BRIDGING KOLMOGOROV COMPLEXITY AND DEEP LEARNING: ASYMPTOTICALLY OPTIMAL DESCRIPTION LENGTH OBJECTIVES FOR TRANSFORMERS

Peter Shaw Google DeepMind
James Cohan Google Research
Jacob Eisenstein Google DeepMind
Kristina Toutanova Google DeepMind

ABSTRACT

The Minimum Description Length (MDL) principle offers a formal framework for applying Occam's razor in machine learning. However, its application to neural networks such as Transformers is challenging due to the lack of a principled, universal measure for model complexity. This paper introduces the theoretical notion of asymptotically optimal description length objectives, grounded in the theory of Kolmogorov complexity. We establish that a minimizer of such an objective achieves optimal compression, for any dataset, up to an additive constant, in the limit as model resource bounds increase. We prove that asymptotically optimal objectives exist for Transformers, building on a new demonstration of their computational universality. We further show that such objectives can be tractable and differentiable by constructing and analyzing a variational objective based on an adaptive Gaussian mixture prior. Our empirical analysis shows that this variational objective selects for a low-complexity solution with strong generalization on an algorithmic task, but standard optimizers fail to find such solutions from a random initialization, highlighting key optimization challenges. More broadly, by providing a theoretical framework for identifying description length objectives with strong asymptotic guarantees, we outline a potential path towards training neural networks that achieve greater compression and generalization.

1 INTRODUCTION

The principle of Occam's razor—that simpler explanations are preferable—is a foundational concept in machine learning. Algorithmic information theory provides an elegant formalization through the *Minimum Description Length* (MDL; Rissanen, 1978) principle. The MDL principle frames learning as data compression: the best model for a dataset is the one that minimizes the combined length of the model's description plus the description of the data encoded with that model. Intuitively, any regularity in the data that is useful for prediction is also useful for compression, and vice versa.

The empirical success of large neural networks might seem to contradict the MDL principle if we consider a naive, uniform encoding of their weights. Such networks often generalize surprisingly well, even when highly over-parameterized and trained with simple maximum likelihood objectives (Zhang et al., 2017). However, various methods have been proposed for training compressible neural networks based on, e.g., quantization (Han et al., 2016), subspace training (Li et al., 2018), low-rank approximation, variational inference (Louizos et al., 2017; Hinton & Van Camp, 1993), or some combination (Lotfi et al., 2022; 2024; DeMoss et al., 2025). These methods have highlighted that neural networks can often be highly compressed, i.e. their true complexity can be much smaller than naive parameter counting would suggest (Blier & Ollivier, 2018). While such methods have thus far not led directly to MDL-inspired regularizers that reliably improve generalization, these methods have been successful for establishing tighter compression bounds (Lotfi et al., 2022; 2024) and inspiring parameter-efficient fine-tuning methods such as LoRA (Hu et al., 2022).

However, key theoretical questions remain. These methods from prior work can be interpreted as defining different description length measures over a network's weights. Different measures may capture different types of regularities, but fail to capture *all* potential regularities in the network. MDL objectives based on such measures may therefore fail to incentivize capturing all of the regu-

larities in the data, if certain patterns cannot be encoded efficiently in the network's weights. This can lead to sub-optimal compression and—per the MDL principle—sub-optimal generalization. Ideally, we want a description length objective that encourages the model to capture *any* regularity in the data. Assuming perfect optimization, such an objective would lead to optimal compression, regardless of the dataset. To what extent can we implement such an idealized objective?

Algorithmic information theory offers a framework to address this question (Li & Vitányi, 2008; Hutter, 2005; Schmidhuber, 1997). MDL was inspired by Solomonoff's theory of inductive inference (Solomonoff, 1964), which proposed to favor hypotheses with low Kolmogorov complexity. Put simply, the Kolmogorov complexity of an object is the length of the shortest program that generates that object (see Section 2). Kolmogorov complexity offers optimal compression relative to any other computable description length measure, up to an additive constant. This *universality* is rooted in the Church-Turing thesis that any sufficiently powerful model of computation can simulate any other. Computable resource-bounded approximations maintain this universality in the limit as resource bounds increase. The notion of Kolmogorov complexity can be directly applied to program induction – priors based on program length have been empirically successful for inducing programs (e.g., Romera-Paredes et al., 2024) and grammars (e.g., Shaw et al., 2021) with strong generalization. However, applying this notion to neural networks is less straightforward. While we can easily compute the length of a discrete program, it is less clear how to quantify the complexity of the function a neural network computes in an analogous way. Therefore, a conceptual gap remains between theoretical frameworks based on algorithmic information theory and practical description length objectives for neural networks. We aim to narrow this gap.

Building on the theory of Kolmogorov complexity and the computational universality of Transformers, we demonstrate the existence of *asymptotically optimal* description length objectives for Transformers, with optimality guarantees analogous to those that hold for Kolmogorov complexity. Our theoretical framework therefore highlights a potential path forward for identifying description length objectives for neural networks that select for models offering greater compression – and potentially greater generalization. Specifically, we develop the theory of asymptotically optimal description length measures for neural networks through the following contributions:

- We define *universal two-part codes* for probabilistic models whose minimum description length for any data sample is provably optimal, up to an additive constant, relative to *any* other two-part code. This framework circumvents the need for arbitrary domain-specific priors by leveraging the universal properties of Kolmogorov complexity. (Section 4)
- We prove the existence of *asymptotically optimal families of codes for Transformers*, which are universal in the limit as resource bounds increase. This result is established via a new demonstration that Transformer encoders are computationally universal in their ability to represent any computable, rational-valued conditional probability distribution. (Section 4)
- We further prove that *tractable and differentiable* objectives for Transformers can be asymptotically optimal, establishing this by constructing and analyzing a variational objective based on an adaptive Gaussian mixture prior. (Section 5)
- We analyze various codelength objectives *empirically* and analytically. Using a manually constructed solution for an algorithmic task, we show that our variational objective selects for a highly compressible model with strong generalization. However, we find that standard optimizers fail to discover such low-complexity solutions from a random initialization, highlighting a key challenge for future work. Additionally, we analytically derive asymptotic bounds for various alternative two-part codes. (Section 6)

2 Background: Kolmogorov Complexity

We give a brief introduction to Kolmogorov complexity, with an extended and more formal version in Appendix A. Intuitively, the *Kolmogorov complexity* $K(x)$ of an object x is the length of the shortest program, written in some standard programming language, that prints x. Similarly, the Kolmogorov complexity $K(f)$ of a discrete function f is the length of the shortest program that computes f. To formalize these notions, Kolmogorov complexity can be defined with respect to *universal prefix Turing machines*, which are multi-tape Turing machines that represent programs as binary strings encoded on a read-only *program tape*. The key *invariance theorem* states that $K(x)$

is independent of the particular choice of universal prefix Turing machine up to an additive constant. While Kolmogorov complexity is not strictly computable due to the halting problem, we write $K_{T,R}(f)$ to denote a computable approximation under a *resource bound* $R = (R_t, R_s) \in \mathbb{N}^2$, which considers only programs that terminate within R_t steps and require at most R_s registers on any tape, with respect to universal prefix Turing machine T. As resource bounds increase, $K_{T,R}(f)$ monotonically decreases to $K(f)$, up to an additive constant, per the invariance theorem. We introduce concise notation to express such relations that hold up to additive constants next.

Inequalities with additive constants Let $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}$ be functions over the same domain $\mathcal{X}$. Then we can write $\forall x,\ f(x) \stackrel{+}{\leq} g(x)$ if and only if there exists some constant $c \in \mathbb{R}$ such that $\forall x,\ f(x) < g(x) + c$, where the additive constant c must not depend on x. We write $\forall x,\ f(x) \stackrel{+}{=} g(x)$ to denote that both $\forall x,\ f(x) \stackrel{+}{\leq} g(x)$ and $\forall x,\ g(x) \stackrel{+}{\leq} f(x)$ hold.[1]

3 Problem Setting and Model Functions

Problem setting Let the input space $\mathcal{X}$ be a discrete and countable set, and the output space $\mathcal{Y}$ be a discrete and finite set. We define the space of all possible datasets as $\mathcal{D} = (\mathcal{X} \times \mathcal{Y})^*$, the set of all finite sequences of input-output pairs. For any dataset $D = ((x_1, y_1), \ldots, (x_n, y_n)) \in \mathcal{D}$, we denote the corresponding sequence of inputs as $X = (x_1, \ldots, x_n)$ and outputs as $Y = (y_1, \ldots, y_n)$. Our analysis focuses on properties of encoding schemes that hold for any such pair of sequences (X, Y).

Model functions Neural networks modeling distributions over a discrete output space are typically characterized by a *model function*, f, which defines a conditional distribution over a discrete space $\mathcal{Y}$ by mapping an input $x \in \mathcal{X}$ to a vector of unnormalized logits, $f(x) \in \mathcal{L}$. Motivated by the finite-precision arithmetic used in neural networks, we assume these logits are rational numbers (i.e. $\mathcal{L} = \mathbb{Q}^{|\mathcal{Y}|}$) which enables a straightforward definition for the Kolmogorov complexity of a model function, $K(f)$.[2] We denote the set of all such computable model functions as $\mathcal{F}$. The conditional distribution for a function $f \in \mathcal{F}$ is given by applying the softmax function, σ, to its output logits:

$$p(Y \mid X; f) = \prod_{x,y \in X,Y} p(y \mid x; f) \qquad p(y \mid x; f) = \sigma(f(x))_y \tag{1}$$

In the following sections, we apply the MDL principle to this setting, where we want to identify *codes* that select for a probabilistic model of this form that optimally compresses a set of labels Y given corresponding inputs X, yielding a *description length objective*.

4 Two-Part Codes

One way to apply the MDL principle is with two-part codes. Consider the cost for a sender (e.g., Alice) to transmit labels Y to a receiver (e.g., Bob), if both are given the inputs X. With a *two-part code*, Alice first transmits a model hypothesis, and then transmits the labels encoded given this model hypothesis (Figure 1). Specifically, we consider two-part codes that are parameterized by a probabilistic model function (Section 3), which we refer to simply as *two-part codes* in this paper.

Definition 1 (two-part code)**.** *A* two-part code M *is specified by a triplet* $\langle \mathcal{H}_M, m_M, \alpha_M \rangle$ *with:*

- *A hypothesis space* $\mathcal{H}_M$ *which we assume is countable, e.g., the parameter space of a particular neural network assuming parameters with some finite-precision.*
- *A mapping* $m_M : \mathcal{H}_M \to \mathcal{F}$ *from hypotheses to model functions (Section 3), e.g., the mapping defined by a particular neural network architecture.*[3]
- *A prior* $\alpha_M(h)$ *defining a distribution over model hypotheses,* $\mathcal{H}_M$*.*[4]

[1] Extending this notation to codelength functions over variables X and Y, $\forall X, Y,\ L_1(Y \mid X) \stackrel{+}{\leq} L_2(Y \mid X)$ implies $\exists c \in \mathbb{R}, \forall X, Y,\ L_1(Y \mid X) < L_2(Y \mid X) + c$, where c does not depend on X or Y. We omit explicit quantifiers when they are clear from context.

[2] See Appendix B.1 for a discussion of the complexities involved with real-valued functions.

[3] This can be a partial function, e.g. a Turing machine with unbounded resources may never halt for some inputs.

[4] Formally we allow the prior to be any lower semicomputable semimeasure (see Appendix A).

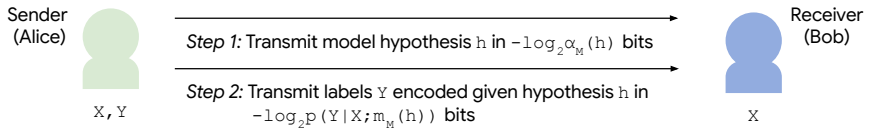

Figure 1: **Two-part code.** One way to formalize the MDL principle is with a two-part code. Assume Alice and Bob agree on a two-part code M and each are given inputs X. Alice then finds the hypothesis (e.g., model parameters) that enables sending Bob the labels Y in the fewest total bits, balancing the complexity of the model with how well it fits the data. One key challenge we address is that the minimum codelength is dependent on the potentially arbitrary choice of prior $\alpha_M(h)$.

Given a two-part code M, the codelength for transmitting Y given X is defined as the sum of the codelength of the hypothesis and the codelength of the data given the hypothesis:

$$L_M^{\text{two-part}}(Y \mid X; h) = -\log_2 \alpha(h) - \log_2 p(Y \mid X; m_M(h)). \tag{2}$$

The $-\log_2$ terms represent the ideal codelength for transmitting the model and data under the distributions specified by the prior and the selected hypothesis, respectively; the latter term is recognizable as the standard negative log-likelihood (NLL) objective. From a Bayesian perspective, minimizing equation 2 can be interpreted as finding the MAP estimate. The minimum achievable codelength for transmitting the labels with the two-part code M is then denoted as:

$$L_M^{*,\text{two-part}}(Y \mid X) = \min_{h \in \mathcal{H}_M} L_M^{\text{two-part}}(Y \mid X; h). \tag{3}$$

Universal two-part codes Notably, there is a considerable degree of freedom in specifying a two-part code, e.g. for a particular neural network architecture specifying the hypothesis space and mapping function, we can choose any prior over the model weights. The prior determines the description length measure over the weights, which, in effect, determines the inductive bias of the codelength objective. Building on the invariance theorem for Kolmogorov complexity, we will show that there exists an equivalence class of two-part codes that are *universal* in the sense that a minimizer of any universal two-part code offers at least as much compression as any other two-part code, for any dataset, up to an additive constant that does not depend on the data. This guarantee holds regardless of the specific prior, as long as it constructs a universal code, and is captured in the following theorem and its corollary.

Definition 2 (universal two-part code)**.** *Let M^1 be a two-part code. M^1 is a* universal two-part code *if and only if, for any other two-part code, M^2, the following holds for any dataset X, Y:*

$$L_{M^1}^{*,\textit{two-part}}(Y \mid X) \overset{+}{\leq} L_{M^2}^{*,\textit{two-part}}(Y \mid X). \tag{4}$$

Proposition 1. *There exists a universal two-part code.*

Corollary 1. *The minimum of any universal two-part code M is equal to the following bound, denoted $C^{\textit{two-part}}(Y \mid X)$, up to an additive constant:*

$$L_M^{*,\textit{two-part}}(Y \mid X) \overset{+}{=} C^{\textit{two-part}}(Y \mid X) = \min_{f \in \mathcal{F}} K(f) - \log_2\ p(Y \mid X; f). \tag{5}$$

The core requirement for a two-part code to be universal is that its underlying model class must be computationally universal. That is, for any computable model function f, there must be some hypothesis h in the hypothesis space that computes it, i.e. where $m_M(h) = f$. Furthermore, the prior, $\alpha(h)$, must assign at least one such hypothesis a codelength, $-\log_2 \alpha(h)$, that is equivalent to the Kolmogorov complexity of the function being computed, $K(f)$, up to an additive constant (see B.4.2 for the formal proofs). This ensures the code can, in principle, efficiently represent any computable regularity in the data. Fully satisfying these conditions requires a model class with unbounded computational resources, like a Turing machine, and strictly minimizing the code is not computable, due to the halting problem. Since any real-world model, such as a Transformer, has finite resources (e.g., a finite number of layers and context window), it cannot be strictly universal. This motivates the notion of *asymptotically optimal* codes, introduced next.

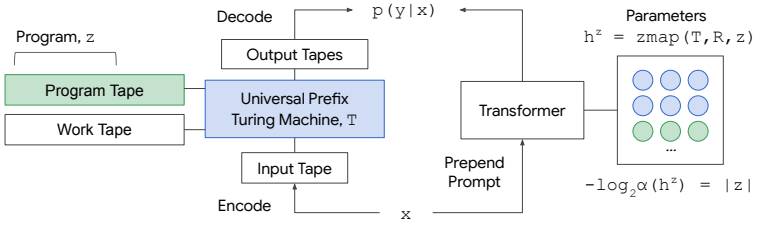

Figure 2: **Constructing asymptotically optimal codes for Transformers.** We construct a function `zmap` that establishes that a Transformer (right) can effectively simulate a prefix Turing machine T with any prefix z encoded on a program tape (left), and therefore can represent any computable model function (as defined in section 3) within some arbitrary time and space resource bound R. With this mapping between model functions and Transformer parameters established, we can select a prior that assigns probability to sets of parameters based on the algorithmic complexity of the function they compute, thus forming an asymptotically optimal code (Section 4.1).

Asymptotically optimal two-part codes Neural network architectures, such as Transformers, define a family of models, with hyperparameters corresponding to finite time and space resource. Certain families of two-part codes can then be shown to be *asymptotically optimal* in the sense that as the resource bounds increase, the minimum code length monotonically decreases to the minimum of a universal two-part code (see B.4.5 for the proof).

Definition 3 (asymptotically optimal families of two-part codes)**.** *A family of two-part codes* $\{M_R \mid R \in \mathbb{N}^2\}$ *is* asymptotically optimal *with respect to a universal prefix Turing machine* T *if:*

$$\forall R \in \mathbb{N}^2,\ L^{*,\textit{two-part}}_{M_R}(Y \mid X) \overset{+}{\leq} C^{\textit{two-part}}_{T,R}(Y \mid X), \tag{6}$$

where $C^{\textit{two-part}}_{T,R}(Y \mid X) := \min_{f \in \mathcal{F}} K_{T,R}(f) - \log_2\ p(Y \mid X; f)$ *and* $K_{T,R}$ *denotes Kolmogorov complexity under resource bound* R*.*

Proposition 2. *Given an asymptotically optimal family of two-part codes* $\{M_R \mid R \in \mathbb{N}^2\}$*:*

$$\lim_{R_t, R_s \to \infty} L^{*,\textit{two-part}}_{M_R}(Y \mid X) \overset{+}{=} C^{\textit{two-part}}(Y \mid X), \tag{7}$$

with the bound $C^{\textit{two-part}}_{T,R}(Y \mid X)$ *monotonically non-increasing with increasing* R_t *or* R_s*.*

We demonstrate the existence of such asymptotically-optimal codes for Transformers next.

4.1 Two-part codes for Transformers

The Transformer architecture (Vaswani et al., 2017) defines a family of models with various hyperparameters determining the time (e.g., number of layers) and space (e.g., context window size) resources of the model. We focus our theoretical analysis on Transformer *encoders* applied to sequence classification, a commonly studied setting (e.g., Devlin et al., 2019).

Theorem 1. *There exists an* asymptotically optimal *family of two-part codes for Transformer encoders.*

Figure 2 captures the main idea. Given a universal prefix Turing machine T and resource bound R, let $\mathcal{Z}_{T,R}$ denote the set of programs such that T halts within R_t steps and requires only up to R_s registers on any tape, for any input $x \in \mathcal{X}$ encoded on the input tape. Let $f^z_T : \mathcal{X} \to \mathcal{L}$ be the *model function* computed by T with prefix z on the program tape. For a given R, we construct a two-part code M_R where the hypothesis space $\mathcal{H}_{M_R}$ and mapping function m_{M_R} are defined by a Transformer encoder, with $\mathcal{O}(R_t)$ layers and a context window size of $\mathcal{O}(R_s)$. We use ALTA (Shaw et al., 2024), a compiler from symbolic programs to Transformer weights, to construct a function `zmap` such that $\forall z \in \mathcal{Z}_{T,R},\ m_{M_R}(h^z) = f^z_T$ where $h^z = \texttt{zmap}(T, R, z)$. While future work could consider alternative ways of constructing such a mapping (see 6.3), our particular construction relies on prepending a sequence of R_s "prompt tokens" to the model input, with learnable embeddings. These prompt tokens represent the program z, while the attention and MLP weights are configured

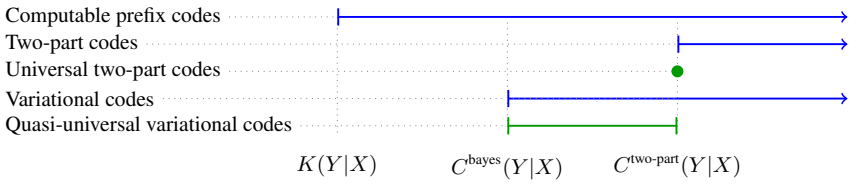

Figure 3: **Upper and lower bounds on minimum codelengths.** The figure shows the minimum number of bits required to transmit labels Y given inputs X, for different classes of codes. The bounds hold for any dataset (X, Y) up to an additive constant that does not depend on the dataset.

to act as an interpreter, emulating the Turing machine T. We can then trivially construct a prior $\alpha_{M_R}(h)$ that such that $\forall z \in \mathcal{Z}_{T,R},\ -\log_2 \alpha_{M_R}(h^z) = |z|$, with $\alpha_{M_R}(h) = 0$ for hypotheses outside the range of `zmap`. This construction can be shown to satisfy the conditions of an asymptotically optimal family of codes with respect to T. Further, any prior satisfying the more relaxed condition $\forall z \in \mathcal{Z}_{T,R},\ -\log_2 \alpha_{M_R}(h^z) < |z| + c_T$ where c_T is a constant not depending on R or z is sufficient. (See B.6 for the formal proof.)

