [y_2], \ldots, [x_n], [y_n]$ where both $x_i$ and $y_i$ are separate tokens which is different from the Transformer with bottleneck. Using a positional encoder, we include information about the ordering of the input sequence and while predicting the output for $x_t$, we use a causal mask to make sure that the position only attends to itself $x_t$ which is the query and the tokens that comes to the left of $x_t$ in the input sequence which forms the context of size $t-1$. For each x-token, the Transformer predicts the next y-token from that x-token's final representation.

**Mamba** We experiment with two state-space model (SSM) architectures, Mamba 1 and Mamba 2, both composed of 4 layers, 256 latent dimensions, state dimensions 8, and local convolution dimension of 4. Additionally, each layer includes a gated MLP with 256 latent dimensions. Similar, to the Transformer with bottleneck, the prediction model is a 5-layer MLP with RELU activations and 256 latent dimensions is used as a separate prediction head.

The SSM takes a dataset $D$ as input in the format $[x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n]$ (where $x_i$ and $y_i$ are concatenated and each $[\cdot]$ is a token). For each context of size $t-1$, we compute the $T_\phi(D_{1:t-1})$ which is a vector that represents the parameters of the output model obtained after processing the first $t-1$ data points. After computing $T_\phi(D_{1:t-1})$, we concatenate it with $x_t$ (i.e., $[T_\phi(D_{1:t-1}), x_t]$) and pass this combined input to an MLP prediction head to predict the next $y$-token.

## F.2 META-TRAINING AND EVALUATION SETUP

In this section, we outline the complete set of hyperparameters and configurations used across different training objectives and model architectures in our experiments.

**In-context learner (prequential and train-risk).** We trained both the Transformer-based meta-learners (with and without bottleneck) for 50 epochs and the Mamba-based meta-learners for 120 epochs. All results were averaged across 5 different random seeds to mitigate the effect of randomness in the pipeline. The training was conducted on a meta-dataset consisting of 10,000 tasks, each with 1,000 data points that serve as context. We used the Adam optimizer (Kingma & Ba, 2017) with a learning rate of $\eta = 0.0001$ and a batch size of 256, without any early stopping. After

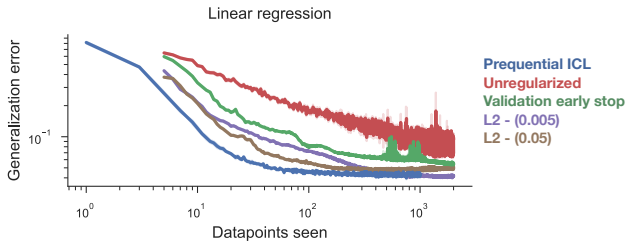


Figure F.1: **Experimental results comparing different regularization techniques.** Figure show average prequential coding curves obtained using both unregularized and regularized Adam optimizers on a linear regression task. Regularized learners exhibit better compression rate (i.e. lower PCL), which implies a stronger incentive toward simple models according to our theory. This experiment confirms the claim that regularization techniques serve as indirect Occam's aligned methods to learn simple models. Analogous to the meta-learning setting, PCL could be minimized with respect to the hyperparameters of the regularization technique.

meta-training, we evaluated the learners on a distinct meta-dataset of 100 tasks, each with 1,000 data points.

**Gradient based learner.** Since gradient-based learner are off-the-shelf learning algorithms which don't require meta-training. The prediction model used is a 5-layers MLP with RELU activations and latent dimensions of 64 or 256 depending on the complexity of the task. We used a meta-dataset of 10000 tasks (with 2000 data points each) split into training (80%) and validation (20%). At each step of prequential coding, we train and evaluate a model by randomly sampling a dataset of fixed size across each of the tasks, starting from 20 to 2000 datapoints. We used an early stopping criteria with minimum loss delta of 0.001 and patience of 10 epochs to avoid overfitting. On each of them, the prediction model was fit using the Adam optimizer (Kingma & Ba, 2017) with a learning rate of $\eta = 0.0001$ and a batch size of 64. All results were averaged across 15 different random seeds.

