ON THE REASONING ABILITIES OF MASKED DIFFUSION LANGUAGE MODELS

Anej Svete*
ETH Zürich
asvete@inf.ethz.ch

Ashish Sabharwal
Allen Institute for AI
ashishs@allenai.org

ABSTRACT

Masked diffusion models (MDMs) for text offer a compelling alternative to traditional autoregressive language models. Parallel generation makes them efficient, but their computational capabilities and the limitations inherent to their parallelism remain largely unexplored. To this end, we characterize what types of reasoning problems MDMs can provably solve and how efficiently. We do this by connecting MDMs to the well-understood reasoning frameworks of chain of thought (CoT) and padded looped transformers (PLTs) in the finite-precision log-width setting: We show that MDMs and polynomially-padded PLTs are, in fact, equivalent in this setting, and that MDMs can solve all problems that CoT-augmented transformers can. Moreover, we showcase classes of problems (including regular languages) for which MDMs are inherently more efficient than CoT transformers, where parallel generation allows for substantially faster reasoning.

1 INTRODUCTION

Many complex problems can be decomposed into smaller, independent sub-problems, making them naturally suited for parallel computation. For example, we can compute the value of a mathematical expression by evaluating its sub-expressions independently and combining the results (see Fig. 2). However, dominant autoregressive language models (LMs) tackle these problems sequentially. Methods like chain of thought (CoT), for instance, generate solutions one step at a time, failing to capitalize on the underlying parallel structure. Parallel generation by masked diffusion models (MDMs) offers a compelling alternative. Recent advances have positioned MDMs as a viable contender to autoregressive LMs in language modeling, code generation, and even molecule design (Lou et al., 2024; Zhang et al., 2025; Sun et al., 2025). However, the fundamental reasoning capabilities of MDMs remain poorly understood, which limits the extent to which we can leverage their potential and apply them to appropriate tasks. This work bridges that gap by providing the first formal characterization of the expressivity of MDMs, clarifying their fundamental computational strengths and weaknesses.
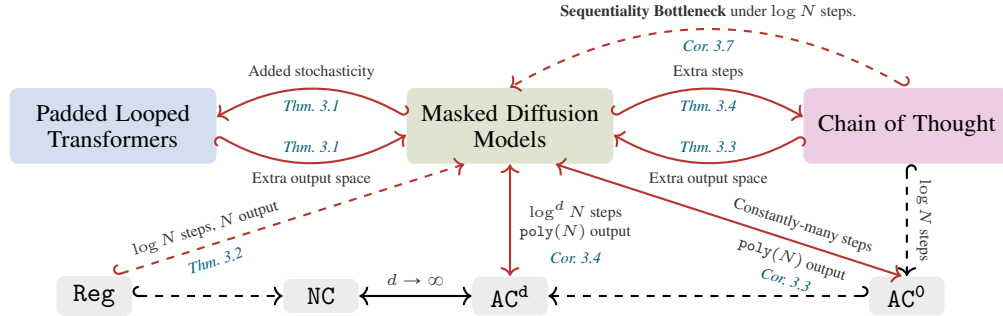

Figure 1: A summary of masked diffusion model expressivity. The colored nodes in the top row correspond to transformer-based computational models and the languages recognized by them. The gray nodes in the bottom row correspond to languages in (L-uniform) classical complexity classes. $\mathcal{X} \hookrightarrow \mathcal{Y}$ indicates the inclusion of $\mathcal{X}$ in $\mathcal{Y}$, and $\mathcal{X} \leftrightarrow \mathcal{Y}$ indicates equality. Dashed arrows represent strict inclusions. Red arrows denote novel results. Reg refers to all regular languages.

---

* This research was conducted while interning at the Allen Institute for AI.
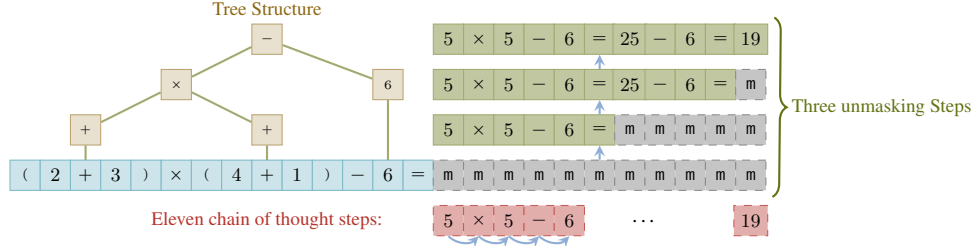
Figure 2: **Two strategies for solving a mathematical expression**. *(a) Parallel*: Parallel computation of intermediate values (three steps). *(b) Sequential*: Step-by-step generation (eleven steps).

Our analysis builds upon the study of LM expressivity—the formal characterization of the problems whose solution the neural architecture of an LM (with appropriate parameters) can express. While this field comprehensively describes autoregressive LMs, its findings do not directly apply to MDMs due to their fundamentally different, non-sequential processing of text, leaving the theoretical studies of these two paradigms largely in isolation. Prior theoretical work on MDMs has focused on the limitations of their factorized backward process and its convergence properties. This research has shown that while MDMs can approximate n-gram LMs with constantly many denoising steps, the number of steps must grow linearly with the string length even for simple LMs such as probabilistic regular languages (Feng et al., 2025; Li & Cai, 2025).[1] However, the asymptotic nature of these findings, together with the strict assumptions on the theoretical model, makes it difficult to draw concrete conclusions about the practical reasoning capabilities of MDMs, leaving a critical gap: A theoretical framework for MDMs that is both (1) formally rigorous enough to provide a comprehensive picture of how MDMs can use the combination of parallelism and iterative refinement for formal reasoning, and (2) faithful to how they are implemented in practice. Our work introduces such a framework, providing a tight characterization of their reasoning capabilities.

Concretely, we connect MDMs implemented as finite-precision transformers with logarithmically-growing model width to known reasoning paradigms of CoT (Wei et al., 2022), looping (Dehghani et al., 2019), and pause tokens (Lanham et al., 2023). For example, we formalize:

Takeaway 1 (*Thms. 3.3 and 3.4*). *MDMs can perform CoT reasoning with some overhead and the MDM denoising process can be (inefficiently) simulated by CoT by generating one symbol at a time.*

We also show how MDMs can solve complex problems by solving easier sub-problems in parallel, which makes them inherently more efficient than CoT on parallelizable problems.

Takeaway 2 (*Cor. 3.7*). *MDMs are provably more efficient at parallelizable problems than CoT.*

We refer to the fact that CoT *cannot* take advantage of this parallelism as the **sequentiality bottleneck** of CoT. Takeaway 2 highlights the potential efficiency gains of *parallel* CoT that generates multiple symbols at once. Our analysis, in fact, identifies a tighter and more natural connection between MDMs and this variant, which we refer to as pCoT. The parallelism of MDMs also facilitates a close connection with *looped* and *padded* transformers, where looping naturally maps to denoising steps and padding tokens to the generated tokens of an MDM. We find the class of problems solvable by MDMs is, in fact, precisely equivalent to that solvable by padded looped transformers.

Takeaway 3 (*Thm. 3.1*). *MDMs are equivalent to PLTs.*

These connections allow us to leverage known characterizations of CoT and PLTs together with classical complexity theory results to understand the fundamental capabilities and limits of MDMs (Li et al., 2024; Saunshi et al., 2025; London & Kanade, 2025; Svete et al., 2025). For example, with N representing the length of the input string and with $\texttt{AC}^\texttt{d}$ for $d \in \mathbb{N}$ being the standard class of Boolean circuits with AND, OR, and NOT gates and depth $\mathcal{O}(\log^d N)$, we obtain:

Takeaway 4 (*Thm. 3.2*). *MDMs with $\log N$ denoising steps can recognize regular languages.*

Takeaway 5 (*Cors. 3.3 and 3.4*). *For $d \in \mathbb{N}$, MDMs with $\mathcal{O}(\log^d N)$ denoising steps and $\texttt{poly}(N)$ output space are equivalent to $\texttt{AC}^\texttt{d}$. As $d \to \infty$, this yields $\texttt{NC}$, the class of all parallelizable problems.*

---

[1]We provide a detailed overview of related work in §A.

With constantly many denoising steps ($d = 0$), MDMs are equivalent to the limited class AC^0, i.e., they are no more powerful than standard transformers and, e.g., cannot recognize all regular languages.

We summarize these takeaways and their relationships in Fig. 1. Our proofs emphasize the affordances and difficulties in solving reasoning problems with MDMs. We find, for example, that the discrete nature of text generation, which serves as a communication channel between individual denoising steps, limits the amount of information that can be passed between steps. This necessitates extra output space to store the intermediate computations and is analogous to the discrepancy between fixed-precision and log-precision transformers (Li et al., 2024), and the difference between the classes AC and TC, the class of circuits with threshold gates. We find that *positional encodings* that carry information not computable by transformers themselves are crucial in locating this information. We also observe that, while unmasked attention makes MDM attention patterns flexible, it complicates left-to-right processing, which is often natural in human language—we find that exact implementation of causal masking requires quadratically more output space.

2 PRELIMINARIES

This section introduces the preliminaries and notation used throughout the paper. We reserve the main text for the high-level intuitions and defer technical details to §B.

2.1 TRANSFORMERS

We analyze (un)masked transformers for string generation and classification. We present the full model in §B.4 and focus here on the most relevant aspects.

Transformer families. We study *finite precision* transformers where the value of each parameter and activation is represented with a fixed number of bits. We allow the model *width*, i.e., the size of the contextual representations, to grow logarithmically with the input length N. This is a standard assumption in the literature on transformer expressivity (Li et al., 2024) since it is necessary and sufficient for the model to uniquely identify input positions, and aligns with modern implementations of *quantized* but *wide* transformers. The growing width results in a separate transformer $\mathcal{T}_N$ for each N, yielding a **family** $\{\mathcal{T}_N\}_{N \in \mathbb{N}}$ of transformers. We enforce L-uniformity in the family by requiring an associated Turing machine that constructs $\mathcal{T}_N$ in $\mathcal{O}(\log N)$ space (cf. §B.4).

(Parallel) CoT transformers. CoT reasoning enables sequential processing by solving problems in multiple steps (Wei et al., 2022). It is an integral part of today's popular "reasoning" models and substantially increases transformers' expressivity (Li et al., 2024; Merrill & Sabharwal, 2024). We define our idealization of CoT transformers in §B.4, including *parallel* CoT transformers, which predict $P' \in \mathbb{N}$ symbols in parallel at each time step, enabling some parallelism if the task allows it.[2] We denote the classes of CoT and parallel CoT transformers by CoT and pCoT, respectively.

Padded looped transformers (PLTs). Looped transformers repeatedly apply a fixed block of transformer layers to the input (Dehghani et al., 2019). This dynamically increases the depth of the model, enabling more complex reasoning, and does not increase the model size, as the same blocks are reused, thus reducing the memory footprint and computational cost (Bae et al., 2025). Such reasoning steps include both sequential and parallel processing, resulting in both efficiency as well as depth of the reasoning process. Padded transformers additionally pad the input with blank symbols, which can be used to perform additional computations *in parallel*. This additional padding space is analogous to increasing the circuit width in circuit complexity. We additionally provide PLTs with external noise applied to the residual stream at each step, which enables stochastic computations (cf. §B.6). We denote the class of padded looped transformers by PLT.

Transformer LMs. An **alphabet** Σ is a finite, non-empty set of **symbols**. Its Kleene closure is $\Sigma^* \stackrel{\text{def}}{=} \bigcup_{n=0}^{\infty} \Sigma^n$, the set of all strings. A **language model** is a distribution over Σ^*. Most LMs are **autoregressive**—they define **next-symbol** distributions $\overrightarrow{p}(\cdot \mid \boldsymbol{w})$ over $\overline{\Sigma} \stackrel{\text{def}}{=} \Sigma \cup \{\text{EOS}\}$ for $\boldsymbol{w} \in \Sigma^*$, where $\text{EOS} \notin \Sigma$ is the end-of-string symbol. A transformer-based LM computes $\overrightarrow{p}(\cdot \mid \boldsymbol{w})$ by linearly transforming the contextual representation of the final symbol to the logits of a distribution over

---

[2]This is different from *speculative decoding* (Leviathan et al., 2023), which generates multiple symbols *one a time* with a smaller LM and then evaluates their probability in a single pass of a larger LM.

$\overline{\Sigma}$. Moreover, contextual representations can be used for **infilling**—predicting symbols at masked positions. Infilling probabilities $p^{\downarrow}(\cdot \mid \boldsymbol{w})$ at masked positions in $\boldsymbol{w} \in \Sigma_{\mathsf{m}}{}^{*}$ are distributions over Σ, where $\mathsf{m} \notin \Sigma$ is the mask symbol and $\Sigma_{\mathsf{m}} \overset{\text{def}}{=} \Sigma \cup \{\mathsf{m}\}$.

Transformers and formal languages. Plenty of work describes transformer capabilities and limitations with formal languages (Strobl et al., 2024). These studies typically frame transformers as **language recognizers**, i.e., classifiers that decide whether a string $\boldsymbol{w} \in \Sigma^{*}$ belongs to some formal language $\mathcal{L} \subseteq \Sigma^{*}$ (Butoi et al., 2025). String membership is usually *deterministic* and can be formalized by determinizing the LM defined by a transformer: The next-symbol and infilling probabilities are used to decode the most probable symbol or decision.[3] The final prediction of $\mathbb{1}\{\boldsymbol{w} \in \mathcal{L}\} \in \{1, 0\}$ can be made (1) *in a single pass* by classifying based on the contextual representation of a particular symbol in the string, analogous to classifying based on the CLS symbol in BERT (Devlin et al., 2019), or (2) after *"reasoning"*, i.e., solving the problem in multiple time steps. In this case, the transformer's prediction is only made after a sequence of intermediate predictions that augment its computation and help the final decision. This is analogous to using CoT reasoning for string recognition and can be thought of as simulating a Turing machine with each step of the CoT process.

Particularly fruitful has been the study of transformers as **Boolean circuits**. In particular, our idealization of transformers falls under AC^0 circuits (Li et al., 2024)—Boolean circuits of constant depth, polynomial size, and with AND, OR, and NOT gates of unbounded fan-in—and captures the entire class if padding is allowed (London & Kanade, 2025). Other idealizations of transformers can compute functions outside of AC^0 (Li et al., 2024; Merrill & Sabharwal, 2025a) but remain in TC^0, the class of *threshold circuits* which add threshold gates (which determine whether the number of inputs exceeds some threshold) to AC^0 circuits. §§ B.2 and B.4 provide more details on circuit classes and their relation to transformers.

2.2 MASKED DIFFUSION LANGUAGE MODELS

Discrete diffusion LMs define a distribution over Σ^{*} by progressively denoising noisy strings sampled from some fixed distribution. Formally, they define a **forward (noising) process** and a **reverse (denoising) process**. The forward process defines a Markov chain over strings that iteratively corrupts them. Common examples include replacing symbols uniformly at random (uniform diffusion) or masking them with the mask symbol m (**masked diffusion models**, MDMs). The latter is the focus of this work. In this setting, the forward process starts from an initial string $\boldsymbol{w}^{(0)} \in \Sigma^{*}$ of some pre-determined length P and, at each of the T (discrete) steps, independently masks symbols with probability determined by a masking schedule $\alpha\left(\frac{t}{T}\right) \in [0, 1]$:

$$q_{t|0}(\boldsymbol{w}^{(t)} \mid \boldsymbol{w}^{(0)}) = \prod_{n=1}^{N} q_{t|0}(w_n^{(t)} \mid w_n^{(0)}), \qquad q_{t|0}(w_n^{(t)} \mid w_n^{(0)}) = \begin{cases} 1 - \alpha(\frac{t}{T}), & \textbf{if } w_n^{(t)} = \mathsf{m} \\ \alpha(\frac{t}{T}), & \textbf{otherwise} \end{cases}.$$

The masking schedule is set such that $\alpha(0) = 1$ (no masking at the start) and $\alpha(1) = 0$ (fully masked at the end, meaning that the noise distribution is the Dirac delta on the fully masked string).

Starting from the fully masked input, the reverse process $q_{0|T}$ inverts the forward process $q_{T|0}$ by (1) uniformly selecting some positions to unmask, and (2) sampling the chosen unmasked symbols. After T denoising steps, $q_{0|T}$ produces a string $\boldsymbol{w}^{(0)}$ sampled from the LM defined by the diffusion process. It is this reverse process that is learned from data. Its analytical form is generally intractable, so one usually models a parameterized approximation of a single denoising step $\hat{q}_{t-1|t}(\boldsymbol{w}^{(t-1)} \mid \boldsymbol{w}^{(t)})$, typically implemented as a transformer, that *factorizes* across positions:

$$\hat{q}_{t-1|t}(\boldsymbol{w}^{(t-1)} \mid \boldsymbol{w}^{(t)}) = \prod_{n=1}^{N} \hat{q}_{t-1|t}(w_n^{(t-1)} \mid \boldsymbol{w}^{(t)}) \tag{1}$$

Eq. (1) enables *parallel generation* but ignores inter-symbol dependencies at each denoising step.

Much of the existing work on MDM expressivity analyzes the convergence of $\hat{q}_{t-1|t}$ to $q_{0|T}$ (Li & Cai, 2025; Chen & Ying, 2024; Feng et al., 2025). Studying convergence properties usually requires

---

[3]Some work also considers transformers as LMs directly (Svete & Cotterell, 2024; Nowak et al., 2024; Borenstein et al., 2024; Svete et al., 2024) and shows the (probabilistic) gains afforded by CoT reasoning.

assuming uniform unmasking and a good approximation of the ground-truth model (e.g., Li & Cai, 2025; Chen & Ying, 2024; Feng et al., 2025; Liu, 2025).

Assumption 2.1 (Uniform unmasking). *The **uniform unmasking assumption** states that $\widehat{q}_{t-1|t}$ generates strings $\boldsymbol{w}^{(t-1)}$ from $\boldsymbol{w}^{(t)}$ by uniformly selecting positions to unmask.*

Assumption 2.2 (Perfect approximation). *Let $q_{t-1|t}$ be the backward processes of an MDM and $p^{\downarrow}$ a transformer-based LM. The **perfect approximation assumption** states that $q_{t-1|t}(w_n^{(t-1)} \mid \boldsymbol{w}^{(t)}) = p^{\downarrow}(w_n^{(t-1)} \mid \boldsymbol{w}^{(t)})$ for all $\boldsymbol{w}^{(t)} \in \Sigma_{\mathtt{m}}^{*}$, $w_n^{(t-1)} \in \Sigma$, and $n \in \{1, \ldots, |\boldsymbol{w}^{(t)}|\}$.*

In words, Assumption 2.2 states that the transformer perfectly models *all* conditional distributions of the diffusion process.[4] While this seems necessary, the following observation, proved in §E, shows that Assumptions 2.1 and 2.2 severely limit the class of functions that the model can compute.

