Fast and Expressive Multi-Byte Prediction with Probabilistic Circuits

Anonymous Authors[1]

Abstract

Multi-token prediction (MTP) is a prominent strategy to significantly speed up generation in large language models (LLMs), especially in byte-level LLMs, which are tokeniser-free but prohibitively slow. However, existing MTP methods often sacrifice expressiveness by assuming *independence* between future tokens. In this work, we investigate the trade-off between expressiveness and latency in MTP within the framework of probabilistic circuits (PCs). Our framework, named MTPC, allows one to explore different ways to encode the *joint* distributions over future tokens by selecting different circuit architectures, generalising classical models such as (hierarchical) mixture models, hidden Markov models, and tensor networks. We show the efficacy of MTPC by retrofitting existing byte-level LLMs, such as EvaByte, and byte-fied subword models, such as Llama3.2 3B. Our experiments show that, when combined with speculative decoding, MTPC substantially speeds up generation compared to MTP with independence assumptions, while guaranteeing to retain the performance of the original verifier LLM. We also rigorously study the optimal trade-off between expressiveness and latency when exploring the possible parameterisations of MTPC, such as PC architectures and partial layer sharing between the verifier and draft LLMs.

1. Introduction

Autoregressive (AR) large language models (LLMs) can only perform single-token prediction (STP) as they generate one token at a time, incurring significantly high latency, energy demand, and deployment costs. This problem affects subword models, but is dramatically exacerbated in byte-level ones (Minixhofer et al., 2025a; Wang et al., 2024; Yu et al., 2023, *inter alia*) as many more decoding steps are required to generate a text of the same length. Among possible alternatives to speed up generation (Ankner et al., 2024; DeepSeek-AI et al., 2024; Nawrot et al., 2023; Pagnoni et al., 2024; Łańcucki et al., 2025), multi-token prediction (MTP) stands out as it promises to predict a window of multiple tokens *all at once*, may they be subwords (Gloeckle et al., 2024; Cai et al., 2024) or bytes (Gloeckle et al., 2024; Zheng et al., 2025). As such, MTP LLMs can achieve a significantly higher throughput than STP ones, as they decrease the number of passes through the LLM to generate the same text. While recent works on MTP focused on subword models (Cai et al., 2024; Li et al., 2024), MTP can improve throughput the most in byte-level models (Pagnoni et al., 2024; Wang et al., 2024) as for their increased computational overhead. If one could model large sequences of future bytes, byte-level LLMs could become more mainstream as they already obviate many limitations of subword tokenizers, such as uneven efficiency (Ahia et al., 2023; Dagan et al., 2024), lack of interoperability (Minixhofer et al., 2025b), and vulnerabilities (Rumbelow & Watkins, 2023; Land & Bartolo, 2024; Geiping et al., 2024).

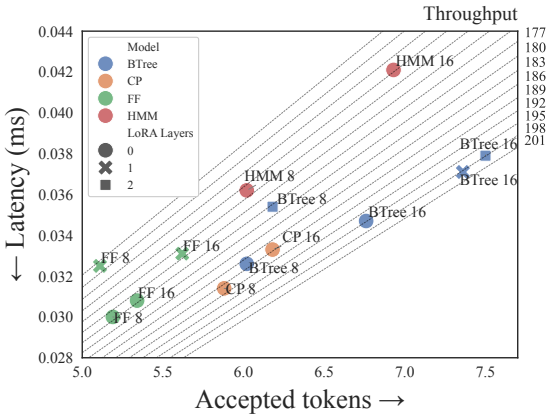


Figure 1. **MTPC allows us to trade-off efficiency (latency) and expressiveness (token acceptance) with different MTP designs** by choosing 1) the probabilistic circuit (PC) architecture (FF, CP, HMM, BTree); 2) the number of layers shared between draft and verifier models in self-speculative sampling. Dotted lines indicate iso-throughput (bytes/sec) regions, highlighting designs such as BTree for 16 bytes and 1 LoRA layer that achieve the best throughput for our retrofitted EvaByte-MTP model (Sec. 4.2).

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

However, modelling the joint distribution over many future bytes as tokens in a window is challenging, as it requires balancing *expressiveness*, i.e., representing all the dependencies between bytes, and *efficiency*, i.e., minimising latency. Existing MTP approaches favour the latter by making an unrealistic assumption: namely, considering all future tokens to be independent (Zheng et al., 2025; Cai et al., 2024; Gloeckle et al., 2024). This clearly comes at the expense of expressiveness (Ankner et al., 2024; Wertheimer et al., 2024), as the choice of a byte for a position within the window cannot influence the probability of the others.

For example, consider the prompt: "*Name a capital of South Africa*", where *Cape Town* and *Pretoria* are equally likely completions. A byte-level MTP model with independence assumptions over an 8-token window could return *Cretoria* as an argmax, because replacing *P* with *C* cannot change the probability of other tokens. More concerningly, a number of "byte-salad" continuations, such as *Crptoria*, *Crpt ria* and *Crpt roa*, can also have high probability, despite having almost zero probability under the STP model. Most importantly, the number of these erroneous continuations grows exponentially w.r.t. the MTP window size, making the independence assumption problematic. Recently, Basharin et al. (2025) introduced dependencies into MTP using a mixture over the future token probabilities, but a single mixture can only add limited expressiveness. Crucially, understanding how to increase expressiveness while optimally trading off efficiency in a systematic way is still an open question.

In this paper, we fill this gap by proposing an MTP framework based on probabilistic circuits (PCs; Choi et al., 2020; Vergari et al., 2021), which we name MTPC. MTPC uses PCs to parameterise the joint distribution over future tokens into tractable computational graphs that can encode hierarchical mixture models. As such, MTPC offers a way to systematically navigate the spectrum of MTP architectural variants, encompassing fully factorised models (Zheng et al., 2025; Cai et al., 2024; Gloeckle et al., 2024) and shallow mixtures (Basharin et al., 2025) but also more expressive parameterisations: hidden Markov models (HMMs) and binary tree factorisations (BTrees), which are novel for MTP.

Moreover, in contrast to previous work on MTP (Zheng et al., 2025; Cai et al., 2024), MTPC guarantees we match the quality of an AR LLM via speculative decoding (Leviathan et al., 2022; Chen et al., 2023; Stern et al., 2018; Xia et al., 2024)—exactly for greedy decoding or in expectation for sampling—showing that the throughput sacrificed for the guarantee is not as large as alluded to previously. We do so by sharing the LLM backbone for the draft and verifier models for different numbers of layers, highlighting how this creates *a second dimension to trade-off expressiveness* (as hidden representations between draft and verifier are allowed to differ) *and latency* (as each non-shared layer requires separate forward passes). We illustrate the two trade-offs at the core of MTPC in Fig. 1.

In summary, we make the following contributions: **C1)** we introduce MTPC, a fast MTP framework based on PCs that overcomes the independence assumptions of previous work and generalises tensor decomposition methods (Basharin et al., 2025); **C2)** we rigorously identify trade-offs between acceptance rates in speculative decoding and latency of generation, based on different choices of PC architectures and partial layer sharing; **C3)** we empirically demonstrate the effectiveness of MTPC by repurposing two byte-level LLMs, EvaByte (Zheng et al., 2025) and Llama 3.2 3B Byte (Minixhofer et al., 2025a), into our framework. By doing so we make byte-level LLMs (Pagnoni et al., 2024; Wang et al., 2024), which obviate the limitations of sub-word tokenisers—including uneven efficiency (Ahia et al., 2023; Dagan et al., 2024), lack of interoperability (Minixhofer et al., 2025b), and vulnerabilities (Rumbelow & Watkins, 2023; Land & Bartolo, 2024; Geiping et al., 2024; Salesky et al., 2021)—at the cost of significantly slowing down generation. We find that MTPC substantially increases the throughput of EvaByte by $5.15\times$ and that of Llama by $2.24\times$ with respect to AR generation and both by $1.17\times$ with respect to MTP with independence assumptions, by enabling scaling to large context windows up to 16 bytes.

2. Speeding up Generation with MTP and Speculative Decoding

As MTP and speculative decoding are two main ingredients in MTPC to speed up generation while guaranteeing the quality of STP, we introduce them next.[1]

MTP. A classical STP LLM encodes a distribution over sequences of tokens $\{\mathbf{x}_t\}$ defined over a vocabulary $\mathcal{V}$ as $\prod_t p(x_{t+1} \mid \mathbf{x}_{\leq t})$, where $\mathbf{x}_{\leq t}$ is the context, *i.e.* the observed tokens at timestep t. MTP (Gloeckle et al., 2024) aims to extend an STP LLM that predicts a single token at a time through $p(x_{t+1} \mid \mathbf{x}_{\leq t})$, to an MTP model, $q_{\boldsymbol{\theta}}$, that models the *joint* probability of a window of n future tokens and generates them *simultaneously*, i.e.,

$$q_{\boldsymbol{\theta}}(x_{t+1}, x_{t+2}, \ldots, x_{t+n} \mid \mathbf{x}_{\leq t}). \tag{1}$$

where $\boldsymbol{\theta}$ denotes a given parameterisation for the joint.[2] The first dimension to trade-off expressiveness and efficiency in MTP pertains to compactly representing $q_{\boldsymbol{\theta}}$. Unlike for $p(x_{t+1} \mid \mathbf{x}_{\leq t})$, we would need to store more than a vector of logits $\mathbf{a} \in \mathbb{R}^v$ of a single univariate categorical distribution for a vocabulary size $v = |\mathcal{V}|$ for every timestep t. The most expressive, but least efficient way to do so, would be to

[1] We adapt notation from the tensor and circuit literature (Loconte et al., 2025a), see also Appendix A.

[2] We drop the index t from $\boldsymbol{\theta}_t$ for readability when not needed.

store an n-dimensional tensor $\mathcal{A} \in \mathbb{R}^{v^{(1)} \times \ldots \times v^{(n)}}$ of logits having v^n entries, but this scales exponentially in n. Next, we review past attempts to avoid storing $\mathcal{A}$ explicitly.

Fully factorised. The most common way to boost efficiency is to assume all n future tokens are independent (Zheng et al., 2025; Cai et al., 2024; Gloeckle et al., 2024), that is, $q_{\boldsymbol{\theta}}$ factorizes as

$$\prod_{i=1}^{n} q_{\boldsymbol{\theta}}(x_{t+i} \mid \mathbf{x}_{\leq t}). \tag{FF}$$

This comes with the benefit that one needs to store only n v-dimensional vectors of probabilities to represent the joint distribution in Eq. (1). At the same time, as already discussed in the introduction, this severely limits model expressiveness (Ankner et al., 2024; Wertheimer et al., 2024).

Canonical polyadic (CP) factorisation. Dependencies between future tokens can be recovered by introducing explicit latent variables (Lee et al., 2018). To this end, Basharin et al. (2025) propose to factorise Eq. (1) via an r-rank CP decomposition. A CP decomposition introduces one discrete latent variable, z, that encodes a mixture of r fully-factorised components, rewriting Eq. (1) as:

$$\sum_{z=1}^{r} q(z \mid \mathbf{x}_{\leq t}) \prod_{i=1}^{n} q_{\boldsymbol{\theta}}(x_{t+i} \mid z, \mathbf{x}_{\leq t}) \tag{CP}$$

where the $q(z \mid \mathbf{x}_{\leq t})$ are the mixture coefficients.[3] Before showing how we can generalize both FF and CP MTP with probabilistic circuits, we review how to ensure MTP models match the quality of a given STP model.

Speculative decoding (Stern et al., 2018; Leviathan et al., 2022; Chen et al., 2023; Xia et al., 2024) can be combined with MTP to speed up generation while guaranteeing no loss in generation quality. Given a target STP LLM that we wish to accelerate, speculative decoding involves two steps: 1) *drafting*, where a cheaper MTP draft model generates n future tokens, and 2) *verification*, where the target STP model accepts or rejects the generated tokens in parallel according to a pre-defined consistency criterion. The closer the distributions of the draft and verifier are, the more often 'speculated' tokens are accepted, speeding up generation. With speculative decoding, we can quantify the trade-off between expressiveness and efficiency in MTP models as their *throughput*, i.e.

$$\text{throughput (tok/s)} = \frac{\text{acceptance rate } \textit{(toks per eval)}}{\text{latency } \textit{(secs per eval)}} \tag{2}$$

where acceptance rates are a function of the total variation distance between the two distributions (Leviathan et al., 2022; Sun et al., 2023) and latency measures how computationally expensive an MTP model is during generation. While previous work, such as Basharin et al. (2025), focused only on measuring acceptance rates, we highlight how both sides of the ratio in Eq. (2) are important, as they create a spectrum. MTPCs provide a systematic way to navigate such a spectrum (see Fig. 1).

[3] Basharin et al. (2025) calls CP a mixture of experts (MoE), but we note this is incorrect as the weights ω_j do not depend on future tokens, but only on past ones. Moreover, while they argue that training CP is challenging and requires insights from the MoE literature, we can train them as well as deeper mixture variants easily without MoE-tailored losses (see Sec. 4).