**Regularization techniques.** Regularization techniques are widely used for gradient-based learners to prevent over-fitted solutions. In this experiment we fit prediction models considering different regularization techniques, namely early-stopping combined with validation data, and weight-decay (L2 regularization). The results are presented in Figure F.1.Experiments with early-stopping halt training when the validation loss does not decrease by more than $1e-4$ over 10 consecutive steps. Experiments with weight-decay consider a regularization parameter $\lambda \in \{0.05, 0.005\}$ and were trained for 1000 epochs. The prediction models used are 5-layers MLPs with RELU activations and latent dimensions of 64. The different prediction models were fit using an Adam optimizer (Kingma & Ba, 2017) with a learning rate of $\eta = 0.0001$ and a batch size of 64. All results were averaged across 15 different random seeds.

### F.3 Pretrained LLM on Mastermind

As described in Section 3.4, we evaluate the performance of a pretrained LLM on the Mastermind task using one of the latest OpenAI models GPT-4 (i.e., gpt-4o). To query the model, we used the OpenAI API with a `temperature` of 0, ensuring that the outputs are deterministic. Along with the responses, we also obtained the log probabilities using the API for calculating the prediction error with respect to each query. This was possible using `logprobs` (boolean) and `top_k_logprobs` (integer) attributes in the API that returns log probabilities for each token in the response and the $k$ tokens with the top log probabilities corresponding to each token in response. By using a structured prompting technique and a retry mechanism (up to 10 retries in case of failure to adhere to the required output format), we were able to consistently obtain appropriate responses to our queries. An example prompt, which includes the task description, context, and the query, is provided below. To calculate the prequential code length, we iteratively query novel examples with an increasing number of in-context examples and obtain the prediction errors. This process emulates the prequential ICL objective.

**Example Prompt**

```
I have a secret code in mind.  It's a 8-digit code with
each digit ranging between 0 and 5.  I'll give you a couple
example guesses, and for each guess I'll tell you two
numbers:

- First number:  the number of correct correct digits at
their correct position.  - Second number:  the number of
correct digits, which aren't necessarily in the correct
position.

Here's a demo to show you what a guess and response would
look like.  Imagine my secret code was:
0 5 2 1 3 4 2 4
And imagine the guess I presented you was:
0 2 1 1 0 2 0 4
Then, the response would be:
3 5

The response is the way it is because the first, forth and
last digit were in the correct place (first response number
is therefore 3) and additionally the second and sixth digit
were in the guess but at the wrong position (second response
number is therefore 5).

The game is about to start.  I'll present you with a series
of guesses and their responses.  Finally, I will present you
with a new guess, and you'll have to predict the correct
response.  Make sure your response is formatted the same
way as in the examples (i.e., with 2 digits between 0-8,
separated by a space).  Let's begin.

----------------------
Guess:  4 2 1 3 4 0 0 5
Response:  3 7

Guess:  1 1 4 3 5 5 0 1
Response:  2 5

Guess:  3 0 2 2 0 5 3 4
Response:  2 6

Guess:  0 2 5 0 4 2 0 1
Response:  1 5

Guess:  4 1 3 2 5 4 2 3
Response:  ?  ?
-----------

What do you think the response is for this final guess?  Make
sure to reply with just 2 digits between 0-8, separated by a
single space character.
```

### F.4 Hidden Markov Model experiment

A prominent theory for why ICL emerges from the next-token prediction objective of LLMs is that sequences $x_{1:n}$ in the pre-training dataset (e.g. large corpuses of text) can be interpreted as

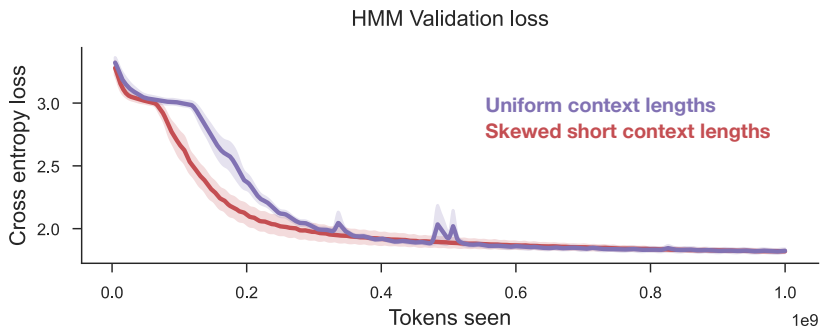


Figure F.2: **Validation loss as a function of the number of tokens seen during training.** The curve is averaged over 5 different datasets (seeds). We can see that the models trained on sequences with shorter length converge faster.