4.2 Limitations & Practical Considerations

While asymptotically optimal codes offers appealing compression bounds, in this section we discuss why these bounds are insufficient to guarantee real-world performance and relevant practical considerations.

First, the theoretical guarantees hold in the limit as model resources increase, and hold up to an additive constant that only becomes negligible as dataset complexity increases. While the community has been training increasingly large models on increasingly large datasets, any real-world models and datasets are, of course, finite. For a finite-resourced Transformer, satisfying the conditions of an asymptotically optimal code only formally constrains the description length of a subset of the computable functions that could be represented. Strictly emulating a Turing machine can be inefficient, failing to leverage a Transformer's capacity for parallel and numerical computation. Therefore, intuitively, we should consider sufficiently *general and flexible priors* that are not overly specialized to a specific Turing machine. For a sufficiently flexible prior, our method used to prove asymptotic optimally – showing that a Transformer can emulate a universal prefix Turing machine – serves primarily to establish a worst-case upper bound on the minimum achievable codelength.

Second, the formal guarantees apply to the *minimum* of the codelength objective. Practically, we need to consider objectives that are computationally tractable and can be efficiently optimized.

Therefore, we should aim to identify codes that are asymptotically optimal (or approximately optimal) but also satisfy these additional informal but intuitive criteria, which can ultimately be assessed empirically. As one potential path toward meeting this goal, we consider *variational codes*, which we study theoretically in Section 5 and empirically in 6.1. We also analyze the asymptotic bounds for various alternative two-part codes in 6.2.

5 Variational Codes

An alternative approach for Alice to transmit the data labels to Bob is to use a *variational code*, which considers a *distribution* over hypotheses, rather than selecting a single best hypothesis. We start with some definitions and formal results, and then work towards a practical implementation.

Definition 4 (variational code)**.** *A* variational code M *consists of a hypothesis space $\mathcal{H}_M$, a mapping $m_M : \mathcal{H}_M \to \mathcal{F}$ from hypotheses to model functions, and prior $\alpha_M(h)$ over $\mathcal{H}_M$, as specified in definition 1. Additionally, a variational code specifies a posterior hypothesis space, Φ_M, and distribution over hypotheses, $\beta_M(h; \phi)$, parameterized by $\phi \in \Phi_M$.*

The conditional distribution over Y given X is therefore specified by the posterior parameters $\phi \in \Phi_M$, and defined by marginalizing over hypotheses:

$$p_M(Y \mid X; \phi) = \mathbb{E}_{h \sim \beta_M(\cdot;\phi)}\, p(Y \mid X; m_M(h)). \tag{8}$$

The codelength for a variational code M is then defined as:

$$L_M^{\text{var}}(Y \mid X, \phi) = \mathbb{E}_{h\sim\beta_M(\cdot;\phi)}\left[-\log_2\ \alpha_M(h) + \log_2\ \beta_M(h;\phi) - \log_2\ p(Y \mid X; m_M(h))\right] \quad (9)$$

$$= \text{KL}\left[\beta_M(\cdot;\phi)\ \|\ \alpha_M(\cdot)\right] - \log_2\ p_M(Y \mid X;\phi), \quad (10)$$

which can be interpreted as the cost of transmitting the labels given the "bits back" argument of Hinton & Van Camp (1993). The intuition is that parameters with higher uncertainty can be transmitted with lower cost. Equation 10 mirrors equation 2, with the KL term capturing the model cost. We denote the minimum of a variational code M as:

$$L_M^{*,\text{var}}(Y \mid X) = \min_{\phi\in\Phi_M} L_M^{\text{var}}(Y \mid X, \phi). \quad (11)$$

A variational code can be seen as approximating an idealized but intractable Bayesian code (B.7.5). As with two-part codes, variational codes require choosing a seemingly arbitrary prior. We address this by defining *quasi-universal variational codes*, which are analogous to universal two-part codes.

Definition 5 (quasi-universal variational code)**.** *A variational code M is a* quasi-universal variational code *if and only if:*

$$L_M^{*,var}(Y \mid X) \overset{+}{\leq} C^{two\text{-}part}(Y \mid X). \quad (12)$$

We use the term *quasi-universal* because such codes are not necessarily strictly universal among the set of variational codes. However, their minima are tightly bound per the following theorem.

Proposition 3. *For any quasi-universal variational code M,*

$$K(Y \mid X) \overset{+}{\leq} C^{bayes}(Y \mid X) \overset{+}{\leq} L_M^{*,var}(Y \mid X) \overset{+}{\leq} C^{two\text{-}part}(Y \mid X), \quad (13)$$

where $C^{bayes}(Y \mid X) := -\log_2 \sum_{f\in\mathcal{F}} 2^{-K(f)} p(Y \mid X; f)$.

The result is visualized in Figure 3. The proof involves deriving $C^{\text{bayes}}(Y \mid X)$ as the minimum of a universal Bayesian code (B.7.6). Under reasonable conditions, the differences between each of these quantities are expected to be small. See B.7 for formal analysis and discussion. The definition of asymptotically optimal families of two-part codes directly extends to variational codes (B.8.2). Additionally, variational codes can be seen as a generalization of two-part codes (B.8.1), and therefore our previous result constructing asymptotically optimal two-part codes for Transformers directly extends to prove the following proposition (B.8.3).

Proposition 4. *There exists an asymptotically optimal family of variational codes for Transformer encoders.*

Finally, we note that we can also generalize variational codes by using an *adaptive prior* (Hinton & Van Camp, 1993), where the prior is itself parameterized. We term such codes *adaptive variational codes*, defined formally in B.8.4. Adaptive priors help generalize our previously constructed codes to form a more flexible implementation, discussed next.

5.1 Differentiable Codes via Gaussian Mixture Models

Per the discussion in 4.2, we aim to identify families of asymptotically optimal codes for Transformers that are based on flexible and general priors – not explicitly constructed around a specific Turing machine emulation – and are amenable to standard gradient-based optimization. As one path towards this goal, we construct a family of adaptive variational codes where the adaptive prior and posterior distributions are both parameterized by sets of independent Gaussian mixture models (GMMs; see B.9.1). Intuitively, a GMM prior shared across a group of weights drives compression by encouraging a low-entropy clustering of weight values around the component means, akin to soft quantization (Ullrich et al., 2017; Achterhold et al., 2018; Han et al., 2016). Such a GMM-based construction is appealing for its generality, and because we can estimate the variational codelength objective using Monte-Carlo sampling and apply standard optimizers via the reparameterization trick (Kingma & Welling, 2014; Jang et al., 2017; Maddison et al., 2017). The proof of Theorem 2 below demonstrates that this construction is also asymptotically optimal, thus establishing a worst-case asymptotic upper bound on the minimal codelength (i.e. compression) achievable with such codes. The proof also serves to illustrate a potential path towards identifying other instances of practical and asymptotically optimal codes.

Theorem 2. *There exists an asymptotically optimal family of adaptive variational codes for Transformer encoders where the adaptive prior and posterior distributions are both specified by products of independent GMMs.*

The key challenge in satisfying this theorem, in contrast to our earlier existence proofs, lies in the stronger constraints imposed on the prior distribution, particularly the independence assumptions. We show that two relatively general conditions are sufficient to establish asymptotic optimality. First, we use layerwise weight sharing, as in a Universal Transformer (Dehghani et al., 2019), ensuring that the number of Transformer weights doesn't scale with the time bound R_t. Second, we share a GMM prior over groups of Transformer weights, ensuring that the number of prior parameters doesn't scale with the space bound R_s.

The formal proof is in B.10. Our proof sketch here builds on the function $\texttt{zmap}$ and the related notation introduced in Section 4.1. Given a universal prefix Turing machine T, for every resource bound $R \in \mathbb{N}^2$ and program $z \in \mathcal{Z}_{T,R}$, we construct GMM posterior parameters $\phi^{R,z}$ and GMM prior parameters ψ^R such that the KL divergence is equal to $|z|$ plus an additive constant and the model function is deterministic and equivalent to f_T^z. Specifically, we construct $\phi^{R,z}$ to approximate a delta function at each weight in $\texttt{zmap}(T, R, z)$, except for weights representing the program tape past the $|z|$th register; these weights do not affect the model function being computed, and we therefore set their posterior to be equal to their respective prior, such that they can be transmitted "for free". Importantly, by construction, the output of $\texttt{zmap}$ consists of a constant number of weights (depending on T but not R) outside of the R_s prompt embeddings (which represent the program tape contents). By sharing a GMM prior across each feature column of the prompt embeddings table, the total KL for these embeddings is shown to be $|z|$ bits. The fixed number of remaining weights can be transmitted in a constant number of bits for any GMM prior assigning non-zero probability to the associated weights in $\texttt{zmap}(T, R, z)$. This establishes the necessary upper bound on the minimum codelength, although there may be prior and posterior parameters offering even greater compression.

6 Experiments & Analysis

6.1 Empirical Analysis of Variational Code

First, we empirically evaluate the GMM-based adaptive variational code discussed in Section 5.1 as a computationally tractable and differentiable instance of an asymptotically optimal code for Transformers.

Computing parity We focus our initial investigation on the task of computing parity, i.e. given a sequence of 0s and 1s, determining whether there is an odd or even number of 1s. Computing parity is a common task for assessing Transformer expressivity and generalization (Hahn, 2020; Bhattamishra et al., 2020; Chiang & Cholak, 2022; Ruoss et al., 2023; Delétang et al., 2022; Anil et al., 2022; Zhou et al., 2022; 2023). Standard Transformers trained with maximum likelihood objectives struggle with consistent length generalization on this task, making it a useful benchmark for assessing whether an alternative objective can encourage stronger out-of-distribution generalization. A key objective of our experiments is to distinguish between (1) failures of the proposed objective to select for compressible models with strong generalization and (2) failures to effectively optimize the proposed objective. To this end, for simple algorithmic tasks such as computing parity, we can again use ALTA, this time to manually determine a set of parameters that fits the training data and also has low complexity according to the proposed objective, establishing an upper bound on the minimum of the variational objective. We can then determine whether our optimization process, starting from a random initialization, can find a set of parameters with similar or lower loss. See C.1 for details.

The results for Transformers with different training objectives and initializations are shown in Table 1. Only the manually initialized model exhibits strong length generalization (OOD accuracy). All randomly initialized models fit the training set, but struggle with length generalization. The model trained with the variational objective from a random initialization does not achieve a loss comparable to the ALTA-compiler initialization. This suggests that while the objective may select for models with stronger generalization, standard optimization techniques fail to effectively find a strong minimizer, which we investigate further next.

Table 1: Transformer performance on parity task for different training objectives and initializations.

Init.	Objective	KL (bits)	Train NLL (bits)	Train Acc.	OOD Acc.
Random	MLE Baseline	—	2.5×10^1	100%	56.4%
Random	Variational	2.8×10^6	2.4×10^2	100%	60.4%
Manual	Variational	8.7×10^2	0.0	100%	100%

Optimization analysis To understand why our Transformer models underfit the proposed objective when randomly initialized, we study the simplified setting of a 2-layer MLP on a simple identity task, with details in C.2. We use a shared GMM prior and Gaussian posteriors for each weight. Again, we see that random initialization fails to find a loss comparable to that achieved via manual initialization, indicating poor optimization of the proposed objective. By inspecting the learned prior and posterior distributions compared to those from manual initialization, we identify that one potential culprit is that the prior distribution collapses to a unimodal distribution (see Figure 10 in C.2). This is in contrast to our manually identified solution which consists of a multimodal distribution with low variance components, which leads to a significantly lower KL divergence. These results suggests that future work should consider alternative optimization procedures that discourage this form of collapse, or consider alternative families of asymptotically optimal codes altogether.

6.2 Asymptotic Bounds for Alternative Two-Part Codes

Next, we analyze asymptotic bounds for alternative two-part codes that may be of practical interest but do not strictly satisfy the conditions of asymptotic optimality. Recall from 4.1 that the key requirement for satisfying the conditions of asymptotic optimality is that $\forall R \in \mathbb{N}^2$ and $\forall z \in \mathcal{Z}_{T,R}$, $-\log_2 \alpha_{M_R}(h^z) < |z| + c_T$ where $h^z = \texttt{zmap}(T, R, z)$ and c_T is an additive constant not depending on R or z. In Table 2, the *Description Length Bound* refers to an upper bound on this model cost $-\log_2 \alpha_{M_R}(h^z)$, up to an additive constant, for various two-part codes that do not strictly satisfy this theoretical ideal. A sub-optimal bound for this quantity leads to a sub-optimal bound for the overall minimum codelength, but close approximations could still be of practical interest.

Table 2: Combining standard methods for quantization, adaptive selection of the prefix length, and layerwise weight sharing yields a two-part code with a description length upper bound (described in 6.2) close to the theoretical ideal of $|z|$. Ablating these methods degrades the bound, shifting the encoding cost away from the algorithmic complexity of the function computed by the network ($|z|$) and causing the encoding cost to scale linearly with the total parameter count of the network, driven by its maximum space (R_s) and time (R_t) resources.

Quantization	Adaptive Prefix Length	Layerwise Weight Sharing	Description Length Bound
✓	✓	✓	$\lvert z\rvert + \log R_s$
✗	✓	✓	$\mathcal{O}(\lvert z\rvert) + \log R_s$
✗	✗	✓	$\mathcal{O}(R_s)$
✗	✗	✗	$\mathcal{O}(R_s) + \mathcal{O}(R_t)$

We show that combining a form of quantization, adaptive selection of the prefix length, and layerwise weight sharing yields a description length bound that is *close* to the theoretically ideal description length bound of $|z|$, differing by only a $\log R_s$ term. *Adaptive prefix length* means dynamically selecting the optimal prefix length, e.g. via a hyperparameter sweep; transmitting this quantity introduces the $\log R_s$ term. Sufficient quantization can be achieved by using an adaptive GMM prior similarly to the previously studied variational code, using the adaptive quantization method proposed by Han et al. (2016), or by restricting the prefix embeddings to a fixed vocabulary as in discrete prompt optimization. The bound becomes progressively worse as these aspects are ablated, ultimately resulting in a uniform encoding of the model weights. Note that by definition $|z| \leq R_s$, and we adopt $\mathcal{O}(\cdot)$ notation to indicate a multiplicative factor > 1. Overall, the $|z| + \log R_s$ bound can perhaps be seen as a relatively positive theoretical result for methods related to discrete prompt optimization. (See B.11 for additional details and discussion.)

6.3 Alternative Transformer Families

The analysis in this paper has focused primarily on scaling the time and space resource bounds of a Transformer encoder by scaling the number of layers and the length of a prefix prepended the input, respectively, as described in 4.1. Here we briefly discuss alternative families of Transformers.

Scaling Transformer Dimensionality Rather than scaling the number of prefix token embeddings to represent increasingly complex programs, we could scale the dimensionality of the Transformer, effectively encoding the program tape in the MLP parameters. In B.12 we discuss how a mapping analogous to `zmap` could be established for such a family of Transformers; however, this requires scaling the model and MLP dimensions linearly with program length. This results in the number of MLP parameters growing quadratically with program length, leading to a bound of $\mathcal{O}(|z|^2) + \log R_s$ in the context of Table 2 if applying similar techniques. The constructed MLP matrices are highly structured, but have rank that scales linearly with program length, and therefore cannot be easily compressed with standard techniques such as rank factorization. Therefore, identifying tighter asymptotic bounds for practical methods for MLP matrix compression would be of interest for future work. Otherwise, these results potentially indicate poor asymptotic bounds for standard methods related to MLP compression.

Transformer Decoders Rather than scaling the number of layers in a Transformer encoder to scale the time resource bound, we could scale the number of intermediate decoding steps in a Transformer decoder. This would not require explicit layerwise weight sharing to maintain a constant description length with respect to a growing time resource bound. The main challenge in implementing a mapping analogous to `zmap` for a Transformer decoder is in representing updates to the Turing machine tape given the decoder's attention mask, but such a result could build on prior work establishing the Turing completeness of Transformer decoders under various conditions (Pérez et al., 2021; Merrill & Sabharwal, 2024a; Li & Wang, 2025). Future work could also study bounds for Transformer decoders that can interact with external tools.

7 Related Work

Our work is closely related to the notion of *universal induction* (Solomonoff, 1964) and related theoretical frameworks (Levin, 1973; Hutter, 2000; Lattimore & Hutter, 2013; Achille et al., 2021; Nakkiran, 2021). For neural networks, a notable early work is Schmidhuber (1997), which introduced a probabilistic search algorithm for discovering neural networks with low Kolmogorov complexity. However, none of these theoretically compelling approaches directly lead to practical and scalable training objectives for neural networks. On the other hand, various MDL-inspired complexity measures have been proposed and evaluated for neural networks, based on variational inference (Hinton & Van Camp, 1993; Blundell et al., 2015; Louizos et al., 2017; Blier & Ollivier, 2018) or other methods (Li et al., 2018; Lotfi et al., 2022; 2024; DeMoss et al., 2025; Abudy et al., 2025), such as quantization and low-rank approximation. However, prior work has not demonstrated asymptotic compression guarantees for such methods. Our main contribution is establishing asymptotic bounds by constructing a bridge between Transformer weights and prefix Turing machines. Our results extend prior work establishing the Turing completeness of Transformers (Pérez et al., 2021; Nowak et al., 2024). We provide an extended discussion of related work in Appendix D.

8 Conclusion

In this paper, we introduced a framework for analyzing asymptotic bounds for description length objectives, grounded in the MDL principle and the universality of Kolmogorov complexity. We established that there exist asymptotically optimal description length objectives for Transformers, while also highlighting potential challenges related to effectively optimizing such objectives. Minimizing an asymptotically optimal description length objective achieves optimal compression for any dataset, up to an additive constant, in the limit as resource bounds increase. Such guarantees are particularly compelling as the computational resources available for training models continue to increase. Our theoretical framework therefore highlights a potential path towards identifying description length objectives leading to greater compression, and potentially greater generalization.

ACKNOWLEDGEMENTS

We thank Jonathan Berant, Kenton Lee, and Tim Genewein for useful discussions and feedback, and the anonymous reviewers for their helpful comments.

REFERENCES

Emmanuel Abbe, Samy Bengio, Aryo Lotfi, and Kevin Rizk. Generalization on the unseen, logic reasoning and degree curriculum. In *International Conference on Machine Learning*, pp. 31–60. PMLR, 2023.

Matan Abudy, Orr Well, Emmanuel Chemla, Roni Katzir, and Nur Lan. A minimum description length approach to regularization in neural networks. *arXiv preprint arXiv:2505.13398*, 2025.

Alessandro Achille, Giovanni Paolini, Glen Mbeng, and Stefano Soatto. The information complexity of learning tasks, their structure and their distance. *Information and Inference: A Journal of the IMA*, 10(1):51–72, 2021.

Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ry-TW-WAb.

Victor Akinwande, Yiding Jiang, Dylan Sam, and J Zico Kolter. Understanding prompt engineering may not require rethinking generalization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=a745RnSFLT.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Andrew R Barron. *Logically smooth density estimation*. PhD thesis, Stanford University, 1985.

Andrew R Barron and Thomas M Cover. Minimum complexity density estimation. *IEEE transactions on information theory*, 37(4):1034–1054, 1991.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

Satwik Bhattamishra, Arkil Patel, Varun Kanade, and Phil Blunsom. Simplicity bias in transformers and their ability to learn sparse boolean functions. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

Léonard Blier and Yann Ollivier. The description length of deep learning models. *arXiv preprint arXiv:1802.07044*, 2018.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pp. 1613–1622. PMLR, 2015.

Jorg Bornschein, Yazhe Li, and Marcus Hutter. Sequential learning of neural networks for prequential MDL. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=dMMPUvNSYJr.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Gregory J Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM (JACM)*, 16(1):145–159, 1969.

Angelica Chen, Ravid Shwartz-Ziv, Kyunghyun Cho, Matthew L Leavitt, and Naomi Saphra. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in MLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=MO5PiKHELW.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. *arXiv preprint arXiv:2202.12172*, 2022.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In *International Conference on Machine Learning*, pp. 5544–5562. PMLR, 2023.

Tristan Cinquin, Alexander Immer, Max Horn, and Vincent Fortuin. Pathologies in priors and inference for bayesian transformers. In *I (Still) Can't Believe It's Not Better! NeurIPS 2021 Workshop*, 2021.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the Chomsky hierarchy. *arXiv preprint arXiv:2207.02098*, 2022.

Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=jznbgiynus.