3. Probabilistic Circuits for MTP

The idea behind MTPCs is to further decompose the joint distribution in Eq. (1) into a deep computational graph encoding a hierarchical mixture model, called a *probabilistic circuit* (Secs. 3.1 and 3.2), and to parameterise it with LLM embeddings (Sec. 3.3).

3.1. Probabilistic Circuits

A ***circuit*** (Darwiche, 2003; Choi et al., 2020; Vergari et al., 2021), c, is a parameterised computational graph[4] over variables $\mathbf{X}$ encoding a function, $c(\mathbf{X})$, and comprises three kinds of computational units: *input*, *product*, and *sum* units. Each product or sum unit n receives the outputs of other units as inputs, denoted with the set $\mathsf{in}(n)$. Each unit n encodes a function, c_n, defined as: (i) $c_n(\mathsf{sc}(n); \phi)$ if n is an input unit, where c_n is a function parameterised by ϕ over variables $\mathsf{sc}(n) \subseteq \mathbf{X}$, called its *scope*; (ii) $\prod_{j \in \mathsf{in}(n)} c_j(\mathsf{sc}(j))$ if n is a product unit; and (iii) $\sum_{j \in \mathsf{in}(n)} \omega_j c_j(\mathsf{sc}(j))$ if n is a sum unit, with $\omega_j \in \mathbb{R}$ denoting the sum parameters. The scope of a product or sum unit n is the union of the scopes of its inputs, i.e., $\mathsf{sc}(n) = \bigcup_{j \in \mathsf{in}(n)} \mathsf{sc}(j)$. Fig. 2 shows examples of circuits, where units of the same scope are grouped into (coloured) layers that can be easily parallelised on a GPU (Mari et al., 2023; Loconte et al., 2025a).

For MTPCs, we use ***probabilistic circuits*** (PCs), i.e., circuits modelling a joint distribution over random variables, in our case tokens. PCs encode Eq. (1) as

$$q_{\boldsymbol{\theta}}(x_{t+1}, \ldots, x_{t+n} \mid \mathbf{x}_{\leq t}) = Z_{\boldsymbol{\theta}_t}^{-1}\, c(x_{t+1}, \ldots, x_{t+n}; \boldsymbol{\theta}_t) \tag{3}$$

where $\boldsymbol{\theta}_t = \{\boldsymbol{\omega}_t, \boldsymbol{\phi}_t\}$ denote the set of circuit parameters, i.e., all sum unit parameters $\boldsymbol{\omega}_t$ and input unit parameterisations $\boldsymbol{\phi}_t$ which depend on the context $\mathbf{x}_{\leq t}$; and $Z_{\boldsymbol{\theta}_t}$ denotes the partition function of c, i.e., $Z_{\boldsymbol{\theta}_t} = \sum_{x_{t+1}, \ldots, x_{t+n} \in \mathcal{V}^n} c(x_{t+1}, \ldots, x_{t+n}; \boldsymbol{\theta}_t)$. Note that the PC architectures we are interested in are already normalised or always allow computing the partition function in a single feed-forward step (see Choi et al. (2020) and Appendix B.1). At the same time, we can easily sample from PCs in a single feedforward pass, as discussed in Appendix B.2. Crucially, within the framework of PCs, we can recover the FF and CP parameterisations for MTP and several other architectures

[4] In Fig. 2, the directionality of the edges is removed for readability, but it is assumed to be from inputs to outputs.

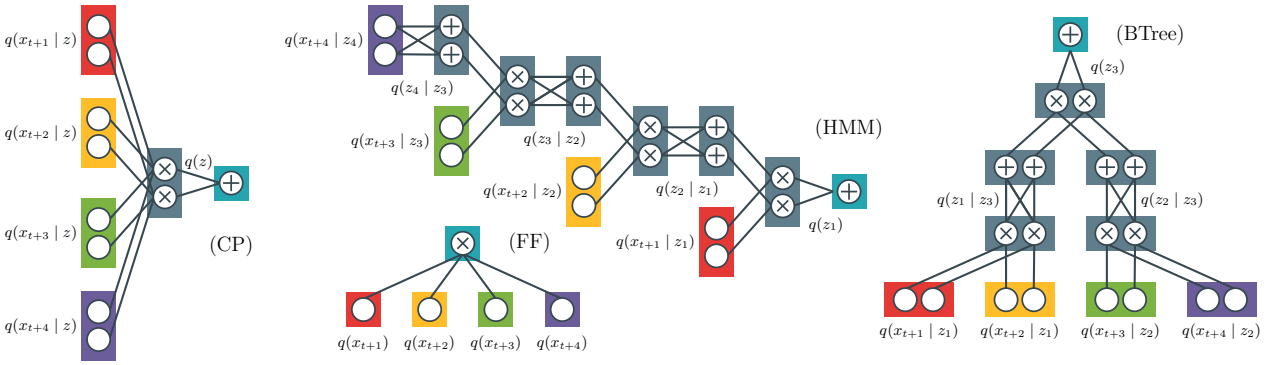


Figure 2. **PCs allow for modelling a spectrum of dependency structures** over sequences of tokens, as shown for the known FF and CP and the novel HMM and BTree MTP variants. Input units are grouped in coloured layers, one for each token, while sum and product layers encoding (hierarchies of) latent variable distributions are in grey. The output unit of each circuit (in blue) computes $q_{\boldsymbol{\theta}}(x_{t+1}, \ldots, x_{t+n} \mid \mathbf{x}_{\leq t})$. In the figure we omit the dependency on the context $\mathbf{x}_{\leq t}$ for readability.

that generalise tensor factorisations (Loconte et al., 2025a), each offering a different expressiveness-efficiency trade-off. We do so while abstracting away from each model's original formulation and obtain a unified way to parameterise MTP LLMs, as discussed next.

3.2. PC Architectures for MTP

MTPC-FF. Representing the commonly used FF MTP parameterisation as a PC is simple: we introduce n input units, each parameterised by ϕ_i, its corresponding token probabilities, and connect them all to a single product unit, as shown in Fig. 2 for a distribution over $n = 4$ tokens.

MTPC-CP. Similarly, we can easily encode a CP factorisation in a *shallow* PC by i) introducing r input units for each token (each parameterised by their own probabilities ϕ_{ij}), then ii) multiplying them to retrieve the r factorised mixture components, which we then iii) aggregate in a sum unit with weights $\omega_j = q(z_j \mid \mathbf{x}_{\leq t})$ (see also Proposition 1 in Loconte et al. (2025a)). Fig. 2 shows this construction for $n = 4$ and $r = 2$. This basic construction suggests that we can create deeper architectures by interleaving sum and product layers, while overparameterising each layer by increasing the number of units r in it. Furthermore, by implementing CP as a PC unlocks a faster sampling routine (Appendix B.2) than the one used in Basharin et al. (2025).

MTPC-HMM. As a further example of the expressiveness increase we get by generalising our approach to deeper PCs, we introduce a factorisation that realises a hidden Markov model (HMM), which better captures distant dependencies in the sequence by introducing *a sequence of latent variables*, in contrast to the single one in CP. More precisely, we define an HMM with r hidden states and truncate its prediction window to n steps into the future. We resort to an inhomogeneous HMM, *i.e.*, we use time-dependent transition matrices, as it is more expressive and worked better in our experiments (Appendix E). This simplifies Eq. (1) into:

$$\sum_{z_1=1}^{r} \cdots \sum_{z_n=1}^{r} q(z_1 \mid \mathbf{x}_{\leq t}) q_{\boldsymbol{\theta}}(x_{t+1} \mid z_1, \mathbf{x}_{\leq t}) \times \prod_{i=2}^{n} q(z_i \mid z_{i-1}, \mathbf{x}_{\leq t}) q_{\boldsymbol{\theta}}(x_{t+i} \mid z_i, \mathbf{x}_{\leq t}). \quad \text{(HMM)}$$

Fig. 2 illustrates the HMM parameterisation above represented as a circuit, comprising $n = 4$ stacked pairs of sum and product layers, where the parameters $\boldsymbol{\omega}_i$ of the former are the transition probabilities $q(z_i \mid z_{i-1}, \mathbf{x}_{\leq t})$. Similarly to CP, we can increase r to overparameterise the circuit with more input units per token and sum units overall, and hence increase expressiveness.

MTPC-BTREE. One drawback of the HMM parameterisation is the asymmetry of its computational graph, which i) provides fewer latent variables for the early tokens, and ii) increases latency when predicting the last tokens due to its autoregressive token dependencies. To solve this, we build a PC whose structure resembles that of a binary tree (BTree), effectively encoding a *hierarchy of latent variables* or a tree tensor factorisation (Grasedyck, 2010; Cheng et al., 2019; Loconte et al., 2025a). This is done recursively: at each step h of the hierarchy, given n tokens, and a parent latent variable Z_l, we split the tokens into two sub-sequences $(x_{t+1}, \ldots, x_{t+\lfloor n/2 \rfloor - 1})$ and $(x_{t+\lfloor n/2 \rfloor}, \ldots, x_{t+n})$, then factorise Eq. (1) as the mixture:

$$\sum_{z_h=1}^{r} q(z_h \mid z_l, \mathbf{x}_{\leq t}) \times q_{\boldsymbol{\theta}}(x_{t+1}, \ldots, x_{t+\lfloor n/2 \rfloor - 1} \mid z_h, z_l, \mathbf{x}_{\leq t}) \times q_{\boldsymbol{\theta}}(x_{t+\lfloor n/2 \rfloor}, \ldots, x_{t+n} \mid z_h, z_l, \mathbf{x}_{\leq t}) \quad \text{(BTree)}$$

which corresponds to creating a sum unit whose weights are $q(z_h \mid z_l, \mathbf{x}_{\leq t})$ followed by products. We repeat the process while caching intermediate units until we reach the base case for $n = 1$, for which we create a layer of input units for the corresponding token. Fig. 2 illustrates the BTree circuit built in this way. Our experiments (Sec. 4.2) show that the

BTree parameterisation obtains the optimal throughput by lowering the latency of the HMM, as it samples more latent variables and tokens in parallel, while achieving similar acceptance rates. In addition, our experiments show that the BTree parameterisation improves throughput on larger MTP window sizes as the balanced tree structure enables parallel sampling of tokens and latent variables.

3.3. Parameterising PCs with LLMs

Parameterising MTPCs requires two functions: an LLM that maps the context $\mathbf{x}_{<t} \in \mathcal{V}^t$ into contextual features, and a neural network head that maps the contextual features to the parameters of the circuit $\boldsymbol{\theta}_t$, realising a *neural conditional circuit* (Shao et al., 2020; 2022; Ahmed et al., 2022). To extract the contextual features $\mathbf{e}_t \in \mathbb{R}^d$, we use $\mathbf{e}_t = \text{LLM}_{\text{LoRA}(k)}(\mathbf{x}_{<t})$ where $\text{LLM}_{\text{LoRA}(k)} : \mathcal{V}^t \to \mathbb{R}^d$ is the STP backbone with LoRA (Hu et al., 2022) applied to the last $k \geq 0$ layers. As we will discuss in Sec. 4.4, the number of LoRA layers can impact throughput significantly. Given $\mathbf{e}_t$, we realise Eq. (3) by computing $\boldsymbol{\theta}_t = g_c(\mathbf{e}_t)$, where g_c is a neural network head that outputs both the input unit parameters, $\boldsymbol{\phi}_t$, and the sum unit parameters, $\boldsymbol{\omega}_t$ (Sec. 3.1). Note that our parameterisation in MTPCs allows us to abstract from the actual structure of the circuit (i.e., FF, CP, HMM or BTree) and just focus on these two sets of tensorised parameters, as we discuss next.

Input unit distributions. All MTPCs produce joint distributions over token windows by combining categorical distributions over individual tokens (Fig. 2). We follow EvaByte (Zheng et al., 2025) and learn n separate unembedding layers, one per window position. For models with mixture coefficients, we also learn one unembedding layer per mixture coefficient.[5] As such, instead of a single unembedding matrix mapping $\mathbb{R}^d \to \mathbb{R}^v$, we have an unembedding tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{n \times r \times v \times d}$, and compute the input distributions with the usual unembedding operation followed by softmax, i.e., $\boldsymbol{\phi}_{tij} = \text{softmax}(\boldsymbol{\mathcal{W}}_{ij}\mathbf{e}_t)$, where i and j index the position in the MTP window and the rank r.

Sum unit parameters. Instead of mapping embeddings to the vocabulary via $\boldsymbol{\mathcal{W}}$, we map to the rank of the sum unit via $\boldsymbol{\mathcal{R}} \in \mathbb{R}^{z \times r \times d}$, where z is the number of sum units, r is its rank, and d the dimensionality of $\mathbf{e}_t$. We compute $\boldsymbol{\omega}_{ti} = \text{softmax}(\boldsymbol{\mathcal{R}}_i\mathbf{e}_t)$, where i indexes the sum unit.