implicitly being sampled from a latent variable generative model $Q(x_{1:n} \mid \boldsymbol{\tau})$ where $\boldsymbol{\tau}$ are some abstract *concepts* underlying samples (Chan et al., 2022; Xie et al., 2022). $\boldsymbol{\tau}$ can range from abstract *style* attributes in natural language (Xie et al., 2022) to *task parameters* such as the teacher weight matrix in linear regression ICL task (Von Oswald et al., 2023a); the important part is that some latent variables can be inferred from the context and subsequently aid prediction. ICL would then emerge as the ability of performing implicit Bayesian inference (i.e. learn from the context) in order to predict $x_t$ :

$$Q(x_t \mid x_{<t}) = \sum_{\tau} \underbrace{Q(x_t \mid x_{<t}, \tau)}_{\text{Condition on the latent}} \underbrace{Q(\tau \mid x_{<t})}_{\text{Infer latent}} \tag{24}$$

We propose to leverage this conceptual framework to devise a novel generation procedure for synthetic LLM pre-training dataset. The general idea is to design a family of sequence models $Q_{\boldsymbol{\tau}}(x_{1:n})$ parameterized by task latents $\boldsymbol{\tau}$, leading to the latent variable generative distribution

$$Q(x_{1:n} \mid \boldsymbol{\tau}) = Q_{\boldsymbol{\tau}}(x_{1:n}).$$

Specifically, we use hidden markov models (HMMs) as the sequences models, and we parameterize the HMMs $Q_{\boldsymbol{\tau}}(x_{1:n})$ with parameters $f_{\boldsymbol{\xi}}(\boldsymbol{\tau}) = \psi_{\boldsymbol{\tau}}$. We use this function $f$ to introduce hyper-parameters $\boldsymbol{\xi}$ which define the whole family of sequence models; i.e. the dataset. Below, we define in details a specific *ad-hoc* function $f_{\boldsymbol{\xi}}(\boldsymbol{\tau})$ which generates a family of HMM where each member share non-trivial structure.

### F.4.1 Detailed description of the generative process

A HMM defines a probability distribution over sequences of *observations* $x_i \in \mathcal{X}$ with a discrete-time probabilistic process over *hidden states* $z_i \in \mathcal{Z}$ paired with a mapping $\mathcal{Z} \to \mathcal{X}$. Both $\mathcal{X}$ and $\mathcal{Z}$ are discrete sets. The hidden process is defined by an initial state distribution $\boldsymbol{\pi}(z)$ and a transition matrix $A \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$ such that

$$Q(z_i | z_j) = A_{ji}$$

Lastly, the mapping between states and observations is governed by the emission matrix $B \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{X}|}$ such that

$$Q(x_j | z_i) = B_{ji}$$

In the rest of the section, we will explicitly define how $f_{\boldsymbol{\xi}}(\boldsymbol{\tau})$ generates $\psi_{\boldsymbol{\tau}} = (\pi^{\tau}, A^{\tau}, b^{\tau})$. We first give a high level description.

The *hyper-parameters* $\boldsymbol{\xi}$ will define a number of building blocks which will be used to create the transition and emission matrix of all HMMs. Then $\boldsymbol{\tau}$ will specify a specific way to combine and manipulate these building blocks to instantiate a specific HMM $Q_{\boldsymbol{\tau}}$. For the transition matrix $A^{\tau}$, the building blocks are pre-defined cycles; which are combined, flipped and accelerated based on $\boldsymbol{\tau}$. For the emission matrix $B^{\tau}$, the building blocks are groups of sub-emission matrices which each

only affect a subset of $|\mathcal{X}|$; which are combined and possibly internal shifted based on $\boldsymbol{\tau}$. Overall, we will have

$$\begin{aligned}\boldsymbol{\xi} = (&\text{N_BASE_CYCLES}, \text{N_BASE_SPEEDS}, \text{N_CYCLE_FAMILIES},\\ &\text{N_GROUP_PER_FAMILY}, \text{N_FAMILY_SPEEDS}, \text{N_EMISSION_GROUPS},\\ &\text{N_EMISSION_PER_GROUP}, \text{N_EMISSION_SHIFT})\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{\tau} = (&\text{BASE_ID}, \text{BASE_SPEED}, \text{FAMILIES_IDS},\\ &\text{FAMILIES_SPEED}, \text{EMISSION_IDS}, \text{EMISSION_SHIFT})\end{aligned}$$

We will refer to the dimensions of $\boldsymbol{\xi}$, $\boldsymbol{\tau}$ as $\xi_i$, $\tau_i$ to avoid clutter and discuss further details below.

