An Information-Theoretic Approach to Understanding Transformers' In-Context Learning of Variable-Order Markov Chains

Ruida Zhou
Amazon AGI

Chao Tian
ECE Dept. Texas A&M University

Suhas Diggavi
EE Dept. UCLA

Abstract

We study transformers' in-context learning of variable-length Markov chains (VOMCs), focusing on the finite-sample accuracy as the number of in-context examples increases. Compared to fixed-order Markov chains (FOMCs), learning VOMCs is substantially more challenging due to the additional structural learning component. The problem is naturally suited to a Bayesian formulation, where the context-tree weighting (CTW) algorithm, originally developed in the information theory community for universal data compression, provides an optimal solution. Empirically, we find that single-layer transformers fail to learn VOMCs in context, whereas transformers with two or more layers can succeed, with additional layers yielding modest but noticeable improvements. In contrast to prior results on FOMCs, attention-only networks appear insufficient for VOMCs. To explain these findings, we provide explicit transformer constructions: one with $D+2$ layers that can exactly implement CTW for VOMCs of maximum order D, and a simplified two-layer construction that uses partial information for approximate blending, shedding light on why two-layer transformers can perform well.

1 INTRODUCTION

The transformer model (Vaswani et al., 2017), the architecture behind current prevailing LLMs, is known to have strong in-context learning (ICL) capabilities, and concrete ICL results for transformers have been established for some simple tasks (Garg et al., 2022; Von Oswald et al., 2023; Bai et al., 2024; Ahn et al., 2024; Zhang et al., 2025a). Despite these advances, the mechanism for transformers to learn in context is still not fully understood, especially when the sequences have complex memory structures. Several recent works studied how transformers can learn fixed-order Markov chains (FOMCs) either in training or in-context (Makkuva et al., 2024; Edelman et al., 2024; Rajaraman et al., 2024a; Nichani et al., 2024; Akyürek et al., 2024; Zhang et al., 2025b), however, FOMC has a pre-defined fixed memory structure, which is highly inefficient in modeling complex memories. In this work, we study the ICL of a considerably more complex model, i.e., variable-order Markov chains (VOMCs), also known as context tree (CT) models (Rissanen, 1983; Willems et al., 1995), which is better suited for sequences with more complex memory structures such as natural languages (Begleiter et al., 2004); we refer to the problem as ICL-VOMC.

Although VOMCs still have finite memory, each new symbol may depend on suffixes consisting of different numbers of previous symbols, i.e., the length of the memory may vary. For example, in the sentence below,

> Language models are useful in a wide variety of applications, from natural language understanding and generation to translation, summarization, and...

"variety" and "summarization" can be modeled as depending naturally (only) on their respective underlined suffixes of different lengths. The inherent structural learning component makes it considerably more challenging, and the naive strategy of adopting an FOMC-based approach is highly sample-inefficient.

Instead of pretraining dynamics, we focus on the ICL accuracy as more in-context examples are given, i.e., the finite in-context sample performance. The well-known context-tree weighting algorithm (CTW) (Willems et al., 1995), originally developed in the information theory community for universal data compression, is in fact Bayesian optimal for the in-context

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

inference task. This is because the underlying core component of the CTW algorithm estimates the next token probability distribution – essentially the same task the transformers perform for ICL. Kontoyiannis et al. (2022) provided a more detailed analysis of the algorithm from a Bayesian inference perspective; earlier work along the Bayesian view can be found in Matsushima and Hirasawa (1994); Matsushima et al. (2002). The connection between ICL and data compression is not entirely surprising, given the recent empirical study (Deletang et al., 2024) showing LLMs' superior performance on various compression tasks.

We empirically observed that one-layer transformers cannot learn VOMCs, but those with two or more layers can learn VOMCs and track the performance of the optimal CTW algorithm closely, with more layers providing small but noticeable improvements. Moreover, attention-only networks suffer materially. We then consider explicit transformer constructions to explain these findings. The CTW algorithm has a recursive structure, which poses a challenge in the transformers' condensed parallel computation pipeline. To resolve this, we first identify an alternative representation of the CTW algorithm, and then construct a transformer with $D+2$ layers, which can learn VOMC of maximum order D in context. The construction relies on the feedforward layers, which explain why attention-only networks perform poorly on VOMCs.

The CTW algorithm needs the counts for the suffixes, and the higher layers of our transformer construction are mostly performing information aggregation. We conjecture that 2-layer transformers can perform reasonably well because even raw counts or partially aggregated information would allow a close-to-optimal approximation. We then consider such a simplified 2-layer transformer, by providing one feed-forward (FF) layer with the probability estimates and various configurations of the counts directly. The FF layer can be trained to approximate the proper blending coefficients using this information. We implement several synthetic transformer layers, and experiments show that their performances indeed match our conjectures.

It may be tempting to directly invoke the universal computation capability of transformers, e.g. Pérez et al. (2021), without seeking explicit interpretations, however, these results usually do not provide any transparency on the underlying mechanisms. In contrast, we focus on constructions that provide meaningful *interpretations* of the empirical observations, with a focus on the mechanisms that extract and aggregate the statistics (more discussions after Theorem 5). We further remark that the VOMC model provides a complex but clean setting where the optimal ICL performance is known and other confounding factors can be well-controlled, which is impossible with real-world language datasets and benchmarks.

Main Contributions. Our work is the first study of ICL-VOMC, particularly the *finite-sample performance*, and the contributions are summarized below.

1. We empirically demonstrate that transformers can learn VOMCs in-context, i.e., tracking the performance of the optimal CTW algorithm in the whole context window;
2. We provide a $(D+2)$-layer transformer construction to imitate CTW, establishing its capabilities, based on a novel Bayesian optimal next token prediction representation;
3. Our construction allows us to investigate the relative performance insensitivity to the number of layers, *i.e.,* why 2-layer transformers perform well.

Notation: Scalars, symbols, and strings are denoted by italic letters like n, N, x, and s. Denote by string $x_i^j := (x_i, x_{i+1}, \ldots, x_j)$ as a sequence of symbols. Define () or x_i^j with $i > j$ as an empty string. Vectors and matrices are in bold like $\vec{x}$, $\vec{H}$, and sets in calligraphic like $\mathcal{A}$ with cardinally $|\mathcal{A}|$.

2 PRELIMINARIES

Universal Data Compression. ICL closely resembles universal compression (Rissanen, 1983) in information theory. Naïvely speaking, the latter aims to adaptively compress the sequence without having direct access to the underlying probabilistic dynamics, but learning it in an in-context fashion. This implies that the CTW algorithm, though originally designed for data compression, is Bayesian optimal for ICL-VOMC. One subtle point is that the *finite-sample* in-context performance is our main interest, i.e., transformers are viewed as performing ICL-VOMC efficiently, only if the performance over the whole context window can track that of CTW; asymptotical (large in-context sample) matching performance is insufficient.

Variable-order Markov Chains. VOMCs have been studied extensively in the information theory literature (Rissanen, 1983; Willems et al., 1995; Begleiter et al., 2004). String $s = (x_{1-l}, x_{2-l}, \ldots, x_0)$ is a suffix of the string $s' = (x'_{1-l'}, x'_{2-l'}, \ldots, x'_0)$, if $0 \le l \le l'$ and $x_{-i} = x'_{-i}$ for $i = 0, 1, \ldots, l-1$; e.g., (a, b, c, b) is suffix of (a, c, a, a, b, c, b). Note that the strings above have non-positive indices.

The behavior of a finite memory VOMC source is specified by a suffix set $\mathcal{S}$ and the associated next token probability distributions. The set $\mathcal{S}$ is a collection of strings $s(k)$, $k = 1, 2, \ldots, |\mathcal{S}|$ with two properties: 1) proper: no string in $\mathcal{S}$ is a suffix of any

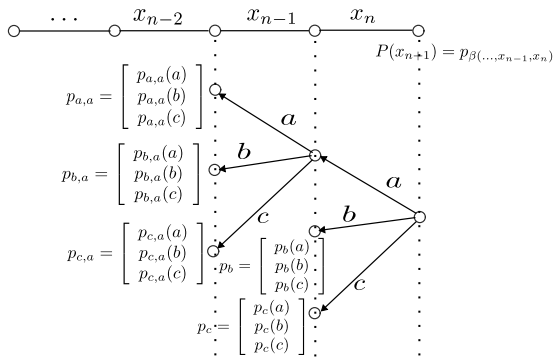

Figure 1: A context tree in the alphabet $\mathcal{A} = \{a, b, c\}$ with suffix set $\mathcal{S} = \{(b), (c), (a, a), (b, a), (c, a)\}$ and the probability distributions. If $(\ldots, x_{n-1}, x_n) = (\ldots, c, a)$, then the probability distribution for the next symbol x_{n+1} is $p_{c,a}$.

other string, and 2) complete: each semi-infinite sequence $(\ldots, x_{n-1}, x_n)$ has a unique suffix that belongs to $\mathcal{S}$, denoted as $\beta_{\mathcal{S}}(\ldots, x_{n-1}, x_n)$. Associated with each suffix $s \in \mathcal{S}$, there is a probability mass function $p_s \in \Delta_{\mathcal{A}}$. A VOMC is best visualized as a context tree, since for any valid suffix set $\mathcal{S}$, there exists a unique tree T with the elements in $\mathcal{S}$ being its leaves; the set of leaves is denoted as $\mathcal{L}(T)$. A CT can thus be equivalently represented by $(T, \{p_s\}_{s \in \mathcal{L}(T)})$; an example is given in Fig. 1. A VOMC has a maximum order D if any suffix in $\mathcal{S}$ has a length at most D. Given a semi-infinite sequence $(\ldots, x_{n-1}, x_n)$, the next symbol x_{n+1} is generated according to the distribution $p_{\beta_{\mathcal{S}}(\ldots, x_{n-1}, x_n)}$.

The CTW algorithm is Bayesian optimal for VOMCs under certain priors, i.e., achieving the optimal cross-entropy loss. An illustration of the algorithm is given in Fig. 2 (the details are deferred to Section 4.2). Prediction by partial matching (PPM) (Cleary and Witten, 1984; Begleiter et al., 2004) and Kneser-Ney (KN) smoothing Ney et al. (1994) can also be used to generate the next token prediction. They take the maximal order D_{ppm} or D_{kn} as parameters, and when a suffix has not been observed at all (or is less often observed), the prediction falls back to the estimate using a lower-order model. PPM uses an escape symbol to indicate the fallback, while KN-smoothing uses a soft blending instead. Both can be viewed as based on FOMC modeling, since they mostly adopt the prediction at the order D_{ppm} or D_{kn} when possible. More details are given in Appendix B and C.

3 IN-CONTEXT LEARNING OF VOMCS

Transformers Can Learn VOMC In-context. We train transformers to conduct experiments. A

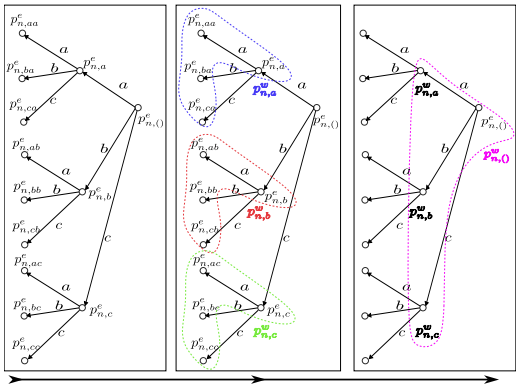

Figure 2: The CTW algorithm aggregates statistics from the leaves to the root in a recursive manner at time position n (here $D = 2$).

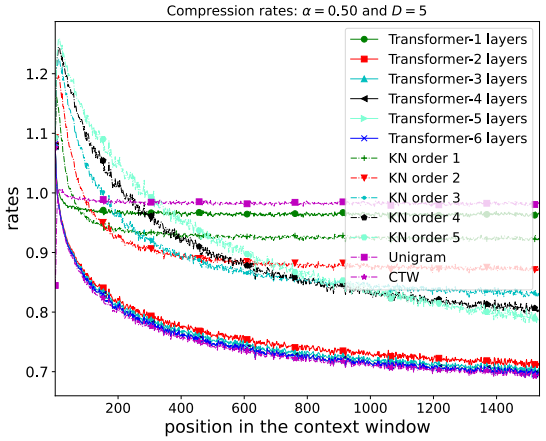

Figure 3: Transformers vs. KN-smoothing vs. CTW.

ternary alphabet $|\mathcal{A}| = 3$ is adopted, and transformers of context window size N are pretrained on data sequences of length-N generated using CTs randomly sampled from the given prior π_{CTW} parameterized by $\boldsymbol{\alpha} = 0.5$, $\lambda = 0.15$ and a maximum tree depth D; training details are given in Appendix D. The training loss is the sum of canonical cross-entropy losses in the context window, or equivalently, the sum of data compression rates. During testing, given a source sequence of length-N generated from an unknown VOMC with an order at most D, can the transformer learn this sequence efficiently, i.e., at a loss close to the optimal value over the whole context window? In Fig. 3, we show the ICL performances (the compression rates, equivalent to the cross-entropy losses) of the trained transformers with various numbers of layers, and the CTW and KN algorithms for $N = 1536$ and CT maximum order $D = 5$. The transformers have 8 attention heads with embedding dimension $E = 128$. The horizontal axis is the position in the context window, and the rates indicate how well transformers and other algorithms learn the underlying VOMCs in context.

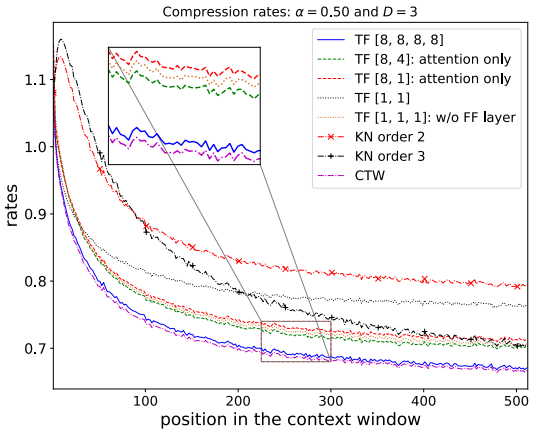

Figure 4: Attention-only vs. full transformers.