Branton DeMoss, Silvia Sapora, Jakob Foerster, Nick Hawes, and Ingmar Posner. The complexity dynamics of grokking. *Physica D: Nonlinear Phenomena*, pp. 134859, 2025.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Raaz Dwivedi, Chandan Singh, Bin Yu, and Martin Wainwright. Revisiting minimum description length complexity in overparameterized models. *Journal of Machine Learning Research*, 24(268): 1–59, 2023.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Fabio James Fehr and James Henderson. Nonparametric variational regularisation of pretrained transformers. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Zu8OWNUC0u.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=qHrADgAdYu.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. 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.

Jordi Grau-Moya, Tim Genewein, Marcus Hutter, Laurent Orseau, Gregoire Deletang, Elliot Catt, Anian Ruoss, Li Kevin Wenliang, Christopher Mattern, Matthew Aitchison, and Joel Veness. Learning universal predictors. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=B1ajnQyZgK.

Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

Peter Grunwald. A tutorial introduction to the minimum description length principle. *arXiv preprint math/0406077*, 2004.

Peter D Grunwald and Paul Vitanyi. Algorithmic information theory. *arXiv preprint arXiv:0809.2754*, 2008.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

Boumediene Hamzi, Marcus Hutter, and Houman Owhadi. Bridging algorithmic information theory and machine learning: A new approach to kernel learning. *Physica D: Nonlinear Phenomena*, 464:134153, 2024.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13, 1993.

Antti Honkela and Harri Valpola. Variational learning and bits-back coding: an information-theoretic view to bayesian learning. *IEEE transactions on Neural Networks*, 15(4):800–810, 2004.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Marcus Hutter. A theory of universal artificial intelligence based on algorithmic complexity. *arXiv preprint cs/0004001*, 2000.

Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2005.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgIPJBFvH.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

A Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:3–11, 1965.

Tor Lattimore and Marcus Hutter. No free lunch versus occam's razor in supervised learning. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence: Papers from the Ray Solomonoff 85th Memorial Conference, Melbourne, VIC, Australia, November 30–December 2, 2011*, pp. 223–235. Springer, 2013.

Leonid A Levin. Universal sequential search problems. *Problems of information transmission*, 9(3): 265–266, 1973.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, 2018.

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

Qian Li and Yuyi Wang. Constant bit-size transformers are turing complete. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=RBWnyDEBKf.

David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Tom McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *Advances in Neural Information Processing Systems*, 36, 2023.

Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew G Wilson. Pac-bayes compression bounds so tight that they can explain generalization. *Advances in Neural Information Processing Systems*, 35:31459–31473, 2022.

Sanae Lotfi, Marc Finzi, Yilun Kuang, Tim GJ Rudner, Micah Goldblum, and Andrew Gordon Wilson. Non-vacuous generalization bounds for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 32801–32818, 2024.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *Advances in neural information processing systems*, 30, 2017.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pp. 2408–2417, 2015.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations*, 2024a.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. In *NeurIPS 2024 Workshop on Mathematics of Modern Machine Learning*, 2024b.

Chris Mingard, Henry Rees, Guillermo Valle-Pérez, and Ard A Louis. Deep neural networks have an inbuilt occam's razor. *Nature Communications*, 16(1):220, 2025.

Preetum Nakkiran. Turing-universal learners with optimal scaling laws. *arXiv preprint arXiv:2111.05321*, 2021.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=O9YTt26r2P.

Franz Nowak, Anej Svete, Alexandra Butoi, and Ryan Cotterell. On the representational capacity of neural language models with chain-of-thought reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12510–12548, 2024.

Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. ISBN 978-0-201-53082-7.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1889–1903, 2023.

Karthik Abinav Sankararaman, Sinong Wang, and Han Fang. Bayesformer: Transformer with uncertainty estimation. *arXiv preprint arXiv:2206.00826*, 2022.

Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

Dale Schuurmans. Memory augmented large language models are computationally universal. *arXiv preprint arXiv:2301.04589*, 2023.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, 2018.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 922–938, 2021.

Peter Shaw, James Cohan, Jacob Eisenstein, Kenton Lee, Jonathan Berant, and Kristina Toutanova. Alta: Compiler-based analysis of transformers. *arXiv preprint arXiv:2410.18077*, 2024.

Ray J Solomonoff. A formal theory of inductive inference. *Information and control*, 7(1):1–22,224–254, 1964.

Nikita Tsoy and Nikola Konstantinov. Simplicity bias of two-layer networks beyond linearly separable data. *arXiv preprint arXiv:2405.17299*, 2024.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 183–196, 2020.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*, 2022.

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. *arXiv preprint arXiv:2310.16028*, 2023.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 14901–14911, 2025.

APPENDIX

A ADDITIONAL BACKGROUND

We first give an overview of relevant concepts related to Kolmogorov complexity and the Minimum Description Length (MDL) principle, and refer the reader to other resources for further reading, such as Li & Vitányi (2008), a reference text for Kolmogorov complexity and related topics. Other texts that cover these topics include Cover & Thomas (2006) and Hutter (2005). Some material is also covered in tutorials Grunwald & Vitanyi (2008) and Grunwald (2004). Other notable resources include Barron (1985) which discusses the Kolmogorov complexity of probability distributions, Rathmanner & Hutter (2011) which offers a more philosophical and intuitive discussion, and the introduction of Schmidhuber (1997) which offers a concise summary for a machine learning audience.

A.1 BINARY STRINGS

Let $\{0,1\}^*$ denote the set of finite *binary strings* or sequences, e.g.:

$$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}, \tag{14}$$

with ϵ denoting the empty string. We denote the *length* of a string $x \in \{0,1\}^*$ as $|x|$, e.g. $|0101| = 4$.

Note that there is a one-to-one correspondence between the set of binary strings and the natural numbers $\mathcal{N}$, e.g. equation 14 shows a standard enumeration. Therefore, there is also a one-to-one mapping between the elements of any countable set and binary strings.

A.2 CODES AND DESCRIPTION LENGTHS

We consider settings where a sender (e.g., Alice) wants to transmit some information to a receiver (e.g., Bob). Formally, consider the case where Alice wants to transmit an element from some countable set $\mathcal{X}$ to Bob by sending a message consisting of some binary string, such that Bob can reconstruct the element from the transmitted message. Thus, prior to communication, Alice and Bob must agree on a *code*, or *description method*, which can be characterized by a *decoding function* $D : \{0,1\}^* \to \mathcal{X}$. We refer to the binary strings in the domain of the decoding function as *codewords* (or *programs*, for reasons which will become clear in the following sections). For each code D, we can therefore associate a *description length measure*, or *complexity measure*, $L_D : \mathcal{X} \to \mathcal{N}$, where the complexity measure's value for every x is equal to the length of the shortest program that generates x:

$$\begin{aligned} L_D(x) &= \text{length of the shortest program } z \text{ such that } D(z) = x \\ &= \min_{z \,:\, D(z)=x} |z| \end{aligned} \tag{15}$$

Therefore, a code D establishes a *compression scheme*, where Alice can transmit an element $x \in \mathcal{X}$ to Bob using $L_D(x)$ bits.

Prefix codes and Kraft's inequality If Alice wants to send Bob a sequence of elements in $\mathcal{X}$, not all codes guarantee that such a sequence is *uniquely decodeable*, i.e. that the sequence can be unambiguously reconstructed when the messages for each of the elements are concatenated. This ambiguity can be resolved by considering a *prefix code*. A set $\mathcal{B} \subset \{0,1\}^*$ is *prefix-free* if no element in $\mathcal{B}$ is a prefix of any other element in $\mathcal{B}$. A decoding function specifies a *prefix code* if its domain is *prefix-free*. A common example of a prefix-free code is a *Huffman code*.

Kraft's inequality captures an important relation between description length measures and prefix codes. It states that there exists a prefix code D over $\mathcal{X}$ with description lengths L_D if and only if:

$$\sum_{\mathcal{X}} 2^{-L_D(x)} \leq 1. \tag{16}$$

Prefix codes and probability distributions Beyond the practical advantages of self-delimiting codes, a primary theoretical motivation for considering prefix-free codes is their close connection with probability distributions. Given a probability distribution $p(x)$ over $\mathcal{X}$, there exists a prefix-free code D for $\mathcal{X}$ such that:

$$L_D(x) = \lceil -\log_2 \; p(x) \rceil, \tag{17}$$

where we round up to form integer-length codewords. One method for constructing such a code is *Shannon-Fano* coding, where more likely elements are assigned shorter codewords. As we are primarily interested in theoretical *codelengths* rather than practical *codes*, we will typically omit the additive constant from this rounding and consider non-integer codelengths.

Universal prefix codes Let us consider the set of all computable prefix-free partial functions mapping from binary strings to elements of some set $\mathcal{X}$, denoted as $\mathcal{D}$. Each decoding function $D \in \mathcal{D}$ has a corresponding description length measure, L_D, given by equation 15. In general, L_D may be an unsatisfying complexity measure for elements of $\mathcal{X}$ due to its strong dependence on the potentially arbitrary choice of D. However, in the following section, we will show that there exists a subset of functions in $\mathcal{D}$, that are *universal* in the sense of offering *at least as much compression* as any other description length measure, up to an additive constant that does not depend on the complexity of the element being encoded.

A.3 KOLMOGOROV COMPLEXITY

The definition of *Kolmogorov complexity* (also called *algorithmic complexity*) and the proof of its universality come from Kolmogorov (1965), with similar formulations appearing independently in Solomonoff (1964) and Chaitin (1969).

Intuitively, we can think of the Kolmogorov complexity $K(x)$ of an object x as the shortest program written in some standard programming language (such as Python) that prints x. To make this notion more precise, Kolmogorov complexity is typically defined with respect to a special class of Turing machines, called prefix Turing machines. (With some effort, it could be defined with respect to any universal model of computation.)

Prefix Turing machines Prefix Turing machines are multi-tape Turing machines. As with all multi-tape Turing machines, the behavior of the machine is specified by a *transition function*. At each step of computation, the transition function takes as input the current state and the register values at current "head" position of any readable tape. The transition function can then optionally update the register values of any writeable tapes at their current "head" positions and optionally move each "head" to an adjacent register. The machine then updates the state for the next step or halts the computation.

The defining characteristic of a *prefix* Turing machine is a binary, unidirectional, read-only *program tape* with no blank symbols. The machine also consists of a bidirectional, read-write work tape. The machine also has at least one read-only input tape, and at least one write-only output tape.[5]

For a prefix Turing machine T, we abuse notation and write T to denote the partial function that the machine computes, i.e. where $y = T(x, z)$ denotes that the Turing machine T, with the input tape initialized with x and the program tape initialized with prefix z, runs for some number of steps before halting with y written to the output tape. The function is undefined for (x, z) that do not halt.

Universal Turing machines Notably, there exists a subset of prefix Turing machines that are *universal* in the sense of being able to simulate the computation of any other prefix Turing machine. Such a machine can effectively take as input the description of any other Turing machine, and emulate the computation of that machine.

Kolmogorov complexity The more commonly used notion of Kolmogorov complexity – used throughout this paper – is called *prefix* Kolmogorov complexity, to distinguish from *plain* Kolmogorov complexity, which lacks some of the desirable formal properties.

The (prefix) Kolmogorov complexity of a string $x \in \{0,1\}^*$ with respect to a universal prefix Turing machine T is:

$$K_T(x) = \min_{z \,:\, T(\epsilon,z)=x} |z|, \tag{18}$$

where ϵ denotes the empty string.

Similarly, the *conditional* Kolmogorov complexity of a string y given x is:

$$K_T(y|x) = \min_{z \,:\, T(x,z)=y} |z|. \tag{19}$$

Note that Kolmogorov complexity $K_T(x)$ satisfies Kraft's inequality because the set of halting programs is prefix-free.

Invariance theorem The *invariance theorem* is the key result that gives Kolmogorov complexity its *universal* property. Given any two *universal* prefix-free Turing machines T^1, T^2:

$$\forall x,\ |K_{T^1}(x) - K_{T^2}(x)| < c, \tag{20}$$

where c is a constant that depends on the choice of T^1 and T^2 but not on x, and therefore we write $\forall x,\ K_{T^1}(x) \stackrel{+}{=} K_{T^2}(x)$, recalling the notation from Section 2. The invariance theorem intuitively

[5]Note that in some constructions, programs and inputs are encoded on the same tape, although this detail only affects the resulting definition of Kolmogorov complexity by an additive constant.

follows from the result that a *universal Turing machine* can simulate the computation of any other Turing machine. Informally, the constant c captures the cost of constructing an *interpreter* for programs written for T^2 in the language of T^1. The result also extends to conditional Kolmogorov complexity.

Therefore, as we are primarily interested in inequalities that hold up to an additive constant, we write Kolmogorov complexity as $K(x)$ and drop the explicit dependence on a particular choice of reference machine.

Computability Kolmogorov complexity is formally uncomputable due to the halting problem. If we try to enumerate programs from shortest to longest, some may never halt. It is, however, upper semicomputable. Any program that halts and generates the given object provides an upper bound. This upper bound becomes increasingly tight as we run more programs for more steps.

Universality of Kolmogorov complexity As a corollary to the invariance theorem, Kolmogorov complexity is a *universal description length measure* as described in Section A.2, i.e. for any computable prefix-free decoding function D and corresponding description length measure L_D,

$$\forall x \;\; K(x) \overset{+}{\leq} L_D(x). \tag{21}$$

In other words, Kolmogorov complexity can be interpreted as a *compression scheme*, where Alice encodes a string x by the shortest program that generates x. This compression scheme offers *at least as much* compression of x, for any x, as any other computable prefix code up to an additive constant that does not depend on x. Thus, as the complexity of x increases, the additive constant becomes negligible, and all such universal description length measures are asymptotically equivalent. Conversely, any description length measure provides an upper bound on Kolmogorov complexity, up to an additive constant, however this bound may be arbitrarily loose for some objects.

Kolmogorov complexity of countable objects The definition of Kolmogorov complexity naturally extends to countable sets other than the non-binary strings. For example, for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are countable, we have:

$$K(y \mid x) = \min_{z \,:\, T(e_{\mathcal{X}}(x),z)=e_{\mathcal{Y}}(y)} |z|, \tag{22}$$

or ∞ if no such z exists, where $e_{\mathcal{X}}(x) : \mathcal{X} \to \{0,1\}^*$ and $e_{\mathcal{Y}}(y) : \mathcal{Y} \to \{0,1\}^*$ are one-to-one computable functions mapping from x and y to binary encodings, and T is a universal prefix Turing machine. While $K(y \mid x)$ therefore depends on a specific choice of $e_{\mathcal{X}}$ and $e_{\mathcal{Y}}$, we leave this implicit in the notation. Since $e_{\mathcal{X}}$ and $e_{\mathcal{Y}}$ are computable functions, the specific choice only affects $K(y \mid x)$ up to an additive constant that does not depend on y or x, similar to the choice of T.

For structured objects such as tuples, we can consider prefix Turing machines with multiple output tapes, in order to simplify the encoding of such objects. For such a universal prefix Turing machine T, we use the same notation as above, assuming that both $T(x, z)$ and $e_{\mathcal{Y}}$ output a tuple of strings with a number of elements equal to the number of output tapes. Again, such a choice only affects the resulting definition of Kolmogorov complex up to an additive constant, as the number of tapes does not affect the machine's expressive power (Papadimitriou, 1994).

Kolmogorov complexity of discrete functions We can extend the definition of Kolmogorov complexity to functions. This is relatively straightforward for a function $f : \mathcal{X} \to \mathcal{Y}$ if the domain $\mathcal{X}$ and range $\mathcal{Y}$ are countable such that the inputs and outputs of the function can be encoded as binary strings. Again, we choose a reference universal prefix Turing machine T.

$$K(f) = \min_{z \,:\, \forall x,\, T(e_{\mathcal{X}}(x),z)=e_{\mathcal{Y}}(f(x))} |z|, \tag{23}$$

or ∞ if no such z exists, which is the case if f is uncomputable, e.g. it solves the halting problem. As a direct analogue to the invariance theorem for Kolmogorov complexity of strings, the same property holds for the Kolmogorov complexity of functions. Extensions to real-valued functions are discussed in Appendix B.1.

Resource-bounded Kolmogorov complexity We can consider computable, resource-bounded approximations to Kolmogorov complexity (Li & Vitányi, 2008, section 7).

Consider a *resource bound* $R = (R_t, R_s) \in \mathbb{N}^2$, representing a bound on the time $R_t \in \mathbb{N}$ and $R_s \in \mathbb{N}$ space resources available.

Given a universal prefix Turing machine T, let $y = T_R(x, z)$ denote the partial function computed by T if the execution given x and z halts within R_t steps and uses at most R_s registers on each tape; otherwise $T_R(x, z)$ is undefined. We can then define time and space bounded Kolmogorov complexity with respect to T and R:

$$K_{T,R}(y \mid x) = \min_{z \,:\, T_R(x,z)=y} |z|, \tag{24}$$

or ∞ if no such z exists, with an analogous definition for functions.

The resource-bounded version lacks the universal properties of unbounded Kolmogorov complexity. However, $K_{T,R}(y \mid x)$ is monotonically non-increasing with respect to increasing R_t and R_s, decreasing to $K_T(y \mid x)$ as R_t and R_s go to ∞.

Coding theorem The *coding theorem* is a central result in algorithmic information theory (Li & Vitányi, 2008, Section 4), and relates to the universality of Kolmogorov complexity. A key result of the coding theorem is that if m is a lower semicomputable semimeasure, then:

$$K(x) \overset{+}{\leq} -\log_2 m(x). \tag{25}$$

where a *semimeasure* is a generalization of probability distribution, such that a semimeasure m over a domain $\mathcal{X}$ satisfies:

$$\sum_{x \in \mathcal{X}} m(x) \leq 1. \tag{26}$$

Note that because $K(x)$ is upper semicomputable and satisfies Kraft's inequality, $2^{-K(x)}$ is a lower semicomputable semimeasure.

This result also extends to conditional Kolmogorov complexity, and, as a direct analogue, to the Kolmogorov complexity of discrete functions.

B Additional Theoretical Analysis and Proofs

B.1 Rationale for Rational-valued Model Functions

Defining Kolmogorov complexity for a function $f : \mathcal{X} \to \mathbb{R}$ with a countable domain but real-valued range is slightly more complicated than for functions with countable (e.g. rational-valued) outputs, but we can accomplish this by considering the shortest program z for a given universal prefix Turing machine T that can approximate f to an arbitrary degree of precision $a \in N$ specified as an additional input:

$$K(f) = \min\{|z| : \forall x \in \mathcal{X}, a \in \mathbb{N} \;\; \left|T(e_{(\mathcal{X}\times\mathbb{N})}(x, a), z) - f(x)\right| \leq 1/a\}. \tag{27}$$

Given this definition, we could define two-part and variational codes with respect to model functions with real-valued logits, analogously to those defined with respect to model functions with rational-valued logits. However, this leads to significantly greater theoretical complexity with limited practical benefit, given that real-world neural networks represent logits with finite precision, and any real-valued function can be approximated to an arbitrary degree of precision by a rational-valued function.

B.2 Note on "Universal" Terminology

The term *universal* is overloaded in the context of codes and compression schemes, with different usages varying with respect to the scope (the class of codes/models being compared against) and the nature of the performance guarantee. Our usage of the term in this paper follows the use of the term in the algorithmic information theory literature to describe the *universality* of Kolmogorov

complexity with respect to any computable prefix code, up to an additive constant. We extend this notion to slightly restricted sets of prefix codes, i.e. the specific classes of two-part, Bayesian, and variational codes discussed in this paper.

In contrast, the term *universal code* also appears in the MDL literature (Grunwald, 2004) to describe codes that are universal with respect to a specific set of candidate codes, in the sense of offering as much compression as any candidate code, up to a factor that is typically logarithmic with respect to the size of the candidate set or data sample. The term "universal code" is also used to describe prefix codes for integers, such as Elias codes, which offer asymptotic guarantees relative to any other prefix code assuming a monotonically decreasing probability distribution over the integers.

B.3 Model Functions and Notation

Here we introduce standard encodings, $e_{\mathcal{X}}$ and $e_{\mathcal{L}}$, used to define the Kolmogorov complexity of model functions, $\mathcal{F} : \mathcal{X} \to \mathcal{L}$, introduced in Section 3. Let us denote the class of universal prefix Turing machines compatible with these encodings as $\mathcal{T}$, with the specific assumptions specified in the following paragraphs. Recall that the specific choice of universal prefix Turing machine, and encoding functions $e_{\mathcal{X}}$ and $e_{\mathcal{L}}$, only affect the resulting definition of Kolmogorov complexity by an additive term (A.3), which does not affect the main inequalities we are interested in proving, which hold up to an additive term. Therefore we choose encodings to simplify the construction of a function `zmap` introduced later (B.5), which generates Transformer parameters to emulate a prefix Turing machine.