3.4. Speculative Decoding with MTPC

For MTPCs, we design an architecture that is *self-drafting* (Zhang et al., 2024b; Cai et al., 2024), i.e. where the draft and verifier models share the same LLM backbone. We use an MTP head (Cai et al., 2024; Ankner et al., 2024) augmented with our circuits to efficiently sample a draft, and an autoregressive STP head as the verifier. Optionally, we also keep a few final transformer layers separate in the two models by fine-tuning LoRA adapters for the draft model.

Unlike previous self-drafting MTP works (Cai et al., 2024; Ankner et al., 2024), we guarantee that the generated tokens are exactly the same (*greedy speculative decoding* (Stern et al., 2018)) or the same in expectation (*speculative sampling*) as those the autoregressive LLM would generate using *speculative decoding* (Leviathan et al., 2022; Chen et al., 2023), *i.e.*, we only generate the subset of drafted tokens accepted by our verifier. To keep latency low, we make only a single LLM call per speculative decoding cycle by re-using the LLM backbone state computed by the verifier for the draft model, where possible. We do this by modifying the speculative decoding algorithm slightly, as detailed in Alg. 2. The GPU memory overhead of speculative decoding with our most expressive PCs is only 10-15% more than MTPC-FF, see Fig. 5. Next, we report results for MTPC both for speculative sampling and greedy speculative decoding.

4. Retrofitting Byte-Level LLMs with MTPCs

We evaluate MTPC on the challenging tasks of speeding up byte-level LLMs. We speed-up two models, EvaByte 6.5B, which already has MTP capabilities, and Llama 3.2 3B (Byte), which does not and is a byte-fied version of the popular subword-level LLM. Detailed parameters of both models can be found in Appendix C. We implement our MTPCs variants in the `cirkit` library (The april Lab, 2024) and provide it in our supplementary materials.

EvaByte 6.5B. EvaByte (Zheng et al., 2025) is an open source, publicly available byte-level model that obtains results that are competitive to subword-level LLMs on benchmarks (Zheng et al., 2025), see Appendix C.1. Importantly, EvaByte has been pre-trained as an MTP model with a prediction window of $n = 8$ bytes. As a result, it can already speed-up generation very effectively with greedy speculative decoding. However, its acceptance rate with speculative sampling is hampered by the unrealistic independence assumptions which we relax with MTPC. Moreover, we will see that EvaByte's pretraining enables us to successfully extend prediction to $n = 16$ tokens. Thus, we retrofit EvaByte-SFT (Zheng et al., 2025), which has been instruction fine-tuned on a data mix of Tülu 3 (Lambert et al., 2025), OpenCoder (Huang et al., 2025) stages one and two, and OpenHermes 2.5. We note that EvaByte's solid performance on benchmarks is obtained via Medusa-style lossy speculative decoding with the MTP head, which in the case of sampling comes with a loss in quality compared to EvaByte-STP ($n = 1$). We therefore set EvaByte-STP as the target model for speculative decoding to *accelerate generation without sacrificing generation quality*.

[5] This is efficient even for PCs with high rank due to the small vocabulary size of byte-level LLMs.

Llama 3.2 3B (Byte). We also retrofit a Llama3.2 3B model (Grattafiori et al., 2024) to show that our findings generalise. Since we focus on byte-level LLMs, we retrofit the byte-level version that has been distilled from the subword-level model in Minixhofer et al. (2025b) while retaining most of Llama's downstream performance.[6] The byte-fied Llama3.2 3B model was fine-tuned on Tülu 3 with a context length of 2048. As opposed to EvaByte, the byte-fied version of Llama was not pretrained for multi-token prediction. As such, the boost we can get from MTPC is smaller than that for EvaByte, as it is more challenging to adapt it to long windows with LoRA adapters only on the last few layers.

Draft models. Each draft model comprises a backbone and an MTP head. We introduce two axes to control the draft model's expressiveness: i) more expressive circuit heads and ii) more LoRA layers in the draft backbone. For i) we combine the target's model backbone with one of our MTPCs heads, including our CP implementation and novel HMM and BTree heads to relax the independence assumptions of the FF model and increase expressiveness. We note that MTPC-CP with $r = 1$ is equivalent to MTPC-FF, as can be seen from Eq. (CP). For ii) we decouple the draft and target backbone by adding LoRA adapters to the last 1, 2 or 4 layers of the draft's backbone.

4.1. Training

We train 55 draft models in total. We improve throughput by making our MTP model's distribution as similar as possible to the target's in the simplest way: we instruction fine-tune our MTP models on a similar data mix to that used for instruction fine-tuning the target.

Training data. We fine-tune on the Tülu 3 SFT mix dataset (Lambert et al., 2025) which contains 939,344 examples of user/assistant interactions on 18 tasks. We split the Tülu 3 dataset into training and validation and evaluate throughput on the unseen validation examples. We make sure all tasks are sampled by shuffling the training data before splitting. To make training possible on 2×80 Gb GPUs, we limit the context length to 8192 bytes and filter out 34,067 examples which are longer. We split the remaining 905,277 examples into 99% train and 1% validation.

Initialisation. For EvaByte, we initialise our MTP heads from EvaByte-SFT in a way that guarantees that our EvaByte-MTP-CP is equivalent to EvaByte-MTP. This guarantees that we leverage previous training: all models start from the same loss and we smoothly move in parameter space from EvaByte-MTP to our more expressive EvaByte-MTP-CP, EvaByte-MTP-HMM and EvaByte-MTP-BTree.

Loss. We train our MTP models on the packed train split of Tülu 3 with a batch size of 256 sequences, or $\approx$ 2m tokens, which is what EvaByte used. We first train our MTP heads for 1 epoch (Sec. 4.3). Then we load the models and continue training for an additional epoch with LoRA (Sec. 4.4). We apply the target model's chat template and only train on the assistant's answers. We use overlapping prediction windows, as we need to be able to begin speculative decoding from any position during generation. We minimise the negative log-likelihood of the observed assistant outputs, see Eq. (4), where N is the number of training sequences and L is the sequence length for each token in the window.[7]

$$\mathcal{L}_j = -\frac{1}{N}\sum_{i=1}^{N}\frac{1}{V_{i,j}}\sum_{t=1}^{L}\log p_\theta(x_{t+j}^{(i)} \mid \mathbf{x}_{<t+j}^{(i)}) \quad (4)$$

This involves locally normalising the loss by the number of valid tokens for example i and output j in the MTP window, $V_{i,j}$. As in Cai et al. (2024), we apply exponential discounting to the loss $\mathcal{L}_j$ of token j in the MTP window, *i.e.*, we minimize $\mathcal{L} = \sum_{j=1}^{n}\gamma^{j-1}\mathcal{L}_j$.[8] We use Adam (Kingma & Ba, 2015) with a fixed learning rate of 3×10^{-4}.

4.2. Metrics

We speed up LLM generation with speculative decoding by balancing **speed** and **expressiveness**. We measure speed using **mean latency** (μ_{lat}) and expressiveness via the **mean acceptance rate** (μ_{acc}; Li et al., 2024), as defined below. Our goal is to increase **throughput**. We obtain a relative throughput speed-up of one method over another by measuring their **wall-time speedup ratio** (Li et al., 2024; Cai et al., 2024). We use a batch size of 1 for all evaluations. Metrics for our main experiments are computed on the server-grade NVIDIA L40S GPU with complete results in Appendix I. We also run experiments on the desktop-grade NVIDIA RTX 3090 in Appendix J. We detail the metrics below.

Mean Latency μ_{lat} is the average time of each speculative decoding step, *i.e.*, the time needed for the draft model to generate a candidate sequence and the verifier to choose which tokens to accept. μ_{lat} is higher for less efficient LLMs and MTP heads, and lower for more powerful GPUs.

Mean acceptance rate μ_{acc} is the percentage of drafted tokens that are accepted by the target model. More expressive draft models will have higher acceptance rate as they will better approximate the target distribution. μ_{acc}depends on the size of the MTP window, n, as we have $\mu_{\text{acc}} \in [0, n]$. **Mean throughput** $\mu_{\text{tok/s}}$ is the ratio μ_{acc} / μ_{lat} (see Eq. (2)). **Wall-time speed-up ratio** is the relative speed-up of a proposed model compared to a baseline model, measured as the ratio of their throughputs. As baselines, we use autoregressive generation from the STP target model and MTP with independence assumptions.

[6] http://hf.co/benjamin/Llama3-2-3B-IT-Byte

[7] Our loss over overlapping windows is a composite log-likelihood (Varin et al., 2011).

[8] We set $\gamma = 0.9$ instead of $\gamma = 0.8$ in the case of $n > 8$.

Table 1. **Increasing the mixture components (r) increases the throughput** ($\mu_{\text{tok/s}}$) as seen for MTPC-CP ($n = 8$) over our baseline, EvaByte-MTP (FF) (in gray) where we report the mean $\pm$ std over three sets of 250 prompts of Tülu3 on an L40S GPU. MTPC-CP increases throughput: it has a larger acceptance rate ($\checkmark\mu_{\text{acc}}$) while latency ($\mu_{\text{lat}}$) increases slowly in r.

		$\mu_{\text{lat}} \downarrow$	$\checkmark\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\max_{\text{tok/s}}$
FF	1	**0.0299**$\pm$0.0003	5.19$\pm$0.05	176.3$\pm$0.9	297.55
	8	0.0314$\pm$0.0005	5.67$\pm$0.05	184.2$\pm$3.9	297.92
	16	0.0323$\pm$0.0020	5.79$\pm$0.08	183.4$\pm$13.3	290.71
CP	32	0.0314$\pm$0.0003	5.88$\pm$0.02	**191.0**$\pm$1.6	287.88
	64	0.0327$\pm$0.0015	5.96$\pm$0.04	185.9$\pm$7.7	281.10
	128	0.0337$\pm$0.0001	**6.02**$\pm$0.03	182.3$\pm$1.1	264.80

4.3. MTPCs without Adapters

RQ1: Can we increase throughput by increasing the number of mixture components?

We begin with the simplest PC from our framework, MTPC-CP, which relaxes the independence assumption of the widely used MTPC-FF ($r = 1$) by increasing the number of mixture coefficients, r. MTPC-CP *increases throughput as it is more expressive than* MTPC-FF *yet still very efficient*.

Table 1 highlights MTPC-CP's increase in expressivity as evidenced by the increase in $\checkmark\mu_{\text{acc}}$as we increase r: MTPC-CP reaches $\checkmark\mu_{\text{acc}} = 6.02$ for $r = 128$, surpassing the MTPC-FF baseline by .83 for EvaByte with speculative sampling. At the same time, the μ_{lat} introduced by MTPC-CP increases only slightly as we increase r, because the forward pass cost of the MTPC-CP head is dominated by the expensive LLM calls. Nevertheless, as the increase in $\checkmark\mu_{\text{acc}}$ tails off for larger r, even the small μ_{lat} increase matters and we see the first of three cases in our experiments where the expressiveness-latency trade-off is vital for understanding the outcome: the best throughput is obtained for $r = 32$ and not $r = 128$. We note that our findings for MTPC-CP also hold for Llama and greedy speculative decoding, see Tables 8, 11 and 14 in the Appendix. In the last column, we also show the maximum attainable throughput ($\max_{\text{tok/s}}$), *i.e.*, we disable speculative decoding and accept all tokens. We see that we pay ≈ 90 tok/s for $r = 32$, which is not a big price to pay for guaranteeing no loss in generation quality.

While MTPC-CP performs well for $n = 8$, the margin for further improving throughput is small. This is because for $n = 8$, we can at best achieve $\checkmark\mu_{\text{acc}} = 8$, and we have already achieved $\checkmark\mu_{\text{acc}} = 6.02$ and have hit diminishing returns. To obtain substantial boosts in throughput, we need to find another axis of expressivity to benefit from. We therefore use MTPC's more expressive circuits to extend our model to longer window sizes. As $r = 32$ worked best, we keep it fixed for the remaining experiments.

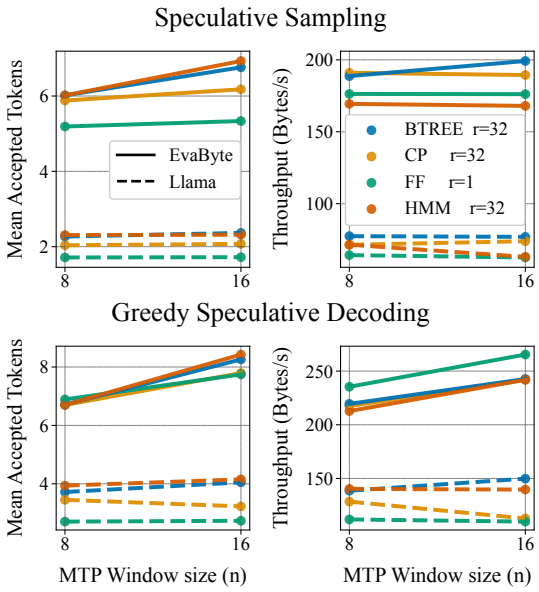


Figure 3. **Extending the MTP window of expressive PCs such as MTPC-HMM and MTPC-BTREE to $n = 16$ boosts their $\checkmark\mu_{\text{acc}}$ in all settings**, as can be seen from the upward trend on the left subplots and in more detail in Fig. 6. Importantly, MTPC-BTREE improves $\mu_{\text{tok/s}}$ for all but Llama with speculative sampling, highlighting its strong expressiveness–latency trade-off. In contrast, MTPC-HMM lags behind due to high latency from its AR nature.