CTW is Bayesian optimal in this setting, whereas the KN-smoothing (with optimized hyperparameter) performs poorly, which is expected since it essentially reduces to an FOMC estimator at the assumed maximum order D_{kn}. If $D_{\text{kn}} < D$, then there is oversimplification; even when D_{kn} is sufficiently large, i.e., $D_{\text{kn}} \geq D$, it is a highly inefficient estimator for those suffixes at lower orders, thus performing poorly in finite-length context window. This effect is particularly pronounced at the latter part of the context window, e.g., a loss floor manifests when $D_{\text{kn}} < D$; at the beginning part of the context window, the potentially inaccurate prior and ill-matched FOMCS updates (escape symbols) jointly make KN's performance poorer. We can view KN as a reference method that does not adapt to variable orders. Additional experimental results with PPM are given in Appendix E.

Almost all trained transformers, except those with a single layer, can track the performance of the CTW algorithm fairly closely in the context window. The single-layer transformer's performance is similar to (only slightly better than) the unigram approach. Transformers' overall performance improves as the number of layers increases in general. Nevertheless, the improvements with increased numbers of layers are relatively small and appear to saturate at four layers. A construction is given in Section 4.4 to explain this phenomenon. We further notice that even transformers with two layers appear to learn in context quite well. In Section 4.5, we provide an explanation of this phenomenon using a simplified transformer construction and additional experimental results.

Insufficiency of Attention-only Networks. Previous works Edelman et al. (2024); Nichani et al. (2024) showed that a two-layer attention-only network (i.e., without attention layer normalization or the FF layer) can effectively perform ICL for FOMCs, where the second layer only requires a single attention head. Moreover, the authors of Rajaraman et al. (2024a) empirically observed that two-layer single-head transformers can perform ICL-FOMCs of at least order 4, and constructed a three-layer single-head attention-only network (without an FF layer but with attention layer normalization) that can perform ICL-FOMCs. We conducted experiments to verify whether these continue to hold for ICL-VOMCs, and the results are shown in Fig. 4, where the bracket after TF (transformer) indicates the number of heads in each layer. Observe that attention-only networks perform considerably worse than CTW and a full four-layer transformer. Two-layer single-head transformers, and three-layer single-head transformers without the FF layer, also perform poorly, particularly the former. These results show that the mechanisms given in Edelman et al. (2024); Nichani et al. (2024); Rajaraman et al. (2024a) for ICL-FOMCs cannot explain well how transformers perform ICL-VOMC, and the feed-forward networks need to play a more significant role.

4 THEORETICAL INTERPRETATIONS

4.1 The Transformer Architecture

We introduce a mathematical description of the L-layer decoder-only transformer model to fix the notation. Transformer interacts with sequential data, e.g., $x_1^N = (x_1, \ldots, x_N)$, where token x_i is a symbol from an alphabet (a.k.a. vocabulary) $\mathcal{A}$ with $A = |\mathcal{A}|$. Each token x_i is embedded into $\vec{h}_i^{(1)} \in \mathbb{R}^E$ by integrating the information of its value x_i and position i, where E is the embedding dimension.

Each layer of the transformer takes matrix $\vec{H}^{(\ell)} = [\vec{h}_1^{(\ell)}, \vec{h}_2^{(\ell)}, \ldots, \vec{h}_N^{(\ell)}]$, where $\vec{h}_i^{(\ell)} \in \mathbb{R}^E$, as its input and applies the multi-head attention (MHA) layer operation and the feed-forward layer operation, and the output is the input to the next layer, denoted as $\vec{H}^{(\ell+1)}$. The output of the multi-head attention (sub-)layer with $M^{(\ell)}$ heads is

$$\vec{a}_i^{(\ell)} = \text{MHA}\left(\vec{h}_i, \vec{H}; \{W_{O,m}^{(\ell)}, W_{Q,m}^{(\ell)}, W_{K,m}^{(\ell)}, W_{V,m}^{(\ell)}\}_{m=1}^{M^{(\ell)}}\right) \triangleq W_O^{(\ell)} \left[\vec{b}_{1,i}^{(\ell)}; \vec{b}_{2,i}^{(\ell)}; \ldots; \vec{b}_{M^{(\ell)},i}^{(\ell)}\right],$$

where $\{W_{Q,m}^{(\ell)}, W_{K,m}^{(\ell)}, W_{V,m}^{(\ell)}\}_{m=1}^{M^{(\ell)}}$ are the $E^{(\ell)} \times E$ query matrices, key matrices, and value matrices[1] at the ℓ-th layer and m is the index of the attention head, respectively, $W_O^{(\ell)}$ is the $E \times (M^{(\ell)} E^{(\ell)})$ output mapping matrix, and $\vec{b}_m^{(\ell)}$ is the output of the m-th attention head at this layer defined as

$$\vec{b}_{m,i}^{(\ell)} = (W_{V,m}^{(\ell)} [\vec{h}_1^{(\ell)}, \vec{h}_2^{(\ell)}, \ldots, \vec{h}_i^{(\ell)}])$$

[1] In practice, embedding dimension E is divisible by the number of heads $M^{(\ell)}$ and $E = M^{(\ell)} E^{(\ell)}$.

$$\cdot \operatorname{softmax}((W_{K,m}^{(\ell)}[\vec{h}_1^{(\ell)}, \vec{h}_2^{(\ell)}, \ldots, \vec{h}_i^{(\ell)}])^\top (W_{Q,m}^{(\ell)} \vec{h}_i^{(\ell)})),$$

where we used ";" to indicate vertical matrix concatenation and "," to indicate horizontal matrix concatenation. The attention layer has a residual connection, and the attention output together with the residual connection also goes through a feed-forward (sub-)layer with a residual connection

$$\begin{aligned}\vec{h}_i^{(\ell+1)} &= \mathrm{FF}(\vec{a}_i^{(\ell)}; W_1^{(\ell)}, W_2^{(\ell)}) \\ &= W_1^{(\ell)} \sigma(W_2^{(\ell)}(\vec{a}_i^{(\ell)} + \vec{h}_i^{(\ell)})) + (\vec{a}_i^{(\ell)} + \vec{h}_i^{(\ell)}),\end{aligned}$$

where σ is a non-linear activation function (e.g., ReLU or sigmoid). The output of the last (L-th) transformer layer $\vec{H}^{(L+1)}$ goes through a linear then softmax to predict the probability of generating the next symbol in the vocabulary $\mathcal{A}$ based on the past observations:

$$\hat{\vec{p}}_{i+1} = \operatorname{softmax}(W_O^{(L+1)} \vec{h}_i^{(L+1)}) \in \Delta_A, \quad i = 1, \ldots, N-1,$$

where Δ_A is the probability simplex on $\mathcal{A}$. The model is illustrated in Appendix F.

Transformers are (pre)-trained to predict next token by minimizing the cumulative log-loss (cross-entropy loss) $\mathbb{E}_{x_1^N}[\sum_{i=1}^{N-1} \vec{x}_{i+1}^\top \log(1/\hat{\vec{p}}_{i+1})]$, where $\vec{x}_i \in \mathbb{R}^A$ is the one-hot encoding of x_i and sequence x_1^N is sampled from some population of sources, e.g., sequences can be articles written by different authors and thus following different dynamics.

4.2 The Bayesian CTW Algorithm

The difficulty to identify and estimate accurately the underlying CT is in learning both of its components: the tree structure itself, and the probability distribution associated with each leaf node. The likelihood of a sequence x_1^i given x_{1-D}^0 for a CT with parameter $(T, \{p_s\}_{s\in\mathcal{L}(T)})$ is

$$\begin{aligned}P_{T,\{p_s\}}(x_1^i | x_{1-D}^0) &= \prod_{j=1}^{i} p_{\beta_{\mathcal{L}(T)}(x_{j-D},\ldots,x_{j-1})}(x_j) \\ &= \prod_{s\in\mathcal{L}(T)} \prod_{a\in\mathcal{A}} p_s(a)^{\vec{n}_{i,s}(a)}, \qquad (1)\end{aligned}$$

where $\vec{n}_{i,s}$ is the *count vector* associated with suffix s

$$\vec{n}_{i,s}(a) := \text{ the number of times symbol } a \in \mathcal{A} \text{ follows suffix } s \text{ in sequence } (x_1, \ldots, x_i). \quad (2)$$

Leveraging the multiplicative nature of the likelihood function, Willems et al. (1995) proposed the context tree weighting (CTW) algorithm. CTW estimates the probability of the sequence x_1^n by the auxiliary parameters $p_{n,s}^e, p_{n,s}^w$'s, where e stands for "estimation", and w stands for "weighted":

1. For each string s with $|s| \le D$, compute

$$p_{n,s}^e = \frac{\Gamma(\sum_{a\in\mathcal{A}} \boldsymbol{\alpha}(a))}{\Gamma(\sum_{a\in\mathcal{A}}(\vec{n}_s(a) + \boldsymbol{\alpha}(a)))} \prod_{q\in\mathcal{A}} \frac{\Gamma(\vec{n}_s(a) + \boldsymbol{\alpha}(a))}{\Gamma(\boldsymbol{\alpha}(a))},$$

   where $\vec{n}_s = \vec{n}_{n,s}$, $\Gamma(\cdot)$ is the Gamma function, and $\boldsymbol{\alpha}$ is a prior-related vector to be specified shortly. $p_{n,s}^e$ is computed from the statistics for that suffix s, i.e., $\vec{n}_s$.
2. From nodes in the D-th level to the 0-th level (i.e., root), iteratively compute

$$p_{n,s}^w := \begin{cases} p_{n,s}^e, & \text{if } |s| = D, \\ \lambda p_{n,s}^e + (1-\lambda)\prod_{q\in\mathcal{A}} p_{n,qs}^w, & \text{otherwise,} \end{cases}$$

   where qs is the string by appending symbol $q \in \mathcal{A}$ before the suffix s. This recursion weights p^e at a node with p^w's at its children nodes.

We illustrate this iterative computation and the corresponding information flow in Fig. 2.

Kontoyiannis et al. (2022) took the Bayesian view, and showed that the probability $p_{n,()}^w$ computed at the root has a Bayesian interpretation. A prior distribution π_{CTW} is introduced, over the CTs in the collection of CTs with a depth at most D, which is denoted as $\mathcal{T}(D)$. Recall that a CT consists of the tree structure T and the set of the transition distributions at the leaves $\{p_s \in \Delta_{\mathcal{A}}\}$. The prior distribution is given as $\pi_{\mathrm{CTW}}(T, (p_s)_{s\in\mathcal{L}(T)}) = \pi_D(T)\prod_{s\in\mathcal{L}(T)} \pi_p(p_s)$, where $\pi_D(\cdot)$ represents a bounded branching process that each node at a level lower than D stops branching with probability λ or branches to $|\mathcal{A}|$ children with probability $(1-\lambda)$; and $\pi_p(p_s)$ is the Dirichlet distribution:

$$\begin{aligned}\pi_D(T) &= (1-\lambda)^{(|\mathcal{L}(T)|-1)/(A-1)} \lambda^{|\mathcal{L}(T)|-|\mathcal{L}_D(T)|}, \\ \pi_p(p_s) &= \mathrm{Dir}(p_s; \{\boldsymbol{\alpha}(a)\}_{a\in\mathcal{A}}), \qquad (3)\end{aligned}$$

where $\mathcal{L}_D(T)$ is the set of the leaves of T at depth D and $\{\boldsymbol{\alpha}(a)\}_{a\in\mathcal{A}}$ are the Dirichlet parameters. A typical choice is $\boldsymbol{\alpha}(a) = 0.5$, corresponding to the Jeffreys prior. The Bayesian view implies that the sequences can be viewed as being generated hierarchically, as illustrated in Fig. 9 in the appendix.

Theorem 1. *(Kontoyiannis et al., 2022, Theorem 3.1) $p_{n,()}^w$ computed by CTW equals the Bayesian predicted probability under the prior π_{CTW} specified by $(D, \lambda, \boldsymbol{\alpha})$:*

$$\begin{aligned}p_{n,()}^w &= P_{\pi_{CTW}}(x_1^n | x_{1-D}^0) \\ &= \sum_{T\in\mathcal{T}(D)} \int P_{T,\{p_s\}}(x_1^n | x_{1-D}^0) \pi(T, \{p_s\}) \prod_{s\in\mathcal{L}(T)} \mathrm{d}p_s.\end{aligned}$$

This implies that CTW computes exactly the probability of sequence x_1^n under the prior π_{CTW} parameterized by $(D, \lambda, \boldsymbol{\alpha})$. The next-token probability can

the be computed sequentially as

$$P_{\pi_{\rm CTW}}(x_{i+1}|x_{1-D}^{i}) = P_{\pi_{\rm CTW}}(x_1^{i+1}|x_{1-D}^{0})/P_{\pi_{\rm CTW}}(x_1^{i}|x_{1-D}^{0}) \quad (4)$$

which achieves a code length at most $\lceil \log_2(1/p^w_{n,()}) \rceil$ (Willems et al., 1995, 1997), i.e., Bayesian optimal.

4.3 Analysis of Attention Maps

We analyzed the attention maps of the trained transformers, where a few distinguished patterns emerge. One pattern is solely relative-position dependent. In the first two panels of Fig. 5 (top), we observe off-diagonal stripes for these two attention heads, which are a few positions below the main diagonal. They can be a single off-diagonal or a collection of several off-diagonals. This indicates that the query position is attending positions at a few fixed but close distances ahead of itself. This pattern usually appears in the first or second layers of the transformers. Combining with the suffix structure in compression algorithms such as CTW, such an attention pattern suggests the suffix is being copied into the current query position for subsequent processing. The off-diagonal stripes may have a width greater than 1, as shown in the second panel, which can be viewed as copying a mixture of the tokens in the suffix, suggesting the flexibility of transformers in forming certain "soft" suffixes.