Input encoding Let $\mathcal{V}$ be a vocabulary of input tokens. We assume some one-to-one encoding of inputs $e_{\mathcal{X}} : \mathcal{X} \to \mathcal{V}^*$ as a token sequence, which is represented on the input tape. For generality, we do not necessarily assume the input vocabulary is binary, and assume the input tape of any $T \in \mathcal{T}$ has a number of symbols greater than or equal to $|\mathcal{V}|$.

Output encoding Recall that the output of a model function is a tuple of rational-valued logits, with one logit for each element in the output space $\mathcal{Y}$, i.e. $\mathcal{L} = \mathbb{Q}^{|\mathcal{Y}|}$. We assume that any $T \in \mathcal{T}$ has $|\mathcal{Y}|$ write-once, unidirectional output tapes, with each logit $\in \mathbb{Q}$ encoded on a separate output tape in the following format. Let $s \in \{0, 1\}$ denote the sign of the logit, $\texttt{numerator} \in \mathbb{N}_0$ be the numerator and $\texttt{denominator} \in \mathbb{N}$ be the denominator. The logit is then encoded on the output tape as:

$$s, \overbrace{1, 1, \cdots, 1}^{\texttt{numerator}}, 0, \overbrace{1, 1, \cdots, 1}^{\texttt{denominator}-1}, \tag{28}$$

followed by blank symbols. Therefore, $e_{\mathcal{L}} : \mathbb{Q}^{|\mathcal{Y}|} \to (\{0, 1\}^*)^{|\mathcal{Y}|}$.

Shorthand notation We introduce the following shorthand notation related to prefix Turing machines and the encoding functions introduced above.

First, let us define a model function $f_T^z \in \mathcal{F}$ computed by $T \in \mathcal{T}$ with program z as follows:

$$f_T^z(x) = e_{\mathcal{L}}^{-1}\left(T(e_{\mathcal{X}}(x), z)\right). \tag{29}$$

Next, let us define the set of programs where $T \in \mathcal{T}$ halts with a valid output for any input:

$$\mathcal{Z}_T = \{z : \forall x \in \mathcal{X},\ T(e_{\mathcal{X}}(x), z) \in \mathrm{dom}(e_{\mathcal{L}})\}. \tag{30}$$

Additionally, let us denote the set of programs where $T \in \mathcal{T}$ halts with a valid output for any input *under resource bound* R as:

$$\mathcal{Z}_{T,R} = \{z : \forall x \in \mathcal{X},\ T_R(e_{\mathcal{X}}(x), z) \in \mathrm{dom}(e_{\mathcal{L}})\}. \tag{31}$$

Note that we can rewrite the definitions of Kolmogorov complexity with respect to these definitions.

$$K_T(f) = \min_{z \in \mathcal{Z}_T\,:\, f_T^z = f} |z|, \tag{32}$$

or ∞ if no such z exists, and similarly:

$$K_{T,R}(f) = \min_{z \in \mathcal{Z}_{T,R}\,:\, f_T^z = f} |z|, \tag{33}$$

or ∞ if no such z exists, where $f_T^z = f$ denotes that $\forall x \in \mathcal{X},\ f_T^z(x) = f(x)$.

B.4 Analysis of Two-Part Codes

Here we provide the proof of Proposition 1, along with supporting lemmas and related analysis of two-part codes.

B.4.1 Lemmas 1 and 2

We introduce the following lemmas to support our later proofs.

Definition 6 (description length of a model function)**.** *The* description length of a model function f *under code M, denoted $L_M^{\mathcal{F}}(f)$, is the codelength of the shortest hypothesis that maps to f:*

$$L_M^{\mathcal{F}}(f) := \min_{h \in \mathcal{H}_M \,:\, m_M(h)=f} -\log_2 \alpha_M(h). \tag{34}$$

Lemma 1. *The minimum codelength of a two-part code M can be expressed as a minimization over the set of model functions realized by M:*

$$L_M^{*,\text{two-part}}(Y|X) = \min_{f \in \mathrm{im}(m_M)} \left(L_M^{\mathcal{F}}(f) - \log_2 p(Y|X;f) \right), \tag{35}$$

where $\mathrm{im}(m_M)$ is the image of the mapping m_M, i.e., the set of all model functions f such that $f = m_M(h)$ for some $h \in \mathcal{H}_M$.

Proof. We start from the definition of the minimum achievable codelength, equation 3:

$$\begin{aligned} L_M^{*,\text{two-part}}(Y|X) &= \min_{h \in \mathcal{H}_M} L_M^{\text{two-part}}(Y|X;h) \\ &= \min_{h \in \mathcal{H}_M} \left(-\log_2 \alpha_M(h) - \log_2 p(Y \mid X; m_M(h)) \right). \end{aligned}$$

We can partition the hypothesis space $\mathcal{H}_M$ into disjoint sets, where each set contains all hypotheses that map to the same model function $f \in \mathrm{im}(m_M)$. The minimization over all $h \in \mathcal{H}_M$ can then be rewritten as a two-level minimization: first, for each function f, we minimize over all hypotheses that produce it, and second, we minimize over all possible functions f:

$$L_M^{*,\text{two-part}}(Y|X) = \min_{f \in \mathrm{im}(m_M)} \left(\min_{h \in \mathcal{H}_M \,:\, m_M(h)=f} \left(-\log_2 \alpha_M(h) - \log_2 p(Y \mid X; m_M(h)) \right) \right).$$

For the inner minimization, all hypotheses h map to the same function f. Therefore, the term $-\log_2 p(Y \mid X; m_M(h))$ is constant and equal to $-\log_2 p(Y \mid X; f)$. We can pull this constant term out of the inner minimization:

$$L_M^{*,\text{two-part}}(Y|X) = \min_{f \in \mathrm{im}(m_M)} \left(\left(\min_{h \in \mathcal{H}_M \,:\, m_M(h)=f} -\log_2 \alpha_M(h) \right) - \log_2 p(Y \mid X; f) \right).$$

By our definition in equation 34, the inner term is precisely the description length of the function f, $L_M^{\mathcal{F}}(f)$. Substituting this back gives the final result:

$$L_M^{*,\text{two-part}}(Y|X) = \min_{f \in \mathrm{im}(m_M)} \left(L_M^{\mathcal{F}}(f) - \log_2 p(Y \mid X; f) \right). \qquad \square$$

Lemma 2. *For any two-part code M and for all $f \in \mathcal{F}$:*

$$K(f) \overset{+}{\leq} L_M^{\mathcal{F}}(f). \tag{36}$$

Proof. First, we show that $2^{-L_M^{\mathcal{F}}(f)}$ is a semimeasure over $\mathcal{F}$.

Substituting and simplifying the definition of $L_M^{\mathcal{F}}(f)$, we have:

$$2^{-L_M^{\mathcal{F}}(f)} = \max_{h \in \mathcal{H}_M \,:\, m_M(h)=f} \alpha_M(h) \tag{37}$$

We now need to show:

$$\sum_{f \in \mathrm{im}\, m_M} \left(\max_{h \in \mathcal{H}_M \,:\, m_M(h)=f} \alpha_M(h) \right) \leq 1. \tag{38}$$

The summation is effectively over some subset of hypotheses in $\mathcal{H}_M$, i.e. those that offer the shortest description length for some function. As each element of the summation is non-negative, we have the following inequality:

$$\sum_{f \in \operatorname{im} m_M} \left(\max_{h \in \mathcal{H}_M \,:\, m_M(h) = f} \alpha_M(h) \right) \leq \sum_{f \in \operatorname{im} m_M} \left(\sum_{h \in \mathcal{H}_M \,:\, m_M(h) = f} \alpha_M(h) \right) = \sum_{h \in \mathcal{H}_M} \alpha_M(h),$$

and because $\alpha_M(h)$ is, by definition, a semimeasure over $\mathcal{H}_M$, we have:

$$\sum_{h \in \mathcal{H}_M} \alpha_M(h) \leq 1,$$

and therefore $2^{-L_M^{\mathcal{F}}(f)}$ is a semimeasure. Similarly, $2^{-L_M^{\mathcal{F}}(f)}$ is lower semicomputable, given that $\alpha_M(h)$ is by definition lower semicomputable.

Therefore, per the coding theorem, we have:

$$\begin{aligned} K(f) &\overset{+}{\leq} -\log_2 2^{-L_M^{\mathcal{F}}(f)} \\ K(f) &\overset{+}{\leq} L_M^{\mathcal{F}}(f). \end{aligned}$$

□

B.4.2 Proof of Proposition 1

Proposition (Proposition 1 restated)**.** *There exists a universal two-part code.*

Proof. We construct a two-part code U with respect to a universal prefix Turing machine $T \in \mathcal{T}$. The components of U are defined as follows:

- The hypothesis space $\mathcal{H}_U = \{0,1\}^*$ is the set of binary strings, i.e. programs.
- The mapping function $m_U : \mathcal{H}_U \to \mathcal{F}$ takes a program $h \in \mathcal{H}_U$ and maps it to a model function. Specifically, $m_U(h) = f_T^h$, i.e. the function computed by running the machine T with program h.
- The prior $\alpha_U(h)$ is defined as $2^{-|h|}$ for $h \in \mathcal{Z}_T$, where $|h|$ is the length of a halting program h. Otherwise, $\alpha_U(h) = 0$ if h does not halt for all inputs. The prior is therefore a lower semicomputable semimeasure.

By the definition of a universal Turing machine, the image of m_U is the set of all computable model functions, $\mathcal{F}$.

Using Lemma 1, we can analyze the description length of a function f under our code U. Recall that $L_M^{\mathcal{F}}(f)$ is the codelength of the shortest hypothesis that maps to f. For our code U, this becomes:

$$\begin{aligned} L_U^{\mathcal{F}}(f) &= \min_{h \in \mathcal{H}_U \,:\, m_U(h) = f} (-\log_2 \alpha_U(h)) \\ &= \min_{h \in \{0,1\}^* \,:\, m_U(h) = f} \left(-\log_2 2^{-|h|} \right) \\ &= \min_{h \in \{0,1\}^* \,:\, f_T^h = f} |h|. \end{aligned} \tag{39}$$

The final expression is, by definition, the Kolmogorov complexity of the function f with respect to machine T, denoted $K_T(f)$. By the invariance theorem, $L_U^{\mathcal{F}}(f) \overset{+}{=} K(f)$. By Lemma 2, for any other two-part code M, we have $K(f) \overset{+}{\leq} L_M^{\mathcal{F}}(f)$. Therefore, combining these results, we have a key inequality that holds for any two-part code M and any function f:

$$L_U^{\mathcal{F}}(f) \overset{+}{=} K(f) \overset{+}{\leq} L_M^{\mathcal{F}}(f). \tag{40}$$

Finally, we show that U is a universal two-part code. We must prove that for any other code M, $L_U^{*,\text{two-part}}(Y \mid X) \overset{+}{\leq} L_M^{*,\text{two-part}}(Y \mid X)$.

Let f_M^* be any model function that minimizes the codelength for code M on a given dataset (X, Y), re-written as a minimum over model functions, per Lemma 1, such that:

$$L_M^{*,\text{two-part}}(Y \mid X) = L_M^{\mathcal{F}}(f_M^*) - \log_2 p(Y|X; f_M^*). \tag{41}$$

The minimum codelength for our universal code U is, by definition, no greater than the codelength achieved by using the specific function f_M^*:

$$L_U^{*,\text{two-part}}(Y \mid X) \leq L_U^{\mathcal{F}}(f_M^*) - \log_2 p(Y|X; f_M^*). \tag{42}$$

Using our key inequality from equation 40, we know that $L_U^{\mathcal{F}}(f_M^*) \overset{+}{\leq} L_M^{\mathcal{F}}(f_M^*)$. Substituting this into the right-hand side of equation 42 yields:

$$L_U^{*,\text{two-part}}(Y \mid X) \overset{+}{\leq} L_M^{\mathcal{F}}(f_M^*) - \log_2 p(Y|X; f_M^*).$$

Now, substituting equation 41 into the right-hand side of the expression above gives us our final result:

$$L_U^{*,\text{two-part}}(Y \mid X) \overset{+}{\leq} L_M^{*,\text{two-part}}(Y \mid X). \tag{43}$$

Since this holds for any two-part code M, we have shown that U is a universal two-part code. □

B.4.3 Discussion of Universal Two-Part Codes

Notably, satisfying the conditions of a universal two-part code does not require that $L_M(h) \overset{+}{\leq} K(h)$ for all hypotheses. This is particularly notable for neural networks, where hypothesis spaces are typically highly redundant – many different parameter sets (hypotheses) compute the same model function. Our definition allows for some of these parameter sets to be inefficiently described, with a description length far exceeding their own Kolmogorov complexity. Universality is maintained as long as for any given model function, at least one of its corresponding parameter sets is described efficiently (with description length equal to the function's Kolmogorov complexity, up to some additive constant). Therefore, we can devise universal two-part codes without necessarily needing to devise description length measures that optimally compress every possible hypothesis, a potentially significant challenge for the vast and redundant parameter spaces of neural networks.

On the other hand, because some (or even most) hypotheses may be coded quite inefficiently, a universal two-part code would not necessarily be useful for post-hoc model selection across models that were not trained to optimize the given two-part description length objective.

B.4.4 Proof of Corollary 1

Corollary (Corollary 1 restated)**.** *The minimum of any universal two-part code M is equal to the following bound, denoted $C^{\text{two-part}}(Y \mid X)$, up to an additive term:*

$$C^{\text{two-part}}(Y \mid X) := \min_{f \in \mathcal{F}} K(f) - \log_2\ p(Y \mid X; f) \overset{+}{=} L_M^{*,\text{two-part}}(Y \mid X). \tag{44}$$

Proof. Let U be the specific universal two-part code constructed in the proof of Proposition 1. Its minimum codelength is given by:

$$L_U^{*,\text{two-part}}(Y \mid X) = \min_{f \in \mathcal{F}} \left(L_U^{\mathcal{F}}(f) - \log_2 p(Y|X; f) \right). \tag{45}$$

In that proof, we established that, $L_U^{\mathcal{F}}(f) \overset{+}{=} K(f)$. Substituting this into the equation above directly shows that the minimum codelength of U is equivalent to the bound $C^{\text{two-part}}(Y \mid X)$:

$$L_U^{*,\text{two-part}}(Y \mid X) \overset{+}{=} \min_{f \in \mathcal{F}} \left(K(f) - \log_2 p(Y|X; f) \right) = C^{\text{two-part}}(Y \mid X). \tag{46}$$

Now, let M be any other universal two-part code. By the definition of a universal two-part code, its minimum codelength must be equivalent to that of U, i.e., $L_M^{*,\text{two-part}}(Y \mid X) \overset{+}{=} L_U^{*,\text{two-part}}(Y \mid X)$. It therefore follows that the minimum codelength for any universal code M is equivalent to the bound. □

B.4.5 Proof of Proposition 2

Proposition (Proposition 2 restated)**.** *Given an asymptotically optimal family of two-part codes* $\{M_R \mid R \in \mathbb{N}^2\}$*:*

$$\lim_{R_t, R_s \to \infty} L^{*,\textit{two-part}}_{M_R}(Y \mid X) \stackrel{+}{=} C^{\textit{two-part}}(Y \mid X), \tag{47}$$

with the bound $C^{\textit{two-part}}_{T,R}(Y \mid X)$ *monotonically non-increasing with increasing* R_t *or* R_s*.*

Proof. The proof has two parts: establishing the limit and showing monotonicity.

First, for the upper bound, the definition of an asymptotically optimal family states that for any resource bound R:

$$L^{*,\text{two-part}}_{M_R}(Y \mid X) \stackrel{+}{\leq} C^{\text{two-part}}_{T,R}(Y \mid X) = \min_{f \in \mathcal{F}} \left(K_{T,R}(f) - \log_2 p(Y \mid X; f) \right). \tag{48}$$

As the resource bounds $R_t, R_s \to \infty$, the resource-bounded Kolmogorov complexity $K_{T,R}(f)$ converges to the standard Kolmogorov complexity $K_T(f)$. By the invariance theorem, $K_T(f) \stackrel{+}{=} K(f)$, so the bound converges to the universal two-part codelength:

$$\lim_{R_t, R_s \to \infty} C^{\text{two-part}}_{T,R}(Y \mid X) \stackrel{+}{=} C^{\text{two-part}}(Y \mid X). \tag{49}$$

This establishes the upper bound for our limit:

$$\lim_{R_t, R_s \to \infty} L^{*,\text{two-part}}_{M_R}(Y \mid X) \stackrel{+}{\leq} C^{\text{two-part}}(Y \mid X). \tag{50}$$

For the lower bound, we know from Corollary 1 that $C^{\text{two-part}}(Y \mid X)$ is the universal lower bound for any two-part code. Thus, for any R, $L^{*,\text{two-part}}_{M_R}(Y \mid X) \stackrel{+}{\geq} C^{\text{two-part}}(Y \mid X)$, which must also hold in the limit.

Since the limit is both upper- and lower-bounded by $C^{\text{two-part}}(Y \mid X)$ up to an additive constant, the equivalence holds.

Now we focus on showing monotonicity. Let R and R' be two resource bounds such that R' provides at least as many resources as R (i.e., $R'_t \geq R_t$, $R'_s \geq R_s$). Any program that is computable within bounds R is also computable within bounds R'. This implies that for any function f, $K_{T,R'}(f) \leq K_{T,R}(f)$. Consequently, the bound on the codelength is monotonic: $C^{\text{two-part}}_{T,R'}(Y \mid X) \leq C^{\text{two-part}}_{T,R}(Y \mid X)$. □

B.5 Details of zmap

We use the ALTA compiler (Shaw et al., 2024) to construct (in Python) a function $\texttt{zmap}(T, R, z)$ that generates Transformer weights such that the Transformer emulates the model function computed by a prefix Turing machine T with program tape contents z under a resource bound R. Formally, we construct zmap to satisfy the following condition for all $x \in \mathcal{X}$, $R \in \mathbb{N}^2$, $z \in \mathcal{Z}_{T,R}$, and $T \in \mathcal{T}$:

$$m_R\left(\,\texttt{zmap}(T, R, z)\,\right) = f^z_T, \tag{51}$$

where:

- The mapping function m_R prepends R_s "prompt tokens" to the input and then runs the forward pass of the Transformer with parameters h, outputting the unnormalized logits prior to the final softmax. Recall that $R_s \in \mathbb{N}$ is a space resource bound.
- The output of zmap is a set of Transformer weights, with the particular model dimensions depending on T and R.
- Recall $f^z_T \in \mathcal{F}$ is the model function from $\mathcal{X}$ to $\mathcal{L}$ computed by prefix Turing machine $T \in \mathcal{T}$ with program z, where $\mathcal{T}$ is a class of universal prefix Turing machines which assumes specific encodings for $\mathcal{X}$ and $\mathcal{L}$ on the input and output tapes, $e_{\mathcal{X}}$ and $e_{\mathcal{L}}$, respectively. Also recall that $\mathcal{Z}_{T,R}$ is the set of programs for T that halt with a valid output under resource bounds R for all inputs. (See B.3)

In this section we detail the construction of zmap and these supporting constructions, which are later used to prove the paper's key theorems.

B.5.1 TRANSFORMER INPUT PREPROCESSING

As described above, the mapping function m_R involves prepending R_s prompt tokens, and then running the forward pass of the Transformer. Given input tokens $e_{\mathcal{X}}(x) = (x_1, x_2, \cdots, x_{|x|}) \in \mathcal{V}^*$, we prepend a sequence of R_s prompt tokens $(p_1, p_2, \cdots, p_{R_s})$ from a separate vocabulary of size R_s. We then form the Transformer input sequence (prior to the embedding lookup) as follows:

$$\texttt{START}, p_1, p_2, \cdots, p_{R_s}, \texttt{SEP}, x_1, x_2, \cdots, x_{|x|}, \texttt{END}, \tag{52}$$

where START, SEP, and END are special reserved tokens.

B.5.2 EMULATING SINGLE-TAPE TURING MACHINES

The function zmap generates Transformer weights that emulate a multi-tape prefix Turing machine, including decoding of the output tapes to a set of logits. Before we describe zmap, we start with a simpler explanation of how a Transformer can emulate a single-tape Turing machine, by giving code for an ALTA program. We refer the reader to Shaw et al. (2024) for an explanation of the ALTA language and compiler.

```
class TransitionIn:
  state: int
  head_symbol: int

class Move(enum.Enum):
  LEFT = enum.auto()
  RIGHT = enum.auto()

class TransitionOut:
  state: int
  symbol: int | None
  move: Move | None
  halt: bool = False

TransitionFn = Callable[[TransitionIn], TransitionOut]

class MachineSpec:
  transition_fn: TransitionFn  # Turing machine transition function.
  num_states: int  # Turing machine number of states.
  num_symbols: int  # Number of tape symbols.
  num_steps: int  # Time resource bound.
  num_registers: int  # Space resource bound.
```

Figure 4: Transition function definition for standard single-tape Turing machine.