RQ2: Do we benefit from more expressive circuit architectures for longer MTP windows?

We now consider the more expressive MTPC-HMM and MTPC-BTREE, and show that they outperform MTPC-CP on $\checkmark\mu_{\text{acc}}$ for longer MTP windows, highlighting the importance of our extension to general PCs. We fix $r = 32$ and explore the different PC architectures for both $n = 8$ and the longer window, $n = 16$. Fig. 3 shows that both MTPC-HMM and MTPC-BTREE increase $\checkmark\mu_{\text{acc}}$. However, MTPC-HMM strikes an unfavourable balance in the expressiveness–latency trade-off: *Due to being AR,* MTPC-HMM *has the largest μ_{lat} (see Tables 6, 9, 12 and 15), and yields poor throughput as a result*. While the gains already obtained by MTPC-BTREE are solid, fine-tuning the MTP head alone can only get us so far. This is because neither Llama nor EvaByte has been trained to produce representations that are good for predicting 16 tokens ahead, as we discuss next.

TAKEAWAY 1: While increasing the mixture components r in CP is initially beneficial, it soon hits diminishing returns. Increasing the MTP future token window size n and adopting more expressive PC architectures unlocks further throughput gains. Moreover, while the HMM achieves the highest acceptance rates, it incurs high latency. Instead, *non-autoregressive* variants such as BTREE strike a better trade-off and should be preferred.

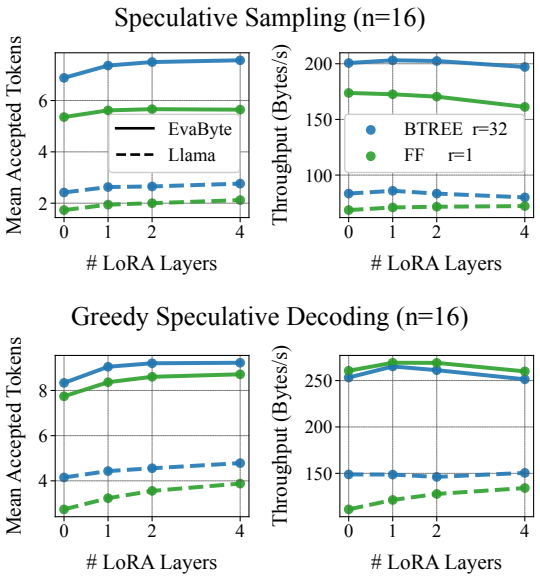


Figure 4. **We increase the expressiveness of the draft model by adding LoRA layers** (x-axis) for $n = 16$. As a result, the ☑μ_{acc} (left subplots) increases substantially for both EvaByte and Llama for both speculative decoding settings. Due to the expressiveness–latency trade-off the optimal $\mu_{\text{tok/s}}$ for EvaByte is obtained with 1 LoRA layer while Llama benefits further from 2 or more.

4.4. MTPCs with Adapters

RQ3: Can we further increase throughput by adapting the draft LLM using LoRA?

We now consider our final axis for increasing expressiveness: adding LoRA layers to the draft model. We note that we also fine-tune a model with 0 LoRA layers for an additional epoch to highlight that our improvements are not due to longer training alone. We show that while we can improve throughput, we need to be strategic when choosing the number of LoRA layers, as the latency introduced can rapidly outweigh the expressiveness gained.

For example, if we train adapters for the last 16 (out of 32) layers of EvaByte, we can improve the acceptance rate by 37%, but we introduce a latency of $1.5\times$ the cost of a forward pass of the LLM.[9] As can be seen for $n = 8$ in Fig. 4, adding LoRA layers almost always improves throughput. An exception is EvaByte-MTP with $n = 8$ where the ☑μ_{acc} cannot be increased by adding LoRA layers, see Fig. 7. This is likely because the backbone has been pre-trained with the MTP loss, and its representations cannot be improved further. In contrast, EvaByte-MTP-BTree once again obtains larger increases in ☑μ_{acc}, especially for speculative sampling for $n = 16$ where by using 4 LoRA layers we obtain a boost of .69 in ☑μ_{acc} over having no LoRA layers. However, because of the latency introduced, the best throughput is obtained with EvaByte-MTP-BTree with 1 LoRA layer, having a speed-up (⏭$\times_{\text{STP}}$) of $\times 5.15$ over EvaByte-STP (see Table 7), while Llama benefits from 2 LoRA layers and obtains $\times 2.24$ speed-up over Llama-STP (see Table 13). Importantly, both obtain a $\times 1.17$ speed-up over the best FF MTP model, highlighting the importance of relaxing the independence assumptions to improve speculative sampling.

TAKEAWAY 2: Fine-tuning a few layers of the draft model with LoRA increases the acceptance rate but also increases latency. The optimal trade-off can be device and backbone specific, but adding LoRAs is always beneficial compared to a fully shared LLM backbone, as long as we are retrofitting a model to an MTP window size it has not already been pretrained for.

5. Conclusions

We have identified key trade-offs between acceptance rates and latency within our framework, MTPC. We enhanced the *expressiveness* of MTP by getting rid of the independence assumption (Gloeckle et al., 2024; Zheng et al., 2025), introducing an explicit probabilistic model for inter-token dependencies that facilitates performance guarantees (Ankner et al., 2024; Li et al., 2024; DeepSeek-AI et al., 2024), and generalising mixture-based methods (Basharin et al., 2025) into the PC framework, unlocking a better expressiveness-latency trade-off. We also showed how to further optimise the expressiveness-latency trade-off by modulating the number of layers shared between draft and verifier model backbones. We showcase the throughput gains of MTPC LLMs *at scale* by retrofitting EvaByte (Zheng et al., 2025), a state-of-the-art 6.5B byte-level LLM, and Llama 3.2 3B, a byte-fied version of the widely used token-level LLM, into our framework.[10] Overall, our results show, for the first time, that throughput in MTP byte-level LLMs can be increased by $1.17\times$ w.r.t. MTP with independence assumptions and $5.15\times$ / $2.24\times$ w.r.t. AR for EvaByte/Llama, while simultaneously guaranteeing the retention of the AR LLM's quality. In future work, our framework can be extended by integrating constraints during generation (Ahmed et al., 2025) or speculative decoding (Nakshatri et al., 2025) via methods such as Gelato (Zhang et al., 2023) and Ctrl-G (Zhang et al., 2024a). Unlike those, we would not need to train an auxiliary HMM in MTPCs and we can integrate constraints directly into our PC head. Moreover, we can further boost expressiveness by leveraging other PC architectures such as subtractive mixtures (Loconte et al., 2024; 2025b) and continuous latent variable circuits (Gala et al., 2024a;b), while reducing latency via recent advancements in scaling up PCs (Liu et al., 2024; Zhang et al., 2025).

[9] We found that training more than 16 layers of EvaByte does not lead to improvements in acceptance rates.

[10] We comment in more depth on related work in Appendix H.

Impact statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Reproducibility statement

To ensure reproducibility for our research, we have attached to our submission the codebase for implementing all model variants and running their training and evaluation. In addition, we have provided full details on sampling in circuits in Appendix B and on our algorithms for speculative decoding in Appendix D.

References

Ahia, O., Kumar, S., Gonen, H., Kasai, J., Mortensen, D., Smith, N., and Tsvetkov, Y. Do all languages cost the same? tokenization in the era of commercial language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9904–9923, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614.

Ahmed, K., Teso, S., Chang, K.-W., Van den Broeck, G., and Vergari, A. Semantic probabilistic layers for neuro-symbolic learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Ahmed, K., Chang, K.-W., and Van den Broeck, G. Controllable generation via locally constrained resampling. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 4 2025. URL https://arxiv.org/pdf/2410.13111.

Ankner, Z., Parthasarathy, R., Nrusimha, A., Rinard, C., Ragan-Kelley, J., and Brandon, W. Hydra: Sequentially-dependent draft heads for medusa decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

Basharin, A., Chertkov, A., and Oseledets, I. Faster language models with better multi-token prediction using tensor decomposition. *arXiv preprint arXiv:2410.17765*, 2025.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv: 2401.10774*, 2024.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. M. Accelerating large language model decoding with speculative sampling. *ArXiv*, abs/2302.01318, 2023.

Cheng, S., Wang, L., Xiang, T., and Zhang, P. Tree tensor networks for generative modeling. *Physical Review B*, 99 (15):155131, 2019.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. Technical report, University of California, Los Angeles (UCLA), 2020.

Dagan, G., Synnaeve, G., and Rozière, B. Getting the most out of your tokenizer for pre-training and domain adaptation, 2024.

Darwiche, A. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50: 280–305, 2003.

Darwiche, A. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

Darwiche, A. and Marquis, P. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y.,

Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Gala, G., de Campos, C., Peharz, R., Vergari, A., and Quaeghebeur, E. Probabilistic integral circuits. In *AISTATS 2024*, 2024a.

Gala, G., de Campos, C., Vergari, A., and Quaeghebeur, E. Scaling continuous latent variable models as probabilistic integral circuits. *arXiv preprint arXiv:2406.06494*, 2024b.

Geiping, J., Stein, A., Shu, M., Saifullah, K., Wen, Y., and Goldstein, T. Coercing LLMs to do and reveal (almost) anything. In *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024. URL https://openreview.net/forum?id=Y5inHAjMu0.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Grasedyck, L. Hierarchical singular value decomposition of tensors. *SIAM journal on matrix analysis and applications*, 31(4):2029–2054, 2010.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J.,

Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Huang, S., Cheng, T., Liu, J. K., Xu, W., Hao, J., Song, L., Xu, Y., Yang, J., Liu, J., Zhang, C., Chai, L., Yuan, R., Luo, X., Wang, Q., Fan, Y., Zhu, Q., Zhang, Z., Gao, Y., Fu, J., Liu, Q., Li, H., Zhang, G., Qi, Y., Yinghui, X., Chu, W., and Wang, Z. OpenCoder: The open cookbook for top-tier code large language models. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33167–33193, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1591. URL https://aclanthology.org/2025.acl-long.1591/.

Kim, Y. and Rush, A. M. Sequence-level knowledge distillation. In Su, J., Duh, K., and Carreras, X. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1139. URL https://aclanthology.org/D16-1139/.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, X., Gu, Y., Malik, S., Graf, V., Hwang, J. D., Yang, J., Bras, R. L., Tafjord, O., Wilhelm, C., Soldaini, L., Smith, N. A., Wang, Y., Dasigi, P., and Hajishirzi, H. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Łańcucki, A., Staniszewski, K., Nawrot, P., and Ponti, E. M. Inference-time hyper-scaling with KV cache compression, 2025.

Land, S. and Bartolo, M. Fishing for magikarp: Automatically detecting under-trained tokens in large language models, 2024.

Lee, J., Mansimov, E., and Cho, K. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1149. URL https://aclanthology.org/D18-1149/.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning (ICML)*, 2022.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Liu, A., Ahmed, K., and Broeck, G. V. d. Scaling tractable probabilistic circuits: A systems perspective. *arXiv preprint arXiv:2406.00766*, 2024.

Loconte, L., Aleksanteri, M. S., Mengel, S., Trapp, M., Solin, A., Gillis, N., and Vergari, A. Subtractive mixture models via squaring: Representation and learning. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Loconte, L., Mari, A., Gala, G., Peharz, R., de Campos, C., Quaeghebeur, E., Vessio, G., and Vergari, A. What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)? *Transactions on Machine Learning Research*, 2025a. ISSN 2835-8856. Featured Certification.

Loconte, L., Mengel, S., and Vergari, A. Sum of Squares Circuits. In *The 39th Annual AAAI Conference on Artificial Intelligence (AAAI)*, 2025b.

Mari, A., Vessio, G., and Vergari, A. Unifying and understanding overparameterized circuit representations via low-rank tensor decompositions. In *6th Workshop on Tractable Probabilistic Modeling*, 2023.

Minixhofer, B., Murray, T., Limisiewicz, T., Korhonen, A., Zettlemoyer, L., Smith, N. A., Ponti, E. M., Soldaini, L., and Hofmann, V. Bolmo: Byteifying the next generation of language models, 2025a. URL https://arxiv.org/abs/2512.15586.

Minixhofer, B., Vulić, I., and Ponti, E. M. Cross-tokenizer distillation via approximate likelihood matching, 2025b. URL https://arxiv.org/abs/2503.20083.

Nakshatri, N. S., Roy, S., Das, R., Chaidaroon, S., Boytsov, L., and Gangadharaiah, R. Constrained decoding with speculative lookaheads. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4681–4700, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.239. URL https://aclanthology.org/2025.naacl-long.239/.