Theorem 2.1. *If Assumptions 2.1 and 2.2 hold for an LM p, p cannot compute non-$\mathtt{AC}^0$ functions.*[5,6]

By assuming that the MDM is unable to choose which positions to unmask, the model has no choice in which sub-problems to solve first, which ignores the possibility of problem decomposition and requires the model to be equally good at solving *any* sub-problem—including predicting the final answer based on the input directly (with no reasoning steps). This implies that the problem is solvable in a single prediction step of a transformer. However, the expressivity of a single transformer pass is limited—Thm. 2.1 uses the fact that fixed-depth transformers lie in $\mathtt{AC}^0$ (Li et al., 2024). However, not *all* conditional probabilities have to be known to be able to solve algorithms in few steps. Intuitively, by *choosing* to solve *simple* subproblems with non-random unmasking, an MDM can avoid the difficult parts. For example, given the current arithmetic expression, one only has to predict the next set of simplifications—which are simple functions of the current expression. This motivates us to *loosen* Assumptions 2.1 and 2.2, which we do in our idealization of an MDM.

2.2.1 OUR IDEALIZATION OF MASKED DIFFUSION MODELS

We aim to understand the expressivity of the *reverse process*. To this end, we introduce an idealization that captures its key aspects—iterative unmasking and infilling—and provides a principled lens for understanding the expressivity of practical MDMs and comparing them to well-known paradigms such CoT. Here, we describe the high-level ideas; see §B.7 for the full formal model.

We formalize the reverse process with two components: A **planner** that *decides* which positions to unmask at each step and a **predictor** that samples the symbols at the unmasked positions. This loosens Assumption 2.1 and generalizes standard MDMs in which the planner is implicitly defined by choosing the positions to unmask uniformly at random. It also mirrors popular MDM implementations that generate text by selecting a subset of masked positions—for example, based on model confidence or according to a learned policy—and predicting the symbols conditioned on the current partially unmasked string (Ghazvininejad et al., 2019; Peng et al., 2025; Zheng et al., 2024; Liu et al., 2025a; Kim et al., 2025; Ben-Hamu et al., 2025).[7] Discarding Assumption 2.1 also sidesteps limitations of the position-wise independence in Eq. (1), a restriction that prevents MDMs with uniform unmasking matching even simple distributions exactly (Feng et al., 2025; Wu et al., 2025). In reasoning problems with a deterministic sequential structure, the ability to decide what to unmask enables problem decomposition into a sequence of deterministic steps that can be solved in parallel.[8] To connect our analysis to practical implementations, we assume that the planner and predictor are implemented as transformers.

We allow the planner to choose to *resample* already unmasked positions. This overcomes another key limitation—the inability to revert decisions and correct earlier mistakes—a challenge that is the focus of much recent research (von Rütte et al., 2025; Song et al., 2025, *inter alia*).[9] While existing

---

[4]Or *approximates* them well, requiring the error to be smaller than some $\epsilon > 0$.

[5]That is, p can only implement LMs whose next-symbol logits can be computed by $\mathtt{AC}^0$ circuits (Liu, 2025).

[6]An analogous version of the theorem applies to transformers in $\mathtt{TC}^0$.

[7]Thm. C.1 in §C.1 shows that the planner and predictor can be fused into a single model that unmasks symbols based on their confidence. This means that all our results apply to this popular model of unmasking.

[8]This is related to the importance of the mutual information and correlations between symbols to MDM performance (Li & Cai, 2025; Wu et al., 2025) and string-level correctness (Feng et al., 2025).

[9]While our results focus on MDMs that can resample generated symbols, they also apply to non-resampling MDMs—Thm. C.2 in §C.2 shows that the latter can simulate the former if given additional output space.

work focuses on reformulating the diffusion process to allow for resampling and refining the resulting training objectives, our idealized MDMs can be seen as a complementary approach that foregoes the complications of training and focuses on the expressivity of the generation process itself—it analyzes what is theoretically possible in a very targeted way when resampling is allowed.

We denote the class our idealized MDMs by MDM. For a more detailed discussion, including the theoretical connection of our idealization to existing MDM variants, see §C.

3 THEORETICAL RESULTS

This section describes two complementary characterizations of MDM expressivity: One based on their connection to PLTs and the other based on their ability to perform sequential CoT reasoning.[10]

Notation. Let $T = T(N)$ and $P = P(N)$ be functions of N. In the following, $\texttt{CoT}[T]$ refers to languages recognized by families $\{\mathcal{T}_N\}_{N \in \mathbb{N}}$ of CoT transformers with at most T steps. For $\texttt{C} \in \{\texttt{pCoT}, \texttt{MDM}, \texttt{PLT}\}$, $\texttt{C}[T, P]$ refers to languages recognized by families in $\texttt{C}$ with at most T generation, denoising, or looping steps, respectively, and P total output or padding symbols. $\widetilde{\mathcal{O}}(N)$ refers to big-$\mathcal{O}$ notation that ignores logarithmic factors, and $\texttt{poly}(N)$ to polynomial functions in N.

3.1 MDMs ARE EQUIVALENT TO PADDED LOOPED TRANSFORMERS

Intuitively, PLTs closely resemble MDMs: Both iteratively refine information in parallel—MDMs by unmasking and predicting discrete symbols, and PLTs by updating the residual stream.[11] Thm. 3.1 formalizes this, assuming PLTs are supplied with external sampling noise (cf. §2, §B.6) like MDMs.

Theorem 3.1 (PLTs and MDMs)**.**

$$\texttt{MDM}[T, P] \subseteq \texttt{PLT}[T, P] \quad (2a) \qquad and \qquad \texttt{PLT}[T, P] \subseteq \texttt{MDM}[T, (N + P)D]. \quad (2b)$$

The simulation of a PLT by an MDM incurs a factor D increase in the required padding (cf. Eq. (2b)), where D is the model width of the PLT. In our setting where $D = \mathcal{O}(\log N)$, this implies that the classes of finite-precision MDMs and PLTs coincide up to a logarithmic factor in the padding length:

Corollary 3.1. *For any $K \geqslant 1$,*

$$\texttt{MDM}[T, \widetilde{\mathcal{O}}(N^K)] = \texttt{PLT}[T, \mathcal{O}(N^K)]. \quad (3)$$

The close connection between MDMs and PLTs allows us to leverage existing results about PLT expressivity to understand MDMs. Saunshi et al. (2025, Thm. 5.1), for example, show that log-depth unpadded transformers can recognize regular languages. Combined with Cor. 3.1, this implies:

Corollary 3.2. *Regular languages are in* $\texttt{MDM}[\log N, N \log N]$*.*

In fact, we obtain a tighter bound with a more specialized construction.[12]

Theorem 3.2. *Regular languages are in* $\texttt{MDM}[\log N, N]$*.*

London & Kanade (2025) show that polynomially padded finite-precision PLTs with constantly many steps are equivalent to L-uniform $\texttt{AC}^0$, the class of $\texttt{AC}^0$ circuits that can be constructed by a logspace Turing machine (cf. §B.2). We leverage this result together with Thm. 3.1 to characterize the expressivity of MDMs with constantly many denoising steps.

Corollary 3.3 (MDMs with constantly many denoising steps)**.**

$$\texttt{MDM}[\mathcal{O}(1), \texttt{poly}(N)] = \texttt{PLT}[\mathcal{O}(1), \texttt{poly}(N)] = \text{L-uniform } \texttt{AC}^0. \quad (4)$$

Allowing for a *constant* number of decoding steps therefore does not increase the expressivity of MDMs beyond the limited class $\texttt{AC}^0$. This further corroborates the empirical observation that the

---

[10]The proofs of all statements in this section are deferred to §E.

[11]PLTs thus also resemble *latent* diffusion LMs that diffuse in the representation space. We do not explore this connection here due to the superior performance and popularity of MDMs (Zhang et al., 2025) in practice.

[12]This is possible because the PLT of Saunshi et al. (2025) only stores discrete values in its residual stream.

number of denoising steps must scale with the input complexity and complements existing results on MDM expressivity as a function of the number of denoising steps (Li & Cai, 2025; Feng et al., 2025).

We can, however, increase expressivity with more denoising steps. Svete et al. (2025) show that finite-precision PLTs with $\mathcal{O}(\log^d N)$ steps and polynomial padding are equivalent to L-uniform $\texttt{AC}^{\texttt{d}}$, similar to the case of log-precision PLTs and L-uniform $\texttt{TC}^{\texttt{d}}$ (Merrill & Sabharwal, 2025a). Thus:

Corollary 3.4 (MDMs with polylogarithmically many denoising steps)**.**

$$\texttt{MDM}[\mathcal{O}(\log^d N), \texttt{poly}(N)] = \texttt{PLT}[\mathcal{O}(\log^d N), \texttt{poly}(N)] = \text{L-uniform } \texttt{AC}^{\texttt{d}} \tag{5}$$

In particular, since $\texttt{NC}^{\texttt{d}} \subseteq \texttt{AC}^{\texttt{d}}$ for $d \in \mathbb{N}$ (where $\texttt{NC}^{\texttt{d}}$ denotes $\texttt{AC}^{\texttt{d}}$ circuits with bounded fan-in) (Vollmer, 1999), we get that with polylogarithmic looping and polynomial padding, MDMs converge to $\texttt{NC}$, the class of all parallelizable problems. Cor. 3.4 also implies that regular languages are in $\texttt{MDM}[\log N, \texttt{poly}(N)]$; Thm. 3.2 provides a more efficient construction with a linear output space. Moreover, Cor. 3.4 implies that L-uniform $\texttt{NC}^1 \subseteq \texttt{MDM}[\log N, \texttt{poly}(N)]$.

3.2 MDMs AND CoT CAN (INEFFICIENTLY) SIMULATE EACH OTHER

While the close connection between MDMs and PLTs provides a useful lens to analyze MDMs in terms of known complexity classes, the lack of practical PLT implementations makes it difficult to draw intuitive conclusions. We therefore complement §3.1 by connecting MDMs and the more popular CoT paradigm, and consider how MDM can behave "autoregressively" like CoT. The intuition is simple: An MDM can simulate CoT by unmasking one symbol at a time, effectively mimicking the sequential generation. More precisely, we connect MDMs to pCoT since the latter's ability to generate multiple symbols at once naturally maps to MDMs' parallel generation. In particular:

Theorem 3.3 (MDMs can simulate pCoT transformers)**.**

$$\texttt{pCoT}[T, P] \subseteq \texttt{MDM}[T, P + (N + P)^2] \tag{6}$$

The simulation incurs a quadratic blow-up in the padding length. This is not due to an inherent feature of the diffusion process but rather the challenge of simulating masked attention with unmasked one (cf. Lem. D.14).[13] In particular, if the MDM transformer is causally masked, the blow-up disappears. Moreover, if the unmasked transformer can simulate masking more efficiently (with, for example, more expressive scoring functions), the blow-up can be alleviated.[14] While Lem. D.14 could possibly be improved, it is interesting to note that the seemingly more general unmasked nature of MDMs might negatively impact their ability to align with human-oriented sequential processing captured by causal masking, which could provide a useful inductive bias for the masked models.[15]

The other direction of Thm. 3.3 shows that pCoT transformers can simulate MDMs.

Theorem 3.4 (pCoT transformers can simulate MDMs)**.**

$$\texttt{MDM}[T, P] \subseteq \texttt{pCoT}[T, LT(P + N)], \tag{7}$$

where L is the number of layers in the transformer implementing the MDM.

Again, the factor L comes from the need to simulate unmasked attention in MDMs with causally masked transformers implementing pCoT (cf. Lem. D.13). However, here, the blow-up is only linear. The additional factor of T comes from the pCoT having to write out every padding token after each denoising step, as the MDM can unmask tokens in an arbitrary order.

The results above can be summarized by the following sequence of inclusions.

---

[13]To the best of our knowledge, Lem. D.14 is the first result showing how to simulate masked attention with unmasked attention and might be of interest in its own right.

[14]This is, for example, used by Saunshi et al. (2025, Thm 5.4) to show that unmasked PLTs can simulate CoT with no blow-up in the padding using a *masking function*—an additional step in the computation of attention scores that allows for zeroing out of irrelevant keys—and with linearly-increasing width.

[15]Interestingly, some existing work finds that MDMs that decode based on the most confident symbols tend to decode autoregressively (Gong et al., 2025). In this sense, the unmasked nature of MDMs could be seen as a hurdle that the model has to overcome to eventually rely on more autoregressive generation. This further motivates the development of hybrid models that combine autoregressive generation of entire blocks with non-autoregressive infilling within the blocks (Nie et al., 2025; Arriola et al., 2025; Song et al., 2025, *inter alia*).

Corollary 3.5. *We have the following set of inclusions:*

$$\texttt{CoT}[T] = \texttt{pCoT}[T, T] \tag{8a, Prop. B.1}$$
$$\subseteq \texttt{MDM}[T, T + (N + T)^2] \tag{8b, Thm. 3.3}$$
$$\subseteq \texttt{pCoT}[T, LT(N + T + (N + T)^2)] \tag{8c, Thm. 3.4}$$
$$\subseteq \texttt{CoT}[LT(N + T + (N + T)^2)] \tag{8d, Prop. B.1}$$
$$\subseteq \texttt{CoT}[LT(N + T + 1)^2]. \tag{8e}$$

In particular, when $T \geqslant N$, we have $\texttt{CoT}[T] \subseteq \texttt{MDM}[T, \mathcal{O}(T^2)] \subseteq \texttt{pCoT}[T, \mathcal{O}(T^3)] \subseteq \texttt{CoT}[\mathcal{O}(T^3)]$.

Cor. 3.5 lower- and upper-bounds MDM expressivity based on the expressivity of CoT transformers. For example, MDM with polynomially many denoising steps remain within the class P, the problems solvable in polynomial time by a non-random-access multitape Turing machine. This follows from the equivalence of CoT transformers with polynomially many steps to P (Li et al., 2024).

Corollary 3.6 (MDMs with polynomially many denoising steps). *For any $K \geqslant 1$, we have that*

$$\texttt{MDM}[T, N^K] \subseteq \texttt{CoT}[\mathcal{O}(TN^K)], \tag{9}$$

meaning that MDMs with polynomially many denoising steps remain in P:

$$\texttt{MDM}[\texttt{poly}(N), \texttt{poly}(N)] \subseteq \texttt{CoT}[\texttt{poly}(N)] \subseteq \mathsf{P}. \tag{10}$$

3.2.1 A SEPARATION BETWEEN MDMS AND COT TRANSFORMERS

Merrill & Sabharwal (2024); Li et al. (2024) show that CoT transformers with logarithmically many decoding steps remain in $\texttt{TC}^0$. Combining this with Thm. 3.2, the widely accepted assumption that $\texttt{TC}^0 \neq \texttt{NC}^1$, and known $\texttt{NC}^1$-completeness of specific regular languages, we obtain the following separation in expressivity under a small (logarithmic) number of decoding steps. We term the inability of CoT to leverage parallelism the **sequentiality bottleneck** of CoT.

Corollary 3.7 (A strict separation in efficient reasoning abilities of MDMs and CoT transformers).

$$\texttt{CoT}[\log N] \subsetneq \texttt{MDM}[\log N, N]. \tag{11}$$

Concretely, $\texttt{MDM}[\log N, N] \setminus \texttt{CoT}[\log N]$, for example, contains all $\texttt{NC}^1$-complete regular languages.

4 DISCUSSION

Strengths and weaknesses of MDMs. §3 provides insights into the suitability of using MDMs for different classes of problems. On the one hand, Cor. 3.7 reveals the sequentiality bottleneck of CoT and an expressivity gap between CoT transformers and MDMs with logarithmically many model evaluations: While CoT transformers remain in $\texttt{TC}^0$, MDMs can solve $\texttt{NC}^1$-complete problems. This formalizes the intuition that MDMs are more suitable for highly-parallelizable problems and has implications for the practical applications of these two paradigms with a limited number of model evaluations. For example, the common *state-tracking* benchmark used to evaluate the reasoning abilities (Liu et al., 2023; Merrill et al., 2024) can be solved with MDMs with logarithmically many steps, while CoT transformers require linearly many steps. On the other hand, the equivalence of MDMs with polylogarithmically many denoising steps to the class NC (cf. Cor. 3.4) reveals problems where efficiency gains from parallelism are limited. For example, assuming the widely-believed hypothesis that $\texttt{NC} \neq \mathsf{P}$, none of the following (P-complete) problems benefit from MDM parallelism:

- **Circuit value problem**: Given a circuit, its inputs, and a gate, calculate the gate's value.
- **Linear programming**: Maximize a linear function subject to linear inequality constraints.
- **Context free grammar (CFG) membership**: Given a CFG $\mathcal{G}$ and a string w, is $w \in \mathcal{L}(\mathcal{G})$?
- **Horn-satisfiability** (P version of SAT): Is there a satisfying assignment to a set of Horn clauses?

In other words, these problems, in general, require a "CoT-style" step-by-step sequential solution. Due to the overhead introduced by unmasked attention of MDMs (such as the inability to store KV-cache), such problems are more efficiently solved by standard autoregressive CoT transformers.

Equivalence to padded looped transformers. Cor. 3.1 reveals a tight connection between MDMs and PLTs. While this suggests these two frameworks are largely interchangeable, important distinctions exist. On the one hand, unlike MDMs, standard PLTs perform sequential computations *deterministically*. This makes MDMs more suitable for ambiguous generation tasks, where decisions early in the generation make subsequent decisions easier. PLTs would, in that case, have to keep track of all possible generations in the residual stream until the final—decoding—step. Moreover, MDMs are easier to train and steer—since their intermediate computation steps are based on partially masked inputs, the model explicitly learns to solve complex tasks from random sub-tasks, which benefits their reasoning abilities (Kim et al., 2025). PLTs, in contrast, only receive training signal from the final decision and have to construct the sub-steps of the computation on their own. This could lead to suboptimal utilization of the sequential computation or even to failure to use it at all. Similarly, the human-understandable sub-tasks that MDMs have to solve make their training more interpretable and easier to control. On the other hand, the more information-rich intermediate states of PLTs make them more efficient at storing and processing information, and the lack of sampling steps makes them more efficient at inference time.