To start, we define data structures to represent the transition function of a single-tape Turing machine in Figure 4. Next, in Figure 5, we include a function that, given a transition function, defines an ALTA program that can be compiled to a Transformer, which emulates the provided Turing machine. The Transformer expects, as input, a sequence representing the initial state of the machine's tape.

While there are potentially many ways to emulate a Turing machine in a Transformer, representing each register at a different input position, and using relative position representations to facilitate shifting the attention head to the right or left, is a relatively straightforward implementation that efficiently leverages the Transformer's element-wise weight sharing and self-attention mechanism.

B.5.3 DEFINING ZMAP

The sketch of the ALTA program for zmap roughly follows the program for a single-tape Turing machine from B.5.2, extended to handle multiple tapes: the input, program, work, and output tapes.

To facilitate "decoding", the output tape registers are represented differently in the Transformer than the registers for other tapes. Recall the encoding of the logits on the output tapes as specified in B.3. The first bit written to the Turing machine's tape specifies the sign of the logit, and therefore for this first bit we set the value of a binary "sign" variable. The number of subsequent 1s written to

```python
from alta import program_builder as pb

def build_turing_machine_program_spec(spec: MachineSpec) -> pb.ProgramSpec:
  """Returns ALTA program spec for a Turing machine emulator."""

  variables = {
      "halted": pb.var(2),
      "state": pb.var(spec.num_states),
      "symbol": pb.input_var(spec.num_symbols),
      "head": pb.input_var(2, init_fn=lambda x: x == START),
      "one": pb.var(2, default=1),  # Constant for queries.
  }
  attention_heads = {
      # Read the symbol at the current head position.
      "head_symbol": pb.qkv("one", "head", "symbol"),
      # Identify tape cells to the left and right of the head.
      "head_left": pb.v_relative("head", -1),
      "head_right": pb.v_relative("head", 1),
  }

  def ffn_fn(x):
    """FFN function for Turing machine emulator."""
    if x["halted"]:
      return

    # Get the next action from the state transition function.
    output = spec.transition_fn(
        TransitionIn(state=x["state"], head_symbol=x["head_symbol"])
    )
    x["state"] = output.state
    if output.halt:
      x["halted"] = 1

    # Write a new symbol at current head position if specified.
    if x["head"] and output.symbol is not None:
      x["symbol"] = output.symbol

    # Move the tape head to left or right if specified.
    if output.move is not None and x["head"]:
      x["head"] = 0
    if output.move == Move.RIGHT and x["symbol"] != START and x["head_left"]:
      x["head"] = 1
    elif output.move == Move.LEFT and x["symbol"] != END and x["head_right"]:
      x["head"] = 1

  return pb.program_spec(
      ffn_fn=ffn_fn, variables=variables, heads=attention_heads,
      output_name="symbol", input_range=spec.num_symbols + 2, position_range=None,
  )
```

Figure 5: ALTA program specification for emulating a single-tape Turing machine.

the Turing machine's tape specifies the numerator. For each 1 we increment a numerical variable "sum" by $+1$ or -1 depending on the "sign" variable. This "sum" variable is only non-zero at the START position. After a 0 is written to the Turing machine's tape, we transition to incrementing the denominator. This is represented by changing a binary "key" variable from 0 to 1 at subsequent positions each time the transition function specifies that a 1 should be written (the "key" value is also initialized to 1 at START). Finally, once the Turing machine has halted, an attention head attends to each position where "key" is 1, and then averages over the "sum" variable, which is non-zero and equal to the numerator at exactly one position. The output of the attention head is therefore the logit value, equivalent to the value decoded from the Turing machine's corresponding output tape given the encoding function.

The pseudocode for both zmap and the corresponding mapping function are given in Figure 6. The PrefixMachineSpec specifies the transition function for a prefix Turing machine $T \in \mathcal{T}$ as well

```
InputSequence = list[int]
Logits = list[float]
ModelFunction = Callable[[InputSequence], Logits]

def zmap(machine_spec: PrefixMachineSpec,
         program_prefix: list[int]) -> TransformerParameters:
  alta_program_spec = build_prefix_turing_machine_program_spec(
      machine_spec, program_prefix)
  return alta_compiler(alta_program_spec)

def mapping_fn(
    machine_spec: PrefixMachineSpec,
    params: TransformerParameters,
) -> ModelFunction:
  def model_fn(input_tokens: InputSequence) -> Logits:
    # Prepend prompt and add special tokens.
    transformer_input = preprocess_input(machine_spec, input_tokens)
    return run_transformer(params, machine_spec, transformer_input)
  return model_fn
```

Figure 6: Pseudocode for `zmap` and corresponding mapping function.

as resource bounds R. We omit the full implementation of the program specification for brevity. We verified the correctness of `zmap` using unit tests.

B.5.4 ALTA Compiler Modifications

We make several modifications to the original ALTA compiler detailed in Shaw et al. (2024) to reduce the number of compiled weights, which is necessary to support our later proofs. ALTA programs represent the behavior of the MLP sub-layer as a set of *MLP rules*. Each MLP rule compiles to a single dimension of the weight matrices in the MLP sub-layer.

- The original ALTA compiler represents all categorical variables as one-hot vectors. We introduce *binary variables*, which function similarly to categorical variables with a range of 2, but are represented in a single activation dimension, as either 0 or 1. This leads to one necessary modification to constructing the first MLP weight matrix in the case where a MLP rule needs to match against a binary variable being 0. In this case, we include a -1 at the index corresponding to the binary variable. This is particularly useful to represent each bit of the program tape as a single weight.
- We don't require specifying fixed buckets for discretizing numerical variables. This means that numerical variables cannot be used as constraints in MLP rules, but this is not required for the `zmap` program. This also means that the number of weights in the MLP layer does not need to scale with the number of values the numerical variable can take on, which is useful given that the scale of the "sum" variables representing the logit numerators can grow arbitrarily.
- We add support for a MLP rule to increment a numerical variable by a fixed scalar. This is relatively straightforward to implement by including the fixed scalar at the index corresponding to the numerical variable in the second MLP weight matrix. This enables us to increment the "sum" variables by $+1$ or -1, with 2 MLP rules covering both cases, regardless of the input value of the variable, by leveraging the Transformer's residual connection.
- We enable specifying multiple output variables for the Transformer. This way, the output projection can select the $|\mathcal{Y}|$ variables corresponding to the outputs of the attention heads that compute the final $|\mathcal{Y}|$ logit values.

B.5.5 Compiled Weights

The weights compiled by `zmap` have the following properties:

- All layers share the same weights, as in a Universal Transformer (Dehghani et al., 2019) or Looped Transformer (Giannou et al., 2023).

- The minimum dimensionality of the activations in the Transformer scales linearly with the number of input ($|\mathcal{V}|$) and output ($|\mathcal{Y}|$) symbols and the number of Turing machine states (specified by $T \in \mathcal{T}$), and does not grow with increasing resource bounds. Therefore, the number of weights in invariant to to the time resource bound R_t.
- The minimum number of attention heads similarly scales linearly with the number of output symbols, as each output tape requires 3 attention heads, and there is one output tape per output symbol. A total of 7 additional heads are needed for the program, input, and work tapes. However, the minimum number of attention heads does not scale with any resource bound.
- The minimum hidden dimension of the MLP layer is determined by the complexity of Turing machine's transition function, but similarly does not scale with any resource bound.
- The Transformer requires only relative position representations, and no absolute position representations. It requires separate bias terms only for relative positions -1 and $+1$, i.e. to attend to positions immediately to the left or right.
- The Transformer's input embedding table has $|\mathcal{V}| + R_s + 3$ rows, i.e. embeddings for `START`, `SEP`, `END`, each token in the input vocabulary $\mathcal{V}$, and R_s prompt tokens, where R_s is the space resource bound. As the embeddings for the prompt tokens therefore are the only weights that scale as a function of any resource bound, we discuss these weights specifically in the following section.

Therefore, the number of necessary weights, i.e. the hypothesis space, is a function of T and R, but not z. The weight dimensions are minimums because the compiled weights can always be padded with 0s or -1s. Also note that Transformers compiled by ALTA exactly compute a symbolic program in the limit as the configurable attention matrix scalar goes to ∞, such that the compiled Transformers implement "hard attention" (Pérez et al., 2021). All weight values belong to a small set of unique values, and therefore require only finite precision to represent.

B.5.6 Prompt Embeddings

The prompt token embeddings for prompt tokens $p_1, p_2, \cdots, p_{R_s}$ produced by `zmap` for a given program $z = z_1, z_2, \cdots, z_{|z|}$ are specified as follows:

$$\begin{matrix} p_1 \\ p_2 \\ \vdots \\ p_{R_s} \end{matrix} \begin{bmatrix} b_1^z & w_1 & \cdots & w_{|w|} \\ b_2^z & w_1 & \cdots & w_{|w|} \\ \vdots & \vdots & \ddots & \vdots \\ b_{R_s}^z & w_1 & \cdots & w_{|w|} \end{bmatrix} \tag{53}$$

where $(w_1, \cdots, w_{|w|})$ is a finite-length weight vector that does not depend on z or any resource bound, and the weights $(b_1^z, \cdots, b_{R_s}^z)$ encode the $|z|$ bits of z in $R_s > |z|$ weights as follows:

$$b_i^z = \begin{cases} 1.0, & i <= |z| \wedge z_i = 1 \\ -1.0, & i <= |z| \wedge z_i = 0 \\ r, & i > |z|, \end{cases} \tag{54}$$

where r can be chosen arbitrarily, as these weights represent bits on the program tape that, by construction, will never be read, because by definition the program halts after reading the final bit of z and does not move the head for the unidirectional program tape any further. In the Transformer, this results in the attention head that "reads" the program tape never attending to these values. Therefore, we arbitrarily choose $r = 0.0$.

B.5.7 Resource Bounds Discussion

Since the compiled weights are the same across all layers, the resource bound R_t denoting the maximum number of Turing machine steps – and therefore the maximum number of Transformer layers – does not affect the compilation. The Transformer requires $R_t + 2$ layers at inference time, as it executes the Turing machine's transition function once per layer, and then requires 2 additional layers to compute the final logit values as described above. The registers of the initially blank work

and output tapes are also represented by the prompt tokens, and therefore the number of prompt tokens determines both the maximum program length and maximum number of registers on the work and output tapes.

Notably, because only the number of prompt tokens grows with increasing space resource bound R_s, and not any of the other aspect of the Transformer, as this bound becomes larger an increasing proportion of the Transformer's overall weights are allocated to embeddings of prompt tokens, rather than, e.g. the weights of the MLP layer. This is quite different from how Transformer parameter counts are conventionally scaled. It's possible there are alternative ways of emulating a prefix Turing machine that would lead to different scaling behavior, e.g. by effectively representing the program within the MLP layer, but we leave consideration of this to future work.

Future work could also consider separate resource bounds for the maximum number of program tape registers (bounding the model's capacity to represent increasingly complex functions) and the number of work tape registers (bounding the model's memory capacity during the forward pass).

B.6 Proof of Theorem 1

Theorem (Theorem 1 restated)**.** *There exists an* asymptotically optimal *family of two-part codes for Transformer encoders.*

Proof. Our proof builds on the construction of the function `zmap`, detailed in B.5, which generates Transformer parameters satisfying equation 51.

Let $T \in \mathcal{T}$ be a universal prefix Turing machine. We can define a family of two-part codes $\{U_R \mid R \in \mathbb{N}^2\}$ that is asymptotically optimal with respect to T as follows.

- The hypothesis space $\mathcal{H}_{U_R}$ is the Transformer parameter space specified by the codomain of `zmap` given T and R.
- The mapping function m_{U_R} is the mapping function described in B.5, where $m_{U_R}(h)$ consists of prepending R_s prompt tokens and running a Transformer forward pass with parameters h.
- We can trivially define the prior as $\alpha_{U_R}(h) = 2^{-|z|}$ if there exists $z \in \mathcal{Z}_{T,R}$ such that $h = \texttt{zmap}(T, R, z)$, or 0 otherwise. This prior therefore assigns non-zero probability only to those specific Transformer parameters that correspond to a valid Turing machine emulation.

Now, we must show that this family is asymptotically optimal with respect to T, meaning we must prove that for any resource bound $R \in \mathbb{N}^2$ and dataset (X, Y):

$$L^{*,\text{two-part}}_{U_R}(Y \mid X) \overset{+}{\leq} C^{\text{two-part}}_{T,R}(Y \mid X). \tag{55}$$

Using Lemma 1, we can express the codelength on the left as a minimum over functions:

$$L^{*,\text{two-part}}_{U_R}(Y \mid X) = \min_{f \in \text{Im}(m_{U_R})} \left(L^{\mathcal{F}}_{U_R}(f) - \log_2 p(Y|X; f) \right). \tag{56}$$

The function description length, $L^{\mathcal{F}}_{U_R}(f)$, is the codelength of the shortest hypothesis that computes f. Given our prior, a hypothesis h has a finite codelength only if it is the output of `zmap` for some program $z \in \mathcal{Z}_{T,R}$. Therefore, the minimization is effectively over programs z:

$$\begin{aligned} L^{\mathcal{F}}_{U_R}(f) &= \min_{h \in \mathcal{H}_{U_R} \,:\, m_{U_R}(h) = f} -\log_2 \alpha_{U_R}(h) \\ &= \min_{z \in \mathcal{Z}_{T,R} \,:\, m_{U_R}(\texttt{zmap}(T,R,z)) = f} -\log_2 2^{-|z|} \\ &= \min_{z \in \mathcal{Z}_{T,R} \,:\, f^z_T = f} |z|. \end{aligned}$$

The final step follows from equation 51, and this final expression is precisely the definition of the resource-bounded Kolmogorov complexity of f with respect to T, i.e., $K_{T,R}(f)$. So, we have shown that $L^{\mathcal{F}}_{U_R}(f) = K_{T,R}(f)$.

Substituting this equality back into the expression for the total codelength, we find:

$$\begin{aligned} L^{*,\text{two-part}}_{U_R}(Y \mid X) &= \min_{f \in \text{im}(m_{U_R})} \left(K_{T,R}(f) - \log_2 p(Y|X; f)\right) \\ &= C^{\text{two-part}}_{T,R}(Y \mid X). \end{aligned}$$

This step holds because the image of our constructed mapping function includes all functions computable by T within bounds R.

Since we have shown that $L^{*,\text{two-part}}_{U_R}(Y \mid X)$ is not just bounded by, but is *equal* to $C^{\text{two-part}}_{T,R}(Y \mid X)$, the condition for being an asymptotically optimal family is satisfied. □

B.7 Relations Between Codelengths

In this section we establish upper and lower bounds on the minimums of various classes of codes. To this end, we review Bayesian codes, and establish a universal lower bound on the minimums of Bayesian and variational codes. These results are used to prove Proposition 3.

B.7.1 Bayesian Codes

Under a Bayesian code, labels are encoded according to their likelihood under a Bayesian posterior distribution, defined below. A Bayesian code M, just like a two-part code, is specified by a hypothesis space $\mathcal{H}_M$, a mapping m_M from hypothesis to model functions, and a lower semicomputable semimeasure (i.e., prior) $\alpha_M(h)$.

The Bayesian codelength is defined with respect to the Bayesian marginal likelihood of the data:

$$L^{*,\text{bayes}}_M(Y \mid X) = -\log_2 p^{\text{marginal}}_M(Y \mid X) \tag{57}$$

$$= -\log_2 \sum_{h \in \mathcal{H}_M} p(Y \mid X; m_M(h))\, \alpha_M(h). \tag{58}$$

While the summation over all hypotheses is typically intractable, the Bayesian codelength serves as a theoretical lower bound and motivation for variational codes.

B.7.2 Universal Bayesian Codes

Analogously to two-part codes, we can show that there exists an equivalence class of universal Bayesian codes.

Definition 7 (universal Bayesian code)**.** *A Bayesian code M^1 is a* universal Bayesian code *if and only if, for any other Bayesian code M^2 and for all X, Y:*

$$L^{*,\textit{bayes}}_{M^1}(Y \mid X) \overset{+}{\leq} L^{*,\textit{bayes}}_{M^2}(Y \mid X). \tag{59}$$

Lemma 3. *There exists a universal Bayesian code.*

Proof. Consider a Bayesian code U defined identically to the two-part code in the proof of Proposition 1, with respect to universal prefix Turing machine T. We will show that U is a universal Bayesian code.

Similarly to Lemma 1, we can re-write the codelength for any Bayesian code M as a sum over model functions rather than hypotheses.

$$L^{\text{bayes}}_M(Y \mid X) = -\log_2 \sum_{h \in \mathcal{H}_M} p(Y \mid X; m_M(h))\, \alpha_M(h) \tag{60}$$

$$= -\log_2 \sum_{f \in \text{im}(m_M)} p(Y \mid X; f)\, \alpha^{\mathcal{F}}_M(f), \tag{61}$$

where:

$$\alpha^{\mathcal{F}}_M(f) = \sum_{h \in \mathcal{H}_M \,:\, m_M(h) = f} \alpha_M(h) \tag{62}$$

or 0 if no such h exists. Similarly to Lemma 2, we have $K(f) \stackrel{+}{\leq} -\log_2 \alpha_M^{\mathcal{F}}(f)$, because $\alpha_M^{\mathcal{F}}(f)$ is a lower semicomputable semimeasure, as by definition $\alpha_M(h)$ is a lower semicomputable semimeasure. Additionally, for our proposed universal code, the quantity $\alpha_U^{\mathcal{F}}(f)$ is recognizable as the *algorithmic probability* of f (Li & Vitányi, 2008), and we have $K(f) \stackrel{+}{=} -\log_2 \alpha_U^{\mathcal{F}}(f)$. We have already shown one side of this inequality above. The other side, $-\log_2 \alpha_U^{\mathcal{F}}(f) \stackrel{+}{\leq} K(f)$ holds because by definition $\alpha_U^{\mathcal{F}}(f)$ includes $2^{-K_T(f)}$ as an element in the summation over non-negative elements. Combining these results, we have:

$$-\log_2 \alpha_U^{\mathcal{F}}(f) \stackrel{+}{=} K(f) \stackrel{+}{\leq} -\log_2 \alpha_M^{\mathcal{F}}(f). \tag{63}$$

Therefore, there exists some positive constant c_M that does not depend on f such that for all f:

$$\alpha_U^{\mathcal{F}}(f) \geq c_M * \alpha_M^{\mathcal{F}}(f). \tag{64}$$

We need to show that the following holds for any other Bayesian code M:

$$L_U^{*,\text{bayes}}(Y \mid X) \stackrel{+}{\leq} L_M^{*,\text{bayes}}(Y \mid X). \tag{65}$$

We can re-write the left side as:

$$\begin{aligned}
L_U^{*,\text{bayes}}(Y \mid X) &= -\log_2 \sum_{f \in \mathcal{F}} p(Y \mid X; f) \alpha_U^{\mathcal{F}}(f) && (66)\\
&\stackrel{+}{\leq} -\log_2 \sum_{f \in \mathcal{F}} p(Y \mid X; f) c_M * \alpha_M^{\mathcal{F}}(f) && (67)\\
&= -\log_2 c_M - \log_2 \sum_{f \in \mathcal{F}} p(Y \mid X; f) \alpha_M^{\mathcal{F}}(f) && (68)\\
&\stackrel{+}{\leq} -\log_2 \sum_{f \in \mathcal{F}} p(Y \mid X; f) \alpha_M^{\mathcal{F}}(f). && (69)
\end{aligned}$$

This expression is the definition of $L_M^{*,\text{bayes}}(Y \mid X)$, and therefore we have demonstrated that U is a universal Bayesian code. □

Corollary 2. *For any universal Bayesian code M:*

$$C^{\textit{bayes}}(Y \mid X) := -\log_2 \sum_{f \in \mathcal{F}} 2^{-K(f)} p(Y \mid X; f) \stackrel{+}{=} L_M^{*,\textit{bayes}}(Y \mid X) \tag{70}$$

Proof. Let U be the universal Bayesian code constructed in the proof of Lemma 2. Its minimum codelength is equivalent to $C^{\text{bayes}}(Y|X)$, shown by rewriting the sum over hypotheses as a sum over functions and applying our previous result that $K(f) \stackrel{+}{=} -\log_2 \alpha_U^{\mathcal{F}}(f)$.

$$\begin{aligned}
L_U^{*,\text{bayes}}(Y \mid X) &= -\log_2 \sum_{f \in \mathcal{F}} p(Y \mid X; f) \alpha_U^{\mathcal{F}}(f)\\
&\stackrel{+}{=} -\log_2 \sum_{f \in \mathcal{F}} p(Y \mid X; f) 2^{-K(f)}\\
&= C^{\text{bayes}}(Y \mid X).
\end{aligned}$$

Now, let M be any other universal Bayesian code. By definition, $L_M^{*,\text{bayes}}(Y|X) \stackrel{+}{=} L_U^{*,\text{bayes}}(Y|X)$. It directly follows that $L_M^{*,\text{bayes}}(Y|X) \stackrel{+}{=} C^{\text{bayes}}(Y|X)$. □

B.7.3 Universal Bayesian Codes and Universal Two-Part Codes

Lemma 4. *The minimum of any universal Bayesian code is less than or equal to the minimum of any universal two-part code up to an additive term, i.e.:*

$$C^{\textit{bayes}}(Y \mid X) \stackrel{+}{\leq} C^{\textit{two-part}}(Y \mid X) \tag{71}$$

Proof. Recall the definitions of C^{bayes} and $C^{\text{two-part}}$:

$$C^{\text{bayes}}(Y \mid X) \stackrel{+}{=} -\log_2 \sum_{f \in \mathcal{F}} 2^{-K(f)} p(Y \mid X; f) \tag{72}$$

$$C^{\text{two-part}}(Y \mid X) \stackrel{+}{=} \min_{f \in \mathcal{F}} K(f) - \log_2 p(Y \mid X; f) \tag{73}$$

$$= -\log_2 \max_{f \in \mathcal{F}} 2^{-K(f)} p(Y \mid X; f). \tag{74}$$

We can substitute these definitions into the inequality above, and then apply $\lambda x \;\; 2^{-x}$ to both sides, which is monotonically decreasing, and therefore we also reverse the direction of the inequality:

$$\sum_{f \in \mathcal{F}} 2^{-K(f)} p(Y \mid X; f) \stackrel{+}{\geq} \max_{f \in \mathcal{F}} 2^{-K(f)} p(Y \mid X; f) \tag{75}$$

The inequality holds because the left-hand side is a sum of nonnegative elements where one of the elements is the element from the right-hand side. □

Note that per the above inequality, it is also clear that $C^{\text{two-part}}(Y \mid X)$ will decrease towards $C^{\text{bayes}}(Y \mid X)$ as more probability converges towards any single hypothesis.