Another pattern, shown in the third panel, has more sophisticated spotty patterns, and the attention appears to depend more explicitly on the current token features instead of the position alone, and they usually appear in the second layer or above in the transformers. Taking query positions 350 and 362 for the attention head shown in the third panel of Fig. 5 (top), we plot in Fig. 5 (bottom) the positions in the data sequence that match their suffixes of length-3 using the stem plots with a black circle on top, and the attention values as the red stems with the diamonds on top. The positions match perfectly, though the attention weights have some variations among them. This attention pattern suggests that it is collecting statistical information for those positions with the matched suffix. Several attention heads present similar patterns but with different suffix lengths.

4.4 Interpretation via Constructions

We next connect transformers' ICL performance to that of CTW via constructions.

A New Representation of CTW. The original CTW algorithm computes the probability of the whole sequence for a particular next token realization. Though there exist studies for sequential computation in the literature Willems et al. (1997); Willems and Tjalkens (1997); Willems (1998a), only binary sequences were considered and they are incompatible with the transformer architecture. To construct meaningful transformers, we propose a novel representation of the predictive probability $P_{\pi_{\rm CTW}}(x_{n+1}|x_{1-D}^n)$ in the theorem below, based on a weighted blending of the next token prediction probability vectors for each potential suffix $s_{n,l} := x_{n-l+1}^n$ of length $l = 0, 1, \ldots, D$. The proof is given in Appendix G.1.

Theorem 2. *The next token probability is given as*

$$\vec{P}_{\pi_{CTW}}(x_{n+1}|x_1^n) = \sum_{l=0,\ldots,D} \omega_{n,l} \cdot \vec{p}_{n,s_{n,l}}(x_{n+1}),$$

$$\text{where } \vec{p}_{n,s_{n,l}}(a) := \frac{\boldsymbol{\alpha}(a) + \vec{n}_{n,s_{n,l}}(a)}{\sum_q (\boldsymbol{\alpha}(q) + \vec{n}_{n,s_{n,l}}(q))},$$

and $\boldsymbol{\omega_n} \in \Delta_{D+1}$ *which are defined recursively using*

$$\ln(\omega_{n,l}) - \ln(\omega_{n,l-1}) = \ln(1-\lambda) - \mathbb{I}_{l=D}\ln(\lambda) + \ell^e_{n,s_{n,l}} - \ell^e_{n,s_{n,l-1}} + \sum_{q\in\mathcal{A}} \ell^w_{n,qs_{n,l-1}} - \ell^w_{n,s_{n,l}},$$

for $l = 1, \ldots, D$, *where* $\ell^e_{n,s} := \ln(p^e_{n,s})$, $\ell^w_{n,s} := \ln(p^w_{n,s})$, *and* $\mathbb{I}_{(\cdot)}$ *is the indicator function.*

Consider an example where $x_{n-1} = c$, $x_n = a$, and $D = 2$. Each suffix $s_{n,l}$, e.g., $s_{n,0} = (), s_{n,2} = ba$, can potentially be the true suffix for the next token, i.e., $s_{n,l} \in \mathcal{L}(T)$. The Bayesian optimal next token prediction for $s_{n,l}$ is $\vec{p}_{n,s_{n,l}}$, and the weight $\omega_{n,l}$ assigns a "credibility" that $s_{n,l}$ is the true suffix. Theorem 2 states that these weights use information on the suffix path (e.g., root-*a*-*c* in this example) and from their siblings $p^w_{n,qs_{n,l-1}}$, but it is independent of the exact realization of the next token (see also Fig. 7). One obvious advantage of this representation is that, unlike the original CTW algorithm, which performs the computation specifically for the next token realization (4), the new representation computes the whole probability distribution (vector) and avoids repeated computation of many identical quantities. The computation of $\omega_{n,\ldots}$'s in the transformer architecture is non-trivial, which will be given shortly.

Transformer Construction for CTW: We next provide a construction of $(2 + D)$-layer transformer with sufficient representation power in the FF layer that can essentially approximate CTW, which demonstrates the capacity of transformers. The first two layers are motivated by the attention map patterns observed in Section 4.3, which collect statistics needed for Theorem 2; the last D layers are induction layers imitating the CTW procedure.

We assume a one-hot initial embedding, with additional scratchpad elements initialized as zeros and a

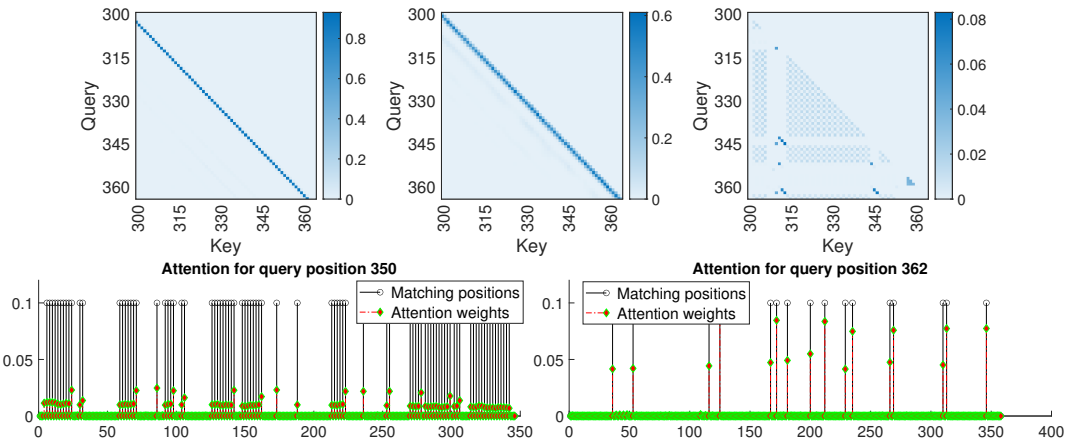

Figure 5: Top: Attention heatmaps. Bottom: Matching between suffix locations and attention values.

positional embedding, i.e., $\vec{h}_i^{(1)} = (\vec{x}_i; \vec{0}; p\vec{o}s_i)$ where $\vec{x}_i \in \mathbb{R}^A$ denotes the one-hot (column) embedding of x_i, $p\vec{o}s_i = (1, \cos(i\pi/N), \sin(i\pi/N))^\top$ is a sinusoidal positional embedding, and the remaining $(E - A - 3)$ elements being zero (E to be specified later).

We begin with the first layer, which is referred to as a finite-memory context-extension layer.

Theorem 3. *There is an M-headed transformer layer that performs finite-memory context-extension with the following output, using the initial one-hot embedded input $\vec{H}^{(1)}$:*

$$\vec{h}_i^{(2)} = (\vec{s}_{i,M+1}; \vec{0}; p\vec{o}s_i), \tag{5}$$

where $\vec{s}_{i,M+1} = (\vec{x}_i; \ldots; \vec{x}_{i-M})$ is the vector version of the M-length suffix $s_{i,M+1} = x_{i-M}^i$.

This layer essentially copies M past embedded symbols to the current position i, and stacks them below the current symbol $\vec{x}_i$. This operation utilizes the positional encoding $p\vec{o}s_i$ via rotation and matching the corresponding positions. Consistent with our empirical observations, we use multiple heads that scale with the parameter D, instead of complex embedding (e.g., embedding history in a high-precision scalar floating number) or that scales as $|\mathcal{A}|^D$. The proof is given in Appendix G.2.1.

The second layer is referred to as the statistics collection layer, which takes a sequence of vectors $\vec{h}_i^{(2)}$, $i = 1, \ldots, N$, defined in (5) as its input. To rigorously specify the function of this layer, we define the k-gram (forward) statistics vector $\vec{g}_{i,s}$ with $|s| = k - 1$, which in plain words, is the empirical probability distribution of the next token associated with the suffix s for sequence x_1^i. Similarly, we define the k-gram backward statistics vector $\vec{g}^{\leftarrow}_{i-1,s}$, which is the empirical probability distribution of the previous token associated with the suffix s for x_1^{i-1}. Mathematically, $\forall a \in \mathcal{A}$

$$\vec{g}_{i,s}(a) = \frac{\vec{n}_{i,s}(a)}{\sum_{q\in\mathcal{A}} \vec{n}_{i,s}(q)}, \; \vec{g}^{\leftarrow}_{i-1,s}(a) = \frac{\sum_{q\in\mathcal{A}} \vec{n}_{i,as}(q)}{\sum_{q\in\mathcal{A}} \vec{n}_{i,s}(q)},$$

where $\vec{n}_{i,s}$ is the counting vector defined in (2), and $\sum_{q\in\mathcal{A}} \vec{n}_{i,s}(q)$ is the number of appears of the string s in the sequence x_1^{i-1}. For both $\vec{g}_{i,s}$ and $\vec{g}^{\leftarrow}_{i-1,s}$, if the suffix s has not appeared in data x_1^{i-1}, it can be initialized arbitrarily as a vector in the probability simplex.

We have omitted the dimensionality of several zero matrices when they are obvious from the context. The first two layers are illustrated in Fig. 6.

Theorem 4. *There is an M'-head attention layer, where $M' \le M + 1$, which can collect statistics, defined by the following output, with $\vec{H}^{(2)}$ in (5) as its input:*

$$\vec{a}_i^{(2)} = (\vec{s}_{i,M+1}; \vec{g}_{i,M'}; \vec{g}^{\leftarrow}_{i-1,M'}; \vec{0}; p\vec{o}s_i), \tag{6}$$

where $\vec{g}_{i,M'} := (\vec{g}_{i,s_{i,0}}; \ldots; \vec{g}_{i,s_{i,M'-1}})$ and $\vec{g}^{\leftarrow}_{i-1,M'} = (\vec{g}^{\leftarrow}_{i-1,s_{i,0}}; \ldots; \vec{g}^{\leftarrow}_{i-1,s_{i,M'-1}})$.

This functional layer essentially collects k-gram statistics for various lengths of $k = 1, 2, \ldots, M'$. For example, when $k = 3$, it collects the normalized frequency associated with the suffix (x_{n-1}, x_n).

It is important to note that for ICL of FOMCs, two layers that collect forward statistics $\vec{g}_{i,M'}$ with $M' = D + 1$ are sufficient (Edelman et al., 2024). However, for the ICL-VOMC task, the underlying CT structure is unknown; therefore, collecting such simple statistics is *no longer sufficient*, and it is also the reason that the FF layer is important for VOMCs while less so for FOMCs. As indicated in Theorem 2, the information of counting statistics $\vec{n}_{i,s_{i,l}}$ is important since the weights heavily depend on $\vec{n}_{i,s}(a)$. Yet due to the softmax function, only (normalized) probability values can

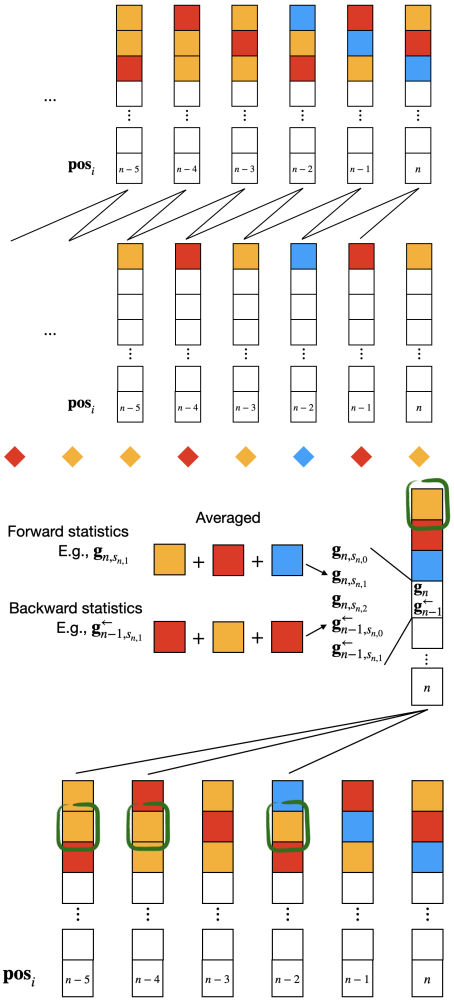

Figure 6: The finite-memory context-extension layer and the statistics collection layer for $D = 2$.

be obtained instead of the counts. With the backward statistics $\vec{g}^{\leftarrow}_{i,s}$, $\vec{n}_{i,s_{i,l}}$ can be derived as

$$\begin{aligned}\vec{n}_{i,s_{i,l}}(a) &= \frac{\vec{n}_{i,s_{i,l}}(a)}{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,l}}(q)}\cdots \\ &\quad \cdots\frac{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,1}}(q)}{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,0}}(q)}\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,0}}(q) \\ &= i\cdot\vec{g}_{i,s_{i,l}}(a)\prod_{j=0}^{l-1}\vec{g}^{\leftarrow}_{i-1,s_{i,j}}(x_{i-j}).\end{aligned}$$

Taking $M = M' - 1 = D$, after the statistics collection layer, a sufficiently wide FF layer with ReLU activation can compute (by the universal approximation capability) (Cybenko, 1989; Hornik et al., 1989).

$$\vec{h}_i^{(3)} = (\vec{s}_{i,D+1}; \vec{p}_{i,D}; \vec{l}^{\,e}_{i,D}; \ell^w_{i,s_{i,D}}; \vec{0}; p\vec{o}s_i),$$

where $\vec{p}_{i,D} = (\vec{p}_{i,s_{i,0}}; \vec{p}_{i,s_{i,1}}; \ldots; \vec{p}_{i,s_{i,D}})$ and $\vec{l}^{\,e}_{i,D} = (\ell^e_{i,s_{i,0}}; \ldots; \ell^e_{i,s_{i,D}})$ in Theorem 2.