Generalizations. By focusing on transformer-based MDMs, we can draw from the rich theory developed on the expressivity of transformers. However, MDMs do not have to be implemented by a transformer—they could, for example, be implemented by a state-space model. Nevertheless, the parallelizable nature of MDMs suggests that any reasonable real-world implementation will include parallelizable components—for example, a model implementable by a constant-depth circuit, such as a $\mathtt{TC}^0$ circuit.[16] The close connection between transformers (with logarithmically-growing precision and padding) and the class $\mathtt{TC}^0$ (Merrill & Sabharwal, 2023; Li et al., 2024) suggests that the results of §3 will largely carry over to such implementations. We conjecture that, regardless of whether MDMs are implemented by finite-precision transformers ($\mathtt{AC}^0$ circuits) or a more expressive $\mathtt{TC}^0$ circuit, $\mathtt{MDM}[\log^d N, \mathtt{poly}(N)]$ would remain in $\mathtt{AC}^d$ (cf. Cor. 3.3) or a similar class like $\mathtt{TC}^d$. Moreover, we state the sequentiality bottleneck only for $\log N$ decoding steps, since the expressivity of $\mathtt{CoT}[\log N]$ is known to be limited. However, we believe that a similar separation exists for polylogarithmically many decoding steps: While the expressivity of $\mathtt{CoT}[\log^d N]$ has not been formalized yet, it likely does not capture all of $\mathtt{AC}^d$, unlike $\mathtt{MDM}[\log^d N, \mathtt{poly}(N)]$ (cf. Cor. 3.3).

Discrepancies. We strive to compare different paradigms fairly by analyzing a specific implementation of MDMs—one based on a specific idealization of transformers. Some inherent differences between the frameworks, however, remain. One is the dichotomy between using causal masking for CoT and unmasked transformers for MDMs and PLTs. We focus on unmasked models to analyze the more natural and popular implementations rather than artificially constraining MDMs and PLTs to causal masking. Another impactful decision is the nature of positional encodings (PEs). While we assume relatively simple PEs standard in theoretical literature (in particular, logspace-computable, cf. §B.4), we allow them to come from an outside source not part of the transformer—they may thus carry information not computable by the model; see §D.1 for additional discussion.

5 CONCLUSION

We describe the expressivity of masked diffusion LMs (MDMs) by connecting them to padded looped transformers (PLTs) and chain-of-thought-augmented (CoT) transformers. This reveals a close connection between PLTs and MDMs, which leads to the equivalence of MDM with polylogarithmically many denoising steps to the class $\mathtt{NC}$ of parallelizable problems, with concrete implications around what problems can benefit from the parallelism afforded by MDMs. We also show that MDMs can (somewhat inefficiently) simulate CoT transformers. We describe the sequentiality bottleneck and the strict expressivity gap between MDMs and CoT transformers with logarithmically many model evaluations. This shows MDMs to be more suitable for highly-parallelizable problems, while CoT transformers are more suitable for inherently sequential ones. Overall, our results provide insights into the strengths and weaknesses of MDMs and their suitability for different classes of problems.

---

[16]A similar modeling assumption is made by Liu (2025); Liu et al. (2025b) for latent diffusion LMs.

ETHICS STATEMENT

This work is theoretical and aims to describe the capabilities of masked diffusion models to better understand their strengths and limitations. We do not foresee any direct negative societal impacts.

REPRODUCIBILITY STATEMENT

All our results are theoretical and thus reproducible from the provided proofs in §§ D and E.

THE USE OF LARGE LANGUAGE MODELS

We used AI-based tools (Gemini and GitHub Copilot) for brainstorming and writing assistance. We used the tools in compliance with the ICLR 2026 policies.

REFERENCES

Afra Amini, Ryan Cotterell, John Hewitt, Luca Malagutti, Clara Meister, and Tiago Pimentel. Generating text from language models. In *ACL*, 2023. URL `https://aclanthology.org/2023.acl-tutorials.4/`.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating between autoregressive and diffusion language models. *arXiv*, 2025. URL `https://arxiv.org/abs/2503.09573`.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv*, 2025. URL `https://arxiv.org/abs/2507.10524`.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv*, 2025. URL `https://arxiv.org/abs/2505.24857`.

Nina L. Corvelo Benz, Stratis Tsirtsis, Eleni Straitouri, Ivi Chatzi, Ander Artola Velasco, Suhas Thejaswi, and Manuel Gomez Rodriguez. Evaluation of large language models via coupled token generation. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL `https://openreview.net/forum?id=FB8FMU99BC`.

Nadav Borenstein, Anej Svete, Robin Chan, Josef Valvoda, Franz Nowak, Isabelle Augenstein, Eleanor Chodroff, and Ryan Cotterell. What languages are easy to language-model? a perspective from learning probabilistic regular languages. In *ACL*, 2024. URL `https://aclanthology.org/2024.acl-long.807/`.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. Training neural networks as recognizers of formal languages. In *ICLR*, 2025. URL `https://openreview.net/forum?id=aWLQTbfFgV`.

Robin S. M. Chan, Reda Boumasmoud, Anej Svete, Yuxin Ren, Qipeng Guo, Zhijing Jin, Shauli Ravfogel, Mrinmaya Sachan, Bernhard Schölkopf, Mennatallah El-Assady, and Ryan Cotterell. On affine homotopy between language encoders. In *NeurIPS*, 2025.

Ivi Chatzi, Nina L. Corvelo Benz, Eleni Straitouri, Stratis Tsirtsis, and Manuel Gomez Rodriguez. Counterfactual token generation in large language models. In *Causality and Large Models Workshop @ NeurIPS 2024*, 2024. URL `https://openreview.net/forum?id=S4pwjKmLGR`.

Hongrui Chen and Lexing Ying. Convergence analysis of discrete diffusion model: Exact implementation through uniformization. *arXiv*, 2024. URL `https://arxiv.org/abs/2402.08095`.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In *ACL*, 2022. URL `https://aclanthology.org/2022.acl-long.527/`.

Ryan Cotterell, Anej Svete, Clara Meister, Tianyu Liu, and Li Du. Formal aspects of language modeling. *arXiv*, 2024. URL `https://arxiv.org/abs/2311.04329`.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv*, 2019. URL `https://arxiv.org/abs/1807.03819`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. URL `https://aclanthology.org/N19-1423/`.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *NeurIPS*, 2023. URL `https://openreview.net/forum?id=qHrADgAdYu`.

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model. *arXiv*, 2025. URL `https://arxiv.org/abs/2502.09622`.

Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, Dec 1984. ISSN 1433-0490. doi: 10.1007/BF01744431. URL `https://doi.org/10.1007/BF01744431`.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *EMNLP-IJCNLP*, 2019. URL `https://aclanthology.org/D19-1633/`.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. DiffuCoder: Understanding and improving masked diffusion models for code generation. *arXiv*, 2025. URL `https://arxiv.org/abs/2506.20639`.

Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. DiffusionBERT: Improving generative masked language models with diffusion models. In *ACL*, 2023. URL `https://aclanthology.org/2023.acl-long.248/`.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. *arXiv*, 2025. URL `https://arxiv.org/abs/2503.14615`.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv*, 2025. URL `https://arxiv.org/abs/2502.06768`.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring faithfulness in chain-of-thought reasoning. *arXiv*, 2023. URL `https://arxiv.org/abs/2307.13702`.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

Gen Li and Changxiao Cai. A convergence theory for diffusion language models: An information-theoretic perspective. *arXiv*, 2025. URL `https://arxiv.org/abs/2505.21400`.

Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of transformer language models. *arXiv*, 2025. URL `https://arxiv.org/abs/2505.23623`.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *ICLR*, 2024. URL `https://openreview.net/forum?id=3EWTEy9MTM`.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *ICLR*, 2023. URL `https://openreview.net/forum?id=De4FYqjFueZ`.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising. *arXiv*, 2025a. URL `https://arxiv.org/abs/2410.06264`.

Yuxi Liu. Perfect diffusion is TC^0—Bad diffusion is Turing-complete. *arXiv*, 2025. URL `https://arxiv.org/abs/2507.12469`.

Yuxi Liu, Konpat Preechakul, Kananart Kuwaranancharoen, and Yutong Bai. The serial scaling hypothesis. *arXiv*, 2025b. URL `https://arxiv.org/abs/2507.12549`.

Charles London and Varun Kanade. Pause tokens strictly increase the expressivity of constant-depth transformers. *arXiv*, 2025. URL `https://arxiv.org/abs/2505.21024`.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *ICML*, 2024. URL `https://arxiv.org/abs/2310.16834`.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *TACL*, 11:531–545, 2023. URL `https://aclanthology.org/2023.tacl-1.31/`.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *ICLR*, 2024. URL `https://openreview.net/forum?id=NjNGlPh8Wh`.

William Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. *arXiv*, 2025a. URL `https://arxiv.org/abs/2505.18948`.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. *arXiv*, 2025b. URL `https://arxiv.org/abs/2503.03961`.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *ICML*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv*, 2025. URL `https://arxiv.org/abs/2502.09992`.

Franz Nowak, Anej Svete, Alexandra Butoi, and Ryan Cotterell. On the representational capacity of neural language models with chain-of-thought reasoning. In *ACL*, 2024. URL `https://aclanthology.org/2024.acl-long.676/`.

Michael Oberst and David Sontag. Counterfactual off-policy evaluation with Gumbel-max structural causal models. In *ICML*, 2019. URL `https://proceedings.mlr.press/v97/oberst19a.html`.

Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling. *arXiv*, 2025. URL `https://arxiv.org/abs/2502.03540`.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *JMLR*, 22(75):1–35, 2021. URL `http://jmlr.org/papers/v22/20-302.html`.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019. URL `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

Shauli Ravfogel, Anej Svete, Vésteinn Snæbjarnarson, and Ryan Cotterell. Gumbel counterfactual generation from language models. In *ICLR*, 2025. URL `https://openreview.net/forum?id=TUC0ZT2zIQ`.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv*, 2025. URL `https://arxiv.org/abs/2502.17416`.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed Diffusion: A large-scale diffusion language model with high-speed inference. *arXiv*, 2025. URL `https://arxiv.org/abs/2508.02193`.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *TACL*, 12:543–561, 2024. URL `https://aclanthology.org/2024.tacl-1.30/`.

Weigao Sun, Jiaxi Hu, Yucheng Zhou, Jusen Du, Disen Lan, Kexin Wang, Tong Zhu, Xiaoye Qu, Yu Zhang, Xiaoyu Mo, Daizong Liu, Yuxuan Liang, Wenliang Chen, Guoqi Li, and Yu Cheng. Speed always wins: A survey on efficient architectures for large language models. *arXiv*, August 2025. URL `https://arxiv.org/abs/2508.09834`.

Anej Svete and Ryan Cotterell. Transformers can represent n-gram language models. In *NAACL*, 2024. URL `https://aclanthology.org/2024.naacl-long.381/`.

Anej Svete, Nadav Borenstein, Mike Zhou, Isabelle Augenstein, and Ryan Cotterell. Can transformers learn n-gram language models? In *EMNLP*, 2024. URL `https://aclanthology.org/2024.emnlp-main.550/`.

Anej Svete, Will Merrill, and Ashish Sabharwal. The exact expressive power of fixed-precision looped padded transformers, 2025. URL `https://anejsvete.github.io/files/fp-lpt.pdf`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, Heidelberg, 1999. ISBN 978-3-540-64310-4. doi: 10.1007/978-3-662-03927-4.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion. In *ICML*, 2025. URL `https://openreview.net/forum?id=rvZv7sDPV9`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv*, 2025. URL `https://arxiv.org/abs/2505.22618`.

Kevin Xu and Issei Sato. To CoT or to loop? A formal comparison between chain-of-thought and looped transformers. *arXiv*, 2025. URL `https://arxiv.org/abs/2505.19245`.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *NeurIPS*, 2024. URL `https://openreview.net/forum?id=FBMsBdH0yz`.

Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S. Yu, and Aiwei Liu. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv*, 2025. URL `https://arxiv.org/abs/2508.08712`.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. A reparameterized discrete diffusion model for text generation. *arXiv*, 2024. URL `https://arxiv.org/abs/2302.05737`.

A   RELATED WORK ON THE EXPERSSIVITY OF MDMs

Existing theoretical work on MDMs focuses on the limitations of the factorized backward process and its convergence properties, which can be linked to formal language generation and recognition.

Feng et al. (2025) analyze the implications of the factorized backward process on the expressivity of MDMs. They find that MDMs can approximate n-gram LM distributions arbitrarily well with constantly many sampling steps and linearly many sampling steps suffice to approximately generate from any regular language. They also provide a linear lower bound on the number of sampling steps required to capture the support of general regular languages—a consequence of the assumed factorization, which is incompatible with the sequential generation of regular languages. Crucially, this differs from our *recognition* setting in which the string to be processed is given and only its membership decision has to be generated.

This line of work is tightened by Li & Cai (2025), who build on work by Chen & Ying (2024) and analyze the approximation error (measured by the KL divergence) of the distribution generated by MDMs with respect to the number of sampling steps and the mutual information (dependence) between the symbols in different positions. Intuitively, the larger the mutual information is, the larger the number of sampling steps has to be to capture the same distribution (equivalently, the fewer symbols per step can be generated). They show that the KL divergence between the generated and the ground-truth distributions decays linearly with the number of sampling steps (with scaling that depends on the mutual information between the symbols in the strings), and show this decay to be optimal in general. This generalizes the results on n-gram distributions and regular languages by Feng et al. (2025).

Liu (2025); Liu et al. (2025b) take a different perspective and analyze the expressivity of latent diffusion LMs. They show that, under an analogous assumption to our Assumptions 2.1 and 2.2, latent diffusion LMs converge to the data distribution in constantly many steps, which means that the computational depth of such models is limited. They show that breaking this assumption makes latent diffusion LMs Turing complete, analogous to our results in §3.2.

While these results provide useful insights into the limitations and affordances of MDMs, their asymptotic and approximate nature makes it difficult to draw concrete conclusions about the (reasoning) capabilities of MDMs in the sense of the work on transformers' expressivity. Our work complements these results by providing a more fine-grained analysis of MDM capabilities based on their connection to PLTs and CoT transformers, which allows us to leverage the existing theory developed on transformer expressivity and apply it directly to MDMs.

B   PRELIMINARIES

B.1   NOTATION

Let Σ be an alphabet. A **language** $\mathcal{L}$ is a subset of Σ^*. A **language recognizer** is a function $R\colon \Sigma^* \to \{0, 1\}$, where 0 and 1 are designated reject and accept symbols. R's language is $\mathcal{L}(R) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid R(w) = 1\}$. Two recognizers R_1 and R_2 are **equivalent** if and only if $\mathcal{L}(R_1) = \mathcal{L}(R_2)$.

We denote the concatenation of two strings $w_1, w_2 \in \Sigma^*$ as $w_1 \circ w_2$ or simply $w_1 w_2$. We define the **intereleaving** of the vectors $x, y \in \mathbb{R}^D$ as $x \frown y \in \mathbb{R}^{2D}$ where

$$x \frown y_d \stackrel{\text{def}}{=} \begin{cases} x_{(d+1)/2} & \textbf{if } d \text{ is odd,} \\ y_{d/2} & \textbf{otherwise } d. \end{cases} \tag{12}$$

We use $[\![w]\!] \in \{0, 1\}^{|\Sigma|}$ to denote the one-hot encoding of symbol $w \in \Sigma$. We use $\mathsf{B}_{\mathsf{p}}(n)$ to denote the binary encoding of natural number n using p binary bits and $\mathsf{B}^{\pm}{}_{\mathsf{p}}(n)$ to denote the signed binary encoding $2\mathsf{B}_{\mathsf{p}}(n) - \mathbf{1}_{\mathsf{p}}$, where $\mathbf{1}_{\mathsf{p}}$ is the D-dimensional vector of all ones. We will leave out p when it is clear from the context.

For $D \in \mathbb{N}$, we define $\mathrm{softmax}\colon \mathbb{R}^D \to \mathbb{R}^D$ as $\mathrm{softmax}(x)_d = \exp(x_d)/\sum_{d=1}^{D} \exp(x_d)$ for $x \in \mathbb{R}^D$ and $d \in [D]$. We also use the shorthand $[x]_+ \stackrel{\text{def}}{=} \max\{x, 0\}$. We denote with $\mathcal{P}(\mathcal{X})$ the set of all probability distributions over a set $\mathcal{X}$.

B.2 Circuit Complexity

Computational circuits are a model of parallel computation. They have been widely used in the study of the expressivity of neural networks. Circuits process binary input strings through a series of logical operations to produce binary outputs.[17] Formally, a **boolean circuit** is a directed acyclic graph where source nodes represent the N-bit input, and a single sink node represents the output. Non-source vertices are called **gates** and are labeled with logical operations (e.g., AND, OR, NOT). The **size** of a circuit is the number of gates, and its **depth** is the longest path from any input to the output.

A circuit computes a function $C\colon \{0,1\}^N \to \{0, 1\}$ for some $N \in \mathbb{N}$, where 0 and 1 are designated reject and accept symbols. The value $C(\boldsymbol{w})$ for input string $\boldsymbol{w} \in \{0,1\}^N$ is computed by evaluating the gates in topological order starting from the input bits. We say that the circuit C **accepts** a string $\boldsymbol{w}$ if $C(\boldsymbol{w}) = 1$.

Circuit families process input strings of variable length. A circuit family is a sequence of circuits $\mathcal{C} \overset{\text{def}}{=} \{C_N\}_{N \in \mathbb{N}}$ where C_N processes inputs of length N. A circuit family is said to recognize a language if for any given input string, the corresponding circuit outputs 1 if and only if the string is in the language.

A **circuit complexity class** is a set of circuit families that satisfy certain constraints on size, depth, and the types of gates used. This paper focuses on two common classes:

- AC^d: Circuits with NOT, AND, and OR gates that have unbounded fan-in and depth $\mathcal{O}(\log^d N)$.
- TC^d: The extension of AC^d that adds **threshold gates**, which output 1 if the sum of their inputs exceeds a given threshold. It is known that $\mathsf{AC}^0 \subsetneq \mathsf{TC}^0$ and $\mathsf{AC}^d \subseteq \mathsf{TC}^d$. For example, PARITY, the language of binary strings with an even number of 1s, is in TC^0 but not in AC^0 (Furst et al., 1984).
- NC^1: This class consists of circuits that can be computed in parallel with a logarithmic depth, a polynomial number of gates, and constant fan-in. It is known that $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

Without additional constraints, circuit families can recognize undecidable languages by having arbitrary, "hard-coded" solutions for each input length. To avoid this and ensure the model of computation is realistic, we can impose a **uniformity** condition. A circuit family is **uniform** if there exists a Turing machine that, given an input of 1^N, can generate a description of the circuit C_N. In particular, a circuit class is L-uniform if a Turing machine using $\mathcal{O}(\log N)$ space can construct its description from the input 1^N. This ensures the circuits for different input lengths are related by a systematic procedure.