B.7.4 Universal Bayesian Codes and Kolmogorov Complexity

How does $C^{\text{bayes}}(Y \mid X)$ relate to $K(Y \mid X)$? Per the coding theorem, we know that $K(Y \mid X) \stackrel{+}{\leq} C^{\text{bayes}}(Y \mid X)$. Establishing a bound in the other direction is more challenging. We have defined the sets of two-part and Bayesian codes with respect a probabilistic model that makes independent predictions for each individual (x, y) pair, while the definition of $K(Y \mid X)$ makes no such probabilistic independence assumptions. However, $K(Y \mid X)$ and $C^{\text{bayes}}(Y \mid X)$ may often be roughly equal for large numbers of independent and identically distributed samples (Li & Vitányi, 2008, Section 5).

B.7.5 Bayesian Codes and Variational Codes

The variational codelength defined in equation 9 has a direct relationship to the Bayesian codelength. The difference between the variational codelength and the ideal Bayesian codelength is given by the KL divergence between the variational posterior $\beta_M(\cdot; \phi)$ and the true Bayesian posterior:

$$L_M^{\text{var}}(Y \mid X, \phi) = \text{KL}\left[\beta_M(\cdot; \phi) \parallel p_M^{\text{posterior}}(\cdot \mid Y, X)\right] - \log_2 p_M^{\text{marginal}}(Y \mid X) \tag{76}$$

where the Bayesian posterior is defined as:

$$p_M^{\text{posterior}}(h \mid X, Y) = \frac{P(Y \mid X; m_M(h))\, \alpha_M(h)}{p_M^{\text{marginal}}(Y \mid X)}, \tag{77}$$

and the Bayesian marginal likelihood is:

$$p_M^{\text{marginal}}(Y \mid X) = \sum_{h \in \mathcal{H}_M} p(Y \mid X; m_M(h))\, \alpha_M(h). \tag{78}$$

Minimizing the variational codelength $L_M^{\text{var}}(Y \mid X, \phi)$ with respect to ϕ is equivalent to maximizing the Evidence Lower Bound (ELBO) from variational inference. This process drives the variational posterior $\beta_M(\cdot; \phi)$ closer to the true Bayesian posterior $P_M^{\text{posterior}}(\cdot \mid X, Y)$ (by minimizing their KL divergence), and consequently brings the variational codelength closer to the corresponding Bayesian codelength $L_M^{*,\text{bayes}}(Y \mid X)$. The variational codelength can thus be seen as an efficiently computable surrogate or approximation for the ideal Bayesian codelength.

B.7.6 Proof of Proposition 3

Proposition (Proposition 3 restated)**.** *For any quasi-universal variational code M,*

$$K(Y \mid X) \stackrel{+}{\leq} C^{\textit{bayes}}(Y \mid X) \stackrel{+}{\leq} L_M^{*,\textit{var}}(Y \mid X) \stackrel{+}{\leq} C^{\textit{two-part}}(Y \mid X), \tag{79}$$

where $C^{\textit{bayes}}(Y \mid X) := -\log_2 \sum_{f \in \mathcal{F}} 2^{-K(f)} p(Y \mid X; f)$.

Proof. This theorem combines several of our previous results.

The relationship $L_M^{*,\text{var}}(Y \mid X) \overset{+}{\leq} C^{\text{two-part}}(Y \mid X)$ follows directly from the definition of a quasi-universal variational code.

The relationship $C^{\text{bayes}}(Y \mid X) \overset{+}{\leq} L_M^{*,\text{var}}(Y \mid X)$ follows from a two-step argument. First, as established by the non-negativity of the KL divergence term in equation 76 (discussed in Section B.7.5), any variational codelength is lower-bounded by its corresponding Bayesian codelength:

$$L_M^{*,\text{bayes}}(Y \mid X) \leq L_M^{*,\text{var}}(Y \mid X).$$

Second, by the existence of a universal Bayesian code (Lemma 3), any specific Bayesian code is lower-bounded by the universal bound:

$$C^{\text{bayes}}(Y \mid X) \overset{+}{\leq} L_M^{*,\text{bayes}}(Y \mid X).$$

Combining these two inequalities yields the desired result, with discussion relating $C^{\text{bayes}}(Y \mid X)$ and $C^{\text{two-part}}(Y \mid X)$ in B.7.3.

Finally, the relationship $K(Y \mid X) \overset{+}{\leq} C^{\text{bayes}}(Y \mid X)$ follows from the discussion in B.7.4. □

B.8 Analysis of Variational Codes

This section provides additional definitions and analysis related to variational codes.

B.8.1 Equivalence Between Two-Part and Variational Codes

Here we show that variational codes can be seen as a generalization of two-part codes, with equivalence when we restrict the posterior hypothesis space to distributions that assign all probability to a single hypothesis.

Recall that for a two-part code we optimize the codelength over the hypothesis space directly. In a variational code, we optimize the codelength over a set of posterior parameters, which define a distribution over the hypothesis space. The codes are equivalent if we can establish a bijective mapping between the hypothesis space of a two-part code and the posterior parameter space of a variational code, such that for these corresponding values the codelengths are the same and the model functions are the same. We formalize this notion in the following definition.

Definition 8. *(equivalency for two-part and variational codes) A variational code M^{var} and two-part code $M^{two\text{-}part}$ are* equivalent *if there exists a bijective mapping $u : \mathcal{H}_{M^{two\text{-}part}} \to \Phi_{M^{var}}$ such that if $\phi = u(h)$ then $L^{var}_{M^{var}}(Y \mid X; \phi) = L^{two\text{-}part}_{M^{two\text{-}part}}(Y \mid X; h)$ and $p_{M^{var}}(y \mid x; \phi) = p(y \mid x; m_{M^{two\text{-}part}}(h))$.*

Lemma 5. *For every two-part code $M^{two\text{-}part}$, there exists an equivalent variational code M^{var}.*

Proof. Given $M^{\text{two-part}}$, we can define M^{var} as:

$$\Phi_{M^{\text{var}}} = \mathcal{H}_{M^{\text{var}}} = \mathcal{H}_{M^{\text{two-part}}} \tag{80}$$
$$m_{M^{\text{var}}} = m_{M^{\text{two-part}}} \tag{81}$$
$$\alpha_{M^{\text{var}}}(h) = \alpha_{M^{\text{two-part}}}(h) \tag{82}$$
$$\beta(h; \phi) = [\![\phi = h]\!], \tag{83}$$

where $[\![P]\!]$ denotes the Iverson bracket, i.e. 1 if P is true and 0 otherwise.

In other words, the posterior hypothesis space is restricted to distributions that assign all probability to a single hypothesis. The bijective mapping u is the identity function, as $\Phi_{M^{\text{var}}} = \mathcal{H}_{M^{\text{two-part}}}$.

The codelength of our variational code with $\phi = u(h) = h$ is defined as:

$$\begin{aligned} L^{\text{var}}_{M^{\text{var}}}(Y|X; u(h)) &= \mathbb{E}_{h \sim \beta_{M^{\text{var}}}(\cdot; h)}\left[-\log_2 \alpha_{M^{\text{var}}}(h) + \log_2 \beta_{M^{\text{var}}}(h; h) - \log_2 p(Y \mid X; m_{M^{\text{var}}}(h))\right] \\ &= -\log_2 \alpha_{M^{\text{two-part}}}(h) + \log_2 [\![h = h]\!] - \log_2 p(Y \mid X; m_{M^{\text{two-part}}}(h)) \\ &= -\log_2 \alpha_{M^{\text{two-part}}}(h) - \log_2 p(Y \mid X; m_{M^{\text{two-part}}}(h)) \\ &= L^{\text{two-part}}_{M^{\text{two-part}}}(Y|X; h). \end{aligned}$$

The conditional distribution for our variational code with $\phi = u(h) = h$ is given as:

$$\begin{aligned} p_M(Y \mid X; \phi) &= \mathbb{E}_{h \sim \beta_{M^{\text{var}}}(\cdot;\phi)}\, p(Y \mid X; m_{M^{\text{var}}}(h)) \\ &= p(Y \mid X; m_{M^{\text{two-part}}}(h)). \end{aligned}$$

Thus $M^{\text{two-part}}$ and M^{var} are equivalent given the bijective mapping u. □

B.8.2 Asymptotically Optimal Variational codes

Asymptotically optimal variational codes are defined in the same way as asymptotically optimal two-part codes.

Definition 9. *(asymptotically optimal variational code) A family of variational codes* $\{M_R \mid R \in \mathbb{N}^2\}$ *is* asymptotically optimal *with respect to a universal prefix Turing machine* T *if for all* R *and for all* X, Y*:*

$$L^{*,var}_{M_R}(Y \mid X) \overset{+}{\leq} \min_{f \in \mathcal{F}} K_{T,R}(f) - \log_2\, p(Y \mid X; f), \tag{84}$$

where $K_{T,R}$ *denotes resource-bounded Kolmogorov complexity.*

The codelength $L^{*,\text{var}}_{M_R}(Y \mid X)$ monotonically decreases to be $\overset{+}{\leq} C^{\text{two-part}}(Y \mid X)$ as the resource bounds R increase.

B.8.3 Proof of Proposition 4

Proposition (Proposition 4 restated)**.** *There exists an asymptotically optimal family of variational codes for Transformer encoders.*

Proof. This trivially follows from Theorem 4, which establishes that there exists an asymptotically optimal family of two-part codes for Transformer encoders, and Lemma 5, which establishes that for every two-part code there exists an equivalent variational code. □

B.8.4 Variational Codes with Adaptive Priors

We extend the definition of a variational code (Section 5) to include an adaptive prior, termed an *adaptive* variational code.

Definition 10 (adaptive variational code)**.** *An* adaptive variational code M *consists of a hypothesis space* $\mathcal{H}_M$*, a mapping* $m_M : \mathcal{H}_M \to \mathcal{F}$ *from hypotheses to model functions, and a prior* $\alpha_M(h)$ *over* $\mathcal{H}_M$*, specified as in Definition 1. Additionally, an adaptive variational code specifies:*

1. *A* prior hypothesis space Ψ_M *and prior distribution over hypotheses,* $\alpha_M(h; \psi)$*, parameterized by* $\psi \in \Psi_M$*.*
2. *A* posterior hypothesis space Φ_M *and posterior distribution over hypotheses,* $\beta_M(h; \phi)$*, parameterized by* $\phi \in \Phi_M$*.*
3. *A description length measure* L^{Ψ}_M *for the prior parameters satisfying Kraft's inequality, which can be interpreted as implicitly specifying a hyperprior over the prior parameters.*

Similar to a standard variational code, the conditional distribution over Y given X is therefore specified by the posterior parameters $\phi \in \Phi_M$, and defined by marginalizing over hypotheses:

$$p_M(Y \mid X; \phi) = \mathbb{E}_{h \sim \beta_M(\cdot;\phi)}\, P(Y \mid X; m_M(h)). \tag{85}$$

Different from a standard variational code, the codelength for a variational code M is then defined with respect to both ϕ and ψ as:

$$L^{\text{adaptive-var}}_M(Y \mid X; \psi, \phi) = L^{\Psi}_M(\psi) + \mathrm{KL}\left[\beta_M(\cdot; \phi) \parallel \alpha_M(\cdot; \psi)\right] - \log_2\, p_M(Y \mid X; \phi). \tag{86}$$

The codelength accounts for the cost of transmitting the prior parameters, $L^{\Psi}_M(\psi)$. Alice can then first send Bob these prior parameters at a cost of $L^{\Psi}_M(\psi)$, and the remainder of the transmission then follows that of a standard variational code, following the bits-back argument.

We similarly denote the minimum of an adaptive variational code M as:

$$L_M^{*,\text{adaptive-var}}(Y \mid X) = \min_{\psi \in \Psi_M, \phi \in \Phi_M} L_M^{\text{adaptive-var}}(Y \mid X; \psi, \phi). \tag{87}$$

The definition of an *asymptotically optimal family of adaptive variational codes* is directly analogous to the definition for standard variational and two-part codes.

Note that a two-part code could also include an adaptive prior, although we don't explicitly consider such codes in this paper.

B.9 Analysis of Gaussian Mixture Models

This section provides additional analysis of variational codes constructed via Gaussian Mixture Models (GMMs).

B.9.1 Parameterization of Gaussian Mixture Models

In formulating variational codes, we consider Gaussian Mixture Models (GMMs) with K components. We parameterize such models with parameters $\omega = \{(\mu_k, \nu_k, w_k)\}_{k=1}^K$, where for the k-th component, μ_k is the mean, ν_k is a parameter controlling variance, and w_k is a logit for the mixture weight. The probability density function is given by:

$$\text{GMM}(x; w) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \sigma_k^2), \tag{88}$$

where the *mixing coefficients*, π_k, are computed via the softmax function applied to the logits w_k to ensure they form a valid probability distribution (i.e., $\sum_k \pi_k = 1$):

$$\pi_k = \frac{e^{w_k}}{\sum_{j=1}^K e^{w_j}}, \tag{89}$$

and the *variances*, σ_k^2, are parameterized using the softplus function of ν_k to ensure they are strictly positive:

$$\sigma_k^2 = \log(1 + e^{\nu_k}). \tag{90}$$

A key property of this parameterization is that as $\nu_k \to -\infty$, the corresponding variance $\sigma_k^2 \to 0$. In this limit, the k-th Gaussian component, $\mathcal{N}(x; \mu_k, \sigma_k^2)$, converges to a Dirac delta function, $\delta(x - \mu_k)$. Consequently, this GMM formulation can approximate any discrete distribution over K points by treating it as a limiting case of the mixture.

B.9.2 Unimodal vs. Multimodal Priors

This section provides a simple, illustrative example of the benefits of a multimodal GMM prior vs. a unimodal Gaussian prior for encoding discrete information.

Consider a setting where Alice wants to send Bob a single random bit $b \in \{0, 1\}$. Let us assume we have a simple variational code consisting of a prior and posterior over a single scalar parameter w. Further, assume Bob "decodes" the bit based on the sign of w. Assume the posterior β is parameterized as a Gaussian, $\beta(w; \mu, \sigma^2) = \mathcal{N}(w; \mu, \sigma^2)$ with mean μ and variance σ^2. We can consider two different parameterizations of the prior α, as a single *unimodal* Gaussian, or as a *multimodal* mixture of 2 Gaussians, which we show enables a much more efficient transmission cost. Under a variational code, Alice selects the parameters of the posterior (μ and σ^2) that minimize the cost of the transmission, which is related to the KL divergence between the prior and posterior distributions ($\text{KL}[\beta(w; \mu, \sigma^2) \parallel \alpha(w)]$) and the probability that the bit can be correctly decoded ($\mathbb{E}_{w \sim \beta(w;\mu,\sigma^2)}[\text{sgn}(w) = b]$).

Figure 7 visualizes posterior distributions for transmitting bits 0 or 1, along with a unimodal Gaussian prior (left) and a multimodal GMM prior (right). Transmission is significantly more efficient with a GMM prior, highlighting why such a prior is critical for constructing our family of asymptotically optimal variational codes. We provide more detailed analysis of each case below.

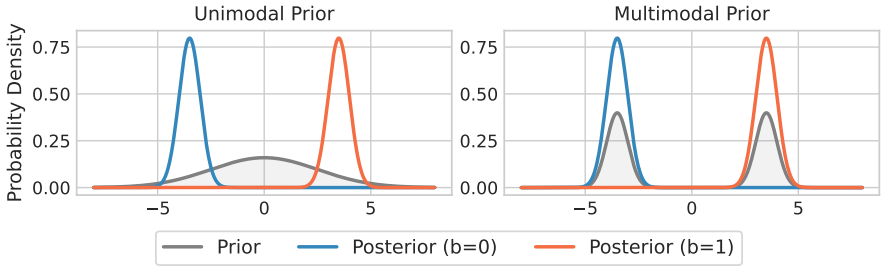

Figure 7: Visualization of posteriors for transmitting a bit b under a unimodal Gaussian prior vs. multimodal GMM prior.

Unimodal (Gaussian) Prior First, consider the case where the prior is a single zero-mean unit Gaussian, $\alpha(w) = \mathcal{N}(w \mid 0, 1)$. The KL divergence (in nats) can be computed analytically:

$$\mathrm{KL}\left[\beta(w; \mu, \sigma^2) \parallel \alpha(w)\right] = \frac{1}{2}\left[\sigma^2 + \mu^2 - 1 - \log(\sigma^2)\right] \tag{91}$$

To make the transmission reliable, the mean μ must be sufficiently far from 0 and the variance σ^2 must be sufficiently small. However, increasing $\mu \gg 0$ or decreasing $\sigma^2 \ll 1$ both increase the KL divergence. Therefore, we must tradeoff the KL divergence with the probability of correct decoding, with the Pareto frontier visualized in Figure 8. Reaching a probability of correct decoding close to 1 requires a KL of $\gg 1$ bits.

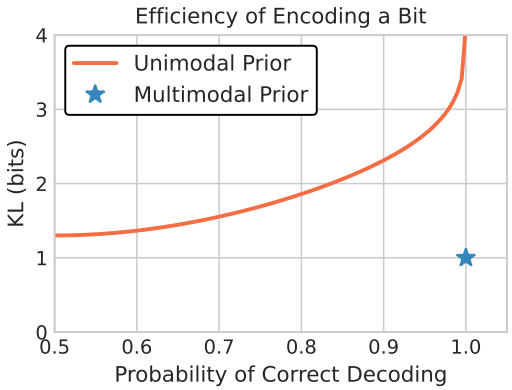

Figure 8: With a unimodal prior, we must tradeoff the KL divergence of the posterior against the expectation that the sign of the sampled parameter matches the sign of a given bit. With a multimodal prior, we can optimally encode the bit.

Multimodal (GMM) Prior Now, let the adaptive prior be a two-component Gaussian Mixture Model (GMM) with equal weights, means -1 and 1, and variance $\hat{\sigma}^2 \approx 0$.

$$\alpha(w) = \frac{1}{2}\mathcal{N}(w; -1, \hat{\sigma}^2) + \frac{1}{2}\mathcal{N}(w; 1, \hat{\sigma}^2)$$

To send a bit $b = 1$ or $b = 0$, Alice can choose a posterior with mean 1 or -1, respectively, and variance $\hat{\sigma}^2$. Consider the $b = 1$ case. Since $\hat{\sigma}^2 \approx 0$, the probability of correct decoding is ≈ 1. Additionally, the probability of any value sampled from the posterior $\beta(w; 1, \hat{\sigma}^2)$ according to the prior mixture component with mean -1 is negligible, so the KL divergence then simplifies to be the

KL divergence between the posterior and the prior component at 1:

$$\begin{aligned}\mathrm{KL}[\beta(w;1,\hat{\sigma}^2)\parallel\alpha(w)] &= \mathbb{E}_{w\sim\beta(w;1,\hat{\sigma}^2)}\left[\log_2\frac{\beta(w;1,\hat{\sigma}^2)}{\alpha(w)}\right]\\ &\approx \mathbb{E}_{w\sim\beta(w;1,\hat{\sigma}^2)}\left[\log_2\frac{\mathcal{N}(w;1,\hat{\sigma}^2)}{\frac{1}{2}\cdot\mathcal{N}(w;1,\hat{\sigma}^2)}\right]\\ &= \mathbb{E}_{w\sim\beta(w;1,\hat{\sigma}^2)}[\log_2 2] = 1 \text{ bit.}\end{aligned}$$

Thus, we can transmit the bit with an optimal KL cost of 1 bit, with near certain probability of correct decoding, as shown in Figure 8.