To fulfill the Bayesian optimal prediction, we use the following induction layer that iteratively computes $\ell^w_{i,s}$ for suffixes on the suffix path and their siblings, as well as the weight differences, which are denoted by $\delta_{i,l} := \ln(\omega_{i,l}) - \ln(\omega_{i,l-1})$ for $l = D, D-1, \ldots, 1$. Specifically, the desired embedding

$$\begin{aligned}\vec{h}_i^{(\ell)} = (&\vec{s}_{i,M^{(1)}+1}; \vec{p}_{i,D}; \vec{l}^{\,e}_{i,D}; \delta_{i,D}; \delta_{i,D-1}; \ldots; \\ &\delta_{i,D-\ell+4}; \ell^w_{i,s_{i,D+3-\ell}}; \vec{0}; p\vec{o}s_i),\end{aligned} \tag{7}$$

for $\ell = 3, 4, \ldots, 3+D$.

Theorem 5. *There exists an A-head transformer layer that can perform the induction: Takes $\vec{H}^{(\ell)}$ in (7) as input and outputs $\vec{H}^{(\ell+1)}$, and the final output layer taking $\vec{H}^{(D+3)}$ as input can output the A-dim Bayesian optimal vector $\sum_{l=0,\ldots,D}\ \omega_{n,l}\vec{p}_{n,s_{n,l}}$.*

The last D-layer construction is shown in Fig. 7 with $D = 2$, where a recursive structure emerges to collect information from different locations of the context window to compute the weights $\omega_{n,\ldots}$'s; the resemblance to the recursion in Fig. 2 is clear.

Parallelization and Layers: A key consideration is for transformers to parallelize the computation at all token positions simultaneously. While a naive construction to perform CTW could certainly process the tokens sequentially as the CTW algorithm suggests, it would require the number of layers to grow linearly with the context window size. Theorem 2 is the theoretical tool allowing us to instead perform the same computation in $D+2$ steps, i.e., the $D+2$ layers.

The construction is certainly not unique. Consider a 2-layer transformer with a large embedding space, where the number of attention heads scales like $|\mathcal{A}|^D$. Each attention head could be precoded to collect the statistics for each possible suffix of length $\le D$, and all such information can be stored in $\vec{h}_i^{(3)}$, assuming a sufficiently large embedding space (or a more delicate embedding structure in a smaller space). Each $\vec{h}_i^{(3)}$ would then have all the information on the context tree, on which a wide FF layer could perform the full CTW algorithm to produce the next token distribution. We conscientiously forgo constructions like this, because in large models with $\mathcal{A}$ in the tens of thousands and D in the tens or hundreds, scaling as $|\mathcal{A}|^D$ is unrealistic, and the computation of the FF layer would be extremely complex and hard to interpret. More discussions on potential tradeoffs between layers and attention heads can be found in Rajaraman et al. (2024a).

4.5 Interpreting Two-Layer Transformers

We next consider how two-layer transformers can learn VOMCs in context. Our construction given above re-

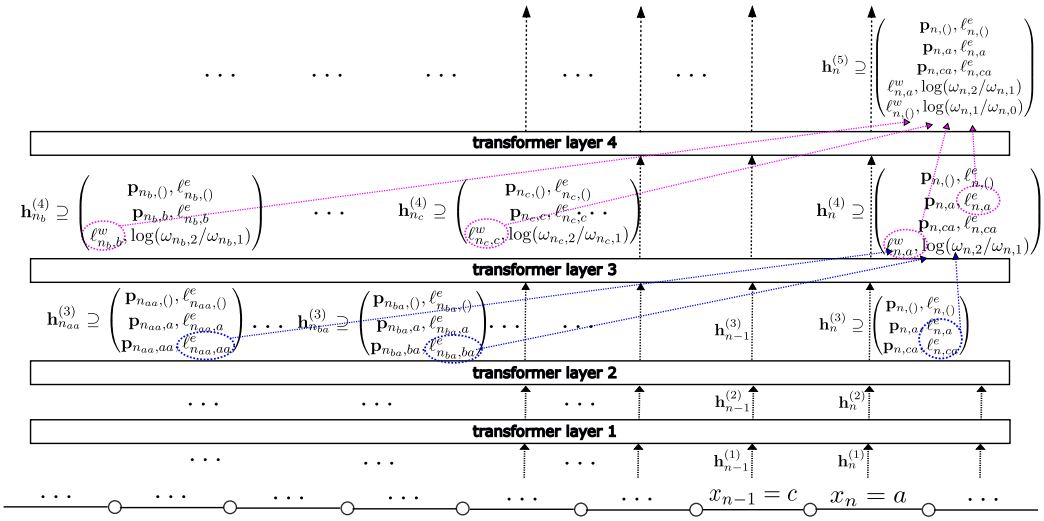

Figure 7: The last D layers mimic the CTW algorithms, and $\omega_{n,\cdot}$'s collect information over different locations.

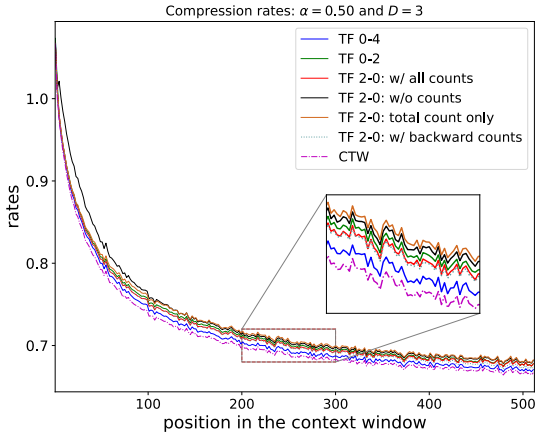

Figure 8: The performance of normal and synthetic 2-layer transformers.

quires $D+2$ layers, and we conjecture that the statistical information collected in the second layer can be used to approximate the optimal weights $\omega_{n,l}$ with an FF network.

To verify our conjecture, we implement the first two constructed transformer layers but control the statistics passed onto the FF layer. In Fig. 8, "TF 0-2" denotes the canonical 2-layer transformer; "TF 2-0" is the transformer with two constructed layers with output $\vec{a}_i^{(2)}$ in (6), followed by a trainable FF layer (the FF layer in the second layer of the transformer) and an output layer. The version "TF 2-0 w/o counts" does not contain $\vec{g}^{\leftarrow}_{i-1,M'}$ or $p\vec{o}s_i$ in $\vec{a}_i^{(2)}$; the version "TF 2-0 total counts only" does not contain $\vec{g}^{\leftarrow}_{i-1,M'}$ in $\vec{a}_i^{(2)}$ and $p\vec{o}s_i$ is replaced by the total count i; "TF 2-0 w/ all counts" replaces $\vec{g}^{\leftarrow}_{i-1,M'}$ and $p\vec{o}s_i$ with $\{\vec{n}_{n,s_{n,l}}\}_{l=0}^{D}$ and i. Even though their performances are rather clustered, we can make the following observations: 1) The performances degrade as more counting information is removed from the representation, and the counting information is clearly important, 2) The performances of "TF 2-0" and "TF 2-0 w/ all counts" almost match exactly, indicating the main function of the backward statistics $\vec{g}^{\leftarrow}_{i-1,M'}$ is to extract the counts, and 3) The performance of the canonical 2-layer transformer is similar to that of the constructed "TF 2-0" and "TF 2-0: w/ all counts" than those with less counting information. Therefore, the performances of these different implementations confirm the conjecture. Interestingly, Akyürek et al. (2024) proposed a similar learned-weight 2&3-gram algorithm to interpret how transformers learn finite automata, and suggested it may be optimal. In contrast, we obtain the (suboptimal) architecture by approximating the provable optimal transformer architecture.

5 CONCLUSION

We considered transformers' in-context learning of VOMCs. By drawing an analogy of ICL and Bayesian universal compression, we leverage the CTW and KN-smoothing as baselines. We observed that two-layer transforms can learn VOMCs in context. To understand the mechanism of transformers' ICL ability, we analyzed the attention maps and provided new transformer constructions. A future work is to extend the approach (Edelman et al., 2024; Nichani et al., 2024) to study the pretraining dynamics, to understand whether gradient-based training would yield the given constructions. Since attention-only networks do not perform well and multilayer networks are needed in this setting, such an analysis is considerably more difficult and may require a less conventional approach.

Acknowledgments

The work of R.D. Zhou and S. Diggavi was supported in part by the DEVCOM Army Research Laboratory under award # W911NF1720196, and by the National Science Foundation under award DMS-2502536. The work of C. Tian was partly supported by the National Science Foundation via grants DMS-2312173 and ECCS-2433631. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

References

Ahn, K., Cheng, X., Daneshmand, H., and Sra, S. (2024). Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36.

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. (2023). What learning algorithm is in-context learning? Investigations with linear models. In *The Eleventh International Conference on Learning Representations*.

Akyürek, E., Wang, B., Kim, Y., and Andreas, J. (2024). In-context language learning: Architectures and algorithms. In *International Conference on Machine Learning*, pages 787–812. PMLR.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. (2024). Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36.

Begleiter, R., El-Yaniv, R., and Yona, G. (2004). On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421.

Bietti, A., Cabannes, V., Bouchacourt, D., Jegou, H., and Bottou, L. (2024). Birth of a transformer: A memory viewpoint. *Advances in Neural Information Processing Systems*, 36.

Burrows, M. (1994). A block-sorting lossless data compression algorithm. *SRS Research Report*, 124.

Cleary, J. and Witten, I. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Cleary, J. G. and Teahan, W. J. (1997). Unbounded length contexts for PPM. *The Computer Journal*, 40(2_and_3):67–75.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dai, D., Sun, Y., Dong, L., Hao, Y., Ma, S., Sui, Z., and Wei, F. (2023). Why can GPT learn in-context? Language models secretly perform gradient descent as meta-optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019.

Deletang, G., Ruoss, A., Duquenne, P.-A., Catt, E., Genewein, T., Mattern, C., Grau-Moya, J., Wenliang, L. K., Aitchison, M., Orseau, L., et al. (2024). Language modeling is compression. In *The Twelfth International Conference on Learning Representations*.

Edelman, B. L., Edelman, E., Goel, S., Malach, E., and Tsilivis, N. (2024). The evolution of statistical induction heads: In-context learning Markov chains. *arXiv preprint arXiv:2402.11004*.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. (2022). What can transformers learn in-context? A case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hu, J., Liu, Q., and Jin, C. (2024). On limitation of transformer for learning HMMs. *arXiv preprint arXiv:2406.04089*.

Kirsch, L., Harrison, J., Sohl-Dickstein, J., and Metz, L. (2022). General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458*.

Kontoyiannis, I. (2023). Context-tree weighting and Bayesian context trees: Asymptotic and non-asymptotic justifications. *IEEE Transactions on Information Theory*.

Kontoyiannis, I., Mertzanis, L., Panotopoulou, A., Papageorgiou, I., and Skoularidou, M. (2022). Bayesian context trees: Modelling and exact inference for discrete time series. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(4):1287–1323.

Li, Y., Ildiz, M. E., Papailiopoulos, D., and Oymak, S. (2023). Transformers as algorithms: Generalization and stability in in-context learning. In *International Conference on Machine Learning*, pages 19565–19594. PMLR.

Makkuva, A. V., Bondaschi, M., Girish, A., Nagle, A., Jaggi, M., Kim, H., and Gastpar, M. (2024). Attention with Markov: A framework for principled

analysis of transformers via Markov chains. *arXiv preprint arXiv:2402.04161*.

Matsushima, T. and Hirasawa, S. (1994). A Bayes coding algorithm using context tree. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, page 386. IEEE.

Matsushima, T., Inazumi, H., and Hirasawa, S. (2002). A class of distortionless codes designed by Bayes decision theory. *IEEE Transactions on Information Theory*, 37(5):1288–1293.

Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Transactions on communications*, 38(11):1917–1921.

Ney, H., Essen, U., and Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Nichani, E., Damian, A., and Lee, J. D. (2024). How transformers learn causal structure with gradient descent. *arXiv preprint arXiv:2402.14735*.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. (2022). In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Pasco, R. (1976). *Source coding algorithms for fast data compression*. PhD thesis, Stanford University.

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

Rajaraman, N., Bondaschi, M., Ramchandran, K., Gastpar, M., and Makkuva, A. V. (2024a). Transformers on Markov data: Constant depth suffices. *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 38.

Rajaraman, N., Jiao, J., and Ramchandran, K. (2024b). An analysis of tokenization: Transformers under Markov data. *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 38.

Reddy, G. (2023). The mechanistic basis of data dependence and abrupt learning in an in-context classification task. *arXiv preprint arXiv:2312.03002*.

Rissanen, J. (1983). A universal data compression system. *IEEE Transactions on information theory*, 29(5):656–664.

Rissanen, J. and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162.

Rissannen, J. (1976). Generalized kraft's inequality and arithmetic coding. *IBM Journal on Research Development*, 20.

Sadakane, K., Okazaki, T., and Imai, H. (2000). Implementing the context tree weighting method for text compression. In *Proceedings DCC 2000. Data Compression Conference*, pages 123–132. IEEE.

Shkarin, D. (2002). PPM: One step to practicality. In *Proceedings DCC 2002. Data Compression Conference*, pages 202–211. IEEE.

Svete, A., Borenstein, N., Zhou, M., Augenstein, I., and Cotterell, R. (2024). Can transformers learn n-gram language models? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9851–9867.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. (2023). Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR.

Willems, F. (1998a). Reducing the complexity of the context-tree weighting method. In *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No. 98CH36252)*, page 347.

Willems, F. M. (1998b). The context-tree weighting method: Extensions. *IEEE Transactions on Information Theory*, 44(2):792–798.

Willems, F. M., Shtarkov, Y. M., and Tjalkens, T. J. (1995). The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664.