B.3 Finite-precision fixed-point arithmetic

We assume that the operations performed by our computational models rely on finite-precision fixed-point arithmetic. This model is based on ones used by Li et al. (2024); Saunshi et al. (2025); London & Kanade (2025).

Definition B.1 (Fixed-Point Representation). *Let $\mathsf{p} \in \mathbb{N}$ be the number of bits devoted to each of the integer and fractional parts. We use $\mathbb{F}_{\mathsf{p}}$ to denote the set*

$$\mathbb{F}_{\mathsf{p}} \overset{\text{def}}{=} \{x_{\pm} \cdot a \cdot 2^{-\mathsf{p}} \mid x_{\pm} \in \{-1, 1\}, a \in \{0, 1, \ldots, 2^{2\mathsf{p}} - 1\}\} \tag{13}$$

We define $B_{\mathbb{F}} \overset{\text{def}}{=} \max \mathbb{F}_{\mathsf{p}} = 2^{\mathsf{p}} - 2^{-\mathsf{p}}$. All values exceeding $B_{\mathbb{F}}$ are considered out of range and are rounded to $B_{\mathbb{F}}$. Note, however, that $B_{\mathbb{F}}$ does *not* behave like infinity—it does not "consume" all subsequent operations. For example, $B_{\mathbb{F}} - x \neq B_{\mathbb{F}}$ for some non-negative $x \in \mathbb{F}_{\mathsf{p}}$ is a valid number.

To handle the results of arithmetic operations that may not be exactly representable in the fixed-point format, we define a standard for rounding.

Definition B.2 (Rounding). *For any $x \in \mathbb{R}$ and any closed subset $\mathbb{F}$ of $\mathbb{R}$ containing 0, we define rounding $\mathsf{round}(x, \mathbb{F})$ as the closest number to x in $\mathbb{F}$. In case of a tie, the value with the smaller absolute value is chosen.*

---

[17]By representing symbols from any alphabet with binary encodings, circuits (or circuit functions) can be used to process strings over any finite alphabet. We focus on binary strings for simplicity.

We denote the rounding operation as $[\cdot]_\mathsf{p} \stackrel{\text{def}}{=} \mathsf{round}(\cdot, \mathbb{F}_\mathsf{p})$. This operation is applied to vectors and matrices element-wise. All binary operations are defined by first performing the ideal mathematical operation and then rounding the result to the nearest representable value in $\mathbb{F}_\mathsf{p}$. Division by zero is considered an error condition resulting in an incorrect output. We also note that $B_\mathbb{F}/2 = 2^{\mathsf{p}-1} - 2^{-\mathsf{p}}$.

For operations involving more than two numbers, rounding is applied iteratively.

Definition B.3 (Summation with Iterative Rounding). *For* $\mathsf{p}, N \in \mathbb{N}$ *and* $\boldsymbol{x} \in \mathbb{R}^N$, *we define summation with iterative rounding to* p *fractional bits as the function* $\mathrm{SUM}_\mathsf{p}\colon \bigcup_{N \in \mathbb{N}}(\mathbb{F}_\mathsf{p})^N \to \mathbb{F}_\mathsf{p}$, *where for any* $N \in \mathbb{N}^+$ *and* $\boldsymbol{x} \in (\mathbb{F}_\mathsf{p})^N$:

$$\mathrm{SUM}_\mathsf{p}(\boldsymbol{x}) \stackrel{\text{def}}{=} \left[\dots \left[[x_1 + x_2]_\mathsf{p} + x_3 \right]_\mathsf{p} + \dots + x_N \right]_\mathsf{p} \tag{14}$$

This iterative rounding process is not associative and the order of operations can affect the final result. Based on this, we can also define more complex operations such as the **fixed-point inner product** $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_\mathsf{p} \stackrel{\text{def}}{=} \mathrm{SUM}_\mathsf{p}(\boldsymbol{x} \odot \boldsymbol{y})$, where $\odot$ denotes the element-wise product of two vectors, and **fixed-point matrix product** for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, where $(\boldsymbol{A} \times_\mathsf{p} \boldsymbol{B})_{i,j} \stackrel{\text{def}}{=} \langle (\boldsymbol{A}_{i,:})^\top, \boldsymbol{B}_{:,j} \rangle_\mathsf{p}$.

B.4 TRANSFORMERS

We consider both **unmasked** (Devlin et al., 2019) and **causally masked** transformers (Radford et al., 2019). Concretely, we work with fixed-point transformers whose underlying arithmetic operations are replaced with their fixed-point counterparts. A transformer $\mathcal{T}$ consists of four parts:

(1) a **symbol embedding** $\boldsymbol{e}\colon \Sigma \to \mathbb{F}^D$ of the form $\boldsymbol{e}(w) \stackrel{\text{def}}{=} \boldsymbol{V}[\![w]\!]$ for $w \in \Sigma$, where $\boldsymbol{V} \in \mathbb{F}^{D \times |\Sigma|}$ and $[\![w]\!] \in \mathbb{F}^{|\Sigma|}$ is the one-hot encoding of w,
(2) a **positional encoding** $\boldsymbol{p}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{F}^D$,
(3) L **layers** $\boldsymbol{\tau}^{(1)}, \dots, \boldsymbol{\tau}^{(L)}$, each of which consists of two sub-layers: A multi-head self-attention layer and a position-wise fully-connected feed-forward network f, and
(4) an **output layer** $\boldsymbol{o}$ of the form $\boldsymbol{o}(\boldsymbol{h}) \stackrel{\text{def}}{=} \mathrm{softmax}(\boldsymbol{E}\boldsymbol{h})$ for $\boldsymbol{h} \in \mathbb{F}^D$, where $\boldsymbol{E} \in \mathbb{F}^{|\overline{\Sigma}| \times D}$.

Each layer has its own parameters and is indexed by the layer name and the depth for attention and feedforward layers. We use D to denote the **width** of a transformer. A transformer with layers $\boldsymbol{\tau}^{(1)}, \dots, \boldsymbol{\tau}^{(L)}$ computes $\boldsymbol{h}_n^{(l)} \in \mathbb{F}^D$ for $l \in \{1, \dots, L\}$ and each position $n \in [N]$ in the input string $\boldsymbol{w} = w_1 \cdots w_N \in \Sigma^*$ as follows:

$$\boldsymbol{h}_n^{(0)} \stackrel{\text{def}}{=} \boldsymbol{e}(w_n) + \boldsymbol{p}(n, N) \in \mathbb{F}^D \text{ for } n \in [N] \tag{15a}$$

$$\boldsymbol{H}^{(l)} \stackrel{\text{def}}{=} \left(\boldsymbol{h}_1^{(l)\top} \quad \cdots \quad \boldsymbol{h}_N^{(l)\top} \right)^\top \in \mathbb{F}^{N \times D} \tag{15b}$$

$$\boldsymbol{Q}^{(l)} \stackrel{\text{def}}{=} \boldsymbol{H}^{(l)} \boldsymbol{W}_Q^{(l)}, \quad \boldsymbol{K}^{(l)} \stackrel{\text{def}}{=} \boldsymbol{H}^{(l)} \boldsymbol{W}_K^{(l)}, \quad \boldsymbol{V}^{(l)} \stackrel{\text{def}}{=} \boldsymbol{H}^{(l)} \boldsymbol{W}_V^{(l)} \quad \in \mathbb{F}^{N \times D} \tag{15c}$$

$$\boldsymbol{G}^{(l)} \stackrel{\text{def}}{=} \mathrm{softmax}(M(\boldsymbol{Q}^{(l)} \boldsymbol{K}^{(l)\top})) \boldsymbol{V}^{(l)} + \boldsymbol{H}^{(l)} \in \mathbb{F}^{N \times D} \tag{15d}$$

$$\boldsymbol{H}^{(l+1)} \stackrel{\text{def}}{=} \boldsymbol{G}^{(l)} + f(\boldsymbol{G}^{(l)}) \in \mathbb{F}^{N \times D} \tag{15e}$$

Here, $M\colon \mathbb{F}^{N \times N} \to (\mathbb{F} \cup \{-\infty\})^{N \times N}$ is the **masking function**.[18,19] We say that the lth layer $\boldsymbol{\tau}^{(l)}$ computes the function $\boldsymbol{\tau}^{(l)}\colon \mathbb{F}^{N \times D} \to \mathbb{F}^{N \times D}$, defined by the function $\boldsymbol{\tau}^{(l)}\colon \boldsymbol{H}^{(l-1)} \mapsto \boldsymbol{H}^{(l)}$ for $l \in \{1, \dots, L\}$. We also denote with $\mathcal{T}$ the function $\mathcal{T}\colon \Sigma^* \to \mathbb{F}^{N \times D}$, defined as $\mathcal{T}\colon \boldsymbol{w} \mapsto \boldsymbol{H}^{(L)}$.

We use the following definition of the multi-layer perceptron.

Definition B.4 (Multi-layer perceptron). *A **multi-layer perceptron (MLP)** is a function* $f\colon \mathbb{F}^D \to \mathbb{F}^{D'}$ *that can be expressed as a composition of affine transformations and the* ReLU *activation function:*

$$f(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{W}_2(\mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)) + \boldsymbol{b}_2), \tag{16}$$

where $\boldsymbol{W}_1 \in \mathbb{F}^{H \times D}$, $\boldsymbol{W}_2 \in \mathbb{F}^{D' \times H}$, $\boldsymbol{b}_1 \in \mathbb{F}^H$, $\boldsymbol{b}_2 \in \mathbb{F}^{D'}$ *for* $D, D', H \in \mathbb{N}$.

---

[18]Similar to Li et al. (2024); Saunshi et al. (2025), we define masking with a function rather than an additive matrix since subtracting $B_\mathbb{F}$ from an arbitrary number in $\mathbb{F}$ does not necessarily result in $-B_\mathbb{F}$.

[19]Examples of masking functions include the identity function (unmasked attention) and the identity function on the upper-triangular portion of the matrix that replaces the lower-triangular part with $-B_\mathbb{F}$ (causally masked attention).

B.4.1 SCALING TRANSFORMER SIZE WITH INPUT LENGTH

The exact expressivity of a transformer model depends on seemingly unimportant details (Jerad et al., 2025). One such example is the interplay between the scaling of the **numerical precision** p of the values stored in the representations $\boldsymbol{h}$ and the scaling of the **width** D. To be able to uniquely identify the N positions in the input string, the "volume" of the embedding space, i.e., the number of possible distinct representations $\boldsymbol{h}$, must be at least N. This implies that the product $\mathsf{p} \cdot D$ must *scale* at least logarithmically with N: $\mathsf{p}(N) \cdot D(N) = \Omega(\log N)$. Existing work focuses on two modeling choices:

(1) **log-precision** transformers where $\mathsf{p}(N) = \Theta(\log N)$ with either constant width $D(N) = \Theta(1)$ or polynomial width $D(N) = \Theta(\texttt{poly}(N))$ (Merrill & Sabharwal, 2023; 2024; Li et al., 2024; London & Kanade, 2025; Merrill & Sabharwal, 2025b;a); and

(2) **constant-precision logarithmic-width** transformers where $\mathsf{p} = \Theta(1)$ and $D(N) = \Theta(\log N)$ (Li et al., 2024; Saunshi et al., 2025; London & Kanade, 2025).

Despite having the same volume of the representation space, the two models differ in their expressivity. For example, constant-precision transformers are constrained to use the volume in a "distributed" manner across model dimensions without the ability to summarize the information into individual values or store pointers to arbitrary positions in them—both of these require precision growing with string length. Such summarization is required in certain steps of the transformer architecture—for example, when computing the attention scores. This limits the expressivity of constant-precision transformers compared to log-precision transformers: While log-precision transformers can compute certain TC^0 functions, constant-precision transformers with polynomial width fall within AC^0 (Li et al., 2024; London & Kanade, 2025). Similar separation results extend to popular variants of transformers: (a) Transformers with chain-of-thought reasoning (cf. §B.5) with logarithmic precision can simulate TC^0 circuits of size corresponding to the number of reasoning steps while constant-precision transformers with polynomial width are constrained to AC^0 circuits (Li et al., 2024; Merrill & Sabharwal, 2024). (b) Transformers with additional padding space (blank thinking tokens; cf. §B.6) with logarithmic precision can express exactly TC^0 functions while constant-precision transformers with polynomial width are equivalent to AC^0 circuits (Merrill & Sabharwal, 2025a; London & Kanade, 2025). However, the trend of coarse quantization while increasing the model size makes fixed-precision logarithmic-width transformers particularly appealing, which is why we focus on *finite precision* transformers with *logarithmic width*.

Analogously to circuit families, each string length N is processed by a separate transformer model. To process all of Σ^*, we therefore define a **transformer family** $\{\mathcal{T}_N\}$ as a sequence of transformers where each $\mathcal{T}_N$ processes strings of length N. Further, we again impose a uniformity condition on the family, which will relate the transformers for different input lengths.

Definition B.5 (Uniform transformer families; variant of London & Kanade, 2025, Def. 3.6). *Let X be a computational complexity class. A transformer family $\{\mathcal{T}_N\}$ is X-**uniform** if there exist Turing machines $\mathcal{M}_1$ and $\mathcal{M}_2$ whose resource usage is constrained by the complexity class X such that:*

(1) $\mathcal{M}_1$ takes input 1^N and outputs a description of $\mathcal{T}_N$, and
(2) $\mathcal{M}_2$ takes input $(1^N, \texttt{B}(n))$ and outputs $\boldsymbol{p}(n, N)$.

Def. B.5 allows for size-dependent transformers while keeping them closely related (as the same Turing machines must construct them for all N). It also facilitates natural connections with uniform circuit classes (cf. §B.2) (London & Kanade, 2025). All our results concern L-uniform transformer families, in which case, the Turing machines in Def. B.5 operate in logarithmic space.

B.4.2 TRANSFORMER LANGUAGE MODELS AND SYMBOL PREDICTORS

Transformers can implement (non-)autoregressive LMs and deterministic symbol predictors. A transformer LM computes the probability distribution over the $N + 1^{\text{st}}$ symbol given the length-N string $\boldsymbol{w}$ as

$$\overrightarrow{p}(\overline{w} \mid \boldsymbol{w}) \stackrel{\text{def}}{=} \boldsymbol{o}(\boldsymbol{h}^{(L)}_{|\boldsymbol{w}|})_{\overline{w}} \qquad \text{for } \overline{w} \in \overline{\Sigma}. \qquad (17)$$

To define **infilling** probabilities, let $\boldsymbol{w} \in \Sigma_{\mathtt{m}}{}^*$ be a possibly (partially) masked string of length N and let $n \in [N]$ such that $w_n = \mathtt{m}$. We then define the probability distribution over the symbol at the masked position n as

$$p^{\downarrow}(w \mid \boldsymbol{w}) \stackrel{\text{def}}{=} \boldsymbol{o}(\boldsymbol{h}^{(L)}_n)_w \qquad \text{for } w \in \Sigma. \qquad (18)$$

Note that the next-symbol probabilities are computed based on the contextual representation of the previous symbol while the infilling probabilities are defined based on the contextual representation at the masked position. That is, for autoregressive next-symbol prediction, the transformer uses the hidden state at the last (previous) position, whereas for infilling, it uses the hidden state at the position to be filled.

To define a deterministic transformer-based symbol predictor, we define a decoding step.

Definition B.6 (Decoding step). *For any $N \in \mathbb{N}$, let $\boldsymbol{H} \in \mathbb{R}^{N \times D}$. The **decoding step** $\mathrm{Dec} \colon \mathbb{R}^{N \times D} \to \overline{\Sigma}^N$ is defined as*

$$\mathrm{Dec}(\boldsymbol{H})_n = \underset{w \in \overline{\Sigma}}{\mathrm{argmax}} \; \boldsymbol{o}(\boldsymbol{H})_{n,:} \tag{19}$$

where the output function $\boldsymbol{o}$ is applied to $\boldsymbol{H}$ row-wise. This can be used either to deterministically infill the masked positions in the string or to deterministically predict the next symbol based on the final representation. For next-symbol prediction, we also define the shorthand $\mathrm{NS} \colon \Sigma^* \to \overline{\Sigma}$ as

$$\mathrm{NS}(\boldsymbol{w}) \stackrel{\mathrm{def}}{=} \mathrm{Dec}(\mathcal{T}(\boldsymbol{w}))_N. \tag{20}$$

B.4.3 Language encoders and model equivalence

We are interested in the expressivity of neural networks as language recognizers and LMs, which at a high level, describes their behavior on input strings. This behavior—either the prediction of language membership or the computation of string probabilities—is completely determined by the **contextual representations** of the input strings produced by the neural network. We abstract the computation of string representations using a **language encoder**—a length-preserving function $\mathrm{Enc} \colon \Sigma^* \to (\mathbb{R}^D)^*$ for some $D \in \mathbb{N}$ (Chan et al., 2025; Cotterell et al., 2024). We regard the output $\mathrm{Enc}(\boldsymbol{w})$ of a language encoder for a string $\boldsymbol{w}$ as a $|\boldsymbol{w}| \times D$ matrix, where each row corresponds to the contextual representation of the symbol at the corresponding position. In the context of transformers, the language encoder is the function $\mathrm{Enc}(\boldsymbol{w}) \stackrel{\mathrm{def}}{=} \boldsymbol{H}^{(L)}$ for $\boldsymbol{w} \in \Sigma^*$.

All aspects of the model's behavior that we might be interested in can be described in terms of the contextual representations—the (logits of the) next-symbol or infilling probabilities are determined by a linear transformation of the contextual representations, and the membership test is determined by a linear classifier based on the contextual representations of the final symbol in the string. Studying the expressivity of a model thus reduces to determining what types of contextual representations can be produced by the model. If we can show that two language encoders $\mathrm{Enc}_1, \mathrm{Enc}_2 \colon \Sigma^* \to (\mathbb{R}^D)^*$ compute the same contextual representations for all input strings, we say that Enc_1 and Enc_2 are **equivalent**. Moreover, we say that two sets of models are equivalent if each model in one set is equivalent to at least one model in the other set.

Model equivalence and variable-length outputs. Sometimes, we will compare the expressivity of models that produce outputs of different lengths, for example when comparing the expressivity of padded and non-padded models, or comparing CoT-augmented transformers with non-augmented ones. This makes direct comparison of contextual representations more difficult. Whenever this is the case, we will explicitly state how the outputs of different lengths are aligned. For example, when comparing a model that produces outputs of length N with a model that produces outputs of length KN for some constant $K \in \mathbb{N}$, we may assume that the first $(K-1)N$ positions of the longer output are used for intermediate computations and only the last N positions are used to produce the final output. Thus, we will only compare the last N positions of the longer output with the N-length output of the shorter model and base model equivalence on that.