Nawrot, P., Chorowski, J., Lancucki, A., and Ponti, E. M. Efficient transformers with dynamic token pooling. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353.

Nawrot, P., Łańcucki, A., Chochowski, M., Tarjan, D., and Ponti, E. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=tDRYrAkOB7.

Pagnoni, A., Pasunuru, R., Rodriguez, P., Nguyen, J., Muller, B., Li, M., Zhou, C., Yu, L., Weston, J., Zettlemoyer, L., Ghosh, G., Lewis, M., Holtzman, A., and Iyer, S. Byte latent transformer: Patches scale better than tokens, 2024. URL https://arxiv.org/abs/2412.09871.

Peharz, R., Tschiatschek, S., Pernkopf, F., and Domingos, P. M. On Theoretical Properties of Sum-Product Networks. In *International Conference on Artificial Intelligence and Statistics*, 2015.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pp. 7563–7574. PMLR, 2020a.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., and Ghahramani, Z. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *35th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 115 of *Proceedings of Machine Learning Research*, pp. 334–344. PMLR, 2020b.

Rumbelow, J. and Watkins, M. Solidgoldmagikarp (plus, prompt generation). https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation (accessed 11-May-2025), 2023.

Salesky, E., Etter, D., and Post, M. Robust open-vocabulary translation from visual text representations. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7235–7252, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.576. URL https://aclanthology.org/2021.emnlp-main.576.

Shao, X., Molina, A., Vergari, A., Stelzner, K., Peharz, R., Liebig, T., and Kersting, K. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *International Conference on Probabilistic Graphical Models*, pp. 401–412. PMLR, 2020.

Shao, X., Molina, A., Vergari, A., Stelzner, K., Peharz, R., Liebig, T., and Kersting, K. Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140:298–313, 2022.

Shpilka, A. and Yehudayoff, A. Arithmetic circuits: A survey of recent results and open questions. *Founddations and Trends in Theoretical Computer Science*, 5:207–388, 2010.

Stern, M., Shazeer, N. M., and Uszkoreit, J. Blockwise parallel decoding for deep autoregressive models. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Sun, Z., Suresh, A. T., Ro, J. H., Beirami, A., Jain, H., and Yu, F. Spectr: Fast speculative decoding via optimal transport. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=SdYHLTCC5J.

The april Lab. cirkit, October 2024. URL https://github.com/april-tools/cirkit.

Varin, C., Reid, N., and Firth, D. An overview of composite likelihood methods. *Statistica Sinica*, 21, 01 2011.

Vergari, A., Di Mauro, N., and Esposito, F. Visualizing and understanding sum-product networks. *Machine Learning*, 108(4):551–573, 2019a.

Vergari, A., Di Mauro, N., and Van den Broeck, G. Tractable probabilistic models: Representations, algorithms, learning, and applications. *Tutorial at the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019b.

Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pp. 13189–13201. Curran Associates, Inc., 2021.

Wang, J., Gangavarapu, T., Yan, J. N., and Rush, A. M. Mambabyte: Token-free selective state space model, 2024. URL https://arxiv.org/abs/2401.13660.

Wertheimer, D., Rosenkranz, J., Parnell, T., Suneja, S., Ranganathan, P., Ganti, R., and Srivatsa, M. Accelerating production llms with combined token/embedding speculators, 2024. URL https://arxiv.org/abs/2404.19124.

Xia, H., Yang, Z., Dong, Q., Wang, P., Li, Y., Ge, T., Liu, T., Li, W., and Sui, Z. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 7655–7671, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.456. URL https://aclanthology.org/2024.findings-acl.456.

Yu, L., Simig, D., Flaherty, C., Aghajanyan, A., Zettlemoyer, L., and Lewis, M. Megabyte: Predicting million-byte sequences with multiscale transformers. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 78808–78823. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f8f78f8043f35890181a824e53a57134-Paper-Conference.pdf.

Zhang, H., Dang, M., Peng, N., and Van Den Broeck, G. Tractable control for autoregressive language generation. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 40932–40945. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/zhang23g.html.

Zhang, H., Kung, P.-N., Yoshida, M., Van den Broeck, G., and Peng, N. Adaptable logical control for large language models. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 115563–115587. Curran Associates, Inc., 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d15c16cf5619a2b1606da5fc88e3f1a9-Paper-Conference.pdf.

Zhang, H., Dang, M., Wang, B., Ermon, S., Peng, N., and Broeck, G. V. d. Scaling probabilistic circuits via monarch matrices. *arXiv preprint arXiv:2506.12383*, 2025.

Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G., and Mehrotra, S. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi:

10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

Zheng, L., Zhao, X., Wang, G., Wu, C., Dong, D., Wang, A., Wang, M., Du, Y., Bo, H., Sharma, A., Li, B., Zhang, K., Hu, C., Thakker, U., and Kong, L. Evabyte: Efficient byte-level language models at scale, 2025. URL https://hkunlp.github.io/blog/2025/evabyte.

A. Notation

We adapt notation and nomenclature from the tensor factorisation (Kolda & Bader, 2009) and circuit (Loconte et al., 2025a) literature.

We denote ordered sets of random variables with $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, and we use $[n]$ to express the set $\{1, 2, \ldots, n\}$ with $n > 0$. The domain of a variable X is denoted as $\mathsf{dom}(X)$, and we denoted as $\mathsf{dom}(\mathbf{X}) = \mathsf{dom}(X_1) \times \cdots \times \mathsf{dom}(X_n)$ the joint domain of variables $\mathbf{X} = \{X_i\}_{i=1}^{n}$. We denote scalars with lower-case letters (e.g., $a \in \mathbb{R}$), vectors with boldface lower-case letters (e.g., $\mathbf{a} \in \mathbb{R}^N$), matrices with boldface upper-case letters (excluding those used for variables, e.g., $\mathbf{A} \in \mathbb{R}^{M \times N}$), and tensors with boldface calligraphic letters (e.g., $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$). Moreover, we use subscripts to denote entries of tensors (e.g., a_{ijk} is the (i, j, k)-th entry in $\boldsymbol{\mathcal{A}}$).

B. Background on circuits

Circuits have a long history in theoretical computer science (Shpilka & Yehudayoff, 2010) and probabilistic reasoning (Darwiche, 2003; 2009). In their more modern definition and application to machine learning (Vergari et al., 2019b; Choi et al., 2020), circuits are introduced as structured computational graphs, simplified neural networks where one is allowed to use units from a restricted set of neurons (sum, product and input units) and whose connections need to abide certain *structural properties* to guarantee tractability (Choi et al., 2020; Vergari et al., 2021), as discussed next.

B.1. Structural properties

Tractability is to be intended as the ability to exactly compute a given function (operation) over the circuit in time that is polynomial in its size, denoted as $|c|$ for a circuit c, and representing the number of edges between the computational units. For example, a circuit c can exactly integrate *any subset of variables* in time $\mathcal{O}(|c|)$ if (i) its input functions can be integrated efficiently and (ii) it is *smooth* and *decomposable* (Darwiche & Marquis, 2002; Choi et al., 2020).

Definition 1 (Smoothness and decomposability (Darwiche & Marquis, 2002; Choi et al., 2020))**.** A circuit is *smooth* if for every sum unit n, all its input units depend on the same variables, i.e., $\forall i, j \in \mathsf{in}(n) \colon \mathsf{sc}(i) = \mathsf{sc}(j)$. A circuit is *decomposable* if the distinct inputs of every product unit n depend on disjoint sets of variables, i.e., $\forall i, j \in \mathsf{in}(n)\, i \neq j \colon \mathsf{sc}(i) \cap \mathsf{sc}(j) = \varnothing$.

Note that all the PC architectures we have discussed in this paper, FF, CP, HMM and BTree, are smooth and decomposable circuits. The reader is encouraged to check this by themselves for the architectures in Fig. 2.

Exactly integrating variables out is relevant to compute marginals such as the normalisation constant of the distribution encoded by the circuit (Eq. (3)). Note that in our implementation, circuits are normalised by design (Peharz et al., 2015), as we assume that input distributions are normalised categoricals and all sum units form a convex combination as their weights are parameterised with a softmax function (see Sec. 3.3).

More importantly for our MTPCs, we can draw samples efficiently from the distribution of a circuit that is both smooth and decomposable, as we discuss in the next sub-section.

B.2. Sampling a circuit

A smooth and decomposable PC can use ancestral sampling to generate a complete sample for all n tokens in a window. In a nutshell, we can iteratively sample each latent variable in the hierarchy encoded by the PC, and then sample the selected input distributions, in the same way one sample one (hierarchical) mixture model by first sampling one component and then drawing a sample from that component.

Operationally, Alg. 1 details the procedure. We have to sample one input branch for each sum unit we encounter when performing a backward traversal of the circuit computational graph (from the circuit output back to the input distributions). Such a branch is sampled proportionally to the sum unit weights ω_j, which encode the mixture components (or equivalently the transition probabilities in an HMM). Then, when we traverse a product unit, we follow all its input branches. When we reach an input unit, we sample a token proportionally to the parameters ϕ_{ij} of the categorical distributions encoded in the unit.

If the circuit is smooth and decomposable, by this process we are guaranteed to end up in a set of input units whose scope is the full set of tokens $\mathbf{X}$ and in which only one input unit is selected per token position i (line 13 of Alg. 1). This procedure can be tensorised to efficiently generate a batch of samples in a single pass over the computational graph of the circuit

Algorithm 1 SAMPLE(c)

Input: A smooth, decomposable and normalised PC c encoding a joint distribution q over the next n tokens $\mathbf{X} = \{X_1, \ldots, X_n\}$ **Output:** a sample $\mathbf{x} \sim q(\mathbf{X})$.

$\mathbf{x} \leftarrow \mathsf{zeroes}(n)$ {init empty sample}
$c_n \leftarrow \mathsf{output}(c)$
$\mathcal{N} \leftarrow \mathsf{queue}(\{c_n\})$ {traverse the computational graph from outputs to inputs}
while $\mathcal{N}$ not empty **do**
  $c_n \leftarrow \mathsf{pop}(\mathcal{N})$ {c_n is a sum unit}
  if $c_n = \sum_{j=1}^{r} \omega_j c_j$ **then**
    $k \leftarrow \mathsf{sampleCategorical}(\omega_1, \ldots, \omega_r)$ {sample from a categorical with r states}
    $\mathcal{N} \leftarrow \mathsf{push}(\mathcal{N}, c_k)$
  else if $c_n = \prod_{j=1}^{d} c_j$ **then**
    for $k = 1 \ldots d$ **do**
      $\mathcal{N} \leftarrow \mathsf{push}(\mathcal{N}, c_k)$ {visit all inputs of product unit c_n}
    end for
  else if c_n is an input unit over variable X_i and parameters ϕ_i **then**
    $x_i \leftarrow \mathsf{sampleCategorical}(\phi_i)$ {sample from a categorical with $v = |\mathcal{V}|$ states}
  end if
end while
return $\mathbf{x}$

(Vergari et al., 2019a; Peharz et al., 2020b;a; Loconte et al., 2025a; Liu et al., 2024).

Lastly, we remark that this routine is potentially computationally more efficient than the one implemented in Basharin et al. (2025), as the latter is based on autoregressive inverse transform sampling (see Loconte et al. (2024) for a discussion) and requires sampling one token at a time.

C. Target Model Details

Table 2. Details of target models we retrofitted using MTPC.

	EvaByte	Llama3.2 3B Byte
MTP Window	8	—
# Transformer Layers	32	28
# Attention Heads	32	24
Embedding Size	4096	3072
dtype	bfloat16	float32
Vocab Size	320	268
Number of Parameters	6.5B	3B
Max Context Window	32768	131072

C.1. EvaByte Details

In Table 3 we provide a copy of the benchmark results of fine-tuned version of EvaByte, EvaByte-SFT, which we retrofit in the paper. The model card can be found at `https://hf.co/EvaByte/EvaByte-SFT`, see also Zheng et al. (2025) for more details. The numbers in the table above were produced by Medusa-style (Cai et al., 2024) tree-based typical decoding. For all benchmarks apart from HumanEval, the results were produced via greedy decoding, *i.e.*, similar to (Stern et al., 2018) with the exception of producing the last "free" token. As such, the produced tokens are equivalent to what EvaByte-STP would produce, and the authors found that the metrics were the same with EvaByte-STP up to some small rounding errors. For HumanEval the authors used tree-based typical decoding, which in this case does not maintain the quality of the EvaByte-STP model. The details above were shared with us by Lin Zheng, the author of EvaByte.