Willems, F. M., Shtarkov, Y. M., and Tjalkens, T. J. (1996). Context weighting for general finite-context sources. *IEEE transactions on information theory*, 42(5):1514–1520.

Willems, F. M. and Tjalkens, T. J. (1997). Complexity reduction of the context-tree weighting method. In *Symposium on Information Theory in the Benelux*.

Willems, F. M., Tjalkens, T. J., et al. (1997). *Complexity reduction of the context-tree weighting algorithm: A study for KPN research*. Euler Institute of Discrete Mathematics and its Applications.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. (2022). An explanation of in-context learning as implicit Bayesian inference. In *International Conference on Learning Representations*.

Zhang, Y., Singh, A. K., Latham, P. E., and Saxe, A. M. (2025a). Training dynamics of in-context learning in linear attention. In *Forty-second International Conference on Machine Learning*.

Zhang, Y., Zhang, F., Yang, Z., and Wang, Z. (2025b). What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. In *International Conference on Artificial Intelligence and Statistics*, pages 1684–1692. PMLR.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator, if your work uses existing assets. [Not Applicable]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

An Information-Theoretic Approach to Understanding Transformers' In-Context Learning of Variable-Order Markov Chains Supplementary Materials

A Related Work

There have been many efforts in studying the ICL capabilities of transformers. A significant recent development is the elucidation of the connection to gradient descent, particularly for linear regression tasks (Von Oswald et al., 2023; Akyürek et al., 2023; Dai et al., 2023; Ahn et al., 2024). Li et al. (2023) formulated the ICL problem as a multi-task learning problem and considered ICL for several simple problem settings for which the authors provide risk bounds for ICL of supervised learning algorithms in these problem settings. Kirsch et al. (2022) viewed the ICL problem as a meta-learner and studied the relation between tasks and model sizes. These approaches focused on the ICL of supervised learning tasks, such as classification and regression, while this work belongs to another direction of studying ICL for the next token prediction of some unknown underlying dynamics.

Olsson et al. (2022) studied the induction head, i.e., the forming of small k-gram attention in LLMs. Reddy (2023) studied the balance between ICL and in-weights learning, and observed the abrupt emergence of the induction head corresponds to the emergence of ICL. The induction head was generalized to the statistical induction head in (Edelman et al., 2024) mainly to study bigrams. We adopted it but further allowed more statistical induction heads for more suffixes to be included together, in the first two layers of the idealized transformer.

There have also been efforts to study transformers and the learning of Markov chains. Xie et al. (2022) viewed ICL as a Bayesian inference problem, where a latent concept determines an HHM, and the observations from the HHM can lead to the identification of the hidden concept. They studied the eventual ICL capability, i.e., when the number of in-context examples goes to infinity. Hu et al. (2024) studied the limitations of transformers on learning to perform belief inference for HMM sources compared to recurrent neural networks. The work in (Bietti et al., 2024) allowed a fixed-order Markov chain to switch to a new deterministic mode, and the authors studied the training behavior of the corresponding ICL task with this mode transition. Akyürek et al. (2024) made a comprehensive empirical comparison of various language models on random finite automata, and showed that the transformer performs the best among these models. Makkuva et al. (2024) studied the loss landscape during transformer training on sequences generated from a single fixed-order Markov chain, using a single-layer transformer. Neither of these studies considered ICL.

More recently, Rajaraman et al. (2024a) considered ICL of FOMCs with single-head transformers, and provided a construction to show that it is possible to use a single attention head to capture longer memory in the sequence. An empirical study on transformers' ICL capability for n-grams was conducted in Svete et al. (2024). Learning certain special Markov sources was studied from the perspective of tokenization usage in Rajaraman et al. (2024b), which showed that tokenizations can play an important role in the overall architecture. Particularly, it was observed that transformers may become difficult to train without a tokenizer, while training became easier with a tokenizer. The work most relevant to us is (Edelman et al., 2024), where ICL of a fixed-order Markov chain was considered, and the training behavior was studied both empirically and theoretically, and the formation of induction heads in a two-layer network was demonstrated. Another closely related work is Nichani et al. (2024), where a more general dependence graph structure among positions in the context window is allowed.

All these existing work assumed fixed-order Markov models or fixed-order HHMs, usually with orders kept at 1 or 2; moreover, they almost all focus on the emergence of the induction heads during training or the training landscape. Our study is different firstly in the variable-order nature of the Markov models, and secondly the focus on the on-time ICL performance instead of the training landscape and behavior.

Lossless data compression has a long history, with many different algorithms being developed over the years. The most popular general-purpose compression algorithms are perhaps the Lempel-Ziv compression algorithms (Ziv

and Lempel, 1977, 1978) and their variants, which belong to dictionary-based compression algorithms. These algorithms do not explicitly maintain any probabilistic models, and their efficiency comes from maintaining an efficiency dictionary of sequences that have been seen before, and are matched with future sequences. More powerful compression algorithms usually maintain probability models explicitly, which are then plugged into an arithmatic coding module (Rissannen, 1976; Pasco, 1976; Rissanen and Langdon, 1979) for efficient compression. The most well-known classes of algorithms in this category are the context-tree weighting algorithm (Willems et al., 1995; Begleiter et al., 2004; Kontoyiannis, 2023) and prediction by partial matching (Cleary and Witten, 1984). The former enjoys a strong theoretical guarantee, particularly on binary sources (Willems et al., 1995), but has some difficulty in its practical implementation (Willems, 1998b; Willems et al., 1996; Sadakane et al., 2000; Begleiter et al., 2004), particularly for large alphabet sizes and sequential data. The latter is based more on heuristics, and has been improved and extended in various ways (Cleary and Teahan, 1997; Moffat, 1990; Shkarin, 2002). Methods based on probabilistic modeling are usually more resource-intensive, though they have gained more popularity recently due to the increased availability of computing resources. The evaluation given in (Begleiter et al., 2004) suggests that CTW and PPM are the two most powerful compression algorithms in practice. There are other compression algorithms, such as those based on the Burrows-Wheeler transformation (Burrows, 1994), which does not explicitly maintain a probabilistic model, but are also not dictionary-based.

Universal compression is often used jointly with arithmetic coding (AC) (Rissanen and Langdon, 1979; Pasco, 1976), which requires an estimated probability distribution $\hat{P}$ for the next symbol to perform compression. AC is complicated, nevertheless, here it suffices to view it as a blackbox that compresses a symbol x with approximately $\ln(1/\hat{P}(x))$ nats, resulting in a rate roughly equal to the cross-entropy between $\hat{P}$ and P, that latter of which is the true distribution.

B The PPM Algorithm

The PPM algorithm (with finite memory of parameter D_{PPM}) blends several CTs by utilizing an escape symbol (Esc), and adaptively refines the CT model using the observed samples. The key idea is that the estimated probability distribution for an emitted symbol is only used when there were past observations of this string. For other cases, the escape symbol is encoded, indicating a shorter suffix needs to be used.

Table 1: PPM counts after observing string (a, b, c, a, b, b, c)

order $k=2$					order $k=1$					order $k=0$				order $k=-1$	
prediction			c	p	prediction			c	p	prediction		c	p	prediction	c p
(a,b)	$\to$	b	1	$\frac{1}{3}$	a	$\to$	b	2	$\frac{2}{3}$	$\to$	a	2	$\frac{1}{4}$	$\frac{1}{\lvert\mathcal{A}\rvert}$	
	$\to$	c	1	$\frac{1}{3}$		$\to$	Esc	1	$\frac{1}{3}$	$\to$	b	3	$\frac{3}{8}$		
	$\to$	Esc	1	$\frac{1}{3}$	b	$\to$	b	1	$\frac{1}{2}$	$\to$	c	2	$\frac{1}{4}$		
(b,b)	$\to$	c	1	$\frac{1}{2}$		$\to$	c	2	$\frac{1}{2}$	$\to$	Esc	1	$\frac{1}{8}$		
	$\to$	Esc	1	$\frac{1}{2}$		$\to$	Esc	1	$\frac{1}{2}$						
(b,c)	$\to$	a	1	$\frac{1}{2}$	c	$\to$	a	1	$\frac{1}{2}$						
	$\to$	Esc	1	$\frac{1}{2}$		$\to$	Esc	1	$\frac{1}{2}$						
(c,a)	$\to$	c	1	$\frac{1}{2}$											
	$\to$	Esc	1	$\frac{1}{2}$											

We illustrate this context tree blending approach by the example shown in Table. 1, where $\mathcal{A} = \{a, b, c\}$, and the memory length $D_{\mathrm{PPM}} = 2$. The escape pattern is assigned a count of one (method-A in (Moffat, 1990)). Suppose the next symbol to emit is a, then the probability prediction is $\frac{1}{2}$ from the $k = 2$ column; if on the other hand, the next symbol to emit is b, then the escape symbol is first encoded with probability $\frac{1}{2}$ since there is no string of (b, c, b) in the history, and then we check the column $k = 1$, and see that another escape symbol will be encoded since there is also no (c, b), and finally b will be encoded at $k = 0$, and the eventual effective probability for b is $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{3}{8}$.

Various refinements of the probability estimation can be adopted to further improve the performance, e.g., the exclusion rule, and other methods to initialize the probability for Esc; see e.g. (Moffat, 1990; Cleary and Teahan, 1997; Begleiter et al., 2004). As the number of observed samples accumulated, all patterns of $(D_{\mathrm{PPM}} + 1)$-grams will be observed at least once and the probability prediction will solely based on the column of maximum order $k = D_{\mathrm{PPM}}$.

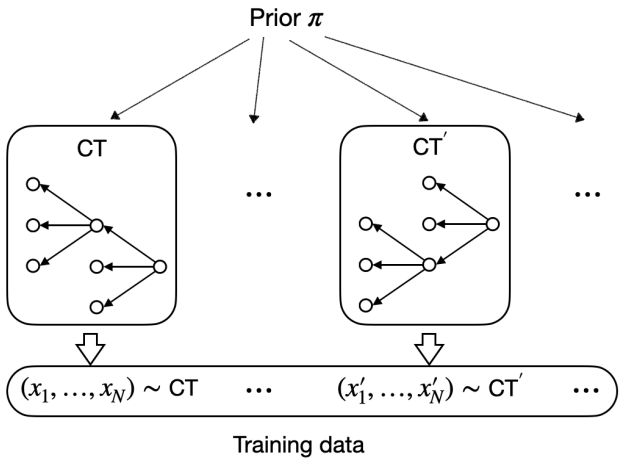

Figure 9: Training data generation

C Kneser–Ney Smoothing

Kneser–Ney smoothing is a method primarily used to calculate the next token probability distribution of $(D_{\text{kn}}+1)$-grams. It uses absolute discounting by subtracting a fixed value from the probability's lower-order terms to omit n-grams with lower frequencies.

First, consider $D_{\text{kn}} = 1$, i.e., Kneser–Ney smoothing based on bigram. Let $c(w, w')$ be the number of occurrences of the token w followed by the token w' in the context window before the current position, for any $w, w' \in \mathcal{A}$. Then the next token prediction using the bigram is given by

$$p_{\text{kn}}(x_{i+1}|x_i) = \frac{\max(c(x_i, x_{i+1}) - \delta, 0)}{\sum_{w' \in \mathcal{A}} c(x_i, w')} + \frac{\delta|\{w' : c(x_i, w') > 0\}|}{\sum_{w' \in \mathcal{A}} c(x_i, w')} p_{\text{kn}}(x_i), \tag{8}$$

where $P_{\text{kn}(w_i)}$ can be the standard unigram probability distribution estimate until current position or any other estimate of the unigram distribution of the same manner, $\delta \in (0, 1)$ is a hyperparameter that can be optimized. For KN-smoothing of order-D, the estimate can be computed recursively as

$$\begin{aligned} p_{\text{kn}}(x_{i+1}|x_{i-D+1}^i) =& \frac{\max(c(x_{i-D+1}^i, x_{i+1}) - \delta, 0)}{\sum_{w' \in \mathcal{A}} c(x_{i-D+1}^i, w')} \\ &+ \frac{\delta|\{w' : c(x_{i-D+1}^i, w') > 0\}|}{\sum_{w' \in \mathcal{A}} c(x_{i-D+1}^i, w')} p_{\text{kn}}(x_{i+1}|x_{i-D+2}^i), \end{aligned} \tag{9}$$

where $c(\vec{w}, w')$ is the occurrences of $(D+1)$-gram $(\vec{w}, w')$ until the current position. The hyperparameter δ controls how the estimates of the different models are blended, with a large δ being a higher preference for the lower-order estimate. The estimate almost reduces to a fixed-order estimate when there are more than a few previous observations of the same $(D_{\text{kn}}+1)$-grams.

D Training Details

We choose the alphabet size to be $|\mathcal{A}| = 3$ in the experiments. For training, we randomly generate $K = 20000$ CTs of various depths (maximum order $D \leq 5$), and then for each CT leaf, we generate a probability distribution. Two different ways of generating these probability distributions are taken: the first approach is to use the Dirichlet distribution to sample such distributions, and the second approach is to randomly select some of the elements in the alphabet to have probability zero, and the others with i.i.d. random values before normalization. Different values of the Dirichlet parameter are tested but only the results do not appear to be sensitive to the choice. For each CT, a source sequence of a certain length (e.g., $N_k = 5120$) is produced. The context window N can vary, but in most cases, we set it at 512 (except when $D = 5$, we set it to be 1536 to allow sufficient data collection in context). Each source sequence is segmented into $\lfloor N_k/N \rfloor$ training sequences.

During testing, we randomly generate multiple (8192 in our experiments) new CTs of varying depths using the same procedure, and for each CT, a sequence of length $N_k = 5120$ is generated, and then again segmented into a length of the context window for testing.

The transformer model is implemented using Pytorch, and trained using the AdamW optimizer with the default parameters. A100/T100 GPUs are used for training. Training a model requires roughly 4 to 6 hours. The batch size is set at 512, and the maximum epoch is set at 100 with early termination allowed after 15 epochs of no improvement. Testing was performed on a local workstation with a GeForce GTX 1660 Ti GPU card.