B.4.4 Sampling from a (transformer) language model

The next-symbol and infilling probabilities from §B.4.2 can be used to sample from the LM by sampling from $\overrightarrow{p}(\cdot \mid \boldsymbol{w})$ or $p^{\downarrow}(\cdot \mid \boldsymbol{w})$, respectively.[20] There are many possible ways to implement the sampling. In this paper, we assume that it is performed using the Gumbel-max trick (Oberst & Sontag, 2019), which both provides a convenient and fast implementation as well as interpretable traces of the sampling procedure (Chatzi et al., 2024; Ravfogel et al., 2025; Benz et al., 2025).

---

[20]In practice, the raw probabilities are often post-processed with temperature scaling or sampling adaptors (Amini et al., 2023). The framework described here can easily be adapted to those settings.

Definition B.7 (Gumbel-max sampling). *Let $\boldsymbol{l} \stackrel{\text{def}}{=} \boldsymbol{E}\boldsymbol{h}$ for $\boldsymbol{h} \in \mathbb{R}^D$ and $\boldsymbol{E} \in \mathbb{R}^{|\overline{\Sigma}| \times D}$ be the logits of a probability distribution over $\overline{\Sigma}$. The **Gumbel-max sampling** from is defined as*

$$w = \operatorname*{argmax}_{w' \in \overline{\Sigma}}(p_{w'} + g_{w'}), \tag{21}$$

where $g_{w'}$ are i.i.d. samples from the Gumbel distribution with the cumulative distribution function $F(x) = \exp(-\exp(-x))$ for $x \in \mathbb{R}$.

It is well known that Eq. (21) results in samples from $\operatorname{softmax}(\boldsymbol{l})$. The Gumbel-max sampling can be used to either sample the next symbol w_{N+1} from the next-symbol distribution $\overrightarrow{p}(\cdot \mid \boldsymbol{w})$ or to sample the masked symbol w_n from the infilling distribution $p^{\downarrow}(\cdot \mid \boldsymbol{w})$.

We rely on the Gumbel-max sampling because it conveniently decouples sampling from the model's representations. In particular, provided that a model is able to implement the argmax operation and is able to receive the Gumbel noise as an input, it can sample from the model's distribution without any additional operations. This will provide a convenient way to precisely link different modeling frameworks in a unified manner.

B.5 CHAIN-OF-THOUGHT REASONING

At a high level, chain-of-thought (CoT) transformers process a string by generating intermediate reasoning steps. These steps can be seen as a generated string itself or as an intermediate thinking process that is used to generate the final output.

Definition B.8 (CoT Generation). *A causally masked transformer implementing the next-symbol distribution $\overrightarrow{p} \colon \Sigma^* \to \mathcal{P}(\overline{\Sigma})$ **generates** strings $y_1 \cdots y_T$, given $\boldsymbol{w} \in \Sigma^*$, as*

$$y_t \sim \overrightarrow{p}(\cdot \mid \boldsymbol{w} \circ y_1 \cdots y_{t-1}) \tag{22}$$

where $y_T = \text{EOS}$ and $y_t \neq \text{EOS}$ for $t < T$.

While some existing work analyzes the distributions induced by CoT transformers (Nowak et al., 2024; Xu & Sato, 2025), much of the existing literature (Pérez et al., 2021; Feng et al., 2023; Merrill & Sabharwal, 2024; Li et al., 2024; Saunshi et al., 2025) focuses on modeling string **acceptance** by CoT transformers by determinizing $\overrightarrow{p}$.

Definition B.9 (CoT Acceptance). *A causally masked transformer implementing the next-symbol predictor $\text{NS} \colon \Sigma^* \to \Sigma_{01}$ generates the sequence of reasoning steps*

$$y_t \stackrel{\text{def}}{=} \text{NS}(\boldsymbol{w} \circ y_1 \cdots y_{t-1}) \tag{23}$$

*for $\boldsymbol{w} \in \Sigma^*$, $t \in [T]$, and a pre-determined $T \in \mathbb{N}$. NS **accepts** a string $\boldsymbol{w}$ in T steps if $y_T = 1$ and rejects it if $y_T = 0$.*

To facilitate a more convenient connection to MDMs, we introduce a *parallel* chain-of-thought process that can be seen as a generalization of CoT transformers that generates multiple symbols at once.

Definition B.10 (Parallel chain of thought (pCoT)). *Let $S \colon \Sigma_{\mathsf{m}}^* \to \Sigma_{\mathsf{m}01}$ be a causally masked transformer symbol predictor and $T, P' \in \mathbb{N}$. A **parallel chain-of-thought transformer** $\text{NS}_{\parallel} \colon \Sigma^* \to \Sigma_{\mathsf{m}}^*$ processes a string $\boldsymbol{w}$ from Σ^* for $t \in [T]$ as follows:*

$$\boldsymbol{t}_n^{(t)} \sim S(\boldsymbol{w} \circ \boldsymbol{t}^{(1)} \circ \cdots \circ \boldsymbol{t}^{(t-1)} \circ \underbrace{\mathsf{m} \cdots \mathsf{m}}_{P'})_{N+(t-1)P'+n} \quad \text{for } n \in [P']. \tag{24}$$

Whenever T and P are clear from the context, we write $\text{NS}_{\parallel}(\boldsymbol{w}) \stackrel{\text{def}}{=} \boldsymbol{t}^{(0)} \circ \cdots \circ \boldsymbol{t}^{(T)}$.

A pCoT transformer generates strings in Σ^* by running the pCoT process on the empty string ε and outputting $\text{NS}_{\parallel}(\varepsilon)$.

Definition B.11 (Acceptance by a pCoT Transformer). *We say that a pCoT transformer **accepts** a string $\boldsymbol{w}$ if there exist $T, P' \in \mathbb{N}$ such that it holds for $\boldsymbol{t}^{(1)} \circ \cdots \circ \boldsymbol{t}^{(T)} = \text{NS}_{\parallel}(\boldsymbol{w})$ that $\boldsymbol{t}_{P'}^{(T)} = 1$. $\text{NS}_{\parallel}$ rejects $\boldsymbol{w}$ if $\boldsymbol{t}_{P'}^{(T)} = 0$.*

We denote the class of languages recognizable by a pCoT transformers that generate P' symbols at a time for T steps as $\texttt{pCoT}[T, P]$, where $P \stackrel{\text{def}}{=} P'T$ is the total number of generated symbols.

The following relationships between CoT and pCoT transformers are clear.

Proposition B.1 (CoT and pCoT)**.** *We have* $\texttt{pCoT}[T, T] = \texttt{CoT}[T]$ *and* $\texttt{pCoT}[T, P] \subseteq \texttt{CoT}[P]$.

B.6 PADDED LOOPED TRANSFORMERS

Looped (or universal) transformers use a fixed block of transformer layers that is applied repeatedly to the input string (Dehghani et al., 2019). This increases the depth of the model, enabling more complex reasoning by applying layers multiple times, and does not increase the model size, as the same block is reused for each iteration, thus reducing the memory footprint and computational cost (Bae et al., 2025). We define looped transformers as follows.

Definition B.12 (Looped transformer (PLT))**.** *Let* $L, T \in \mathbb{N}$ *and let* $1 \leqslant l_1 \leqslant l_2 \leqslant L$. *Given a depth-$L$ transformer, a **looped transformer** (PLT) computes symbol contextual representations* $\boldsymbol{H}$ *by*

(1.) Computing the initial hidden states $\boldsymbol{H}^{(0)}$ *for the input string* $\boldsymbol{w} = w_1 \cdots w_N$ *and computing* $\boldsymbol{H}^{(l_1)}$ *as with the first* l_1 *layers of the transformer*

(2.) Applying the transformer layers $l_1 + 1, \ldots, l_2$ *T times to the hidden states* $\boldsymbol{H}^{(l_1)}$ *to obtain* $\boldsymbol{H}^{(l_1 + T(l_2 - l_1))}$.

(3.) Applying the transformer layers $l_2 + 1, \ldots, L$ *to the hidden states* $\boldsymbol{H}^{(l_1 + T(l_2 - l_1))}$ *to obtain the final representations* $\boldsymbol{H}$ *that are passed to the output layer.*

The representations $\boldsymbol{H}$ can then be used in the same way as described in §B.4.2.

The dynamic computational depth of PLTs endows them with the ability to perform more complex reasoning tasks by iteratively refining their hidden states over multiple timesteps. Importantly, these reasoning steps include both sequential and parallel processing of the input symbols, allowing for both parallel efficiency as well as depth in the reasoning process.

Padded transformers additionally pad the input string with padding (pause) symbols.

Definition B.13 (Padded Transformer)**.** *Given* $P \in \mathbb{N}$, *a **padded transformer** $\mathcal{T}$ transformer computes the contextual representations* $\boldsymbol{H}$ *of a string* $\boldsymbol{w} \in \Sigma^*$ *by processing the padded input* $\boldsymbol{w} \circ \underbrace{\square \cdots \square}_{P}$

(possibly by looping, cf. Def. B.12), where $\square \notin \Sigma$ *is a designated padding symbol.*

Instead of being restricted to the contextual representations of the N input symbols, a padded transformer can determine string membership or symbol probabilities based on the contextual representations of the P additional padded symbols as well. This additional space can be used to perform more operations and is analogous to increasing the circuit width in circuit complexity.

Padding and looping together increase the expressivity of transformers.

Remark 1 (Expressivity of padded looped transformers)**.** *The following characterizations of padded looped transformers are known:*

(1) Regular languages $\subseteq \texttt{PLT}[\log N, 0]$ *(Saunshi et al., 2025, Thm. 5.1),*

(2) $\texttt{CoT}[T] \preceq \texttt{PLT}[T, 0]$, *where the width of the PLT scales linearly with T (Saunshi et al., 2025, Thm. 5.4),*

(3) $\texttt{PLT}[\log N, \texttt{poly}(N)] = \texttt{AC}^0$ *(London & Kanade, 2025, Thm. 4.1),*

(4) $\texttt{PLT}[\log N, \texttt{poly}(N)] = \texttt{TC}^0$, *where, in contrast to our model, the precision* $\texttt{p}$ *scales logarithmically with string length N (London & Kanade (2025, Thm. 4.5), Merrill & Sabharwal (2025a, Thm. 1)),*

(5) $\texttt{PLT}[\log^d N, \texttt{poly}(N)] = \texttt{TC}^d$, *where the precision* $\texttt{p}$ *scales logarithmically with string length N (Merrill & Sabharwal, 2025a, Thm. 3).*

Stochastic padded looped transformers. The looping mechanism naturally accommodates the unmasking steps of MDMs. However, unlike PLT transformers, the MDM unmasking steps can be *stochastic*—the predictor *samples* the unmasked symbol from the infilling probability distribution defined by the transformer. To offer a more suitable analogue to MDMs, we introduce *stochastic* PLTs, which receive Gumbel-distributed noise as additional input to each loop, mimicking the sampling process of MDMs.

Definition B.14 (Stochastic padded looped transformer). *A **stochastic padded looped transformer** is a padded looped transformer in which $\boldsymbol{H}^{(l_1 + t(l_2 - l_1))}$ is augmented by a matrix of Gumbel-distributed noise variables at each time step $t \in [T]$.*

The fact that the MLPs in a transformer layer can implement the argmax operation used for Gumbel sampling (cf. Lem. D.11) allows PLTs to implement the planner and predictor of an MDM as we detail later (cf. Thm. E.1).

Including stochasticity in PLTs is required for a natural connection to stochastic models such as MDMs and CoT transformers. While this departs from the standard definitions of PLTs, it is a natural extension that allows us to capture the stochastic nature of MDMs while retaining the looping structure. Since the Gumbel noise is assumed to come from an external source in the sampling procedure of MDMs and CoT transformers, we believe adding it as input to PLTs is natural.

B.7   OUR IDEALIZATION OF MASKED DIFFUSION MODELS

In the following, $\Sigma_{\mathtt{m}} \stackrel{\text{def}}{=} \Sigma \cup \{\mathtt{m}\}$ where $\mathtt{m} \notin \Sigma$ is the mask symbol and $\Sigma_{01} \stackrel{\text{def}}{=} \Sigma \cup \{\mathtt{0}, \mathtt{1}\}$, where $\mathtt{0}$ and $\mathtt{1}$ are the reject and accept symbols.

Definition B.15 (Planner). *A **planner** is a length-preserving function $U \colon \Sigma_{\mathtt{m}}^* \to \mathcal{P}(\{\mathtt{0}, \mathtt{1}\})$. We distinguish two cases:*

*Case 1: An **unrestricted planner** is a planner that can choose to resample any symbol.*

*Case 2: A **mask dominated** planner is one where, if $w_n \neq \mathtt{m}$, then $(U(\boldsymbol{w})_n)_{\mathtt{0}} = 1$, i.e., U never tries to resample already unmasked symbols.*

*A **deterministic** planner is a length-preserving function $U \colon \Sigma_{\mathtt{m}}^* \to \{\mathtt{0}, \mathtt{1}\}^*$.*

Definition B.16 (Symbol Predictor). *A **symbol predictor** is a length-preserving function $S \colon \Sigma_{\mathtt{m}}^* \to \mathcal{P}(\Sigma_{01})^*$. A **deterministic** symbol predictor is a length-preserving function $S \colon \Sigma_{\mathtt{m}}^* \to \Sigma_{01}^*$.*

Definition B.17 (Masked Diffusion Model). *Given a planner U and a symbol predictor S, we call $M = (U, S)$ a **masked diffusion model (MDM)**.*

An MDM with a mask dominated planner corresponds closely to standard MDMs, where at each step, a subset of masked positions is selected for unmasking and then filled in. For example, setting the planner to implement uniformly random unmasking recovers the true reverse process of an MDM. We, however, additionally allow for unrestricted planners, which can also choose to *resample* already unmasked positions and we take this to be our default setting—whenever we refer to an MDM, we mean one with an unrestricted planner. While this departs from standard MDM formulations, it allows us to sidestep the limitations of the inability of the MDM to "change" its decisions and thus to correct earlier mistakes. This has recently been identified as a key limitation of MDMs and is the focus of much recent work on improving MDMs (von Rütte et al., 2025; Song et al., 2025, *inter alia*).

MDMs generate strings by iteratively unmasking and filling in symbols over a series of discrete denoising steps. We call this the **unmasking process**.

Definition B.18 (Unmasking process). *Let M be an MDM and $T, P \in \mathbb{N}$. Given a string $\boldsymbol{w} \in \Sigma^*$, M generates the string $M(\boldsymbol{w}) \stackrel{\text{def}}{=} \boldsymbol{y}^{(T)}$ as follows for $t \in [T]$:*

$$\boldsymbol{y}^{(0)} = \underbrace{\mathtt{m} \cdots \mathtt{m}}_{P} \tag{25a}$$

$$\boldsymbol{u}^{(t)} \sim U(\boldsymbol{w} \circ \boldsymbol{y}^{(t-1)}) \tag{25b}$$

$$\boldsymbol{y}_n^{(t)} = \begin{cases} y_n^{(t-1)} & \textbf{if } \boldsymbol{u}_n^{(t)} = 0 \\ y_n \sim S(\boldsymbol{w} \circ \boldsymbol{y}^{(t-1)})_{N+n} & \textbf{otherwise} \end{cases} \quad \textit{for } n \in [P] \tag{25c}$$

21

By running the MDM on the empty string ε and outputting $M(\varepsilon)$ (constraining the generated symbols to $\Sigma \subsetneq \Sigma_{01}$), an MDM generates strings in Σ^* and thus defines an LM.

We can also use the MDM to define a membership test for languages by using the unmasking process to emit reasoning (intermediate) computations and looking at the final symbol of the generated string.

Definition B.19 (Acceptance by an MDM). *We say that the diffusion model M with a deterministic U and S **accepts** a string $\boldsymbol{w} \in \Sigma^*$ if there exist $T, P \in \mathbb{N}$ such that it holds for $\boldsymbol{y}^{(T)} = M(\boldsymbol{w})$ that $y_P^{(T)} = 1$. M rejects $\boldsymbol{w}$ if $y_P^{(T)} = 0$.*

Transformer MDMs. We focus on MDMs where both the planner and predictor are implemented by transformers.

Definition B.20 (Transformer planner and predictor). *A **transformer planner** is a planner $\Sigma_\mathrm{m}^* \to \mathcal{P}(\{0, 1\})$ where the logits over $\{0, 1\}$ are computed by a transformer.*

*A **transformer predictor** is a predictor $S \colon \Sigma_\mathrm{m}^* \to \Sigma_{01}^*$ where the logits over Σ_{01} at each position are computed by a transformer.*

C A DISCUSSION OF THE THEORETICAL MODEL

Our formalization of MDMs, centered around the planner and predictor, is motivated by the need to analyze their expressivity. While this definition intentionally departs from the exact analytical reverse process, it remains well-grounded in both empirical practice and theoretical considerations. In the following, we justify our modeling choices and clarify their connections to practical implementations.

C.1 THE PLANNER AS A PRINCIPLED DEPARTURE FROM UNIFORM UNMASKING

At an intuitive level, our formalism distills the essential structure of practical MDMs: A process that iteratively unmasks and fills in symbols. In our idealization, this process is governed by a model that chooses which symbols to unmask and when. This is motivated by practical considerations—although the analytical reverse process of the MDM forward masking process would unmask symbols uniformly at random, this is not how practical MDMs operate, since they empirically benefit from strategic, context-aware unmasking (Ghazvininejad et al., 2019; Peng et al., 2025; Zheng et al., 2024; Liu et al., 2025a; Kim et al., 2025; Ben-Hamu et al., 2025). Forcing a trained MDM to follow a uniform unmasking schedule is thus suboptimal—it prevents the model from decoding in an order that makes tasks easier to solve (Kim et al., 2025) and can lead to uncontrolled error propagation (Ghazvininejad et al., 2019). The dedicated planner captures many empirically successful planned unmasking strategies that consistently outperform random unmasking such as:

- **Confidence-based unmasking** that unmask based on the model's prediction confidence.
- **Difficulty-based scheduling** that masks informative symbols longer so they are generated last with maximum context (He et al., 2023).
- **Structured schedules** such as blockwise-autoregressive unmasking (Nie et al., 2025).
- **Learned planners** that use a separate trained model to guide unmasking decisions (Peng et al., 2025).
- **Confidence thresholding** that unmasks symbols whose confidence lies above a theory-suggested threshold (Wu et al., 2025).