B.9.3 Optimal Mixture Priors

In this section we discuss how adaptive GMM priors enable efficient encoding of quantized weights, as previously discussed in Ullrich et al. (2017) and Nowlan & Hinton (1992). Specifically, we will derive the optimal GMM prior parameters for the following case:

- We have a group of N weights with a shared, adaptive GMM prior with K mixture components (B.9.1).
- We want to optimize the KL divergence between this prior and a posterior over these weights.
- The posterior for each weight approximates a delta function at a particular value.
- These posterior delta functions are all clustered at a small set of $M \leq K$ unique values. Let $\{m_1, m_2, \cdots, m_M\}$ denote these unique values, and let c_i be the count of weights that have the value m_i, such that $\sum_{i=1}^{M} c_i = N$.

The optimal GMM prior (that minimizes KL divergence) is a mixture of M components, where each component is a Dirac delta function centered at one of the unique value m_i. The remaining $K - M$ components should have a mixing weight of zero. Specifically, the optimal parameters, $\{(\mu_k, \nu_k, w_k)\}_{k=1}^{K}$, for this prior are:

- *Means*: The mean of each active component is set to one of the unique weight values. For $i = 1, \ldots, M$, the optimal mean is $\mu_i = m_i$.
- *Variances*: As the components converge to Dirac delta functions, their variances must approach zero. Thus, the optimal softplus parameter is $\nu_i \to -\infty$, which implies $\sigma_i^2 \to 0$.
- *Mixing coefficients*: The mixing coefficient for each active component should be equal to the empirical frequency of the corresponding unique value in the data. For $i = 1, \ldots, M$, we select w_i such that the optimal mixing coefficient is $\pi_i = \frac{c_j}{N}$. The mixing coefficients for the remaining $K - M$ components are zero.

By using a GMM with delta function components, the prior perfectly matches the empirical distribution of the weights, leading to the minimum KL divergence. In this idealized setting, the total KL divergence is equivalent to the Shannon entropy of the empirical distribution of the weights, scaled by the total number of weights N. Let $p_i = c_i/N$ be the empirical frequency of the unique weight value m_i. The KL divergence for a single weight whose posterior is a delta function at m_i simplifies to the negative log-likelihood under the prior, $-\log_2 \alpha(m_i)$. Since the optimal prior sets $\alpha(m_i) = p_i$, the cost for this single weight is $-\log_2 p_i$. Summing this cost over all N weights in the group, we get the total KL divergence:

$$\sum_{i=1}^{M} c_i(-\log_2 p_i) = \sum_{i=1}^{M} Np_i(-\log_2 p_i) = N\left(-\sum_{i=1}^{M} p_i \log_2 p_i\right) = N \cdot H(p)$$

where $H(p)$ is the Shannon entropy of the empirical distribution $p = (p_1, \cdots, p_M)$. This is the theoretical minimum number of bits required to encode the specific weight values given their empirical frequencies.

This highlights a close connection between the dynamic quantization method of Han et al. (2016), as employed by Lotfi et al. (2022; 2024), and this special case of optimizing a variational objective with an adaptive GMM prior.

B.10 PROOF OF THEOREM 2

Theorem (Theorem 2 restated)**.** *There exists an asymptotically optimal family of adaptive variational codes for Transformer encoders where the adaptive prior and posterior distributions are both specified by products of independent GMMs.*

Proof. We construct a family of adaptive variational codes $\{M_R \mid R \in \mathbb{N}^2\}$ and show it is asymptotically optimal with respect to a universal prefix Turing machine $T \in \mathcal{T}$.

Construction of the code family For each resource bound $R \in \mathbb{N}^2$, the code M_R is defined as follows. The hypothesis space $\mathcal{H}_{M_R}$ consists of the weights for a Transformer encoder architecture that uses layerwise weight sharing, as specified by the construction of $\texttt{zmap}(T, R, z)$ in Section B.5. For a given $h \in \mathcal{H}_{M_R}$, let $h_i \in h$ denote the specific Transformer weight value for some index i, assuming some arbitrary enumeration of the weights. The mapping function m_{M_R} is also as defined in that section, prepending R_s prompt tokens to the input before the Transformer forward pass.

The prior and posterior hypotheses spaces, Ψ_{M_R} and Φ_{M_R}, are both parameterized by sets of independent Gaussian mixture models (GMMs), as described in Section B.9.1. For posterior parameters $\phi \in \Phi_{M_R}$, let ϕ_i denote the parameters of a GMM with index i. Similarly, for adaptive prior parameters $\psi \in \Psi_{M_R}$, let $\psi_{\texttt{group}(i)}$ denote the parameters of a GMM with index $\texttt{group}(i)$, where $\texttt{group} : \mathbb{N} \to \mathbb{N}$ is a function defining a grouping of Transformer weights that share the same prior GMM.

The posterior distribution is defined as:

$$\beta_{M_R}(h; \phi) = \prod_i \text{GMM}(h_i; \phi_i). \tag{92}$$

The adaptive prior distribution is similarly defined as:

$$\alpha_{M_R}(h; \psi) = \prod_i \text{GMM}(h_i; \psi_{\texttt{group}(i)}). \tag{93}$$

Recall that the number of weights in the Transformer constructed by $\texttt{zmap}$ is finite, except for the number of prompt token embeddings, where the number of rows (i.e. number of prompt tokens) grows with the space resource bound R_s, and therefore our asymptotic analysis largely focuses on these weights (see B.5.6).

For the prompt token embeddings, a separate GMM prior is shared across each feature column. For all weights other than the prompt token embeddings, this grouping does not formally affect our results, e.g. we can simply share a single GMM prior across each matrix or bias vector in the Transformer.

We require that the prior and posterior GMMs corresponding to the weights of the prompt embedding table have at least 2 components. We will show that the other GMMs only formally require a single component for the desired asymptotic bounds to hold, although a prior with more components can lead to a more efficient code in practice, as discussed below.

Finally, we construct the encoding of the prior parameters, $L^{\Psi}_{M_R}$, as follows. Our constructed grouping of weights sharing the same prior ensures that the total number of prior parameters in Ψ_{M_R} is a small constant that does not depend on the resource bound R. This is because the number of parameters generated by $\texttt{zmap}$ is constant with respect to R, other than the number of rows in the prompt token embedding table, and a single prior for each column is shared across every row of the prompt embedding table. Assuming the prior parameters can be encoded with some finite precision, we select some uniform encoding for the prior parameters, $L^{\Psi}_{M_R}$, which assigns non-zero probability to any specific set of prior parameters we construct below, such that they can be transmitted in c_Ψ bits, which does not depend on R.

Proof of asymptotic optimality To prove that the family $\{M_R : R \in \mathbb{N}^2\}$ is asymptotically optimal, we must show that for any resource bound $R \in \mathbb{N}^2$ and any dataset X, Y:

$$L^{*,\text{adaptive-var}}_{M_R}(Y \mid X) \overset{+}{\leq} C^{\text{two-part}}_{T,R}(Y \mid X). \tag{94}$$

Recall that the minimum codelength is achieved by minimizing over all possible prior and posterior parameters,

$$L^{*,\text{adaptive-var}}_{M_R}(Y \mid X) = \min_{\psi \in \Psi_{M_R}, \phi \in \Phi_{M_R}} L^{\text{adaptive-var}}_{M_R}(Y \mid X; \psi, \phi),$$

where,

$$\begin{aligned} L^{\text{adaptive-var}}_{M_R}(Y \mid X; \psi, \phi) &= L^{\psi}_{M_R}(\psi) + \text{KL}\left[\beta_{M_R}(\cdot; \phi) \parallel \alpha_{M_R}(\cdot; \psi)\right] - \log_2 \, p_{M_R}(Y \mid X; \phi) \\ &= c_\Psi + \text{KL}\left[\beta_{M_R}(\cdot; \phi) \parallel \alpha_{M_R}(\cdot; \psi)\right] - \log_2 \, p_{M_R}(Y \mid X; \phi). \end{aligned}$$

We will show that for any resource bound $R \in \mathbb{N}^2$ and program $z \in \mathcal{Z}_{T,R}$, we can construct a specific set of prior parameters ψ^R – not dependent on program z – and posterior parameters $\phi^{R,z}$ such that the resulting variational codelength satisfies the following bound with respect to the function f^z_T computed by prefix Turing machine T with program z. That is, we will show that there exist $\psi^R \in \Psi_{M_R}$ and $\phi^{R,z} \in \Phi_{M_R}$ such that:

$$L^{\text{adaptive-var}}_{M_R}(Y \mid X, \psi^R, \phi^{R,z}) \leq c_\Psi + |z| + c_T - \log_2 p(Y \mid X; f^z_T), \tag{95}$$

where c_T and c_Ψ are constants that do not depend on X, Y, z, or R. We address the data likelihood and KL divergence terms individually next. Then we will show that this condition implies asymptotic optimality.

Let $h^{R,z} = \texttt{zmap}(T, R, z)$ be the set of weights generated by the ALTA compiler to emulate the Turing machine T with program z.

Data likelihood For a given program $z \in \mathcal{Z}_{T,R}$, we construct posterior parameters such that any weights sampled from the posterior distribution compute f^z_T, i.e. any sampled weights compute the same function as the weights $h^{R,z}$.

To accomplish this, we partition the Transformer weights into two disjoint subsets, which we will denote with respect a set of indexes $\mathcal{I} \in \mathbb{N}$. For the first subset, consisting of weights with indexes $\in \mathcal{I}$, we can set the parameters of the corresponding posterior GMM $\phi^{R,z}_i$ to approximate a Dirac delta function at the desired weight value, $h^{R,z}_i$. This is possible because a GMM component's variance can approach zero (see B.9.1). The second subset, consisting of weights with indexes $\notin \mathcal{I}$, consists of only those weights in the first column of the prompt token embedding table (see B.5.6) corresponding to weights encoding the values of bits on the program tape after z, i.e. rows $|z|+1$ to R_s. By construction, these parameters are never "read" by the attention head scanning the program tape, so their value does not affect the function being computed by the Transformer. We therefore allow these weights to take on a "random" value by setting the corresponding posterior parameters to approximate the Rademacher distribution. This construction allows these bits to be transmitted "for free", as we will show in the next section. Formally, the posterior distribution specified by the posterior parameters $\phi^{R,z}$ is therefore:

$$\text{GMM}(w; \phi^{R,z}_i) = \begin{cases} \delta(w - h^{R,z}_i) & , i \in \mathcal{I} \\ \frac{1}{2}\delta(w+1) + \frac{1}{2}\delta(w-1) & , i \notin \mathcal{I} \end{cases}. \tag{96}$$

Therefore, given the posterior parameters $\phi^{R,z}$, the negative log likelihood of the data is equivalent to that under a two-part code with weights $h^{R,z}$:

$$\begin{aligned} -\log_2 p_{M_R}(Y \mid X; \phi^{R,z}) &= \mathbb{E}_{h \sim \beta_{M_R}(\cdot; \phi^{R,z})}\left[-\log_2 p(Y|X; m_{M_R}(h))\right] && (97) \\ &= -\log_2 p(Y|X; m_{M_R}(h^{R,z})) && (98) \\ &= -\log_2 p(Y|X; f^z_T). && (99) \end{aligned}$$

KL divergence We now show that we can select prior parameters $\psi^R \in \Psi_{M_R}$ such that the KL divergence term is bounded by $|z| + c_T$, where c_T is a constant depending only on T but not on z or R. Because the prior and posterior are parameterized by *independent* GMMs, the overall KL computation factors across the individual weights of the Transformer:

$$\text{KL}\left[\beta(\cdot; \phi^{R,z}) \parallel \alpha(\cdot; \psi^R)\right] = \sum_i \text{KL}\left[\beta(\cdot; \phi^{R,z}_i) \parallel \alpha(\cdot; \psi^R_{\texttt{group}(i)})\right] \tag{100}$$

We consider the weights in the prompt embedding table first, and then discuss the remaining weights. Recall that a GMM prior is shared for each column in this table. The weights in the first column encodes the contents of the program tape. We set the prior for this first column of the program embedding table to approximate the Rademacher distribution, such that the prior and posterior distribution are the same for the weights corresponding to the $R_s - |z|$ rows in the table after row $|z|$. The KL divergence between two equal distributions is 0. For the weights in the first $|z|$ rows, the posterior is either a delta function at 1 or -1. In either case, the KL divergence is 1 bit (see B.9.2, which also highlights the importance of a *multimodal* prior in our construction). For the remaining columns of the prompt embedding table, each row has the same value, and the posterior for all rows within a column approximates a delta function at this value. Therefore, we can set the prior to be equal to the posterior, and the KL divergence is 0 for these weights. Therefore, the overall KL divergence for the prompt embedding table is $|z|$.

For the finite set of weights outside of the prompt embedding table, the posterior distribution for each weight approximates a delta function. The weights generated by `zmap` contain a relatively small set of unique values determined by T. We can choose any prior that assigns non-zero probability to these specific weight values. Summing over all fixed weights gives a total constant cost c_T that depends on T but not on z or R. A single mixture component is formally sufficient, since it can have sufficiently large variance to assign non-zero probability to each unique value in the corresponding weight matrix. However, since the weight values generated by `zmap` have values drawn from a finite set determined by T, with a sufficient number of components, we can construct the shared GMM priors to be mixtures of delta functions centered at these values, as described in B.9.3, and therefore c_T can be relatively small.

Combining these parts, the total KL divergence is $|z| + c_T$.

Conclusion We have shown that for any program $z \in \mathcal{Z}_{T,R}$ and resource bound $R \in \mathbb{N}^2$, there exists a choice of prior and posterior parameters $(\psi^R, \phi^{R,z})$ such that:

$$L_{M_R}^{\text{adaptive-var}}(Y \mid X, \psi^R, \phi^{R,z}) \leq -\log_2 p(Y \mid X; f_T^z) + |z| + c_T + c_\Psi. \tag{101}$$

The minimum codelength for the family M_R is found by minimizing over all (ψ, ϕ), so it must be less than or equal to the codelength for this specific choice, minimized over all programs z:

$$\begin{aligned}
L_{M_R}^{*,\text{adaptive-var}}(Y \mid X) &= \min_{\psi \in \Psi_{M_R}, \phi \in \Phi_{M_R}} L_{M_R}^{\text{adaptive-var}}(Y \mid X; \psi, \phi) \\
&\leq \min_{z \in \mathcal{Z}_{T,R}} L_{M_R}^{\text{adaptive-var}}(Y \mid X; \psi^R, \phi^{R,z}) \\
&\leq \min_{z \in \mathcal{Z}_{T,R}} (c_\Psi + |z| + c_T - \log_2 p(Y \mid X; f_T^z)) \\
&\overset{+}{\leq} \min_{z \in \mathcal{Z}_{T,R}} (|z| - \log_2 p(Y \mid X; f_T^z))
\end{aligned}$$

This last expression is equal to the definition of $C_{T,R}^{\text{two-part}}(Y \mid X)$:

$$\begin{aligned}
C_{T,R}^{\text{two-part}}(Y \mid X) &= \min_{f \in \mathcal{F}} (K_{T,R}(f) - \log_2 p(Y \mid X; f)) \\
&= \min_{z \in \mathcal{Z}_{T,R}} (|z| - \log_2 p(Y \mid X; f_T^z)) .
\end{aligned}$$

Therefore our construction satisfies the condition for an asymptotically optimal family of codes. □

B.11 Analysis of Alternative Two-Part Codes

Here we provide additional details and discussion related to the bounds in Table 2 introduced in Section 6.2. The first row of the table shows a description length bound of $|z| + \log R_s$, leading to an overall codelength bound:

$$L_{M_R}^*(Y \mid X) \overset{+}{\leq} C_{T,R}^{\text{two-part}}(Y \mid X) + \log_2 R_s. \tag{102}$$

First let us detail the construction of this family of two-part codes in contrast to the construction of the variational code detailed in B.10. One of the key challenges addressed by the adaptive variational code is that we wanted to transmit the R_s prefix token embeddings representing the program tape contents in $|z|$ bits in order to satisfy the conditions of asymptotic optimality. In order for the "unused" capacity to be transmitted "for free", our proposal required a multimodal GMM posterior that could be set equal to the prior for the $R_s - |z|$ "unused" weights. However, implementing and optimizing a multimodal posterior can be challenging. Instead, Alice can adaptively determine the optimal prefix length by selecting $|z| \in \{1, \cdots, R_s\}$, e.g. by sweeping over prefix length as a hyperparameter, using structured dropout, or using some other approach. Alice can then communicate $|z|$ to Bob in $\log_2 R_s$ bits, assuming a naive uniform prior over the integers $\{1, \cdots, R_s\}$ for encoding $|z|$. Then, Alice can simply communicate z in $|z|$ bits assuming a Rademacher prior over the $|z|$ weights, as in our previous construction in B.10. As in our previous construction, this can be generalized by using an adaptive GMM prior. Alternatively, such a code could potentially be constructed using the quantization method of Han et al. (2016), which has also been employed by Lotfi et al. (2022; 2024). Alternatively, we could restrict the prefix embeddings to some fixed vocabulary, akin to discrete prompt optimization. Most directly, in the special case where the vocabulary size is 2, we reach the same bound.

Next we discuss the bounds in Table 2 resulting from ablating aspects of this recipe. First, if we do not use any form of quantization, and instead use a fixed prior, such as the unimodal Gaussian prior implicit in methods such as weight decay, or a uniform prior implicit in standard MLE objectives, then each weight encoding a bit of z requires more than a single bit to encode. Therefore, the program length term in the bound changes from $|z|$ to $\mathcal{O}(|z|)$, where the multiplicative factor is > 1. Second, if we do not adaptively select the prefix length, then we must encode the entire prefix embedding table, which contains R_s rows, regardless of the length of the program it encodes. Therefore, the bound degrades further to $\mathcal{O}(R_s)$, where $R_s \geq |z|$ as by definition R_s determines the maximum program length that can be represented. Finally, if we do not use layerwise weight sharing, then our bound simply becomes a function of the number of parameters in the Transformer. As R_t specifies the number of layers, we must encode $\mathcal{O}(R_t)$ weights, in addition to the $\mathcal{O}(R_s)$ weights in the prefix embedding table. Note that the bounds we have derived assume we are using the Transformer family and specification of `zmap` discussed in Section 4.1. Alternative constructions could potentially lead to different bounds.

Overall, a weaker bound on the model description length leads to a weaker bound on the minimum of the overall codelength. Therefore, optimizing such an objective may lead to sub-optimal compression, and—from an MDL perspective—sub-optimal model selection. Regardless, the $|z| + \log R_s$ bound is close to the theoretically optimal bound of $|z|$. While pre-trained Transformer decoders are a different setting than the one we have studied here, if our results could be extended to that setting it would provide a strong asymptotic bound for the compression achievable via prompt optimization methods, complementing the positive empirical results from Akinwande et al. (2024).

B.12 Scaling Transformer Dimensionality

Here we provide additional details on the alternative Transformer family discussed in Section 6.3. In contrast to the Transformer family detailed in B.5, the program tape contents are implicitly encoded in the MLP parameters. In this construction, we still require prepending R_s "padding" tokens to the input in order to scale the context window so that the Transformer can represent Turing machine tapes of size R_s. However, the program z is encoded in the MLP layer, and we do not require a prefix embedding table. An ALTA program for populating the program tape contents based on a program encoded in the MLP weights is shown in Figure 9. The overall construction first iteratively populates the program tape contents at sequential Transformer positions, and then the Turing machine emulation continues as in our previous construction. Therefore, this construction requires $R_t + R_s + 2$ layers.

The MLP matrices generated by the ALTA program in Figure 9 require $\mathcal{O}(|z|^2)$ parameters, and have rank $\mathcal{O}(|z|)$, where $|z|$ is the program length. While the existence of a mapping analogous to `zmap` for this Transformer family indicates that asymptotically optimal description length objectives exist, implementing such a prior introduces complex dependencies across the parameters of the MLP. Given the rank requirements on the MLP matrices under this construction, rank factorization would not be an effective means of compression, but it would be interesting for future work to consider

```
from alta import program_builder as pb

def build_program_spec(spec: MachineSpec) -> pb.ProgramSpec:
  """Returns ALTA program spec for populating program tape."""

  variables = {
      "done": pb.var(2),
      "is_start": pb.input_var(2, init_fn=lambda x: x == START),
      "head": pb.input_var(2, init_fn=lambda x: x == START),
      "symbol": pb.input_var(2),
      "program_index": pb.var(len(program_input)),
  }
  attention_heads = {
      "head_left": pb.v_relative("head", -1),
  }

  def ffn_fn(x):
    if x["program_index"] == len(program_input):
      x["done"] = 1
      return

    if x["head"]:
      x["symbol"] = int(program_input[x["program_index"]])
      x["head"] = 0

    if not x["is_start"] and x["head_left"]:
      x["head"] = 1

    x["program_index"] = x["program_index"] + 1

  return pb.program_spec(
      ffn_fn=ffn_fn, variables=variables, heads=attention_heads,
      output_name="symbol", input_range=2, position_range=None,
  )
```

Figure 9: ALTA program specification for populating a program tape given a program encoded in the MLP layer.

alternative constructions or practical methods for matrix compression that could yield asymptotic bounds approaching the theoretical ideal.