E Additional Experiments on PPM Algorithm

We replace the KN-smoothing used in Fig. 3 by the PPM algorithm (method A). Since there are many PPM variants, we do not choose and optimize the algorithm in our study, unlike the case for the KN-smoothing, where we optimize the hyperparameter. The results are shown in Fig. 10. It can be seen that the fixed-order estimate behavior is similar between PPM and KN-smoothing, though KN-smoothing appears to be overall more efficient than the specific PPM. For algorithms based on FOMC models, the mismatch between the underlying source and the model is most clear in the latter part of the context window, where the prior and update variations have mostly subsided. In the early part of the context windows, the prior mismatch and sub-optimal probability estimate update both manifest, and it may not be as informative. The similarity between the PPM behavior and the KN-smoothing behavior highlights the fact that the performance gap between CTW (and the transformers) and KN-smoothing (and PPM algorithms) is due to the underlying variable-order and fixed-order modelings, instead of the specific algorithms chosen.

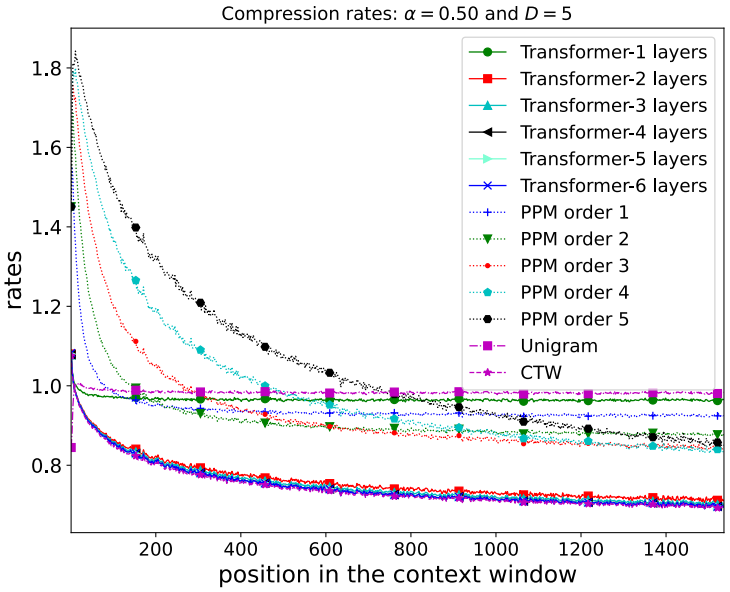

Figure 10: Performances of the PPM algorithm in comparison to trained transformers.

F Transformer Architecture

The transformer architecture used in our construction is illustrated in Fig. 11.

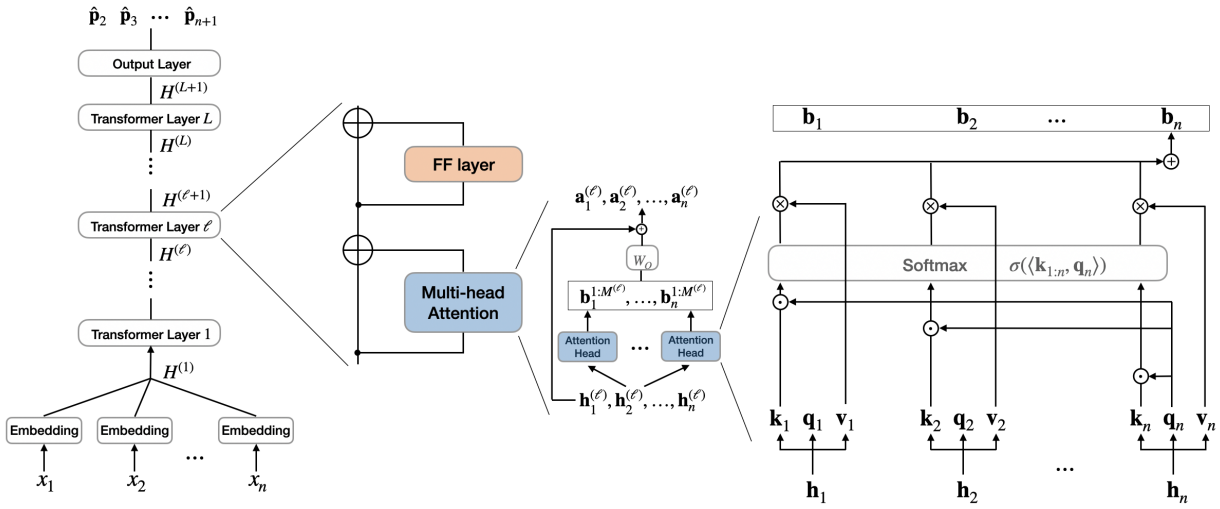

Figure 11: Transformer model

G Proofs of the Theorems for VOMC Sources

G.1 A New Formula for Bayesian Next Token Prediction

We aim to predict the next token x_{n+1} based on the observations $x_{1-D}^n = (x_{1-D}, \ldots, x_n)$ via a transformer-friendly formula. Note that x_{1-D}^0 is a place holder or dummy initialization sequence, which does not contain any information of the CT $(T, \{p_s\})$; or alternatively, we can view $P(T, \{p_s\}|x_{1-D}^0) = \pi_{\mathrm{CTW}}(T, \{p_s\})$ parameterized by $\lambda, \boldsymbol{\alpha}$.

Theorem 6 (Restate Theorem 2)**.** *The predicted probability can be computed as*

$$\vec{P}_{\pi_{CTW}}(x_{n+1}|x_{1-D}^n) = \sum_{l=0,\ldots,D} \omega_{n,l} \cdot \vec{p}_{n,s_{n,l}}(x_{n+1}), \tag{10}$$

where $\vec{p}_{n,s_{n,l}}(a) = \frac{\boldsymbol{\alpha}(a)+\vec{n}_{n,s_{n,l}}(a)}{\sum_q(\boldsymbol{\alpha}(q)+\vec{n}_{n,s_{n,l}}(q))}$*; and* $\omega_{n,\cdot} \in \Delta_{D+1}$ *with* $\ln(\omega_{n,l}) - \ln(\omega_{n,l-1}) = \ln(1-\lambda) - \mathbb{I}_{l=D}\ln(\lambda) + \ell^e_{n,s_{n,l}} - \ell^e_{n,s_{n,l-1}} + \sum_{q\in\mathcal{A}} \ell^w_{n,qs_{n,l-1}} - \ell^w_{n,s_{n,l}}$*, where* $\ell^e_{n,s} = \ln(p^e_{n,s})$*,* $\ell^w_{n,s} = \ln(p^w_{n,s})$*.*

Discussion. Note that $p^e_{n,s}, p^w_{n,s}$ can be efficiently calculated by the CTW procedure, and compared to calculate $\frac{P_{\pi_{\mathrm{CTW}}}(x_1^{n+1}|x_{1-D}^0)}{P_{\pi_{\mathrm{CTW}}}(x_1^n|x_{1-D}^0)}$ for each x_{n+1} the extra computation besides the CTW procedure is A times larger than that by Eq (10). The weighted average formula in Eq (10) gives a natural interpretation for the Bayesian optimal next token predicted probability. Each suffix along the root the leaf path $s_{n,0} - s_{n,1} - \cdots - s_{n,D}$ can potentially be the true suffix, i.e., $s_{n,l} \in \mathcal{L}(T)$, and $\vec{p}_{n,s_{n,l}}$ is in fact the Bayesian optimal next token prediction given $s_{n,l} \in \mathcal{L}(T)$.

The blending weights $\omega_{n,l}$'s are based on stopping probability λ, the information in the potential suffix path such as $p^e_{s_{n,s_{n,l}}}$ as well as the information from their siblings $p^w_{n,qs_{n,l-1}}$. We can interpret $p^e_{n,s}$ as the evidence (unnormalized likelihood) that $s \in \mathcal{L}(T)$, and $p^w_{n,s}$ as the evidence that $s \in T$, i.e., the underlying tree covers node s. Theorem 6 indicates that more weights are assigned to $s_{n,l}$ than $s_{n,l-1}$, i.e., $\omega_{n,l} > \omega_{n,l-1}$, if λ is smaller (i.e., node $s_{n,l-1}$ is more likely to branch and thus less likely to be a leaf node), $p^e_{n,s_{n,l}} - p^e_{n,s_{n,l-1}}$ is larger (i.e., $s_{n,l}$ has more evidence than $s_{n,l-1}$) and $\sum_{q\in\mathcal{A}} \ell^w_{n,qs_{n,l-1}} - \ell^w_{n,s_{n,l}}$ is larger (i.e., $s_{n,l}$'s siblings have more evidence to explain the data and thus $s_{n,l-1}$ is less likely to be a leaf node). The indicator function $\mathbb{I}_{l=D}$ is due to the maximum depth constraint on the branching process. Nodes at level $l = D$ automatically stop the branching, i.e., the branching-stopping probability is 1 for such nodes.

We remark that the form in (10) is similar in spirit to a formula in Matsushima and Hirasawa (1994). However, $\omega_{n,l}$ was not viewed as a computable quantity in that early work, but only as a conditional distribution value for their subsequent theoretical analysis.

Proof of Theorem 6. Recall $s_{i,l} = (x_{i-l+1}, \ldots, x_i)$ is the suffix at position i of length l. We omit D by writing

$\mathcal{T} = \mathcal{T}(D)$ when D is clear from the context. Define partition $\{\mathcal{T}_{s_{n,l}}\}_{0\le l\le D}$, that $\mathcal{T}_s = \{T \in \mathcal{T} : s \in \mathcal{L}(T)\}$ is the set of trees containing leaf s. The predicted probability can then be computed as

$$\begin{aligned}
\vec{P}_{\pi_{\mathrm{CTW}}}(x_{n+1}|x_{1-D}^n) &= \sum_{T\in\mathcal{T}} \int p(x_{n+1}|T,\{p_s\},x_{1-D}^n)\pi_{\mathrm{CTW}}(T,\{p_s\}|x_{1-D}^n)\Big(\prod_{s\in\mathcal{L}(T)}\mathrm{d}p_s\Big)\\
&= \sum_{l=0,\dots,D}\ \sum_{T\in\mathcal{T}_{s_{n,l}}} \int p_{s_{n,l}}(x_{n+1})\pi_{\mathrm{CTW}}(T,\{p_s\}|x_{1-D}^n)\Big(\prod_{s\in\mathcal{L}(T)}\mathrm{d}p_s\Big)\\
&= \sum_{l=0,\dots,D}\ \sum_{T\in\mathcal{T}_{s_{n,l}}} \int p_{s_{n,l}}(x_{n+1})\pi_D(T|x_{1-D}^n)\pi_p(p_{s_l}|T,x_{1-D}^n)\mathrm{d}p_{s_l}\\
&= \sum_{l=0,\dots,D}\ \sum_{T\in\mathcal{T}_{s_{n,l}}} \pi_D(T|x_{1-D}^n)\int p_{s_{n,l}}(x_{n+1})\pi_p(p_{s_l}|T,x_{1-D}^n)\mathrm{d}p_{s_l}\\
&= \sum_{l=0,\dots,D}\ \sum_{T\in\mathcal{T}_{s_{n,l}}} \pi_D(T|x_{1-D}^n)\int p_{s_{n,l}}(x_{n+1})\pi_p(p_{s_l}|\mathcal{T}_{s_{n,l}},x_{1-D}^n)\mathrm{d}p_{s_l}\\
&= \sum_{l=0,\dots,D}\ \Big(\sum_{T\in\mathcal{T}_{s_{n,l}}}\pi_D(T|x_{1-D}^n)\Big)\Big(\int p_{s_{n,l}}(x_{n+1})\pi_p(p_{s_l}|\mathcal{T}_{s_{n,l}},x_{1-D}^n)\mathrm{d}p_{s_l}\Big)\\
&= \sum_{l=0,\dots,D}\ \omega_{n,l}\cdot\vec{p}_{n,s_{n,l}}(x_{n+1}),
\end{aligned}\tag{11}$$

where the last equality is by the definition that

$$\omega_{n,l} = \sum_{T\in\mathcal{T}_{s_{n,l}}}\pi_D(T|x_{1-D}^n),\tag{12}$$

and the optimal prediction probability given suffix $s_{n,l}$ is

$$\vec{p}_{n,s_{n,l}}(a) = \frac{\boldsymbol{\alpha}(a)+\vec{n}_{n,s_{n,l}}(a)}{\sum_{q\in\mathcal{A}}(\boldsymbol{\alpha}(q)+\vec{n}_{n,s_{n,l}}(q))},\tag{13}$$

since for any $T\in\mathcal{T}_{s_l}$, the posterior of p_s follows the Dirichlet distribution

$$\pi_p(p_{s_l}|T,x_{1-D}^n) = \mathrm{Dir}(p_{s_l};\boldsymbol{\alpha}+\vec{n}_{n,s_{n,l}}),\tag{14}$$

with posterior mean $\mathbb{E}[p_{s_l}|T,x_{1-D}^n]\in\Delta_{\mathcal{A}}$ and proportional to $\boldsymbol{\alpha}+\vec{n}_{n,s_{n,l}}$, which we had simply denoted as $\pi_p(p_{s_l}|\mathcal{T}_{s_{n,l}},x_{1-D}^n)$.

Since the length of data n is fixed and clear from the context, let $\underline{x} = x_{1-D}^n$ be the sequence, and we omit n in the subscript of $p_{n,s}^e$, $p_{n,s}^w$ and $s_{n,l}$ for simplicity.