**Transition matrix $A^\tau$.** We define a cycle as sequence of hidden states $\boldsymbol{c} = (c_0, \ldots, c_{|c|-1}),\ c_i \in \mathcal{Z}$, and the following manipulation functions

$$\text{DIR}(\boldsymbol{c}, k) = \begin{cases}(c_0, c_{|c|-1}, \ldots, c_1) & \text{if } k = 1\\ c & \text{otherwise.}\end{cases}$$

$$\text{SPEED}(\boldsymbol{c}, k) = (c_0, c_{k(\text{mod } |c|)}, c_{2k(\text{mod } |c|)}, \ldots)$$

In words, $\text{SPEED}(\boldsymbol{c}, k)$ changes the speed at which the cycle is traversed and $\text{DIR}(\boldsymbol{c}, k)$ change its direction. We finally define the transition matrix $\mathcal{T}(\boldsymbol{c})$ associated with cycle $\boldsymbol{c}$ such that

$$\mathcal{T}(\boldsymbol{c})_{ij} = \begin{cases}1 & \text{if } \exists k < n \text{ s.t } (i, j) = (c_k, c_{k+1(\text{mod } n)})\\ 0 & \text{otherwise.}\end{cases}$$

Initially, we randomly generate $\xi_0$ *base cycles* $\boldsymbol{b}_i$ which go through all states $z_i$. Further, we initialize $\xi_2$ families of $\xi_3$ groups of cycles $\boldsymbol{g}^i_j,\ i \in [\xi_1],\ j \in [\xi_2]$. Each HMM's transition matrix is then built from these "building blocks" cycles. Specifically,

$$A^\tau = \mathcal{T}(\text{SPEED}(\text{DIR}(\boldsymbol{b}_{\tau_0}, \tau_1), \tau_2)) + \sum_{i=1}^{\xi_2} \tau_{4,i} \sum_{j=1}^{\xi_3} \cdot \mathcal{T}(\text{SPEED}(\text{DIR}(\boldsymbol{g}^i_j, \tau_5), \tau_6))$$

In words, each transition matrix is made of a) one of $\xi_0$ base cycle, possibly sped up and flipped and b) $\xi_2$ groups of smaller cycles (each from a pool of $\xi_3$ groups), possibly sped up and flipped. The number of possible speeds for the base cycle is defined by $\xi_1$. For the cycle families, it is defined by $\xi_4$

**Emission matrix $B^\tau$.** We separate the states $z \in \mathcal{Z}$ in $\xi_5$ groups $\boldsymbol{h}_i \subset \mathcal{Z}$ and for each group we initialize $\xi_6$ sub-emission matrices $H^i_j \in \mathbb{R}^{|\boldsymbol{h}_i| \times |\mathcal{Z}|}$. Then, we define the manipulation function $\text{SHIFT}(H, k)$ which applies a circular shift of $k$ to the indices of the matrix. Finally, we have

$$B^\tau = \sum_{i=1}^{\xi_5} \text{SHIFT}(B^i_{\tau_{7,i}}, \tau_8)$$

In words, each emission matrix is made of $\xi_5$ possibly overlapping sub-emission matrix, each picked from a pool of $\xi_6$ unique ones. The number of possible shifts is $\xi_7$.

**Initial distribution.** We always use the uniform distribution.

#### F.4.2 HMM HYPER-PARAMETERS

For experiments in this paper, we use $|\mathcal{X}| = 50$ and $|\mathcal{Z}| = 20$. The hyper-parameters of $f$, $\boldsymbol{\xi}$, are given in Table F.1. This results in a total of 512 different transition matrices and 24 different emission matrices, for a total of 12,228 different HMMs. We show results averaged from 5 different seed.

#### F.4.3 TRAINING

We hold out 1,000 HMMs for validation and train on the 11,228 others. Training consists on next-token prediction with a cross-entropy loss, using sequences coming from the training HMMs.