C Additional Experimental Results and Details

Here we provide additional details and results for the Transformer and MLP experiments discussed in Section 6.

C.1 Transformer Experiments

Parity task details We follow the setting of Shaw et al. (2024). The train set consists of examples with lengths ranging from 1 to 20. The train set includes $100,000$ examples, with roughly an equal number of examples per number of ones. We evaluate out-of-distribution (OOD) accuracy on a test set with lengths ranging from 21 to 40.

ALTA program for parity The ALTA program for parity that we use for our manual initialization is based on the sequential algorithm with relative position representations presented in Shaw et al. (2024). This algorithm computes parity by iterating through each position (one per layer), flipping a running parity bit every time a one is encountered. We simply reverse the direction of iteration from left-to-right to right-to-left, to accommodate that our model's classification decision depends on the `SEP` token at the beginning of the input sequence (see below).

Architecture and ALTA weight conversion One implementation challenge is that our experiments require a trainable Transformer that is compatible with ALTA-compiled weights. The ref-

erence ALTA Transformer implementation in `numpy` is not trainable (Shaw et al., 2024). To accomplish this, we implement a trainable version in Jax (Bradbury et al., 2018), following the same parameterization expected by the ALTA compiler. This parameterization uses relative position representations (Shaw et al., 2018) following the parameterization of T5 (Raffel et al., 2020), which uses a single scalar bias term for relative positions within some window. It also uses an alternative parameterization for the attention head output transformations, as proposed by Elhage et al. (2021), to simplify compilation; however, this alternative attention output parameterization is shown to be equivalent to that of the original Transformer. We select the logits for the `SEP` token as the output of the model, as our experiments focus on classification tasks.

A more significant challenge is that ALTA-compiled weights are not designed to be used with layer normalization (Ba et al., 2016). As the sequential algorithm for parity requires at least as many layers as bits in the longest input sequence, and our test set contains sequences up to length 40, we require a relatively deep Transformer, using 42 layers in our experiments. In our initial experiments we found that while a shallow Transformer could converge without layer normalization, training a Transformer with ≥ 40 layers was not feasible on the parity task without some form of normalization. We also evaluated using tanh normalization, as an alternative to layer normalization, as proposed by Zhu et al. (2025), which we found to perform at least as well as layer normalization in our experiments, with or without using the variant with a dynamic scalar. We therefore use tanh normalization, and adapted the ALTA compiler to generate weights compatible with this form of normalization.

This required a few changes to the ALTA compiler. However, as our parity program only requires categorical variables, the changes are relatively straightforward. First, we changed how categorical variables are represented in the residual stream. In the original ALTA compiler, these are represented as "one-hot" vectors, i.e. a sequence of 0s and 1s. Instead, we represent categorical variables as "signed one-hot" vectors, i.e. a sequence of -1s and 1s, with the position of the 1 still representing the value of the variable. This change only requires minor modifications to the compiled embedding and MLP parameters. We additionally scale the compiled parameters by configurable scalars > 1 to ensure the outputs of every sub-layer sufficiently saturate the tanh function. This also makes the compiled parameters more robust to noise. We zero-pad the ALTA-generated weights up to the dimensions specified by the given hyperparameters. Finally, as we are using only categorical variables, we can use the more standard ReLU activation in the MLP layer, as opposed to the clipped ReLU activation required to handle numerical variables in the original ALTA compiler.

We can then convert from the scalar weights produced by the ALTA compiler to posterior distributions used by our experiments with variational codes by setting the mean of the posterior distribution to the scalar weight and setting the variance to be a small, configurable scalar (e.g. $\nu = -10$). As ALTA-generated weights contain few unique values, we can analytical determine the optimal parameters for the adaptive prior (B.9.3).

We otherwise use a "tiny" Transformer encoder, which was previously shown to be sufficient to fit the parity task (Shaw et al., 2024), with 2 attention heads, 128 model dimensions, and 512 hidden MLP dimensions. We prepend 20 prompt tokens to the model input.

Relation to asymptotic bounds The experiment setting, using a relatively small model on a simple synthetic task, is potentially not well aligned with where the asymptotic guarantees from our theory are most likely to apply, which is in the limit as model size and dataset complexity increase. Regardless, this experiment setting is a useful initial investigation of the proposed variational code. Relatedly, the ALTA program for parity does *not* emulate a Turing machine; rather it represents the algorithm within the weights of the MLP and attention sub-layers and leaves the prompt tokens effectively unused. However, this is not entirely at odds with our theory; as discussed in 4.2, the minimizer of an asymptotically optimal code may be quite different from the Turing machine emulation used to prove the asymptotic bound, especially under finite resource constraints.

Model training We use Monte-Carlo sampling to estimate the expected data likelihood in the adaptive variational codelength objective during training. Each gradient step is computed over 2 MC weight samples shared across a minibatch of 128 examples, for a total of 256 forward passes of the model (computed in parallel) per step. Prior work has commonly shared a single MC weight sample across a minibatch. In our initial experiments, we did not find significant improvements from increasing beyond 2 MC weight samples per minibatch. For the posteriors corresponding to

weights in the prompt token embeddings table, we use a GMM posterior with 2 components, and use Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017) sampling with a temperature of 0.1. For all other posteriors, we use a single Gaussian, and use the standard Gaussian reparameterization trick (Kingma & Welling, 2014). We did not implement the local reparameterization trick of Kingma et al. (2015) for reducing variance, although this could be useful for future work.

Following Blundell et al. (2015), we also use MC samples to estimate KL divergence, as there is no closed form expression for the KL divergence of GMMs with multiple components. Specifically, we use 100 MC samples for estimating KL. For sampling from posterior distributions consisting of mixtures with more than one component, we use the Straight-Through (ST) Gumbel-Softmax estimator (Jang et al., 2017), as soft estimates can be problematic and lead to negative KL estimates. For example, if the components of the true posterior distribution have relatively low variance, then samples from the Gumbel-Softmax distribution with a sufficiently large temperature can actually have higher probability under a higher-variance prior distribution than under a prior matching the posterior.

We use the Adam optimizer (Kingma & Ba, 2014), with a learning rate of 10^{-3}, 1000 warmup steps, and $50,000$ total steps with exponential decay of the learning rate. We determined some hyperparameters such as learning rate on an in-distribution validation set.

As we train the model on minibatches consisting of 128 samples from the $100,000$ training examples, we need to scale the coefficient of the KL term in the loss to accommodate that the data likelihood estimate is over a small sample, $\approx 10^{-3}$, of the full dataset. Therefore, we scale the KL term in the loss by a coefficient of 10^{-3}, which we found to produce the lowest total codelength. Sweeping the KL coefficient produces a tradeoff between the data likelihood and the KL divergence, as observed in Table 4 for the MLP experiments.

Random initialization For the MLE baseline experiment, we initialize the weight matrices using the standard variant of normal random initialization proposed by He et al. (2015), as our network uses ReLU activations. Bias vectors are initialized to zeros. For the experiments with random initialization and the variational objective, there are many possibilities for initializing the parameters of the prior and posterior GMMs, and we did not explore this space exhaustively. However, we did find in our initial experiments that it was necessary to initialize the posterior distributions with relatively low variance in order for the model to converge towards a solution that fits the data. Therefore, for the means of the posterior distributions, we sample random weight values following the same method as the MLE experiments, set these sampled weights as the GMM means, and then set the GMM variance parameter ν (i.e. the variance prior to the softplus function) to be -10. We initialize the prior parameters similarly, but set the initial variance parameter ν to be 1 to avoid instability in KL estimates at initialization. We set all mixing weights for the GMMs to be initially equal. Future work could explore alternative initializations.

Optimizer Comparison We evaluated SGD in comparison to Adam as the optimizer for Transformers with the variational objective for the parity task, following the setting reported in Section 6.

Table 3: Comparison of Adam and SGD for optimizing variational objective.

Optimizer	**Train NLL** (bits)	**KL** (bits)	**OOD Accuracy**
Adam	2.36×10^2	2.81×10^6	60.4%
SGD	1.58×10^5	2.44×10^6	49.6%

Adam and SGD exhibit different optimization behavior, and a different trade-off between optimizing the model cost (KL divergence) and data cost (negative log likelihood). Both methods fail to find a codelength comparable to that of the manual initialization. SGD more severely underfits the data, leading to poor generalization. This highlights the sensitivity of optimizing the variational objective to the specific optimization procedure. Future work could more fully explore alternative hyperparameters, learning rate and KL coefficient schedules, or alternative families of optimizers, such as approximate second-order methods, e.g., K-FAC (Martens & Grosse, 2015).

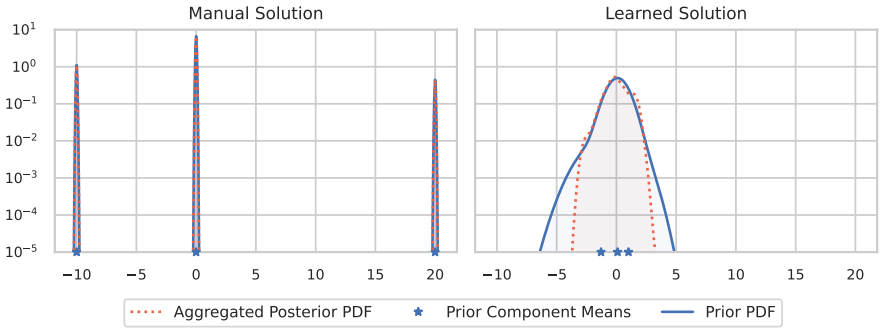

Figure 10: Manually specified vs. learned distributions for MLP trained with a variational objective on the identity task with an adaptive GMM prior. The aggregated posterior shown in the figure is an equally-weighted mixture of the individual Gaussian posteriors for each MLP weight.

C.2 MLP Experiments

Here we provide additional details on the MLP experiments discussed in Section 6. To understand why our Transformer models underfit the proposed objective when randomly initialized (as evidenced by the significantly lower loss achieved by ALTA-initialized models), we study the simplified setting of a 2-layer MLP. Specifically, we study the task of learning an identity function over a vector of 4 binary values. The model is a MLP with 4 input dimensions, 16 hidden dimensions, and 4 output dimensions, which makes 4 independent binary classifications. We can again compare random initialization with initializing the model with manually chosen weights that fit the data and have low complexity according to the proposed objective, by selecting weights inspired by how the ALTA compiler generates MLP layers.

Table 4: MLP performance on identity task.

Objective	**Init.**	**KL Coef.**	**KL** (bits)	**Train NLL** (bits)	**Codelength** (bits)	**Accuracy**
MLE Baseline	Random	—	—	8.82	—	100%
Variational	Random	1.0	2.87	2.64×10^3	2.64×10^3	54%
		10^{-1}	3.19×10^1	1.90×10^3	1.93×10^3	63%
		10^{-2}	2.24×10^2	1.12×10^2	3.37×10^2	100%
		10^{-3}	2.88×10^2	5.39×10^1	3.42×10^2	100%
		10^{-4}	2.98×10^2	5.09×10^1	3.49×10^2	100%
Variational	Manual	10^{-1}	9.87×10^1	0.0	9.87×10^1	100%

The results are shown in Table 4, and show similar trends as the Transformer experiments. We see that when starting from a random initialization, the optimization process fails to find a loss comparable to that achieved via manual initialization, indicating poor optimization of the proposed objective. To understand why the randomly initialized models underfit, we can inspect the properties of the learned prior and posterior distributions compared to those from manual initialization. Notably, the prior distribution appears to converge to a unimodal distribution, as visualized in Figure 10, in contrast to the multimodal prior of the manual solution, with lower-variance components.

Task and network Our network is a 2 layer MLP with 4 inputs dimensions and 16 hidden dimensions, that computes the function:

$$f(x;\theta) = \text{sigmoid}(W^2\ \text{ReLU}(W^1(x) + b^1) + b^2) \tag{103}$$

with parameters $\theta = (W^1, b^1, W^2, b^2)$.

Our goal is to learn the identity function such that $f(x;\theta) \approx x$ for any binary vector $x \in \{0,1\}^4$.

Posterior and prior distributions We parameterize the posterior distribution with an independent Gaussian corresponding to each weight in the MLP. We use a single adaptive GMM prior with 3 components shared across all weights.

Manual solution A relatively simple "manual" solution to the identity task can be constructed by choosing some scalar $\lambda \gg 0$ (we choose $\lambda = 20$ in our experiments) and defining the network weights as follows:

$$W^2 = (W^1)^\top = \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \lambda & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \lambda & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \lambda & 0 & \cdots & 0 \end{bmatrix} \qquad b^1 = \begin{bmatrix} -\frac{\lambda}{2} \\ \vdots \\ -\frac{\lambda}{2} \end{bmatrix} \qquad b^2 = \begin{bmatrix} -\frac{\lambda}{2} \\ \vdots \\ -\frac{\lambda}{2} \end{bmatrix}$$

Training details The negative log-likelihood of y given x with respect to θ follows the standard binary cross-entropy loss:

$$-\log_2 p(y \mid x; \theta) = \sum_{i=1}^{4} -y_i \log_2(f(x;\theta)_i) - (1 - y_i) \log_2(1 - f(x;\theta)_i) \tag{104}$$

We train the MLP with a variational objective. We specify posterior and prior distributions over θ as specified above, minimizing the variational objective consisting of the sum of the KL divergence between posterior and prior distributions, and the expected binary cross entropy loss with respect to sampling weights θ from the posterior distribution. We over-sample the space of 2^4 possible binary vectors of length 4 to create minibatches for training of 128 examples.

Sampling, initialization, and training details are similar to the Transformer setting detailed above, except training only required $1,000$ steps. We show a sweep over different KL coefficients in Table 4.

D Extended Related Work

Here we offer an extended discussion of related work previously summarized in Section 7.

D.1 Theoretical Foundations

Our work is closely related to the theoretical notion of *universal induction* (Solomonoff, 1964; Hutter, 2000; Lattimore & Hutter, 2013; Achille et al., 2021) – and related resource-bounded variants (Levin, 1973; Nakkiran, 2021). Previous work has developed theoretical frameworks applying these notions to density estimation (Barron, 1985; Barron & Cover, 1991), sequential decision theory (Hutter, 2005), and kernel methods (Hamzi et al., 2024). For neural networks, Schmidhuber (1997) introduced a probabilistic search algorithm for discovering neural networks with low Kolmogorov complexity. However, none of these theoretical frameworks directly lead to practical and scalable training objectives for neural networks.

D.2 Variational Inference

Variational inference was proposed as a regularizer to improve generalization (Hinton & Van Camp, 1993), under the "bits back" argument, which can also be viewed as a form of Bayesian inference (Honkela & Valpola, 2004). Such variational methods were popularized by the success of variational autoencoders (VAEs) (Kingma & Welling, 2014), and related advances that enabled applying such methods to neural networks using standard gradient-based optimizers (Graves, 2011; Blundell et al., 2015). Variational methods have found success for network compression – enabling weight pruning (Louizos et al., 2017) and quantization (Achterhold et al., 2018; Ullrich et al., 2017) – as well as for probing (Voita & Titov, 2020) and improving uncertainty modeling (Sankararaman et al., 2022; Gal & Ghahramani, 2016). Other work has pursued applying variational bottlenecks to network *activations* (Fehr & Henderson, 2024), as opposed to network *weights*. However, in general, the effectiveness of variational methods for implementing MDL-inspired regularizers for improving generalization has remained elusive (Blier & Ollivier, 2018; Cinquin et al., 2021). Prior work has typically found tighter compression bounds via prequential coding methods (Blier & Ollivier, 2018; Bornschein et al., 2023). Our work provides some new perspectives on the poor performance of variational approaches: objectives evaluated by prior work may have poor asymptotic bounds, and variational objectives can be difficult to optimize.

D.3 Non-Variational Complexity Measures

As regularization is a foundational concept in machine learning, there are many related methods for neural networks (Jiang et al., 2020), and we give only a brief survey here. We focus on recently proposed compression-based methods with connections to the MDL principle, as these are most relevant to the description length objectives discussed in this paper.

Motivated by prior work on the intrinsic dimensionality of neural networks (Li et al., 2018), Lotfi et al. (2022) propose a complexity measure based on subspace sampling combined with dynamic quantization, similar to the quantization method of Han et al. (2016). Lotfi et al. (2024) combines the method of Lotfi et al. (2022) with LoRA (Hu et al., 2022; Dettmers et al., 2023), to further drive compression. Both Lotfi et al. (2022) and Lotfi et al. (2024) discuss generalization bounds for neural networks related to compression, a relevant topic, but not one we study explicitly in this paper. DeMoss et al. (2025) studies the connection between the complexity of a neural network and "Grokking", i.e. the transition from memorization to generalization during training. To this end, they propose a new complexity measure and regularizer based on coarse-grained quantization and spectral entropy, which encourages low rank parameter matrices. Abudy et al. (2025) similarly advocate for regularizers grounded in MDL instead of standard norm-based penalties on weights. They demonstrate how norm-based regularizers actively push RNN weights away from perfect initializations on algorithmic tasks, while their MDL-inspired objective does not. Their proposed MDL measure can be interpreted as a two-part code, using a non-differentiable prior that encodes individual weight values according to their representation as a rational number. Dwivedi et al. (2023) similarly studies MDL-inspired complexity measures in over-parameterized models where parameter count does not provide a suitable measure of complexity, and proposes a principled measure, MDL-COMP. However, their scope is limited to specific classes of linear models and kernel methods.

The variational code proposed in Section 5.1 drives compression by encouraging soft quantization of model weights around the means of the components of the GMM prior, and by reducing the effective number of weights by encouraging higher uncertainty posterior distributions close to the prior for "unused" capacity. The non-variational approaches discussed here similarly combine some form of quantization with some alternative means of reducing the effective number of parameters, e.g. through low-rank approximation. While prior work has not established asymptotic guarantees for these alternative encoding schemes, future work could aim to construct families of asymptotically optimal, or quasi-optimal, codes based on these alternative methods.

D.4 Computational Universality of Transformers

There has been considerable interest in establishing the theoretical expressivity of various classes of Transformers (Pérez et al., 2021; Chiang et al., 2023; Yun et al., 2020; Feng et al., 2023; Merrill & Sabharwal, 2024a; Nowak et al., 2024; Merrill & Sabharwal, 2024b). Most relevant to our constructed mapping between computable, rational-valued distributions and Transformer weights is prior work establishing various equivalences between Transformers and Turing machines. Pérez et al. (2021) established Turing completeness of an encoder-decoder Transformer in a non-probabilistic language recognition setting, with Merrill & Sabharwal (2024a) and Feng et al. (2023) extending this result to decoder-only variants. Most relevant to our result is Nowak et al. (2024), which demonstrates Turing completeness in a probabilistic setting. One difference is that our result holds for Transformer encoders (in the limit as context size and number of layers increase) compared with Nowak et al. (2024)'s result for Transformer decoders with intermediate decoding steps. Their result also does not directly extend to prefix Turing machines, which is necessary to support our theoretical results. However, their work could be potentially helpful for future work seeking to establish the existence of families of asymptotically optimal codes for Transformer decoders with intermediate decoding steps. Finally, Schuurmans (2023) shows that a Transformer decoder augmented with an external memory tape is computationally universal. Broadly, considering description length objectives over models with external tools could be of interest for future work.

We use the ALTA compiler (Shaw et al., 2024) to support our demonstration of universality. ALTA was inspired by RASP (Weiss et al., 2021), an alternative language for compiling programs to Transformers, and the related Tracr (Lindner et al., 2023) compiler. ALTA offers better support for programs with loops, which is useful for emulating a Turing machine.

D.5 SIMPLICITY BIAS

While our work focuses on implementing a simplicity bias *explicitly* in a training objective, several studies have evaluated the *implicit* simplicity bias of neural networks (Zhou et al., 2023; Abbe et al., 2023; Bhattamishra et al., 2023; Tsoy & Konstantinov, 2024; Chen et al., 2024; Mingard et al., 2025), or lack thereof (e.g., Nikankin et al., 2025). Others study the simplicity bias of *in-context learning* with pre-trained models (Goldblum et al., 2023; Deletang et al., 2024), or proposed tasks to enhance this bias (Grau-Moya et al., 2024).