For any model $T\in\mathcal{T}(D)$, the posterior probability $\pi(T|\underline{x})$ is given by:

$$\pi_D(T|\underline{x}) = \frac{\pi_D(T)P_\pi(\underline{x}|T)}{P_\pi(\underline{x})} = \frac{\pi_D(T)\prod_{s\in\mathcal{L}(T)}p_s^e}{p_{()}^w},\tag{15}$$

where the denominator $P_\pi^*(\underline{x}) = p_{()}^w$ is the prior predictive likelihood computed by CTW given in Theorem 1, and the numerator is by $P_\pi(\underline{x}|T) = \prod_{s\in\mathcal{L}(T)}p_s^e$ in (Kontoyiannis et al., 2022, Lemma 2.2). Since $\omega_l = \sum_{T\in\mathcal{T}_{s_l}}\pi(T|\underline{x})$ by definition, we have for any $l = 1,2,\dots,D$,

$$\frac{\omega_l}{\omega_{l-1}} = \frac{\sum_{T'\in\mathcal{T}_{s_l}}\pi_D(T'|x)}{\sum_{T\in\mathcal{T}_{s_{l-1}}}\pi_D(T|x)} = \frac{\sum_{T'\in\mathcal{T}_{s_l}}\pi_D(T')\prod_{s\in\mathcal{L}(T')}p_s^e}{\sum_{T\in\mathcal{T}_{s_{l-1}}}\pi_D(T)\prod_{s\in\mathcal{L}(T)}p_s^e}.\tag{16}$$

Note that tree in $\mathcal{T}_{s_l}$ and trees in $\mathcal{T}_{s_{l-1}}$ share similarities. For any $T\in\mathcal{T}_{s_{l-1}}$, let $\mathcal{T}_{s_l;T} = \{T'\in\mathcal{T}_{s_l} : \mathcal{L}(T)\subset\mathcal{L}(T')\cup\{s_{l-1}\}\}$ be the set of trees that differs from T only at subtree $\mathrm{sub}(T';s_l) := \{\text{subtree of } T' \text{ with root at } s\}$.

Take any $l = 1, 2, \ldots, D-1$. For any $T \in \mathcal{T}_{s_{l-1}}$ and $T' \in \mathcal{T}_{s_l;T}$. Based on the definition of $\pi_D = (1-\lambda)^{(|\mathcal{L}(T)|-1)/(A-1)}\lambda^{|\mathcal{L}(T)|-|\mathcal{L}_D(T)|}$, it is not hard to verify that

$$\begin{aligned}\frac{\pi_D(T')}{\pi_D(T)} &= \frac{\pi_{D-l+1}(\mathrm{sub}(T';s_{l-1}))}{\pi_{D-l+1}(\mathrm{sub}(T;s_{l-1}))} \\ &= \frac{(1-\lambda)\pi_{D-l}(\mathrm{sub}(T';s_l))\prod_{s'_l\in\mathrm{sib}(s_l)}\pi_{D-l}(\mathrm{sub}(T';s'_l))}{\lambda} \\ &= (1-\lambda)\prod_{s'_l\in\mathrm{sib}(s_l)}\pi_{D-l}(\mathrm{sub}(T';s'_l)),\end{aligned}$$

where $\mathrm{sib}(s_{l+1}) = \{qs_l : q \in \mathcal{A} \text{ and } qs_l \neq s_{l+1}\}$ is set of siblings of s_{l+1}. We can interpret the ratio as follows. T' and T only differs at the $\mathrm{sub}(T';s_{l-1})$ and $\mathrm{sub}(T;s_{l-1})$. Since T' branches at node s_{l-1}, we thus have the numerator in the second equation, where $(1-\lambda)$ corresponds to the branching, and then compute for the subtrees. Note that T stops branching at s_{l-1} and T' stops branching at s_l, then $\pi_{D-l+1}(\mathrm{sub}(T;s_{l-1})) = \pi_{D-l}(\mathrm{sub}(T';s_l)) = \lambda$ equals to the stopping probability.

Given any suffix s with $|s| \le D$, it has been shown in (Kontoyiannis et al., 2022, Proof of Theorem 3.1) that for any $l \le D$,

$$p^w_s = \sum_{U\in\mathcal{T}(D-l)}\pi_{D-l}(U)\prod_{u\in\mathcal{L}(U)}p^e_{us}, \tag{17}$$

where $\mathcal{T}(D-l)$ is the set of trees with maximum depth $D-l$ and π_{D-l} is the prior for bounded branching process with maximum depth $D-l$. We thus have

$$\frac{\sum_{T'\in\mathcal{T}_{s_l;T}}\pi_D(T')\prod_{s\in\mathcal{L}(T')}p^e_s}{\pi_D(T)\prod_{s\in\mathcal{L}(T)}p^e_s} = \frac{\sum_{T'\in\mathcal{T}_{s_l;T}}\pi_D(T')\prod_{s\in\mathcal{L}(T')}p^e_s}{\pi_D(T)\prod_{s\in\mathcal{L}(T)}p^e_s} \tag{18}$$

$$= \sum_{T'\in\mathcal{T}_{s_l;T}}\frac{\pi_D(T')}{\pi_D(T)}\frac{\prod_{s\in\mathcal{L}(T')\setminus\mathcal{L}(T)}p^e_s}{p^e_{s_{l-1}}} \tag{19}$$

$$= \sum_{T'\in\mathcal{T}_{s_l;T}}\left((1-\lambda)\prod_{s'_l\in\mathrm{sib}(s_l)}\pi_{D-l}(\mathrm{sub}(T';s'_l))\right)\left(\frac{p^e_{s_l}\prod_{s'_l\in\mathrm{sib}(T;s_l)}\prod_{s\in\mathcal{L}(\mathrm{sub}(T';s'_l))}p^e_s}{p^e_{s_{l-1}}}\right) \tag{20}$$

$$= (1-\lambda)\frac{p^e_{s_l}}{p^e_{s_{l-1}}}\sum_{T'\in\mathcal{T}_{s_l;T}}\left(\prod_{s'_l\in\mathrm{sib}(s_l)}\pi_{D-l}(\mathrm{sub}(T';s'_l))\right)\left(\prod_{s'_l\in\mathrm{sib}(T;s_l)}\prod_{s\in\mathcal{L}(\mathrm{sub}(T';s'_l))}p^e_s\right) \tag{21}$$

$$= (1-\lambda)\frac{p^e_{s_l}}{p^e_{s_{l-1}}}\sum_{T'\in\mathcal{T}_{s_l;T}}\left(\prod_{s'_l\in\mathrm{sib}(s_l)}\pi_{D-l}(\mathrm{sub}(T';s'_l))\prod_{s\in\mathcal{L}(\mathrm{sub}(T';s'_l))}p^e_s\right) \tag{22}$$

$$= (1-\lambda)\frac{p^e_{s_l}}{p^e_{s_{l-1}}}\prod_{s'_l\in\mathrm{sib}(s_l)}\left(\sum_{U\in\mathcal{T}(D-l)}\pi_{D-l}(U)\prod_{u\in\mathcal{L}(U)}p^e_{us'_l}\right) \tag{23}$$

$$= \frac{(1-\lambda)p^e_{s_l}\prod_{a\neq s_l\setminus s_{l-1}}p^w_{as_{l-1}}}{p^e_{s_{l-1}}}. \tag{24}$$

Similarly, for any $T \in \mathcal{T}_{s_{D-1}}$ and $T' \in \mathcal{T}_{s_D;T}$, $\frac{\pi_D(T')}{\pi_D(T)} = \frac{1-\lambda}{\lambda}$, we can derive

$$\frac{\omega_D}{\omega_{D-1}} = \frac{(1-\lambda)p^e_{s_D}\prod_{a\neq s_D\setminus s_i}p^w_{as_{D-1}}}{\lambda p^e_{s_{D-1}}}, \tag{25}$$

in the same manner. The proof can then be concluded by taking the logarithm on both sides. □

G.2 Construction of Transformer for CTW

To make the presentation clear, in the following, we separate the layers by their functionality and present them separately. Recall that

$$\vec{a}_i^{(\ell)} = \mathrm{MHA}\left(\vec{h_i}, \vec{H}; \{W_{O,m}^{(\ell)}, W_{Q,m}^{(\ell)}, W_{K,m}^{(\ell)}, W_{V,m}^{(\ell)}\}_{m=1}^{M^{(\ell)}}\right) \triangleq W_O^{(\ell)}\left[\vec{b}_{1,i}^{(\ell)}; \vec{b}_{2,i}^{(\ell)}; \ldots; \vec{b}_{M^{(\ell)},i}^{(\ell)}\right],$$

where $\{W_{Q,m}^{(\ell)}, W_{K,m}^{(\ell)}, W_{V,m}^{(\ell)}\}_{m=1}^{M^{(\ell)}}$ are the $E^{(\ell)} \times E$ query matrices, key matrices, and value matrices and $W_O^{(\ell)}$ is the $E \times M^{(\ell)}E^{(\ell)}$ output mapping matrix. For simplicity of presentation, we take $E^{\ell} = E$ and $W_O^{\ell} = [\vec{I}; \vec{I}; \ldots; \vec{I}]$. It is not hard to see the following constructions can be applied to much smaller $E^{(\ell)}$ while taking W_O as a permutation matrix.

G.2.1 Finite-memory context-extension layer

We begin with the first layer, which is referred to as a finite-memory context-extension layer.

Theorem 7 (Restate Theorem 3)**.** *There is an M-headed transformer layer that can perform finite-memory context extension, defined by the following output, with the initial one-hot embedded input $\vec{H}^{(1)}$:*

$$\vec{h}_i^{(2)} = (\vec{s}_{i,M+1}; \vec{0}; p\vec{o}s_i) = (\vec{x}_i; \vec{x}_{i-1}; \ldots; \vec{x}_{i-M}; \vec{0}; p\vec{o}s_i), \tag{26}$$

where $\vec{s}_{i,M+1} = (\vec{x}_i; \ldots; \vec{x}_{i-M})$ is the vector version of the M-length suffix $s_{i,M+1} = x_{i-M}^i$.

Proof of Theorem 3. The multi-head attention in the first layer is consisted of $M^{(1)} = M$ heads parameterized by $(W_{Q,m}^{(1)}, W_{K,m}^{(1)}, W_{V,m}^{(1)})_{m=1,2,\ldots,M^{(1)}}$. Specifically, for $m = 1, 2, \ldots, M^{(1)}$,

$$W_{Q,m}^{(1)} = \begin{pmatrix} \vec{0} & \mathrm{Rot}(m) \\ \vec{0} & \vec{0} \end{pmatrix}, \quad W_{K,m}^{(1)} = \begin{pmatrix} \vec{0} & c\vec{I}^{2\times 2} \\ \vec{0} & \vec{0} \end{pmatrix}, \quad W_{V,m}^{(1)} = \begin{pmatrix} \vec{0}^{mA\times A} & \vec{0} \\ \vec{I}^{A\times A} & \vec{0} \\ \vec{0} & \vec{0} \end{pmatrix}, \tag{27}$$

where $\mathrm{Rot}(m) = \begin{pmatrix} \cos(m\pi/N) & \sin(m\pi/N) \\ -\sin(m\pi/N) & \cos(m\pi/N) \end{pmatrix}$ is a rotation matrix that rotates clockwise by an angle of $m\pi/C$, and $c \in \mathbb{R}_+$ is a temperature factor. The query, key, and value after the mapping are

$$W_{Q,m}^{(1)}\vec{h}_n^{(1)} = \begin{pmatrix} p\vec{o}s_{n-m} \\ \vec{0} \end{pmatrix}, \quad W_{K,m}^{(1)}\vec{h}_i^{(1)} = c\begin{pmatrix} p\vec{o}s_i \\ \vec{0} \end{pmatrix}, \quad W_{V,m}^{(1)}\vec{h}_i^{(1)} = \begin{pmatrix} \vec{0}^{mA\times 1} \\ \vec{x}_i \\ \vec{0} \end{pmatrix}. \tag{28}$$

Take $c = \infty$ or sufficiently large. It is seen that the m-th head essentially copies the m-th earlier symbol to stack at the $(m+1)$-th position below the original symbol $\vec{x}_i$. Together with the residual link, the attention layer gives exactly the $\vec{h}_i^{(2)}$ shown in (29) while the FF layer in this layer can be set to all zeros.

$$\vec{h}_i^{(2)} = (\vec{x}_i; \vec{x}_{i-1}; \vec{x}_{i-2}; \vec{x}_{i-M^{(1)}}; \vec{0}; p\vec{o}s_i) = (\vec{s}_{i,M^{(1)}+1}; \vec{0}; p\vec{o}s_i), \tag{29}$$

where $\vec{s}_{i,l} = (\vec{x}_i; \vec{x}_{i-1}; \cdots; \vec{x}_{i-l+1})$ is the one-hot embedded version of suffix $s_{i,l} = (x_{i-l+1}, \ldots, x_{i-1}, x_i)$. □

G.2.2 Statistics collection layer

Theorem 8 (Restate Theorem 4)**.** *There is an M'-head attention layer, where $M' \le M+1$, that can perform statistics collection, defined by the following output, with $\vec{H}^{(2)}$ in (5) as its input:*

$$\vec{a}_i^{(2)} = (\vec{s}_{i,M+1}; \vec{g}_{i,M'}; \vec{g}_{i-1,M'}^{\leftarrow}; \vec{0}; p\vec{o}s_i), \tag{30}$$

where $\vec{g}_{i,M'} := (\vec{g}_{i,s_{i,0}}; \ldots; \vec{g}_{i,s_{i,M'-1}})$ and $\vec{g}_{i-1,M'}^{\leftarrow} = (\vec{g}_{i-1,s_{i,0}}^{\leftarrow}; \ldots; \vec{g}_{i-1,s_{i,M'-1}}^{\leftarrow})$.