Theoretically, uniform unmasking is only optimal if the symbol predictor is perfect and unfactorized, which is not the case in practice (Peng et al., 2025). Furthermore, under a factorized backward process, uniform unmasking is computationally inefficient even with a perfect predictor. It requires at least a linear number of denoising steps in the string length to capture dependencies in even simple formal languages, and the KL divergence to the true data distribution converges slowly (Feng et al., 2025; Li & Cai, 2025). This lower bound negates any potential speed benefit over autoregressive models. A planner is therefore a necessary component for efficient, accurate generation, as it provides an educated choice of which symbols to unmask and generate in parallel, mitigating the weakness of the independence assumption. Frameworks such as that by Zheng et al. (2024) and augmented MDM

evidence lower bounds that incorporate explicit planner terms (Peng et al., 2025; Liu et al., 2025a) demonstrate that a planner can be a principled, optimizable component of the generative process.

To further justify our choice of modeling the planner as a separate component, we show that any MDM that can be implemented as a combination of a planner and a predictor can in fact be implemented by a single transformer that unmasks symbols based on their confidence—top-k decoding. This means that all our results apply to the popular model of unmasking symbols based on their confidence (Ghazvininejad et al., 2019; Peng et al., 2025; Zheng et al., 2024; Liu et al., 2025a; Kim et al., 2025; Ben-Hamu et al., 2025).

Definition C.1 (top-k unmasking). *Let $\mathcal{T}$ be a transformer and $k \in \mathbb{N}$. The* top-k ***unmasking process** of $\mathcal{T}$ is defined by the planner of the form*

$$U(\boldsymbol{y})_n = \begin{cases} 1 & \textbf{\textit{if}} \ n \in \textsf{top-}k(\mathcal{T}(\boldsymbol{y})) \\ 0 & \textbf{\textit{otherwise}} \end{cases} \text{ for } n \in [N], \tag{26}$$

where $\boldsymbol{y} \in \Sigma_{\mathsf{m}}{}^*$ with $N = |\boldsymbol{y}|$ and $\textsf{top-}k(\mathcal{T}(\boldsymbol{y}))$ selects the k positions in $[N]$ with the largest maximal logits in $\mathcal{T}(\boldsymbol{y})$. The predictor is defined as

$$S(\boldsymbol{y})_n = p^{\downarrow}(w \mid \boldsymbol{y}) \text{ for } n \in [N]. \tag{27}$$

where $p^{\downarrow}(w \mid \boldsymbol{y})$ is the infilling probability distribution of the nth token defined by $\mathcal{T}$.

Theorem C.1 (Combining the planner and predictor). *Let $M = (U, S)$ be a masked diffusion model with a planner U and a symbol predictor S. Then, there exists a transformer $\mathcal{T}$ such that it holds for any $\boldsymbol{y} \in \Sigma_{\mathsf{m}}{}^*$ with $N = |\boldsymbol{y}|$ and $n \in [N]$ that*

$$n \in \textsf{top-}k(\mathcal{T}(\boldsymbol{y})) \iff U(\boldsymbol{y})_n = 1 \tag{28a}$$

$$p^{\downarrow}(w \mid \boldsymbol{y}) = S(\boldsymbol{y})_n \tag{28b}$$

where $p^{\downarrow}(w \mid \boldsymbol{y})$ denotes the infilling probability distribution of the nth token defined by $\mathcal{T}$.

Proof. The construction of $\mathcal{T}$ combines the planner U and predictor S into a single model that predicts the next symbol based on the planner's decision and the predictor's distribution. In particular, $\mathcal{T}$ runs U and S in parallel. The planner's output decision $U(\boldsymbol{y})_n$ of 1 or 0 can be made based on implementing the argmax function with an MLP (cf. Lem. D.11) if the planner is deterministic or by inserting the noise from the Gumbel-max sampling (cf. Def. B.7) if the planner is probabilistic. $\mathcal{T}$ can then use this information to down-weight the logits of that are chosen not to be unmasked by the predictor by subtracting $B_{\mathbb{F}}$ from the logits of the symbols that are not chosen by the planner. This ensures that only the symbols that are chosen by the planner can be predicted by the predictor. The predictor S can in parallel compute its residual stream and the accompanying logits before combining them with the planner's decisions. The subtraction of the $B_{\mathbb{F}}$ from the logits of the symbols that are not chosen by the planner ensures that $\textsf{top-}k$ will only select the symbols that are chosen by the planner and the simulation of the predictor ensures that the infilling distributions match. ∎

C.2 Editing and non-editing MDMs.

The choice to allow the MDM to resample already unmasked symbols also departs from the analytical reverse process of MDMs, which only unmask masked symbols. This is, however, a principled choice that allows the MDM to correct earlier mistakes and is supported by empirical evidence that resampling already unmasked symbols can improve generation quality (von Rütte et al., 2025). Practically, this choice enables conciser connections to PLTs and pCoT transformers. Moreover, all results in this work can easily be adapted to MDMs with mask dominated planners that do not resample already unmasked symbols. This is justified by the following theorem that shows that any MDM with an unrestricted planner can be simulated by an MDM with a mask dominated planner by increasing the output space by a factor of T to account for the inability to resample. In the following, we refer to MDMs with mask dominated planners as *simple* MDMs (sMDMs).

Theorem C.2 (sMDMs can simulate MDMs).

$$\texttt{MDM}[T, P] \subseteq \texttt{sMDM}[T, TP]. \tag{29}$$

Proof. At a high level, the sMDM's planner selects the decoding space by selecting the next P positions to unmask, and the predictor *(1)* reads the string generated so far, *(2)* simulates a step of the MDM transformer on the string, and *(3)* writes the updated values into a *new* portion of the padding space.

More precisely, we can implement the sMDM transformer as follows:

(1) The sMDM transformer uses TP masked symbols to store the generated symbols at each of the T unmasking steps of the MDM transformer.

(2) The planner is implemented as the transformer from Lem. D.9 that acts independently of the input string and uses positional encodings to select the next P positions to unmask.

(3) The predictor is implemented as a transformer that predicts the values of the P masked symbols based on the current input string. It reads the values from the padding space by attending to the last P unmasked positions in the padding space analogous to the dump-decode-read mechanism from Lem. E.1. The only difference is that the predictor has to attend to the *last* unmasked block of decoded values. This can be done with a two additional transformer layers by *(a)* ignoring all masked symbols and *(b)* ignoring positions with identical positional encodings to their right.

■

D THEORETICAL GADGETS

This section contains various theoretical gadgets that are used in the proofs of the main results. Not all of these are novel, and some are modified restatements from their original sources.

In the following, $N \in \mathbb{N}$ always refers to the length of the original input string. If the string is additionally padded, the number of padding symbols is denoted by P, meaning that the entire input to the transformer is of length $N + P$.

D.1 POSITIONAL ENCODINGS

Uniquely identifying positions in a string requires the "volume" of the representation space to grow with the string length. In the case of finite-precision logarithmic-width transformers, this is achieved with positional encodings that encode the binary representation of the position in the string. The following lemma follows from the definition of fixed-point arithmetic, and the rounding and thresholding applied therein.

Lemma D.1. *Let $x \in \mathbb{F}_p$ for some $p \in \mathbb{N}$ such that $x > \log 2(p + 1)$. Then, it holds that*

$$\exp(x) = B_{\mathbb{F}}, \tag{30a}$$
$$\exp(-x) = 0. \tag{30b}$$

Lemmata D.2 and D.3 readily follow from Lem. D.1.

Lemma D.2 (Li et al. (2024, Lemmata E.1 and E.2)). *For $B_{\mathbb{F}}$ from §B.3, it holds that*

$$\exp(B_{\mathbb{F}}) = B_{\mathbb{F}}, \tag{31a}$$
$$\exp(-B_{\mathbb{F}}) = 0. \tag{31b}$$

Lemma D.3 (Li et al. (2024, Lem. E.3)). *For $N \in \mathbb{N}$, $n \in [N]$, define the vectors $\boldsymbol{q}_n \in \mathbb{R}^{2\lceil \log N \rceil}$ and $\boldsymbol{k}_n \in \mathbb{R}^{2\lceil \log N \rceil}$ as follows:*

$$\boldsymbol{q}_n \stackrel{\text{def}}{=} B_{\mathbb{F}} \cdot (\mathsf{B}^{\pm}(n) ^\frown \mathbf{1}_{\lceil \log N \rceil}) \tag{32a}$$
$$\boldsymbol{k}_{n'} \stackrel{\text{def}}{=} \mathsf{B}^{\pm}(n') ^\frown (-\mathbf{1}_{\lceil \log N \rceil}). \tag{32b}$$

Then, it holds that

$$\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} = \begin{cases} 0 & \textbf{\textit{if}} \ n = n' \\ -B_{\mathbb{F}} & \textbf{\textit{otherwise}} \ . \end{cases} \tag{33}$$

Thus,

$$\exp(\boldsymbol{q}_n^\top \boldsymbol{k}_{n'}) = \begin{cases} 1 & \textbf{if } n = n' \\ 0 & \textbf{otherwise} . \end{cases} \tag{34}$$

The following slight generalization of Lem. D.3 is also easy to show.

Lemma D.4. *For $N \in \mathbb{N}$, $n \in [N]$, define the vectors $\boldsymbol{q}_n \in \mathbb{R}^{2\lceil \log N \rceil}$ and $\boldsymbol{k}_n \in \mathbb{R}^{2\lceil \log N \rceil}$ as follows:*

$$\boldsymbol{q}_n \stackrel{\text{def}}{=} B_{\mathbb{F}}/m \cdot (\mathsf{B}^\pm(n) \frown \mathbf{1}_{\lceil \log N \rceil}) \tag{35a}$$

$$\boldsymbol{k}_{n'} \stackrel{\text{def}}{=} \mathsf{B}^\pm(n') \frown (-\mathbf{1}_{\lceil \log N \rceil}). \tag{35b}$$

Then, it holds that

$$\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} = \begin{cases} 0 & \textbf{if } n = n' \\ x & \textbf{otherwise} . \end{cases} \tag{36}$$

where $x \leqslant -^{2B_{\mathbb{F}}}/m$.

Our constructions heavily rely on specific positional encodings. We assume that this positional information comes from an outside source and is not computed by the transformer model directly. We note that this is in contrast to some existing work with causally masked transformers where the positional encodings are inferred by the model itself (Yang et al., 2024; Li & Cotterell, 2025; Jerad et al., 2025). This matters for multiple reasons. First, including positional information from an external source allows us to decouple the computation of positional information from the computations performed by the transformer model. Furthermore, it facilitates providing the model with structured information that it would not be able to compute on its own. While this could be abused to give the model unrealistic computational power, we assume that the positional information is easily computable and thus realistic.

In that vein, it is interesting to consider what is the minimal amount of positional information that the model has to be provided with for it to be able to construct the useful positional encodings. This is described by the following lemma. At a high level, it says that our positional encodings require the binary encodings of the relevant numbers, along with any modular and division operations performed on them. Addition, thresholding, and multiplication by a power of two, in contrast, can be performed by the MLPs in the transformer model.

Lemma D.5 (Arithmetic operations performed by MLPs). *Let $m, n \in \mathbb{N}$. Then, given the binary encodings of the numbers, $\mathsf{B}(m), \mathsf{B}(n)$, there exist MLPs that can compute the following operations:*

    (a) computing the signed binary encoding $\mathsf{B}^\pm(m)$,

    (b) computing the sum $\mathsf{B}(m) + \mathsf{B}(n)$ and the difference $\mathsf{B}(m) - \mathsf{B}(n)$,

    (c) computing $\mathsf{B}(2^k m)$ for $k \in \mathbb{N}$,

    (d) computing the indicator function $\mathbb{1}\{m \geqslant 0\}$,

    (e) computing the indicator function $\mathbb{1}\{m = 0\}$, and

    (f) computing the positive-part function $[m]_+$.

Proof.

    (a) The transformation $\mathsf{B}^\pm(m) \stackrel{\text{def}}{=} 2\mathsf{B}(m) - \mathbf{1}_{\lceil \log m \rceil}$ is an affine function that can be implemented by the affine part of the MLP followed by the identity function, which can be implemented by an MLP as well.

    (b) It is easy to implement AND and NOT gates by an MLP. This suffices to implement any AC^0 (and thus addition and subtraction) circuit.

    (c) The transformation $\mathsf{B}(m) \mapsto \mathsf{B}(2^k m)$ can be performed by shifting the binary representation of $\mathsf{B}(m)$ to the left by k positions, which can be implemented by a linear transformation.

(d) The indicator function $\mathbb{1}\{m \geqslant 0\}$ can be computed by checking the sign bit of the signed binary representation $\mathsf{B}^{\pm}(m)$, which can be implemented by an MLP.

(e) The indicator function $\mathbb{1}\{m = 0\}$ can be computed by checking if all bits of the binary representation $\mathsf{B}(m)$ are zero, which can also be implemented by an MLP.

(f) Computing $[m]_{+}$ can be done by first computing $\mathbb{1}\{m \geqslant 0\}$ with *(d)* and then conditionally outputting m or 0 based on the result, which can be implemented by an MLP similar to Lem. D.12.

∎

Simplification of positional encodings. Lem. D.5 leads us to conclude that the relatively complicated positional encodings used in the following proofs can be thought of as simple transformations of the "basic" information captured by

- $\mathsf{B}(n)$,
- $\mathsf{B}(N)$,
- $\mathsf{B}(n \mod m)$ for some relevant $m \in \mathbb{N}$, and
- $\mathsf{B}(n/m)$ for some relevant $m \in \mathbb{N}$.

This makes the positional encodings in our constructions streamlined and easily computable. While somewhat non-standard, we note that positional encodings based on both the symbol position n and string length N are common in theoretical literature (Chiang & Cholak, 2022).

Despite being simple to compute, these positional encodings are powerful enough to allow the transformer to uniquely identify positions in the string and to perform useful computations based on them. In a sense, the inclusion of this information is also necessary, as the operations such as division and modular arithmetic—including the computation of the binary encodings—lie outside of AC^0 and thus cannot be performed by finite-precision transformers (Li et al., 2024). We note, however, that more expressive transformers such as those with logarithmic precision could possibly implement the required functions to compute the information in $\mathsf{B}(n)$, $\mathsf{B}(n \mod m)$, and $\mathsf{B}(n/m)$ from n and N directly since, unlike fixed-precision transformers, they are not constrained to AC^0 operations (Li et al., 2024).[21] The simplicity and uniformity of these encodings lies in contrast to more complex (non-uniform) positional encodings that directly serialize the circuits to be simulated when analyzing expressivity lower bounds (Li et al., 2024; Saunshi et al., 2025; London & Kanade, 2025).

D.2 USEFUL ATTENTION PATTERNS

The following lemmata describe how a transformer layer can either ignore or exclusively focus on specific positions in the input string.

Lemma D.6 (Ignoring Marked Positions with a Transformer). *Let $N, D \in \mathbb{N}$, $\mathcal{N} \subseteq [N]$, and let $\boldsymbol{H} \in \mathbb{R}^{N \times D}$ be a matrix representing the residual stream such that*

$$\llbracket \square \rrbracket \in \boldsymbol{H}_{n,:} \iff n \in \mathcal{N}. \tag{37}$$

Here, the notation $\llbracket \square \rrbracket \in \boldsymbol{H}_{n,:}$ means that the vector $\boldsymbol{H}_{n,:}$ contains the one-hot encoding of the symbol $\square$ at position n. Further, let $\boldsymbol{G} \stackrel{\text{def}}{=} \boldsymbol{H}_{[N] \setminus \mathcal{N},:} \in \mathbb{R}^{(N - |\mathcal{N}|) \times D}$, where $\boldsymbol{H}_{[N] \setminus \mathcal{N},:}$ denotes the projection of the matrix $\boldsymbol{H}$ onto the rows not in $\mathcal{N}$. Finally, let $\boldsymbol{\tau}$ be a transformer layer. Then, there exists a logarithmic-width transformer layer $\boldsymbol{\tau}'$ such that it holds for $\boldsymbol{G}' \stackrel{\text{def}}{=} \boldsymbol{\tau}(\boldsymbol{G}) \in \mathbb{R}^{(N - |\mathcal{N}|) \times D}$ and $\boldsymbol{H}' \stackrel{\text{def}}{=} \boldsymbol{\tau}'(\boldsymbol{H}) \in \mathbb{R}^{N \times D}$ that

$$\boldsymbol{G}' = \boldsymbol{H}'_{[N] \setminus \mathcal{N},:}. \tag{38}$$

Informally, Lem. D.6 states that a transformer layer can ignore positions containing one-hot encodings of specific "marker" symbols, such as additional symbols not in the original alphabet. Here, ignoring means that the content of the positions with the marker symbols does not affect the output of the transformer layer at other positions.

---

[21]We also note that the modular information in our encodings resembles the periodic nature of original sinusoidal positional encodings used by the transformer architecture (Vaswani et al., 2017). Moreover, the modular nature of such positional encodings has been analyzed by theoretical work before and is known to increase the expressivity of certain idealizations of transformers (Li & Cotterell, 2025; Jerad et al., 2025).