Model	BBH	GSM8k	IFEval	MATH	MMLU	HumanEval*	TruthQA
Gemma-2-9B-it	20.0	79.7	69.9	29.8	69.1	71.7	61.4
Ministral-8B-Instruct	56.2	80.0	56.4	40.0	68.5	91.0	55.5
Qwen-2.5-7B-Instruct	25.3	83.8	74.7	69.9	76.6	93.1	63.1
Llama-3.1-8B-Instruct	69.7	83.4	80.6	42.5	71.3	86.3	55.1
Tülu 3 8B	66.0	87.6	82.4	43.7	68.2	83.9	55.0
OLMo-7B-Instruct	35.3	14.3	32.2	2.1	46.3	28.7†	44.5
OLMo-v1.7-7B-Instruct	34.4	23.2	39.2	5.2	48.9	49.7†	55.2
OLMoE-1B-7B-0924-Instruct	37.2	47.2	46.2	8.4	51.6	54.8	49.1
MAP-Neo-7B-Instruct	26.4	69.4	35.9	31.5	56.5	72.1†	51.6
OLMo-2-7B-SFT	50.7	71.2	68.0	25.1	62.0	67.0†	47.8
OLMo-2-7B-1124-Instruct	48.5	85.2	75.6	31.3	63.9	67.6†	56.3
EvaByte-SFT	34.6	52.9	60.2	29.8	49.5	73.7	46.3

Table 3. Downstream benchmark performance of EvaByte-SFT, table taken verbatim from Zheng et al. (2025). Entries with † were computed by the EvaByte authors. All numbers were computed with Medusa-style tree-based greedy decoding using the multi-token head, apart from HumanEval*, which used typical sampling. The authors of EvaByte followed Tulu 3, and evaluated the Pass@10 rate for HumanEval with 20 samples at temperature 0.8.

D. Speculative Decoding

We give pseudocode for our self-speculative decoding algorithm below. The algorithm accepts between 0 and n tokens, but always generates between 1 and n tokens, where n is the MTP window size. The algorithm is very similar to vanilla speculative decoding (Leviathan et al., 2022), but our algorithm includes a modification that reduces latency for the self-speculative scenario, and for this it needs to sacrifice the last "free" token typically obtained from the verifier. The gain in latency is possible because we can evaluate the shared LLM once per draft/verification cycle, while a naive implementation of Leviathan et al. (2022) for self-speculative decoding would need two, approximately halving the possible throughput.

In our self-speculative setup, the verifier and draft LLMs share some layers of the backbone. Importantly, the verifier is always computing LLM states ahead of the draft. As such, we can get away with a single forward pass through the shared LLM, similar to Medusa (Cai et al., 2024), by re-using the LLM backbone state computed by the verifier for the draft model. For this to work, we cannot accept a "last sample for free" from the verifier (lines 23-30) Alg. 3), as we would not have the backbone state for this new token and it is not worth paying an extra LLM evaluation for it. Therefore, in our algorithm we only sample the "free" token from the verifier in the rare case that no tokens are accepted. This is necessary because the model can get caught in successive no-accept states in the sampling case, or get stuck in an infinite loop if we use greedy decoding. If any tokens were accepted, we use the last state of the shared backbone computed during the verify phase to seed the draft phase. In what follows, if we have no LoRA layers, the algorithm is modified to have a single component: the shared encoder.

Algorithm 2 SHAREDSTATESELFSPECULATIVEDECODING

Architecture Components:
Three components: Shared Encoder (S), Verifier (V), Draft (D)
Each with their own KV-cache
Given: A prompt of length L
Initialisation:
Prefill V to $L-1$, and D and S to L
Set S and D state

Switch to draft/verify cycle:
while true **do**
Draft stage:
if S state is not set **then**
Compute S by conditioning on the additional token
end if
Use S state to compute D state
Parameterize MTPC with D state
Draft n tokens

Verify stage:
Compute S state on $n+1$ tokens (draft + predecessor)
Compute V state using S state
Obtain up to $n+1$ tokens from speculative decoding
if 0 tokens accepted **then**
Keep "free" token sampled from last valid logits
Unset S state (stale)
else
Accept n tokens (drop "free" token)
Set S state (hidden state for last accepted token)
end if
end while

Algorithm 3 SELFSPECULATIVEDECODING($\mathbf{x}_{\leq t}, f, h, c, g$)

Input: A prefix $\mathbf{x}_{\leq t}$ of length t, an LLM backbone $f\colon \mathcal{V}^* \to \mathbb{R}^d$, an LLM head $h\colon \mathbb{R}^d \to \mathbf{\Theta}$ parameterising a PC c encoding a joint distribution q over the next n tokens, and an LLM head $g\colon \mathbb{R}^d \to \Delta^v$ computing the next token probabilities.
Output: A sentence $(\mathbf{x}_{\leq t} || \mathbf{x}_{t+1:t+s}) \in \mathcal{V}^{t+s}$ where $1 \leq s \leq n+1$. Moreover, we have that $\mathbf{x}_{t+1:t+s} \sim p(x_{t+1}, \ldots, x_{t+s} \mid \mathbf{x}_{\leq t})$ as equivalently encoded by the autoregressive single token prediction model consisting of f and g only (Leviathan et al., 2022).

$\mathbf{e}_t \leftarrow f(\mathbf{x}_{\leq t})$ {Compute the last embedding}
$\boldsymbol{\theta} \leftarrow h(\mathbf{e}_t)$ {Compute the circuit parameters}

Let $q(\mathbf{X}_{t+1:t+n} \mid \mathbf{x}_{\leq t}) = \frac{1}{Z_{\boldsymbol{\theta}}} c(\mathbf{X}_{t+1:t+n} \mid \boldsymbol{\theta})$
$\mathbf{x}_{t+1:t+n} \sim q(\mathbf{X}_{t+1:t+n} \mid \mathbf{x}_{\leq t})$ {Sample n tokens from c in time $\mathcal{O}(|c|)$}
$\mathbf{x} \leftarrow \mathbf{x}_{\leq t} \,||\, \mathbf{x}_{t+1:t+n}$ {Concatenate the prefix with the n tokens}

Compute in parallel for $1 \leq i \leq n$: {Compute marginals in time $\mathcal{O}(|c|)$}
    $q(\mathbf{x}_{t+1:t+i} \mid \mathbf{x}_{\leq t}) = \sum_{x_{t+i+1}, \ldots, x_{t+n} \in \mathcal{V}} q(\mathbf{x}_{t+1:t+n} \mid \mathbf{x}_{\leq t})$

Compute in parallel for $1 \leq i \leq n+1$: {Compute target model conditionals}
    $p(X_{t+i} \mid \mathbf{x}_{\leq t+i-1}) = g(\mathbf{e}_{t+i-1})$, where $\mathbf{e}_{t+i-1} = f(\mathbf{x}_{\leq t+i-1})$

$s \leftarrow 0$ {Determine the number of accepted tokens s, $0 \leq s \leq n$}
while $s < n$ **do**
    $\alpha \sim \mathcal{U}(0, 1)$
    if $s > 0$ **then**
        $q(x_{t+s+1} \mid \mathbf{x}_{\leq t+s}) \leftarrow q(\mathbf{x}_{t+1:t+s+1} \mid \mathbf{x}_{\leq t}) / q(\mathbf{x}_{t+1:t+s} \mid \mathbf{x}_{\leq t})$
    end if
    if $\alpha > p(x_{t+s+1} \mid \mathbf{x}_{t+s}) / q(x_{t+s+1} \mid \mathbf{x}_{t+s})$ **then**
        exit loop
    end if
    $s \leftarrow s + 1$
end while{Sample one last token from the autoregressive LLM model}
if $s < n$ **then**
    Let $s(X_{t+s+1}) = q(X_{t+s+1} \mid \mathbf{x}_{\leq t+s})$ {Adjust the distribution first, if we accept fewer tokens}
    Let $m(X_{t+s+1}) = \max\left(0,\ p(X_{t+s+1} \mid \mathbf{x}_{\leq t+s}) - s(X_{t+s+1})\right)$
    $r(X_{t+s+1} \mid \mathbf{x}_{t+s}) = m(X_{t+s+1}) / Z$, with $Z = \sum_{x' \in \mathcal{V}} m(x')$
    $x_{t+s+1} \sim r(X_{t+s+1} \mid \mathbf{x}_{\leq t+s})$
else
    $x_{t+s+1} \sim p(X_{t+s+1} \mid \mathbf{x}_{\leq t+s})$
end if
return $\mathbf{x}_{\leq t+s} \,||\, x_{t+s+1}$

D.1. GPU Memory Usage For Speculative Decoding

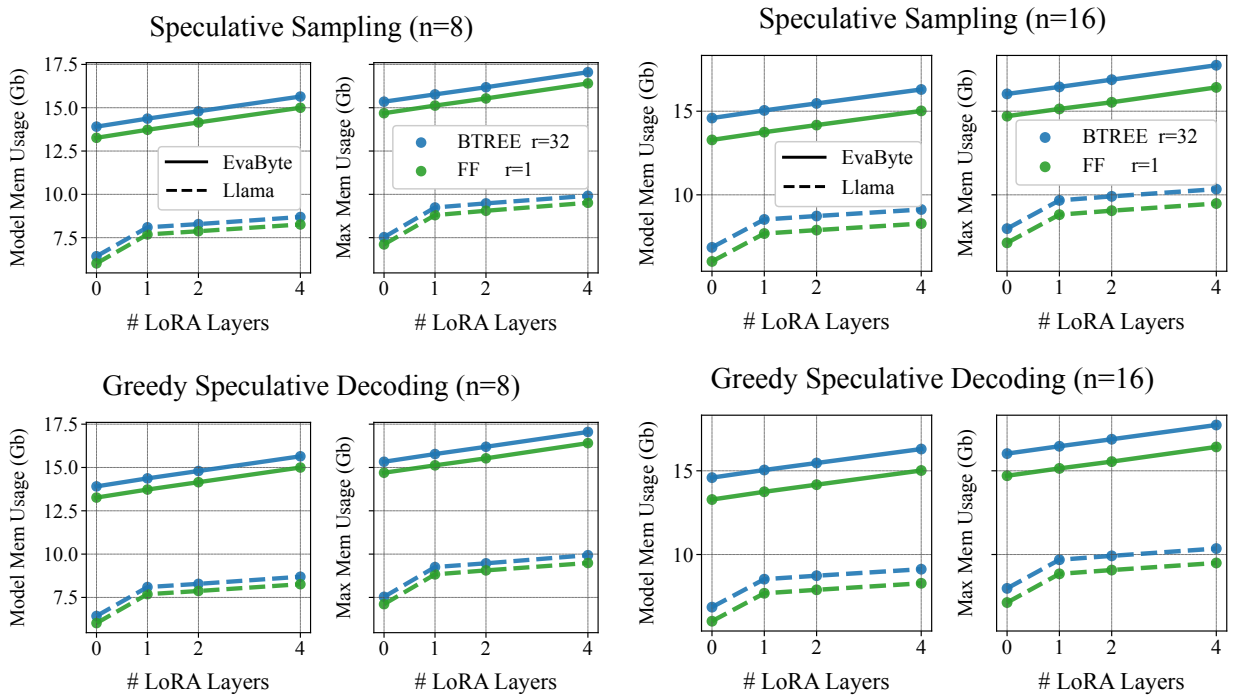


Figure 5. GPU memory usage as a function of the number of LoRA layers (x-axis) during model loading (left subplot) and maximum memory usage during inference (right subplot). As can be seen, MTPC-BTREE has similar memory requirements to MTPC-FF and scales graciously with the MTP window size, n. Moreover, retrofitting LoRA on the last 1-4 layers barely changes the footprint: we pay a memory overhead of approximately k transformer layers when using k LoRA layers. Measurements taken on the L40S GPUs during the speculative decoding experiments with KV cache enabled, BF16 computations, and 1,024 generated tokens.

E. Hidden Markov Models Setup

In our experiments we use **contextual**, **inhomogeneous** hidden Markov models (HMMs) with **identity initialisation** (see Table 4). We chose the above after preliminary experiments where we assessed the following configuration choices for training our HMMs.

Parameterisation We can parameterise HMMs to either be contextual, i.e. we can make the transition probabilities depend on the input, or we can make the transition probabilities be independent of the input (non-contextual).

Transition Type The transition matrix can be the same at each time step (homogeneous) or it can be different (inhomogeneous). The former would correspond to additional parameter sharing across sum layers in the circuit representation. We note that inhomogeneous HMMs subsume homogeneous HMMs. This is because inhomogeneous HMMs could in theory learn parameters that do not vary from timestep to timestep, thus becoming equivalent to a homogeneous HMM.

Initialisation A crucial setup is to initialise the HMM with transition matrices that are identity matrices, which make the HMM equivalent to CP at the beginning of training. We achieve this by adding a bias term to allow the HMM model to be initialised to identity matrices. This setting in combination with extending to larger token windows, i.e. $n = 16$ lead to a scenario where HMMs outperform CP. The other alternative is to initialise the transition matrices uniformly at random (before softmax), but this complicates learning and yields performance that is lower than CP models.

Table 4. Most Successful HMM Configuration

Parameterisation		Transition Type		Initialisation	
Contextual	Non-Contextual	Homogeneous	Inhomogeneous	Identity Init.	Uniform Init.
✓	✗	✗	✓	✓	✗

F. Further Experimental Details

To make the comparison between methods fair, we: a) constrained the models to not produce end-of-sequence symbols during generation, as the latency of retrieving KV cache items from memory increases with sequence length (Nawrot et al., 2024) and b) we filtered the validation set of the models to only include examples with both prompts and responses in English, as acceptance rates may vary dramatically based on the language chosen for the response.