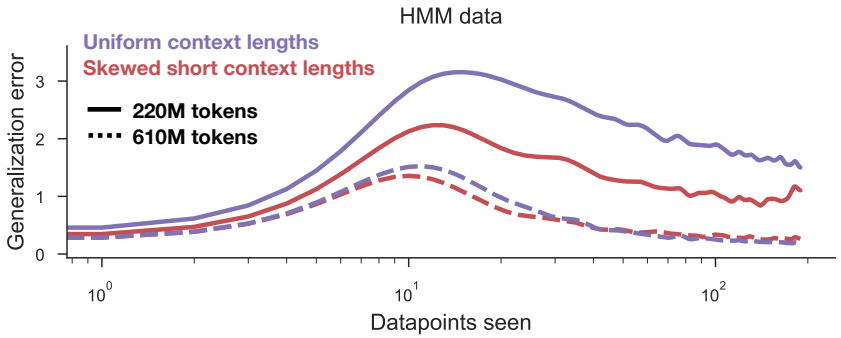


Figure F.3: **Prequential code curves at different stages of training** Reproduction of Figure 3b but with the prequential curve at 610M tokens also. At this point, the models trained with uniform context length have essentially the same performance as the ones trained with smaller context lengths.

| | |
|---|---|
| N_BASE_CYCLES ($\xi_0$) | 4 |
| N_BASE_SPEEDS ($\xi_1$) | 2 |
| N_CYCLE_FAMILIES ($\xi_2$) | 3 |
| N_GROUP_PER_FAMILY ($\xi_3$) | 2 |
| N_FAMILY_SPEEDS ($\xi_4$) | 2 |
| N_EMISSION_GROUPS ($\xi_5$) | 3 |
| N_EMISSION_PER_GROUP ($\xi_6$) | 2 |
| N_EMISSION_SHIFT ($\xi_7$) | 3 |

Table F.1: **HMM dataset hyper-parameters**

Specifically, each epochs consists of one sequence sampled from each training HMM. Every epochs, the sequence sampled from a given HMM is different (using a different seed). As such, the model most likely never sees the same sequence twice. We evaluate on sequences from the 1,000 held-out HMMs. Finally, we use Transformers with 6 layers, 8 heads and embedding dimension of 512. We use a batch size of 512 and a learning rate of $0.001$ with Adam.

#### F.4.4 Evaluation

To obtain the curve in Figure 3b, we compute the $KL$ divergence between the next-token distribution of trained models to the ground truth which we can compute explicitly with Equation (24):

$$KL[p_{\text{model}}(x_t \mid x_{<t}), p_{\text{true}}(x_t \mid x_{<t})] \tag{25}$$

We can compute Equation (24) explicitly because HMMs afford very efficient and parallelizable inference through the forward algorithm. Also, we observe that this "backward" KL divergence is simply a better version of the cross-entropy loss, used to train the model. Indeed, in the cross-entropy loss, $p_{\text{true}}(x_t \mid x_{<t})$ is replaced by a delta-dirac distribution on the observed $x$. While training on it also ends up minimizing Equation (25), it is not the best evaluation metric. Indeed, cross-entropy doesn't take into account the stochasticity of the ground-truth, while Equation (25) does.

Note that the non-monotonicity of the KL prequential coding curve is a consequence of using the above $KL$. Indeed, when very few datapoints have been seen, the model can learn memorise the marginal probability $p_{\text{true}}(x_t \mid x_{<t})$ quite easily, bypassing the to perform ICL. This doesn't show when displaying cross-entropy because $p_{\text{true}}(x_t \mid x_{<t})$ has often very high entropy for small $t$.

#### F.4.5 Training with shorter sequences

When training sequence models like LLMs, the typical approach is to fill the maximum context window of the model with sequences, possibly concatenating multiple ones. This ensures that every batch contains as much tokens—i.e. training signal—as possible. However, because of this, most tokens seen during training are preceded by a lot of tokens: putting more pressure on correctly predicting late tokens than rapidly adapting with small amount of context. According to our theory, this leads to more complex models, possibly worse at generalizing.

Based on this reasoning, we propose a simple way to bias the meta-learner towards simpler models: training on sequences with random context length, typically much shorter than the maximal one. We show the efficacy of our method using our HMM dataset: models trained with **uniform context length** (i.e. all sequences have maximal length) need less tokens to arrive at simple models than the ones trained with **skewed short context lengths** (i.e. sequences of random lengths), as shown in Figure F.2 and Figure F.3. However, there are diminishing returns: with enough data training on long context catches up. Exploring this approach on large-scale language modeling is an interested future work.