Proof. Notice that, since $\boldsymbol{H}$ and $\boldsymbol{G}$ match on all positions not in $\mathcal{N}$, *ignoring* the positions in $\mathcal{N}$ (marked by $\square$) by $\boldsymbol{\tau}'$ will ensure that the outputs of the two layers $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$ are identical on the positions not in $\mathcal{N}$. We now construct a transformer layer that ignores the contributions of rows marked by $[\![\square]\!]$. To do so, we modify each attention head of $\boldsymbol{\tau}$ such that the head computes its attention scores with queries and keys of the form

$$\boldsymbol{q}'_n \stackrel{\text{def}}{=} \cdot \begin{pmatrix} \boldsymbol{q}_n \\ -B_{\mathbb{F}} \cdot [\![\square]\!] \\ -B_{\mathbb{F}} \cdot [\![\square]\!] \end{pmatrix} \tag{39a}$$

$$\boldsymbol{k}'_{n'} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{k}_{n'} \\ [\![w_{n'}]\!] \\ [\![w_{n'}]\!] \end{pmatrix} \tag{39b}$$

where $\boldsymbol{q}_n$ and $\boldsymbol{k}_{n'}$ are the original head's query and key vectors of $\mathcal{T}$ at position n and n', respectively, and $[\![w_{n'}]\!]$ is the one-hot encoding of the symbol at position n'. We can then compute the dot product of the two vectors as

$$\boldsymbol{q}'^{\top}_n \boldsymbol{k}'_{n'} = \boldsymbol{q}^{\top}_n \boldsymbol{k}_{n'} - B_{\mathbb{F}} \cdot \mathbb{1}\{w_{n'} = \square\} - B_{\mathbb{F}} \cdot \mathbb{1}\{w_{n'} = \square\}. \tag{40}$$

Thus, if the symbol at position n' is not $\square$, the attention score is $\boldsymbol{q}^{\top}_n \boldsymbol{k}_{n'}$ and the head behaves as it did in $\mathcal{T}$. If the symbol at position n' is $\square$, the last two components of the vectors $\boldsymbol{q}'_n$ and $\boldsymbol{k}'_{n'}$ ensure that the exponentiated value of the attention score becomes 0, thus not contributing to the attention weights. $\mathcal{T}'$ can thus simulate $\mathcal{T}$ on the rest of the positions. $\blacksquare$

Lemma D.7 (Focusing on Marked Positions with a Transformer). *Let $N, D \in \mathbb{N}$, $R\colon [N] \to [N]$, $r \in [N]$, $\mathcal{N} \stackrel{\text{def}}{=} R^{-1}(r) \subseteq [N]$, and let $\boldsymbol{H} \in \mathbb{R}^{N \times D}$ be a matrix representing the residual stream such that*

$$\bar{\mathsf{B}}(R(n)) \in \boldsymbol{H}_{n,:} \quad \text{for all } n \in [N] \tag{41}$$

Here, the notation $\bar{\mathsf{B}}(R(n)) \in \boldsymbol{H}_{n,:}$ means that the vector $\boldsymbol{H}_{n,:}$ contains the signed binary encoding (cf. §2) of $R(n)$. Further, let $\boldsymbol{G} \stackrel{\text{def}}{=} \boldsymbol{H}_{\mathcal{N},:} \in \mathbb{R}^{|\mathcal{N}| \times D}$, where $\boldsymbol{H}_{\mathcal{N},:}$ denotes the projection of the matrix $\boldsymbol{H}$ onto the rows in $\mathcal{N}$. Finally, let $\boldsymbol{\tau}$ be a transformer layer. Then, there exists a logarithmic-width transformer layer $\boldsymbol{\tau}'$ such that it holds for $\boldsymbol{G}' \stackrel{\text{def}}{=} \boldsymbol{\tau}(\boldsymbol{G}) \in \mathbb{R}^{|\mathcal{N}| \times D}$ and $\boldsymbol{H}' \stackrel{\text{def}}{=} \boldsymbol{\tau}'(\boldsymbol{H}) \in \mathbb{R}^{N \times D}$ that

$$\boldsymbol{G}' = \boldsymbol{H}'_{\mathcal{N},:}. \tag{42}$$

Informally, Lem. D.7 states that a transformer layer can focus on positions containing signed binary encodings of a number r computed as a function of the position while ignoring the rest of the positions.

Proof. The idea of the construction of $\boldsymbol{\tau}'$ is similar to that of Lem. D.6, but instead of ignoring the positions in $\mathcal{N}$, we want the transformer layer to focus on them while ignoring the rest of the positions. This can be done by including $\bar{\mathsf{B}}(R(n))$ in the positional encodings of the attention heads and then using the key and query vectors of the form

$$\boldsymbol{q}'_n \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{q}_n \\ B_{\mathbb{F}}/2 \cdot (\mathsf{B}^{\pm}(r) \frown \boldsymbol{1}_{\lceil \log N \rceil}) \\ B_{\mathbb{F}}/2 \cdot (\mathsf{B}^{\pm}(r) \frown \boldsymbol{1}_{\lceil \log N \rceil}) \end{pmatrix} \tag{43a}$$

$$\boldsymbol{k}'_{n'} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{k}_{n'} \\ \mathsf{B}^{\pm}(R(n')) \frown (-\boldsymbol{1}_{\lceil \log N \rceil}) \\ \mathsf{B}^{\pm}(R(n')) \frown (-\boldsymbol{1}_{\lceil \log N \rceil}) \end{pmatrix} \tag{43b}$$

where $\boldsymbol{q}_n$ and $\boldsymbol{k}_{n'}$ are the original head's query and key vectors of $\mathcal{T}$ at position n and n', respectively, and $B_{\mathbb{F}}$ is the maximal representable number in the fixed-point arithmetic (which might depend on the string length N). We can then compute the dot product of the two vectors as

$$\boldsymbol{q}'^{\top}_n \boldsymbol{k}'_{n'} = \boldsymbol{q}^{\top}_n \boldsymbol{k}_{n'} + \underbrace{B_{\mathbb{F}}/2 \cdot (\mathsf{B}^{\pm}(r) \frown \boldsymbol{1}_{\lceil \log N \rceil})^{\top} (\mathsf{B}^{\pm}(R(n')) \frown (-\boldsymbol{1}_{\lceil \log N \rceil}))}_{\stackrel{\text{def}}{=} G} \tag{44a}$$

$$+ \underbrace{B_{\mathbb{F}}/2 \cdot (\mathsf{B}^{\pm}(r) \frown \boldsymbol{1}_{\lceil \log N \rceil})^{\top} (\mathsf{B}^{\pm}(R(n')) \frown (-\boldsymbol{1}_{\lceil \log N \rceil}))}_{\stackrel{\text{def}}{=} G}$$

Note that Eq. (44a) uses fixed-point arithmetic. We compute the inner product in Eq. (44a) by analyzing individual cases:

1. **Case 1:** $R(n') \neq r$.

   All intermediate computations of Eq. (44a) are thresholded at $\min(B_{\mathbb{F}}, \boldsymbol{q}_n^\top \boldsymbol{k}_{n'} + {}^{B_{\mathbb{F}}}/_2)$. In particular, by Lem. D.4, the value after adding the first term G is at most $\min(B_{\mathbb{F}}, \boldsymbol{q}_n^\top \boldsymbol{k}_{n'} + {}^{B_{\mathbb{F}}}/_2) - 2^{B_{\mathbb{F}}}/_2 \leqslant B_{\mathbb{F}} - B_{\mathbb{F}} = 0$. After adding the second term G, the value is at most $-B_{\mathbb{F}}$, resulting in a 0 exponentiated attention score, as required.

2. **Case 2:** $R(n') = r$. We analyze three sub-cases based on the value of $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'}$.

   1. **Sub-case 2a:** $\left| \boldsymbol{q}_n^\top \boldsymbol{k}_{n'} \right| < \log 2(\mathsf{p} + 1)$. All intermediate computations in Eq. (44a) are bounded by $\log 2(\mathsf{p} + 1) + {}^{B_{\mathbb{F}}}/_2$ in absolute value, so they fall within the range of $\mathbb{F}_{\mathsf{p}}$. Moreover, addition of ${}^{B_{\mathbb{F}}}/_2$ can be exactly represented in $\mathbb{F}_{\mathsf{p}}$. This makes addition in Eq. (44a) associative and commutative. By Lem. D.3, the terms G in Eq. (44a) are 0, meaning that the final attention score equals $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'}$.

   2. **Sub-case 2b:** $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} \geqslant \log 2(\mathsf{p} + 1)$. In this case, the intermediate computations of Eq. (44a) are either exact or thresholded at $B_{\mathbb{F}}$. In both cases, the exponent of the resulting attention score is $B_{\mathbb{F}}$ by Lem. D.1, preserving the attention score.

   3. **Sub-case 2c:** $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} \leqslant \log 2(\mathsf{p} + 1)$. In this case, all intermediate computations are representable in $\mathbb{F}_{\mathsf{p}}$ analogously to the case 2a. The attention score is therefore preserved.

Taken together, this means that the attention scores between positions n and n' of $\boldsymbol{\tau}'$ are identical to those of $\boldsymbol{\tau}$ on the positions in $\mathcal{N}$, while the attention scores on the rest of the positions are 0. This completes the proof. ∎

Lemma D.8 (Detecting a symbol occurrence). *Let $\boldsymbol{w} \in \Sigma^*$ and $w \in \Sigma$. Then, there exists a single-layer unmasked fixed-precision logarithmic-width transformer $\mathcal{T}$ such that, on input $\boldsymbol{w}$, an entry of its final residual stream contains the entry $\mathbb{1}\{w \in \boldsymbol{w}\}$.*

Proof sketch. Note that $\mathcal{T}$ cannot use the commonly-used *exact* uniform attention over all symbols to detect $\mathbb{1}\{w \in \boldsymbol{w}\}$ due to fixed precision. Nevertheless, rounded uniform attention suffices. By attending to all symbols in the string with weight 1, the denominator of the attention scores is at most $B_{\mathbb{F}}$. Using one-hot encodings of symbols w_n as the attention values $\boldsymbol{v}_n$, it is easy to see that the final contextual representation at the final position will have a positive value at the entry corresponding to w if and only if $w \in \boldsymbol{w}$, since $^c/_{B_{\mathbb{F}}} > 0$ for any $c \geqslant 1$. This condition can be checked by the MLP applied after the attention aggregation operation. ∎

Lemma D.9. *Let $\boldsymbol{w} \in \Sigma^*$ be a string of length N and PT the number of padding symbols where $P, T = \mathtt{poly}(N)$. There exists a fixed-precision and logarithmic-width transformer $U \colon \Sigma_{\mathsf{m}}^* \to \{0, 1\}^*$ that, given $\boldsymbol{w}$ and the current partially masked string $\boldsymbol{t}^{(t)}$, selects the next P positions to unmask by outputting 1 for the next P positions to unmask and 0 for all other positions.*

Proof. The idea of the construction is for U to *(1)* output 0 for any position that does not contain the padding symbol □, and *(2)* output 1 for the first P positions that contain the padding symbol □. Step *(1)* can be implemented by checking whether the symbol at the current position is □ and outputting 0 otherwise. Step *(2)* can be implemented by attending to position P positions back and outputting 1 if the symbol at that position $\neq$ □ and 0 otherwise. These steps can be performed by two attention heads in a single transformer layer using the positional encodings

$$\mathsf{PE}(n, N) \stackrel{\text{def}}{=} \begin{pmatrix} \mathsf{B}^{\pm}(n) \\ \mathsf{B}^{\pm}(n - P) \end{pmatrix} \in \{0, 1\}^{\mathcal{O}(\log N)}. \tag{45}$$

∎

Lemma D.10 (Converting a binary representation into a positional encoding). *Let $N \in \mathbb{N}$ and $n \in [N]$. Then, there exists an unmasked fixed-precision logarithmic-width padded looped transformer $\mathcal{T}$ such that, on input* & $\mathsf{B}(n)$*, after $\lceil \log N \rceil$ iterations, the residual stream at position $\lceil \log N \rceil + 1$ contains the value $\mathsf{B}(n)$.*

Proof sketch. The transformer $\mathcal{T}$ has to convert the binary representation $\mathsf{B}(n)$ of n contained in across $\lceil \log N \rceil$ positions in the input string into a single $\lceil \log N \rceil$-dimensional binary vector in the residual stream. This is done as follows:

1. In the first layer, each symbol $w_{n'} \in \{0, 1\}$ checks if it is immediately preceded by the & symbol, which denotes the beginning of the pointer in the string. If it is, $w_{n'}$ stores $\boldsymbol{e}_1$ and $\boldsymbol{d}_1 \stackrel{\text{def}}{=} w_{n'} \boldsymbol{e}_1$ in designated parts of its residual stream. Here, $\boldsymbol{e}_1$ is the first unit vector of $\mathbb{R}^{\lceil \log N \rceil}$.
2. In the subsequent layers $l \in \{2, \dots, \lceil \log N \rceil\}$, each symbol $w_{n'}$ checks if the entry $\boldsymbol{e}_{l-1}$ has already been written to the designated space of the previous symbol's residual stream. If it has, $w_{n'}$ copies and shifts $\boldsymbol{e}_{l-1}$ into $\boldsymbol{e}_l$, and stores $\boldsymbol{e}_l$ and $\boldsymbol{d}_l \stackrel{\text{def}}{=} \boldsymbol{d}_{l-1} + w_{n'} \boldsymbol{e}_l$ in designated parts of its residual stream.

After $\lceil \log N \rceil$ layers, the residual stream at position $\lceil \log N \rceil + 1$ thus contains $\mathsf{B}(n)$. ∎

D.3 Neural network constructions

Lemma D.11 (Saunshi et al. (2025, Lem. B.6)). *For every $D \in \mathbb{N}$ and precision $\mathsf{p} \in \mathbb{N}$, there exists a ReLU-activated MLP $f \colon \mathbb{R}^D \to \{0, 1\}^D$ such that for any $\boldsymbol{x} \in \mathbb{F}_{\mathsf{p}}^D$, if there is $d \in [D]$, such that $x_d > \max_{j \in [D] \setminus \{d\}} x_j$, then $f(\boldsymbol{x}) = \boldsymbol{e}_d$, the dth unit vector.*

Proof. The proof is based on the construction of a 3-layer ReLU network that computes the argmax of a vector $\boldsymbol{x} \in \mathbb{F}_{\mathsf{p}}^D$. The first layer computes the differences between each pair of elements in $\boldsymbol{x}$. The second layer computes the maximum of these differences. The third layer then checks if the maximum difference is greater than zero, indicating that there is a unique maximum element in $\boldsymbol{x}$.

More, concretely, define

$$g_d \stackrel{\text{def}}{=} 2^{\mathsf{p}} \cdot \mathrm{ReLU}\left(2^{-\mathsf{p}} - \sum_{j \neq d} \mathrm{ReLU}(x_j - x_d)\right), \tag{46}$$

which can be computed by a 3-layer ReLU network. We have that $g_d = 1$ if and only if $x_d > \max_{j \neq d} x_j$, or, equivalently, if and only if $x_d - \max_{j \neq d} x_j \geqslant 2^{-\mathsf{p}}$. Indeed if $x_d - \max_{j \neq d} x_j \geqslant 2^{-\mathsf{p}}$, we have that

$$g_d = 2^{\mathsf{p}} \cdot \mathrm{ReLU}\left(2^{-\mathsf{p}} - \sum_{j \neq d} \mathrm{ReLU}(x_j - x_d)\right) = 1. \tag{47}$$

In contrast, for $d' \neq d$, we have that $x_{d'} - x_d < 2^{-\mathsf{p}}$, and thus

$$g_{d'} = 2^{\mathsf{p}} \cdot \mathrm{ReLU}\left(2^{-\mathsf{p}} - \sum_{j \neq d'} \mathrm{ReLU}(x_j - x_{d'})\right) \leqslant 2^{\mathsf{p}} \cdot \mathrm{ReLU}\left(2^{-\mathsf{p}} - \mathrm{ReLU}(x_d - x_{d'})\right) = 0. \tag{48}$$

∎

Lemma D.12. *Let $\boldsymbol{x} \in \{0, 1\}^D$ and $\boldsymbol{e}_d \in \{0, 1\}^D$ be the dth unit vector. Then, there exists a ReLU-activated MLP $f \colon \{0, 1\}^{D+D} \to \{0, 1\}^D$ such that*

$$f(\boldsymbol{x}, \boldsymbol{e}_d) = \boldsymbol{x}^\top \boldsymbol{e}_d = x_d. \tag{49}$$

Proof. We have that

$$x_d = \mathbf{1}^\top (\mathrm{ReLU}(\boldsymbol{x} - (\mathbf{1} - \boldsymbol{e}_d))) \tag{50}$$

where $\mathbf{1}$ is the all-ones vector of length D. This can be implemented by a ReLU-activated MLP. ∎

D.4 MASKED AND UNMASKED TRANSFORMERS

Lemma D.13 (Unmasked to causally masked; Merrill & Sabharwal (2025a, Lem. 1)). *Let $\mathcal{T}$ be an unmasked fixed-precision logarithmic-width transformer with L layers. Then, there exists a fixed-precision logarithmic-width causally-masked transformer $\mathcal{T}'$ with L layers such that for any input string $\boldsymbol{w} \in \Sigma^*$ of length N padded with $L(N-1)$ padding symbols, the representations $\boldsymbol{H}'^{(L)}_{(L-1)N:LN}$ computed by $\mathcal{T}'$ on $\boldsymbol{w}$ equal the final representations $\boldsymbol{H}^{(L)}$ computed by $\mathcal{T}$.*

Proof. We adapt the proof of Merrill & Sabharwal (2025a, Lem. 1) to our setting. The idea is for $\mathcal{T}'$ to unroll the computation of $\boldsymbol{H}^{(L)}$ into a sequence of L "blocks" of padding, each of width N. Each block will attend to the previous block—representing the values in the preceding layer—and will thus be able to see all symbols despite causal masking. To do so, $\mathcal{T}'$ uses additional positional encodings of the form

$$\mathsf{PE}(n, N) \stackrel{\text{def}}{=} \begin{pmatrix} \mathsf{B}^{\pm}(l_n) \\ \mathsf{B}^{\pm}(l_n - 1) \end{pmatrix} \in \{0, 1\}^{2\lceil \log N \rceil} \tag{51}$$

Here, $l_n \stackrel{\text{def}}{=} \lfloor n/L \rfloor + 1$ represents the layer that each padding position belongs to. To construct $\mathcal{T}'$, we then modify each original head from $\mathcal{T}$ with Lem. D.7 to ensure that the attention is focused on the correct padding positions, where the attention head computes the same function as the one in $\mathcal{T}$. ∎

Lemma D.14 (Causally masked to unmasked). *Let $\mathcal{T}$ be an L-layer finite-precision logarithmic-width causally-masked transformer. Then, there exists a finite-precision logarithmic-width unmasked transformer $\mathcal{T}'$ with $2L + 1$ layers such that for any input string $\boldsymbol{w} \in \Sigma^*$ of length N padded with $(N-1)N$ padding symbols, the representations $\boldsymbol{H}'^{(L)}_{(N-1)N:N^2}$ computed by $\mathcal{T}'$ on $\boldsymbol{w}$ equal the final representations $\boldsymbol{H}^{(L)}$ computed by $\mathcal{T}$.[22]*

Proof. The idea is for $\mathcal{T}'$ to unroll the computation of $\boldsymbol{H}^{(L)}$ into a sequence of N "blocks" of padding, each of width N. Each block will compute the representation of one of the symbols in the string. To do so, $\mathcal{T}'$ uses additional[23] positional encodings of the form

$$\mathsf{PE}(n, N) \stackrel{\text{def}}{=} \begin{pmatrix} \mathsf{B}^{\pm}(b_n) \\ \mathsf{B}^{\pm}(r_n) \\ \mathbb{1}\{n \leqslant N\} \\ \mathbb{1}\{b_n \geqslant r_n\} \\ \mathbb{1}\{b_n = r_n\} \end{pmatrix} \in \{0, 1\}^{\mathcal{O}(\log N)}. \tag{52}$$

Here, $b_n \stackrel{\text{def}}{=} \lfloor n/N \rfloor + 1$ represents the block that position n falls into and $r_n \stackrel{\text{def}}{=} (n \mod N) + 1$ represents the position within that layer.

$\mathcal{T}'$ then processes a string $\boldsymbol{w} \in \Sigma^*$ of length N padded with N^2 padding symbols as follows.