We compute throughput by generating answers to 250 prompts and report the mean and std of 3 runs with different prompts.

G. Alternative Losses

In early versions of this work we also experimented using a Kullback-Leibler divergence (KL) loss as recommended by Basharin et al. (2025). However, we found that training with the KL loss doubled the training time while requiring a lot more memory, and the benefits in acceptance rate did not outweigh the additional complexity. For completeness we include the loss below. **KL Loss** $\mathcal{L}$

$$\mathcal{L} = \sum_{j=1}^{n} \mathcal{L}_j \gamma^{j-1}, \quad \mathcal{L}_j = \sum_{i=1}^{N} \sum_{t=1}^{L} \frac{f_{\text{KL}}\left(p_\theta(x_{t+j}^{(i)} \mid \mathbf{x}_{<t+j}^{(i)}) \,\middle\|\, q_{\theta'}(x_{t+j}^{(i)} \mid \mathbf{x}_{<t+j}^{(i)})\right)}{N\text{valid}(i,j)} \tag{5}$$

In the above we condition both the draft model, $q_{\theta'}$, and the target model, p_θ, on the gold data. The above is equivalent to the KL term from the word-level distillation loss in (Kim & Rush, 2016).[11]

H. Further Related Work

MTP for byte-level LLMs Gloeckle et al. (2024) and Zheng et al. (2025) both pretrain byte-level LLMs which predict $n = 8$ future bytes. This window size was found to be optimal for downstream performance in Gloeckle et al. (2024). Both make conditional independence assumptions, but the approaches are architecturally different. Gloeckle et al. (2024) uses a transformer head per token to provide different feature vectors to each head and uses a shared unembedding matrix. On the other hand, Zheng et al. (2025) uses a shared feature vector across heads with different unembedding matrices per token.[12] However, they only focus on greedy self-speculative decoding, while in our work, we also explore speculative sampling.

Speculative Decoding with MTP drafts Previous MTP work either ignores speculative sampling and only focuses on greedy self-speculative decoding (Gloeckle et al., 2024), or abandons guarantees altogether (possibly at the expense of quality): specifically, Cai et al. (2024) and Zheng et al. (2025) use a tree decoding mechanism to consider multiple candidates at each speculative decoding step. Since their approach may accept multiple continuations but only generate the longest accepted one, they bias the distribution and break the guarantees. (Wang et al., 2024) use a subword-level draft model to speed up their byte-level STP model via speculative decoding. Most prior work has introduced sequential dependencies in the draft model through architecture modifications. Hydra (Ankner et al., 2024) modifies the Medusa heads such that the predicted probabilities are also a function of the input embeddings of predicted draft tokens. Eagle (Li et al., 2024) introduces sequential dependencies by autoregressively predicting future feature representations. While these works relax the independence assumption, they have no explicit probabilistic model for the dependencies introduced.

An exception is Basharin et al. (2025), who study the effect of relaxing the conditional independence assumption by using a CP factorisation. While they obtain some first promising results, showing that increasing the rank can increase the acceptance rate of tokens for speculative decoding, they focus on subword-level models which have very large vocabulary sizes (e.g. $v \geq 100$k). This makes CP very expensive, both in terms of the number of parameters needed, and the GPU memory required. Moreover, they evaluate their models on unrealistic scenarios, i.e. datasets used for pre-training LLMs rather than instruction fine-tuning. Finally, despite the fact that a lot of previous work exists on MTP for subword-level LLMs, they use different models from those widely used for benchmarking speculative decoding methods, despite the existence of a benchmark, Spec-Bench (Xia et al., 2024) and common models (e.g. Vicuna). In our case, since there is a limited amount of work on MTP for byte-level LLMs (Gloeckle et al., 2024; Zheng et al., 2025), we directly compare with the results of the EvaByte model.

[11] While performing sequence-level distillation, *i.e.*, conditioning on data sampled from the teacher model may improve distillation (Kim & Rush, 2016), we did not explore this.

[12] It is worth noting that Gloeckle et al. (2024) also do an ablation in the appendix and find that linear heads were competitive.

There has been increasing interest in multi-token prediction not only for generation speed-up, but also for improved model performance on tasks due to the lookahead offered by MTP. For example, DeepSeek-AI et al. (2024, Table 4) show that MTP for a token window of 2 tokens leads to improvements in benchmark metrics even when MTP is not used at inference time. Furthermore, they report an increase in the throughput of the model by $\times 1.8$ when using speculative decoding.

Token granularity In MTP a token can vary in granularity from bytes (Zheng et al., 2025) to subword tokens provided by tokenisers (Basharin et al., 2025).

In addition to the choice of token granularity, there are generally 3 axes related works differ on:

- Training from scratch vs distilling an existing STP model into a MTP model
- Neural network architectures for the token heads (e.g. Linear, MLP, Transformer)
- Probabilistic modelling assumptions (conditional independence vs more expressive models)

H.1. Differences in Scenario

Training from Scratch Evabyte (Zheng et al., 2025) train a MTP byte-level model from scratch using $n = 8$.

Retrofitting STP to MTP Some works explore both training from scratch and retrofitting an STP model into an MTP one (Cai et al., 2024; Basharin et al., 2025). In our work, we focus on the second setting.

H.2. Differences in Architectures

Linear Heads Basharin et al. (2025) use a linear parameterisation for each token head in their distillation experiments.

MLP Heads Cai et al. (2024) use a MLP with a single hidden layer for each output token head. Ankner et al. (2024) extend this to multiple layers of MLPs per output token. Gloeckle et al. (2024) make the heads context-aware by including a transformer head in each token head.

Autoregressive Head While the main point of having future token heads is to avoid the expensive autoregression of the target model, current state of the art speculative decoding models rely on "cheap" autoregression. Eagle (Li et al., 2024), which is the best performing on the speculative decoding benchmark, Spec-bench (Xia et al., 2024), fits an autoregressive model to predict future feature vectors of the model (i.e. the inputs to the softmax layer). A similar architecture was also used for the DeepSeekV3 model (DeepSeek-AI et al., 2024).

MLP Heads Cai et al. (2024) propose the Medusa model which uses a MLP with a residual connection for each token head. While in theory they could use many MLP layers, they choose to use MLPs with single hidden layer. Ankner et al. (2024) explore using deeper MLPs for each token head.

Transformer Heads Gloeckle et al. (2024) use a shared unembedding matrix and use a separate transformer for each token head. More precisely, in order to predict the token at offset s, i.e. x_{t+s}, they compute $\mathrm{softmax}\left(\mathbf{W}\mathbf{z}_s\right)$, where $\mathbf{z}_s$ is produced by a separate transformer head for each s, i.e. $\mathbf{z}_s = H_s(\mathbf{x}_{\leq t})$.

Sharing the Unembedding layer. While the decoding of Zheng et al. (2025) is based on Medusa, instead of using a MLP for each token head, they use a different unembedding matrix per token head.[13] This modelling choice is possible due to the small vocabulary size of byte-level models, i.e. $|\mathcal{V}| = 320$. In their training from scratch scenario, Basharin et al. (2025) use different unembedding layers $\mathbf{W}_a^{(s)}$ in order to predict x_{t+s} for the mixture component with index a. As such, they parameterise $s \times |a|$ unembedding matrices. This seems non-ideal, since the last layer in LLMs can have a large number of parameters, i.e. $(V \times d)$. In their distillation scenario they use a shared unembedding matrix.

[13] https://github.com/OpenEvaByte/evabyte/blob/98d5f48d32197b803e7560a798da35c7a4bdcf4d/evabyte_hf/modeling_evabyte.py#L753

I. Results on L40S GPU

I.1. EvaByte 6.5B

I.1.1. Speculative Sampling

Table 5

		$\mu_{\text{lat}}\downarrow$	$\mu_{\text{acc}}\uparrow$	$\mu_{\text{tok/s}}\uparrow$	$\max_{\text{tok/s}}$
FF	1	**0.0299**$\pm$0.0003	5.19$\pm$0.05	176.3$\pm$0.9	297.55
	8	0.0314$\pm$0.0005	5.67$\pm$0.05	184.2$\pm$3.9	297.92
	16	0.0323$\pm$0.0020	5.79$\pm$0.08	183.4$\pm$13.3	290.71
CP	32	0.0314$\pm$0.0003	5.88$\pm$0.02	**191.0**$\pm$1.6	287.88
	64	0.0327$\pm$0.0015	5.96$\pm$0.04	185.9$\pm$7.7	281.10
	128	0.0337$\pm$0.0001	**6.02**$\pm$0.03	182.3$\pm$1.1	264.80

Table 6

			$\mu_{\text{lat}}\downarrow$	$\mu_{\text{acc}}\uparrow$	$\mu_{\text{tok/s}}\uparrow$	$\times_{\text{STP}}$
1	1	STP	**0.0254**	—	39.5	1.00
8	1	FF	0.0299$\pm$0.0003	5.19$\pm$0.05	176.3$\pm$0.9	4.47
	32	HMM	0.0362$\pm$0.0006	**6.02**$\pm$0.02	169.4$\pm$2.6	4.29
		BTree	0.0326$\pm$0.0008	**6.02**$\pm$0.05	188.7$\pm$6.1	4.78
		CP	0.0314$\pm$0.0003	5.88$\pm$0.02	**191.0**$\pm$1.6	**4.84**
16	1	FF	0.0308$\pm$0.0003	5.34$\pm$0.03	176.1$\pm$2.3	4.46
	32	HMM	0.0421$\pm$0.0022	**6.93**$\pm$0.06	168.0$\pm$9.4	4.26
		CP	0.0333$\pm$0.0009	6.18$\pm$0.02	189.4$\pm$5.4	4.80
		BTree	0.0347$\pm$0.0000	6.76$\pm$0.07	**199.2**$\pm$2.2	**5.05**

Table 7

			$\mu_{\text{lat}}\downarrow$	$\mu_{\text{acc}}\uparrow$	$\mu_{\text{tok/s}}\uparrow$	$\times_{\text{STP}}$
1	STP	—	**0.0254**	—	39.5	1.00
8	FF	0	0.0308$\pm$0.0004	5.11$\pm$0.03	168.9$\pm$2.9	4.28
		1	0.0325$\pm$0.0011	5.11$\pm$0.07	160.0$\pm$7.8	4.06
		2	0.0332$\pm$0.0009	5.12$\pm$0.07	157.1$\pm$6.3	3.98
		4	0.0348$\pm$0.0004	5.12$\pm$0.04	149.5$\pm$3.1	3.79
	BTree	0	0.0332$\pm$0.0006	6.05$\pm$0.05	**186.1**$\pm$1.8	**4.72**
		1	0.0348$\pm$0.0010	6.16$\pm$0.06	180.9$\pm$6.9	4.58
		2	0.0354$\pm$0.0014	6.18$\pm$0.03	178.2$\pm$7.9	4.52
		4	0.0373$\pm$0.0017	**6.22**$\pm$0.01	170.9$\pm$7.5	4.33
16	FF	0	0.0313$\pm$0.0012	5.36$\pm$0.07	173.8$\pm$7.9	4.40
		1	0.0331$\pm$0.0010	5.62$\pm$0.08	172.6$\pm$7.1	4.37
		2	0.0339$\pm$0.0010	5.66$\pm$0.15	170.5$\pm$8.9	4.32
		4	0.0357$\pm$0.0012	5.64$\pm$0.11	161.2$\pm$8.7	4.09
	BTree	0	0.0350$\pm$0.0002	6.88$\pm$0.10	200.7$\pm$2.6	5.09
		1	0.0371$\pm$0.0008	7.36$\pm$0.03	**203.1**$\pm$5.0	**5.15**
		2	0.0379$\pm$0.0013	7.50$\pm$0.03	202.5$\pm$7.0	5.13
		4	0.0393$\pm$0.0006	**7.57**$\pm$0.13	197.1$\pm$6.3	5.00

I.1.2. Greedy Speculative Decoding

Table 8

		$\mu_{\text{lat}}\downarrow$	$\mu_{\text{acc}}\uparrow$	$\mu_{\text{tok/s}}\uparrow$	$\max_{\text{tok/s}}$
FF	1	**0.0300**$\pm$0.0006	**6.89**$\pm$0.01	**235.4**$\pm$5.4	299.64
	8	0.0310$\pm$0.0002	6.68$\pm$0.00	221.5$\pm$1.3	295.79
	16	0.0312$\pm$0.0000	6.70$\pm$0.02	221.3$\pm$0.5	294.16
CP	32	0.0317$\pm$0.0002	6.70$\pm$0.01	217.4$\pm$1.6	288.68
	64	0.0319$\pm$0.0002	6.69$\pm$0.01	215.9$\pm$1.1	282.55
	128	0.0340$\pm$0.0003	6.68$\pm$0.02	202.3$\pm$2.1	264.70