Proof of Theorem 4. To make the proof self-contained, we first recall some key notations. The second layer is referred to as the statistics collection layer, which uses a sequence of vectors $\vec{h}_i^{(2)}$, $i = 1, 2, \ldots, N$, defined in (5) as its input, restated as follows.

$$\vec{h}_i^{(2)} = (\vec{s}_{i,M+1}; \vec{0}; p\vec{o}s_i), \tag{31}$$

where $\vec{s}_{i,M+1} = (\vec{x}_i; \ldots; \vec{x}_{i-M})$. To rigorously specify the function of this layer, recall the definition of the k-gram statistics vector $\vec{g}_{i,s}$, which in plain words, is the empirical probability distribution of the next token associated with the suffix s for a sequence x_1^i. Mathematically, for a suffix s whose length is $k-1$ and the current position i,

$$\vec{g}_{i,s}(a) = \frac{\vec{n}_{i,s}(a)}{\sum_{q\in\mathcal{A}} \vec{n}_{i,s}(q)} \quad \forall a \in \mathcal{A}, \tag{32}$$

where $\vec{n}_{i,s}$ is the counting vector defined in (2).

The k-gram backward statistics vector $\vec{g}^{\leftarrow}_{i-1,s}$ is defined similarly, which is the empirical probability distribution of the previous token associated with the suffix s for data x_1^{i-1}, and mathematically

$$\vec{g}^{\leftarrow}_{i-1,s}(a) = \frac{\sum_{q\in\mathcal{A}} \vec{n}_{i,as}(q)}{\sum_{q\in\mathcal{A}} \vec{n}_{i,s}(q)} \quad \forall a \in \mathcal{A}, \tag{33}$$

where $\sum_{q\in\mathcal{A}} \vec{n}_{i,s}(q)$ is the number of appears of the sub-string s in the sequence x_1^{i-1}.

The multi-head attention in the second layer is consisted of $M^{(2)} = M' \le M^{(1)} + 1 = M + 1$ heads parameterized by $(W_{Q,m}^{(2)}, W_{K,m}^{(2)}, W_{V,m}^{(2)})_{m=0,1,2,\ldots,M^{(2)}-1}$. Specifically, for $m = 1, 2, \ldots, M^{(2)} - 1$,

$$W_{Q,m}^{(2)} = \begin{pmatrix} \vec{I}^{(m-1)A\times(m-1)A} & \vec{0} \\ \vec{0} & \vec{0} \end{pmatrix}, \; W_{K,m}^{(2)} = \begin{pmatrix} \vec{0}^{(m-1)A\times A} & c\vec{I}^{(m-1)A\times(m-1)A} & \vec{0} \\ \vec{0} & \vec{0} & \vec{0} \end{pmatrix}, \tag{34}$$

$$W_{V,m}^{(2)} = \begin{pmatrix} \vec{0}^{(M^{(1)}+m)A\times A} & \vec{0} \\ \vec{I}^{A\times A} & \vec{0} \\ \vec{0}^{(M^{(2)}-1)A\times A} & \vec{0} \\ \vec{0}^{A\times A} & [\vec{0}^{A\times(m-1)A}, \vec{I}^{A\times A}, \vec{0}] \\ \vec{0} & \vec{0} \end{pmatrix}. \tag{35}$$

The corresponding query, key, and value vectors after the mapping are

$$W_{Q,m}^{(2)}\vec{h}_n^{(2)} = \begin{pmatrix} \vec{s}_{n,m-1} \\ \vec{0} \end{pmatrix}, \quad W_{K,m}^{(2)}\vec{h}_i^{(2)} = c\begin{pmatrix} \vec{s}_{i-1,m-1} \\ \vec{0} \end{pmatrix}, \quad W_{V,m}^{(2)}\vec{h}_i^{(2)} = \begin{pmatrix} \vec{0}^{(M^{(1)}+m)A\times 1} \\ \vec{x}_i \\ \vec{0}^{(M^{(2)}-1)A\times 1} \\ \vec{x}_{i-m} \\ \vec{0} \end{pmatrix}.$$

For $m = M^{(2)}$, $W_{Q,m}^{(2)}, W_{K,m}^{(2)}$ are of the same structure, while $W_{V,m}^{(2)}$ does not contains that $\vec{I}^{A\times A}$ in that $[\vec{0}^{A\times(m-1)A}, \vec{I}^{A\times A}, \vec{0}]$ block, and thus $W_{V,m}^{(1)}\vec{h}_i^{(1)}$ does not have $\vec{x}_{i-m}$.

It is not hard to see that taking $c \to \infty$ gives

$$(\vec{s}_{i,M^{(1)}+1}; \vec{g}_{i,M^{(2)}-1}; \vec{g}^{\leftarrow}_{i-1,M^{(2)}-1}; \vec{0}; p\vec{o}s_i) = [\mathrm{MHA}(\vec{H}^{(2)}) + \vec{H}^{(2)}]_i, \tag{36}$$

where

$$\vec{g}_{i,M'} = (\vec{g}_{i,s_{i,0}}; \ldots; \vec{g}_{i,s_{i,M'-1}})$$
$$\vec{g}^{\leftarrow}_{i-1,M'} = (\vec{g}^{\leftarrow}_{i-1,s_{i,0}}; \ldots; \vec{g}^{\leftarrow}_{i-1,s_{i,M'-1}}).$$

□

Note that the counting vector can be obtained via

$$\vec{n}_{i,s_{i,l}}(a) = \frac{\vec{n}_{i,s_{i,l}}(a)}{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,l}}(q)}\frac{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,l}}(q)}{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,l-1}}(q)}\cdots\frac{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,1}}(q)}{\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,0}}(q)}\left(\sum_{q\in\mathcal{A}}\vec{n}_{i,s_{i,0}}(q)\right) \tag{37}$$

$$= \vec{g}_{i,s_{i,l}}(a)\left(\prod_{j=0}^{l-1}\vec{g}^{\leftarrow}_{i-1,s_{i,j}}(x_{i-j})\right)\cdot i, \tag{38}$$

by the information contained in vector $(\vec{s}_{i,M^{(1)}+1}; \vec{g}_{i,M^{(2)}-1}; \vec{g}^{\leftarrow}_{i-1,M^{(2)}-1}; \vec{0}; p\vec{o}s_i)$.

Since $p^e_{i,s_{i,l}}$ and $\vec{p}_{i,s_{i,l}}$ in (13) are simple functions of $\vec{n}_{i,s_{i,l}}$, they can be approximated by a sufficiently wide FF layer, yielding the following output

$$\vec{h}^{(3)}_i = (\vec{s}_{i,M^{(1)}+1}; \vec{p}_{i,D}; \vec{l}^{\,e}_{i,D}; \ln(p^w_{i,s_{i,D}}); \vec{0}; p\vec{o}s_i), \tag{39}$$

where $\vec{l}^{\,e}_{i,D}$ contains the logarithm of p^e along the path from root () to $(x_{i-d+1},\ldots,x_i)$, and $\vec{p}_{i,D}$ stacks the optimal prediction given suffices $s_{i,0},\ldots,s_{i,D}$, i.e.,

$$\vec{l}^{\,e}_{i,D} = (\ell^e_{i,s_{i,0}}; \ell^e_{i,s_{i,1}}; \ldots; \ell^e_{i,s_{i,D}}) = (\ln(p^e_{i,s_{i,0}}); \ln(p^e_{i,s_{i,1}}); \ldots; \ln(p^e_{i,s_{i,D}})), \tag{40}$$

$$\vec{p}_{i,D} = (\vec{p}_{i,s_{i,0}}; \vec{p}_{i,s_{i,1}}; \ldots; \vec{p}_{i,s_{i,D}}), \tag{41}$$

and $\ln(p^w_{i,s_{i,D}}) = \ln(p^e_{i,s_{i,D}})$ with suffix $|s_{i,D}| = D$. These quantities can be extracted, since they are functions of the statistics collected from $\vec{a}^{(2)}_i$.

This functional layer essentially collects k-gram statistics for various lengths of $k = 1, 2, \ldots, M^{(2)}$ via multi-head attention and then processes the statistics for the follow-up optimal scheme.

G.2.3 Inductive CTW layer

Recall the inputs of the inductive CTW layers at $\ell = 3, 4, \ldots, 3+D$ (layer-$(3+D)$ is a fictitious layer), which are also the outputs of layers $2, 3, \ldots, 2+D$, should be

$$\vec{h}^{(\ell)}_i = (\vec{s}_{i,M^{(1)}+1}; \vec{p}_{i,D}; \vec{l}^{\,e}_{i,D}; \delta_{i,D}; \delta_{i,D-1}; \ldots; \delta_{i,D-\ell+4}; \ell^w_{i,s_{i,D+3-\ell}}; \vec{0}; p\vec{o}s_i), \tag{42}$$

for $\ell = 3, 4, \ldots, 3+D$, where $\delta_{i,l} := \ln(\omega_{i,l}) - \ln(\omega_{i,l-1})$ for $l = D, D-1, \ldots, 1$ are the weight difference, and we take $M^{(1)} = D$.

Theorem 9 (Restatement of Theorem 5)**.** *There exists a A-head transformer layer that can perform the induction: Takes $\vec{H}^{(\ell)}$ in (42) as input and outputs $\vec{H}^{(\ell+1)}$. And the final readout layer taking $\vec{H}^{(D+2)}$ as input can output the A-dimensional Bayesian optimal next token prediction vector $P_{\pi_{CTW}}(\cdot|x^n_{1-D}) = \sum_{l=0,\ldots,D}\ \omega_{n,l}\vec{p}_{n,s_{n,l}}$.*

Proof of Theorem 5. For any fixed $\ell = 3, 4, \ldots, 2+D$, we specify the construction for the ℓ-th transformer layer. It contains A heads and for each $m = 1, 2, \ldots, A$, the Q, K, V matrices are

$$W^{(\ell)}_{Q,m} = \begin{pmatrix} \vec{I}^{(D+1-\ell)A\times(D+1-\ell)A} & \vec{0} \\ \vec{0} & [\vec{e}_m, \vec{0}^{A\times 2}] \\ \vec{0} & \vec{0} \\ \vec{0} & \vec{I}^{2\times 2} \end{pmatrix},\ W^{(\ell)}_{K,m} = \begin{pmatrix} c\vec{I}^{(D+2-\ell)A\times(D+2-\ell)A} & \vec{0} \\ \vec{0} & \vec{0} \\ \vec{0} & c\vec{I}^{2\times 2} \end{pmatrix},$$

$$W^{(\ell)}_{V,m} = \begin{pmatrix} \vec{0}^{(\mathrm{place}_\ell+m)\times(\mathrm{place}_\ell+m)} & \vec{0} \\ [\vec{0}^{1\times(\mathrm{place}_\ell-1)}, 1] & \vec{0} \\ \vec{0} & \vec{0} \end{pmatrix},$$

where $\vec{e}_m$ is the A-dimensional one-hot vector at position m, and $\mathrm{place}_\ell = (M^{(1)}+D+2)A + D + \ell - 1$ is the index of element $\ell^w_{i,s_{i,D+3-\ell}}$ in $\vec{h}^{(\ell)}_i$. The corresponding query, key, and value vectors after the mapping are

$$W^{(\ell)}_{Q,m}\vec{h}^{(\ell)}_n = \begin{pmatrix} \vec{s}_{n,D+1-\ell} \\ \vec{e}_m \\ \vec{0} \\ p\vec{o}s_n \end{pmatrix},\quad W^{(\ell)}_{K,m}\vec{h}^{(\ell)}_i = c\begin{pmatrix} \vec{s}_{i,D+2-\ell} \\ \vec{0} \\ p\vec{o}s_i \end{pmatrix},\quad W^{(\ell)}_{V,m}\vec{h}^{(\ell)}_i = \begin{pmatrix} \vec{0}^{(\mathrm{place}_\ell+m)\times 1} \\ \ell^w_{i,s_{i,D+3-\ell}} \\ \vec{0} \end{pmatrix}.$$

At position n, the query of m-th head will select the latest (due to positional embedding) position with suffix $[\vec{s}_{n,D+1-\ell}; \vec{e}_m]$, and append its ℓ^w at the end. It is not hard to see that taking $c \to \infty$ gives

$$\begin{aligned}\vec{a}_i^{(\ell)} &= [\mathrm{MHA}(\vec{H}^{(2)}) + \vec{H}^{(2)}]_i \\ &= (\vec{s}_{i,D+1}; \vec{p}_{i,D}; \vec{l}^e_{i,D}; \delta_{i,D}; \delta_{i,D-1}; \ldots; \delta_{i,D+4-\ell}; \ell^w_{i,s_{i,D+3-\ell}}; [\ell^w_{i,qs_{i,D+2-\ell}}]_{q\in\mathcal{A}}; \vec{0}; p\vec{o}s_i)\end{aligned}$$

Recall $\ln(\omega_{n,l}) - \ln(\omega_{n,l-1}) = \ln(1-\lambda) - \mathbb{I}_{l=D}\ln(\lambda) + \ell^e_{n,s_{n,l}} - \ell^e_{n,s_{n,l-1}} + \sum_{q\in\mathcal{A}} \ell^w_{n,qs_{n,l-1}} - \ell^w_{n,s_{n,l}}$ by Theorem 2. $\delta_{i,D+3-\ell} = \ln(\omega_{i,D+3-\ell}) - \ln(\omega_{i,D+2-\ell})$ can be computed by $\vec{a}_i^{(\ell)}$ and thus $\vec{h}_i^{(\ell+1)}$ can be conveniently computed via the FF layer following the ℓ-th multi-head attention layer.

The final readout layer approximates an A-dimensional vector

$$P_{\pi_{\mathrm{CTW}}}(\cdot|x_{1-D}^n) = \sum_{l=0,\ldots,D} \omega_{n,l} \cdot \vec{p}_{n,s_{n,l}}(\cdot), \tag{43}$$

by an FF layer taking input

$$\vec{h}_n^{(D+3)} = (\vec{s}_{n,M^{(1)}+1}; \vec{p}_{n,D}; \vec{l}^e_{n,D}; \delta_{n,D}; \ldots; \delta_{n,1}; \vec{0}; p\vec{o}s_i). \tag{44}$$

The proof is now complete. □