(1) $\mathcal{T}'$ uses an additional "copy" layer to copy the input symbols from the first N positions to the residual stream for access in later layers. In particular, each position $n \in \left[N^2\right]$ is assigned the value of the input symbol at position r_n. This can be done by the symbol at position n attending to the symbol at position r_n in the input string, i.e., $\boldsymbol{H}^{(1)}_n = \boldsymbol{H}^{(0)}_{r_n}$, if $b_n \geqslant r_n$, which can be ensured by attending only to positions with non-zero entry $\mathbb{1}\{b_n \geqslant r_n\}\mathsf{B}^{\pm}(r_n)$ in the positional encoding. The latter condition ensures that the bth block contains only symbols $\boldsymbol{w}_{\leqslant b}$—$w_b$ attending to the *entire* block is then equivalent to w_b attending to the string with causal attention. Concretely, the attention scores $s^{(1)}_{n,n'} = {\boldsymbol{q}^{(1)}_n}^{\top} \boldsymbol{k}^{(1)}_{n'}$ are computed

---

[22]This simulation is somewhat inefficient in that only a subset of the N positions are used at each of the N blocks (specifically, n positions in the nth block). While this could be made more efficient with a more sophisticated construction, the asymptotic complexity would remain quadratic in N.

[23]By additional, we mean that these positional encodings are appended to the ones used by $\mathcal{T}$.

with query and key vectors

$$q_n^{(1)} \stackrel{\text{def}}{=} B_{\mathbb{F}} \cdot \begin{pmatrix} \mathrm{B}^{\pm}(r_n) \frown \mathbf{1}_{\lceil \log N \rceil} \\ -1 \end{pmatrix} \tag{53a}$$

$$k_{n'}^{(1)} \stackrel{\text{def}}{=} \begin{pmatrix} \mathrm{B}^{\pm}(r_{n'}) \frown (-\mathbf{1}_{\lceil \log N \rceil}) \\ \mathbb{1}\{n \leqslant N\} \end{pmatrix} \tag{53b}$$

(2) Once the symbols have been copied to the appropriate positions, $\mathcal{T}'$ can simulate one layer of $\mathcal{T}$ by augmenting its heads with Lem. D.7 to ensure that the computations at position n are restricted to the block b_n, which, as described above, contains information about $w_{\leqslant b_n}$.[24] This ensures that the attention scores are non-zero only for *(i)* positions n in the same block b_n as n', and *(ii)* positions that should be unmasked in the current block.

(3) After simulating the layer from $\mathcal{T}$ in step (2), the contextual representation in the $b_n{}^{\text{th}}$ block contain the information about $w_{\leqslant b_n}$ computed based on the symbols $w_{\leqslant b_n}$. In particular, the representations of the symbols $w_{<b_n}$ in the $b_n{}^{\text{th}}$ block contain information *not* obtained by causal masking since they attend to *all* the symbols $w_{\leqslant b_n}$ in the $b_n{}^{\text{th}}$ block. To amend that, an additional transformer layer discards the information about symbols $w_{<b_n}$ in the $b_n{}^{\text{th}}$ block by overwriting the representation of $w_{n'}$ in the $b_n{}^{\text{th}}$ block with the representation of $w_{n'}$ in the n'^{th} block for $n' < b_n$. This is done by attending to the positions n' in which the block index $b_{n'}$ matches the position $r_{n'}$, i.e., with the query and key vectors

$$q_n \stackrel{\text{def}}{=} B_{\mathbb{F}} \cdot \begin{pmatrix} \mathrm{B}^{\pm}(b_n) \frown \mathbf{1}_{\lceil \log N \rceil} \\ -1 \end{pmatrix} \tag{54a}$$

$$k_{n'} \stackrel{\text{def}}{=} \begin{pmatrix} \mathrm{B}^{\pm}(r_{n'}) \frown (-\mathbf{1}_{\lceil \log N \rceil}) \\ \mathbb{1}\{b_n = r_n\} \end{pmatrix} \tag{54b}$$

which ensures that n uniquely attends to the symbols $w_{\leqslant b_n}$ in the $b_n{}^{\text{th}}$ block.

■

E   PROOFS

This section contains the proofs of all novel theoretical results. Many constructions in the proofs rely on the theoretical gadgets introduced in §D.

Theorem 2.1. *If Assumptions 2.1 and 2.2 hold for an LM p, p cannot compute non-AC^0 functions.*[25,26]

Proof. This is a consequence of the established result that fixed-depth transformers can only compute AC^0 functions (Li et al., 2024; Saunshi et al., 2025; London & Kanade, 2025, *inter alia*). Note that this is similar to the proof in Liu (2025); Liu et al. (2025b) but is simpler due to the focus on discrete predictions directly rather than the continuous modeling of the diffusion process in the latent space. ■

E.1   PROOFS OF RESULTS IN §3.1

Theorem E.1 (PLTs can simulate MDMs)**.**

$$\mathrm{MDM}[T, P] \subseteq \mathrm{PLT}[T, P]. \tag{55}$$

Proof. We can simulate an MDM transformer with a PLT transformer by "composing" the planner and predictor into a single transformer model. This model

---

[24]The augmented attention mechanism additionally downweights positions with $\mathbb{1}\{b_{n'} \geqslant r_{n'}\} = 1$, which can be done by subtracting $B_{\mathbb{F}}$ from the attention score.

[25]That is, p can only implement LMs whose next-symbol logits can be computed by AC^0 circuits (Liu, 2025).

[26]An analogous version of the theorem applies to transformers in TC^0.

(1) computes the planner's contextual representations while passing the input symbol in the residual stream,

(2) computes the planner's decision at each position by simulating the argmax of the planner's output logits as in Lem. D.11,

(3) computes the predictor's contextual representations based on the planner's decision, and

(4) predicts the next symbol at each position by simulating the argmax of the predictor's output logits as in Lem. D.11.

∎

Lemma E.1. *Let $\mathcal{T}$ be a fixed-precision and polynomial-width transformer and $\boldsymbol{H} \in \{0,1\}^{N \times D}$ the residual stream of $\mathcal{T}$ after l layers on some (possibly padded) input string $\boldsymbol{w} \in \Sigma^*$. Then, there exist fixed-precision and polynomial-width transformer layers τ_{dump} and τ_{read} with such that $\tau_{\mathsf{dump}}(\boldsymbol{H}) \in \{0,1\}^{\mathcal{O}(\log N) \times D}$ and*

$$\tau_{\mathsf{read}}(\mathsf{Dec}(\tau_{\mathsf{dump}}(\boldsymbol{H})))_{:N,:} = \boldsymbol{H} \tag{56}$$

for some output matrix $\boldsymbol{E} \in \mathbb{R}^{2 \times D}$.

Proof. The idea of the construction of the layers τ_{dump} and τ_{read} is to store the contents of the residual stream in the padding space and then read it out at the next iteration. To do that, we allocate $N \cdot D$ symbols of additional (masked) padding space in the decoded string. Each of the N length-D blocks corresponds to a position in the input string and each symbol in the block to a dimension in the hidden representation. The layers thus have to be augmented with positional encodings that will allow for the identification of the position in the residual stream and the dimension in the hidden representation. This will suffice for τ_{dump} to write out the contents of the residual stream into the padding space and for the reading layer τ_{read} to read it out again.

More precisely, τ_{dump} and τ_{read} use the following positional encodings:

$$\mathsf{PE}(n, N) \overset{\text{def}}{=} \begin{pmatrix} \mathsf{B}^{\pm}(n) \\ \mathsf{B}^{\pm}(b_n) \\ [\![d_n]\!] \\ \mathbb{1}\{n \leqslant N\} \end{pmatrix} \in \{0,1\}^{\mathcal{O}\log N} \tag{57}$$

to the input of τ_{dump}. In particular, for a masked padding symbol at position n, $b_n \overset{\text{def}}{=} [n - N]_+/D$ and $d_n \overset{\text{def}}{=} [n - N]_+ \bmod D$ correspond to the position in the residual stream and the dimension in the hidden representation that the position will store, respectively. τ_{dump} can then be implemented as follows:

(1) Using $\mathsf{B}^{\pm}(b_n)$ as the query at masked position n and $\mathsf{B}^{\pm}(n')$ as the key at position $n' \leqslant N$, τ_{dump} can individually identify the corresponding position $\mathsf{B}^{\pm}(b_n)$ in the residual stream.

(2) Feeding $\boldsymbol{h}_{\mathsf{B}^{\pm}(b_n)} \in \{0,1\}^D$ together with $[\![d_n]\!]$ as the value at the masked position n into the MLP, τ_{dump} can write the value of the dimension d_n of the residual stream at position $\mathsf{B}^{\pm}(b_n)$ into the padding space by Lem. D.12.

It is then easy to construct the output matrix $\boldsymbol{E}$ as part of the decoding step Dec such that $\mathsf{Dec}(\tau_{\mathsf{dump}}(\boldsymbol{H}))$ decodes the contents of the residual stream. τ_{read} can then be implemented as follows:

(1) Include a transformer layer that ignores the attention mechanism and reads the input string and the decoded residual stream values, passes them through the residual connection, and encodes the values into the embedding space with the position-wise MLP. In particular, combining the information in $[\![d_n]\!]$ with the information in the input symbols, the MLP can convert the one-hot encoding of the dimension d_n into the vector $h_{d_n} \boldsymbol{e}_{d_n}$, where h_{d_n} corresponds to the value of the appropriate dimension of the residual stream at the appropriate position.

(2) Using $\mathsf{B}^{\pm}(n)$ as the query at the input string position $n \leqslant N$ and $\mathsf{B}^{\pm}([n' - N]_+/D)$ as the key, $\boldsymbol{\tau}_{\mathsf{read}}$ can identify all the padding positions that contain the values of the individual dimensions of the residual stream at position n. The positional encodings ensure that the attention scores satisfy (cf. Lem. D.2)

$$\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} = \begin{cases} 0 & \textbf{if } n = [n' - N]_+/D \\ -B_{\mathbb{F}} & \textbf{otherwise} \end{cases} . \tag{58}$$

Exponentiating and normalizing the attention scores, the attention mechanism will then only attend to the positions n' that correspond to the position n in the residual stream. More concretely, the attention mechanism computes

$$s_{n,n'} = \frac{\exp(\boldsymbol{q}_n^\top \boldsymbol{k}_{n'})}{\sum_{n''} \exp(\boldsymbol{q}_n^\top \boldsymbol{k}_{n''})} = \frac{1}{\sum_{n''} \exp(\boldsymbol{q}_n^\top \boldsymbol{k}_{n''})} = \frac{1}{\min(D, B_{\mathbb{F}})} \geqslant \frac{1}{B_{\mathbb{F}}} \tag{59}$$

for all n' that correspond to the position n in the residual stream and $s_{n,n'} = 0$ otherwise. Summing over the values at these positions (which project the constructed vectors $h_{d_n} \boldsymbol{e}_{d_n}$), $\boldsymbol{\tau}_{\mathsf{read}}$ can then reconstruct the value of the residual stream at position n (normalized by $\min(D, B_{\mathbb{F}})$).

(3) Use the position-wise MLP to convert the normalized value of the residual stream at position n into the vector $h_{d_n} \boldsymbol{e}_{d_n}$. This can be done by a ReLU-activated MLP that maps $(-\infty, 0]$ to 0 and $\left[\frac{1}{B_{\mathbb{F}}}, \infty\right)$ to 1 position-wise.

$\blacksquare$

Theorem E.2 (MDMs can simulate PLTs).

$$\mathtt{PLT}[T, P] \subseteq \mathtt{MDM}[T, (N + P)D]. \tag{60}$$

Proof. The proof uses Lem. E.1 to simulate a single iteration of the PLT transformer loop with a single denoising step in the MDM transformer. In particular, by adding $(N + P)D$ padding space, the MDM has enough room to store the residual stream values, allowing it to decode the values at the next iteration. At a high level, the MDM's planner deterministically selects the entire padding space to unmask or resample, and the predictor *(1)* reads the input string or the currently stored residual stream values, *(2)* simulates a single pass of the PLT transformer on the input string $\boldsymbol{w}$ and the current residual stream values and thus computes the updated value of the residual stream, and *(3)* uses Lem. E.1 to write the updated values into the padding space.

More precisely, we can implement the MDM transformer as follows:

(1) The MDM transformer uses $(N + P)D$ masked symbols to store the residual stream values.

(2) The planner deterministically outputs a 1 for positions $n > N$ and 0 for positions $n \leqslant N$.

(3) The predictor predicts the values of the $(N + P)D$ symbols based on the current input string and the residual stream values. It reads the values from the residual stream by making the first N positions attend to the $(N + P)D$ padding positions analogous to the dump-decode-read mechanism from Lem. E.1.

By treating input symbols $\in \Sigma$ separately to the decoded values of the residual stream (which enables the MDM transformer to simulate both the initial as well as the looping blocks of the PLT transformer), the MDM transformer can thus simulate the PLT transformer with TPD padding symbols. $\blacksquare$

Theorem 3.2. *Regular languages are in* $\mathtt{MDM}[\log N, N]$.

Proof. The proof adapts the construction from the proof of Saunshi et al. (2025, Thm. 5.1) to the MDM setting. The key difference lies in adapting the positional encodings to allow for the padded tokens to attend to appropriate positions in the residual stream. In particular, the first $\lceil N/2 \rceil$ positions of the padding space will attend to the N input symbols while the remaining positions in the padding

space will attend to the other padding positions of the residual stream in later unmasking steps. Concretely, defining $\tilde{n} \stackrel{\text{def}}{=} [n - N]_+$, the positional encodings take the form

$$\mathsf{PE}(n, N) \stackrel{\text{def}}{=} \begin{pmatrix} \mathsf{B}^{\pm}(n) \\ \mathsf{B}^{\pm}(\tilde{n}) \\ \mathsf{B}^{\pm}\left([2n - N]_+\right) \\ \mathsf{B}^{\pm}\left([2n - N - 1]_+\right) \\ \mathsf{B}^{\pm}\left([2\tilde{n} - N]_+\right) \\ \mathsf{B}^{\pm}\left([2\tilde{n} - N - 1]_+\right) \\ \mathbb{1}\{n > N\} \\ \mathbb{1}\{\tilde{n} \geqslant N/2\} \end{pmatrix} \in \{0, 1\}^{\mathcal{O}(\log N)}. \tag{61}$$

This information gives padding tokens enough information to either attend to the input symbols ($\mathbb{1}\{\tilde{n} \geqslant N/2\} = 1$) or to the residual stream values, enabling the simulation of the algorithm from Saunshi et al. (2025, Thm. 5.1). ∎

E.2 PROOFS OF RESULTS IN §3.2

Theorem 3.3 (MDMs can simulate pCoT transformers)**.**

$$\mathsf{pCoT}[T, P] \subseteq \mathsf{MDM}[T, P + (N + P)^2] \tag{6}$$

Proof. Let $P = TP'$. The idea of the simulation is straightforward: The planner first determines the next P' symbols to unmask. Then, the predictor determines the symbols at those positions by simulating the behavior of the pCoT transformer. This is trivial if the MDM is causally masked like the pCoT transformer. However, if the MDM is not causally masked, the planner must take additional steps to ensure that the prediction is equivalent to the pCoT transformer.

Concretely, the MDM simulates the pCoT transformer on the input string $\boldsymbol{w}$ of length N as follows:

1. We first note that the pCoT transformer predicts the next P' symbols at every timestep t based on the input string $\boldsymbol{w}_{\leqslant N+(t-1)P'} \underbrace{\square \cdots \square}_{(T-t)P'}$ rather than $\boldsymbol{w}_{\leqslant N+(t-1)P'}$. This is because, by Lem. D.6, the pCoT transformer can ignore all symbols containing the padding symbol and thus produce equivalent predictions at every step t. This will help us make use of the same padding space at every step of the simulation.

2. We assume that the final output of the MDM will be stored in the first P positions of the padding space. It will be filled in T generation steps, where at each step $t \in [T]$, a new block of P' symbols will be predicted. In particular, by Lem. D.9, the planner can select the next P' positions to unmask at time t by including the values $\mathsf{B}^{\pm}(\lceil [n - N]_+/P' \rceil)$ and $\mathsf{B}^{\pm}(\lceil [n - N]_+/P' \rceil - 1)$ in the positional encodings.

3. The predictor then uses an initial layer to copy the input string $\boldsymbol{w}_{\leqslant N+(t-1)P'} \underbrace{\square \cdots \square}_{(T-t)P'}$ into the residual stream of the first $N + P$ positions of the padding space.

4. The predictor then uses the $(N + P)^2$ padding positions to predict the next P' symbols predicted by the pCoT transformer. These are written to the positions chosen to be unmasked by the planner.

∎

Theorem 3.4 (pCoT transformers can simulate MDMs)**.**

$$\mathsf{MDM}[T, P] \subseteq \mathsf{pCoT}[T, LT(P + N)], \tag{7}$$

where L is the number of layers in the transformer implementing the MDM.

Proof. For simplicity, we assume that the input to the pCoT transformer is padded by P symbols. Intuitively, the pCoT transformer simulates an L-layer MDM transformer on the input string $\boldsymbol{w}$ of length N by simulating each MDM generation step with additional padding to account for causal masking. Whereas the MDM transformer "overwrites" its previous input and bases its predictions at time step t on the current version of the unmasked input, the pCoT transformer bases its predictions of the P symbols on the entire string of $(t-1) \cdot P$ symbols generated so far. For correct simulation, the pCoT transformer therefore has to ignore all the symbols not generated at the previous time step, which will be ensured by appropriate positional encodings. The pCoT transformer can then predict the next $N + P$ symbols based on the current input string and the previously predicted symbols, simulating the behavior of the MDM transformer on that input. However, to predict $N + P$ symbols, the pCoT transformer uses $P' \overset{\text{def}}{=} L(N + P)$ padding space at each step to account for the unmasked nature of the MDM transformer (cf. Lem. D.14).

More concretely, the simulation happens as follows.

(1) The pCoT transformer uses additional positional encodings with the information about $\lfloor [n - N]_+/T \rfloor$, $[n - N]_+ \mod P'$, $\lfloor [n - N]_+/L \rfloor$, and $[n - N]_+ \mod P$. These positional encodings allow the pCoT transformer to identify *(1)* the previous block of P' predicted symbols, *(2)* the last P symbols within that block (which is where the actual predictions of the previous step will be stored), and *(3)* the current position in the block with Lem. D.7.

(2) The pCoT transformer first uses an initial layer to copy the output of the previous generation step (captured in the previous $N + P$ positions) into the next $N + P$ positions of the padding space (this is where we use the assumption that the input to the pCoT transformer is padded—if that is not the case, a more complicated construction could specifically handle the initial step of the generation where only the initial input string would be copied).

(3) The pCoT transformer can then predict the next P symbols by simulating the behavior of the composed MDM planner and predictor as in Thm. E.1.

∎