Table 9

			$\mu_{\text{lat}}\downarrow$	$\mu_{\text{acc}}\uparrow$	$\mu_{\text{tok/s}}\uparrow$	$\times_{\text{STP}}$
1	1	STP	**0.0251**	—	39.9	1.00
8	1	FF	0.0300$\pm$0.0006	**6.89**$\pm$0.01	**235.4**$\pm$5.4	5.90
	32	HMM	0.0324$\pm$0.0001	6.70$\pm$0.03	212.8$\pm$1.5	5.33
		CP	0.0317$\pm$0.0002	6.70$\pm$0.01	217.4$\pm$1.6	5.45
		BTree	0.0315$\pm$0.0004	6.71$\pm$0.03	219.5$\pm$3.6	5.50
16	1	FF	0.0299$\pm$0.0002	7.75$\pm$0.02	265.4$\pm$1.8	**6.65**
	32	HMM	0.0360$\pm$0.0009	**8.43**$\pm$0.02	241.8$\pm$6.0	6.06
		CP	0.0333$\pm$0.0002	7.79$\pm$0.05	242.2$\pm$1.8	6.07
		BTree	0.0353$\pm$0.0020	8.26$\pm$0.08	**242.6**$\pm$12.4	6.08

Table 10

			$\mu_{\text{lat}}\downarrow$	$\mu_{\text{acc}}\uparrow$	$\mu_{\text{tok/s}}\uparrow$	$\times_{\text{STP}}$
1	STP	—	**0.0251**	—	39.9	1.00
8	FF	0	0.0297$\pm$0.0003	6.89$\pm$0.01	237.3$\pm$2.2	5.95
		1	0.0311$\pm$0.0003	6.89$\pm$0.01	**227.2**$\pm$2.7	**5.69**
		2	0.0319$\pm$0.0002	**6.90**$\pm$0.02	221.2$\pm$1.7	5.54
		4	0.0336$\pm$0.0006	6.88$\pm$0.01	209.6$\pm$4.1	5.25
	BTree	0	0.0316$\pm$0.0004	6.72$\pm$0.02	218.6$\pm$2.6	5.48
		1	0.0333$\pm$0.0006	6.73$\pm$0.01	208.3$\pm$3.6	5.22
		2	0.0337$\pm$0.0002	6.73$\pm$0.01	205.6$\pm$1.2	5.15
		4	0.0356$\pm$0.0009	6.72$\pm$0.03	194.5$\pm$4.0	4.87
16	FF	0	0.0305$\pm$0.0005	7.74$\pm$0.03	260.6$\pm$4.5	6.53
		1	0.0319$\pm$0.0004	8.36$\pm$0.04	**269.2**$\pm$3.4	**6.74**
		2	0.0329$\pm$0.0002	8.60$\pm$0.02	269.1$\pm$1.9	6.74
		4	0.0346$\pm$0.0002	8.72$\pm$0.02	259.9$\pm$1.1	6.51
	BTree	0	0.0341$\pm$0.0003	8.33$\pm$0.10	253.2$\pm$5.0	6.35
		1	0.0354$\pm$0.0001	9.05$\pm$0.06	265.2$\pm$2.3	6.65
		2	0.0365$\pm$0.0002	9.20$\pm$0.09	261.2$\pm$3.6	6.54
		4	0.0381$\pm$0.0001	**9.23**$\pm$0.07	251.3$\pm$2.5	6.30

I.2. Llama 3.2 3B

I.2.1. Speculative Sampling

Table 11

		$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\max_{\text{tok/s}}$
FF	1	0.0291±0.0031	1.71±0.01	64.2±6.4	342.39
	8	0.0298±0.0031	1.92±0.04	70.2±5.9	355.81
CP	16	0.0301±0.0036	1.98±0.02	**71.7**±8.0	357.04
	32	**0.0311**±0.0032	**2.04**±0.04	71.4±8.2	357.82

Table 12

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	1	STP	**0.0256**	—	39.2	1.00
8	1	FF	0.0291±0.0031	1.71±0.01	64.2±6.4	1.64
	32	CP	0.0311±0.0032	2.04±0.04	71.4±8.2	1.82
		HMM	0.0351±0.0030	**2.31**±0.05	71.5±6.7	1.82
		BTree	0.0318±0.0040	2.27±0.04	**77.4**±8.7	**1.98**
16	1	FF	0.0299±0.0023	1.72±0.03	62.6±4.3	1.60
	32	HMM	0.0400±0.0045	2.32±0.03	63.0±6.3	1.61
		CP	0.0303±0.0028	2.07±0.01	73.9±6.8	1.89
		BTree	0.0333±0.0037	**2.37**±0.05	**76.9**±6.9	**1.96**

Table 13

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	STP	—	**0.0256**	—	39.2	1.00
8	FF	0	0.0269±0.0001	1.73±0.02	69.4±0.9	1.77
		1	0.0323±0.0041	1.95±0.06	67.0±9.4	1.71
		2	0.0308±0.0003	2.02±0.08	72.1±3.1	1.84
		4	0.0318±0.0001	2.18±0.04	75.2±1.0	1.92
	BTree	0	0.0301±0.0004	2.31±0.02	82.9±0.6	2.12
		1	0.0334±0.0021	2.54±0.02	82.0±5.8	2.09
		2	0.0332±0.0011	2.71±0.04	**87.7**±1.9	**2.24**
		4	0.0362±0.0042	**2.82**±0.08	84.1±10.1	2.15
16	FF	0	0.0273±0.0002	1.73±0.04	68.6±0.6	1.75
		1	0.0300±0.0004	1.94±0.03	70.9±1.3	1.81
		2	0.0308±0.0002	2.00±0.02	71.6±0.1	1.83
		4	0.0323±0.0005	2.12±0.05	72.2±1.2	1.84
	BTree	0	0.0312±0.0004	2.42±0.03	83.4±1.4	2.13
		1	0.0329±0.0006	2.63±0.07	**85.9**±3.3	**2.19**
		2	0.0343±0.0002	2.65±0.03	83.4±1.2	2.13
		4	0.0376±0.0041	**2.76**±0.02	80.0±8.3	2.04

I.2.2. Greedy Speculative Decoding

Table 14

		μ_{lat} ↓	μ_{acc} ↑	$\mu_{\text{tok/s}}$ ↑	$\max_{\text{tok/s}}$
FF	1	**0.0248**±0.0010	2.70±0.00	111.8±4.3	330.31
	8	0.0264±0.0003	3.22±0.03	128.2±1.1	310.61
CP	16	0.0286±0.0020	3.39±0.05	125.2±7.9	315.91
	32	0.0285±0.0026	**3.45**±0.04	**128.4**±12.2	282.88

Table 15

			μ_{lat} ↓	μ_{acc} ↑	$\mu_{\text{tok/s}}$ ↑	$\times_{\text{STP}}$
1	1	STP	0.0255	—	39.4	1.00
8	1	FF	**0.0248**±0.0010	2.70±0.00	111.8±4.3	2.84
	32	CP	0.0285±0.0026	3.45±0.04	128.4±12.2	3.26
		BTree	0.0283±0.0022	3.72±0.10	138.6±7.7	3.52
		HMM	0.0295±0.0023	**3.94**±0.06	**140.3**±11.5	**3.56**
16	1	FF	0.0257±0.0027	2.73±0.01	109.8±11.2	2.79
	32	CP	0.0304±0.0025	3.23±0.06	112.7±9.9	2.86
		HMM	0.0312±0.0023	**4.15**±0.07	139.6±8.0	3.55
		BTree	0.0285±0.0001	4.05±0.06	**149.8**±1.9	**3.81**

Table 16

			μ_{lat} ↓	μ_{acc} ↑	$\mu_{\text{tok/s}}$ ↑	$\times_{\text{STP}}$
1	STP	—	0.0255	—	39.4	1.00
8	FF	0	**0.0252**±0.0014	2.71±0.00	110.8±6.2	2.81
		1	0.0266±0.0003	3.27±0.03	126.8±0.6	3.22
		2	0.0288±0.0018	3.57±0.04	128.4±8.0	3.26
		4	0.0300±0.0003	3.89±0.06	134.8±3.2	3.42
	BTree	0	0.0287±0.0017	3.86±0.07	141.7±8.8	3.60
		1	0.0291±0.0005	4.12±0.05	148.7±0.8	3.78
		2	0.0297±0.0002	4.29±0.03	**151.7**±1.8	**3.85**
		4	0.0320±0.0019	**4.48**±0.05	147.0±9.1	3.73
16	FF	0	0.0253±0.0016	2.74±0.01	111.2±7.0	2.82
		1	0.0275±0.0013	3.23±0.05	121.3±5.3	3.08
		2	0.0289±0.0018	3.56±0.04	127.8±8.6	3.25
		4	0.0300±0.0006	3.88±0.05	134.2±3.2	3.41
	BTree	0	0.0294±0.0014	4.15±0.04	148.8±7.5	3.78
		1	0.0314±0.0019	4.43±0.10	148.7±9.6	3.78
		2	0.0329±0.0014	4.56±0.05	146.2±6.5	3.71
		4	0.0335±0.0016	**4.78**±0.02	**150.5**±6.6	**3.82**

J. Results on RTX 3090 GPU

J.1. EvaByte

J.1.1. Speculative Sampling

		$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\max_{\text{tok/s}}$
FF	1	0.0548	5.14	96.2	160.66
CP	8	0.0591	5.64	98.0	157.02
	16	0.0577	5.73	102.3	158.35
	32	0.0574	5.89	105.8	154.99
	64	0.0567	5.98	108.8	154.92
	128	0.0590	5.98	104.4	151.02

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	1	STP	0.0472	—	21.2	1.00
8	1	FF	0.0548	5.14	96.2	4.53
	32	HMM	0.0657	6.03	94.3	4.44
		BTree	0.0597	6.05	104.4	4.91
		CP	0.0574	5.89	105.8	4.98
16	1	FF	0.0556	5.37	99.2	4.67
	32	HMM	0.0774	6.88	91.2	4.29
		CP	0.0594	6.18	107.1	5.04
		BTree	0.0639	6.71	108.2	5.09

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	STP	—	0.0472	—	21.2	1.00
8	FF	0	0.0553	5.13	95.0	4.47
		1	0.0569	5.05	90.9	4.28
		2	0.0586	5.11	89.3	4.20
		4	0.0624	5.12	84.1	3.96
	BTree	0	0.0595	6.04	104.5	4.92
		1	0.0614	6.19	104.0	4.90
		2	0.0625	6.21	102.6	4.83
		4	0.0667	6.22	96.3	4.53
16	FF	0	0.0554	5.35	99.1	4.67
		1	0.0597	5.46	94.0	4.42
		2	0.0605	5.61	95.3	4.49
		4	0.0635	5.64	91.4	4.30
	BTree	0	0.0620	6.85	114.0	5.37
		1	0.0648	7.24	115.6	5.44
		2	0.0666	7.41	115.3	5.43
		4	0.0697	7.50	111.7	5.26

J.1.2. Greedy Speculative Decoding

		$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\max_{\text{tok/s}}$
FF	1	0.0552	6.87	128.9	162.65
	8	0.0579	6.66	119.7	160.28
	16	0.0580	6.67	119.7	158.39
CP	32	0.0581	6.65	119.2	161.22
	64	0.0584	6.65	118.7	155.42
	128	0.0584	6.64	118.7	151.29

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	1	STP	0.0466	—	21.5	1.00
8	1	FF	0.0552	6.87	128.9	5.98
	32	HMM	0.0603	6.69	115.3	5.35
		BTree	0.0589	6.66	117.7	5.46
		CP	0.0581	6.65	119.2	5.53
16	1	FF	0.0550	7.71	145.7	6.76
	32	HMM	0.0657	8.38	133.6	6.20
		CP	0.0602	7.70	134.3	6.23
		BTree	0.0621	8.13	137.5	6.38

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	STP	—	0.0466	—	21.5	1.00
8	FF	0	0.0563	6.87	126.6	5.88
		1	0.0588	6.87	121.0	5.62
		2	0.0598	6.88	119.5	5.55
		4	0.0633	6.86	112.5	5.22
	BTree	0	0.0598	6.68	116.4	5.40
		1	0.0624	6.70	112.0	5.20
		2	0.0652	6.69	107.0	4.97
		4	0.0669	6.70	104.4	4.85
16	FF	0	0.0569	7.70	140.8	6.54
		1	0.0621	8.30	139.5	6.48
		2	0.0624	8.51	142.6	6.62
		4	0.0655	8.66	138.4	6.43
	BTree	0	0.0636	8.22	136.0	6.31
		1	0.0657	8.94	143.4	6.66
		2	0.0680	9.03	140.2	6.51
		4	0.0709	9.10	135.5	6.29

J.2. Llama 3.2 3B

J.2.1. Speculative Sampling

		$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\max_{\text{tok/s}}$
FF	1	0.0503	1.71	36.8	193.61
	8	0.0519	1.92	40.0	190.51
CP	16	0.0524	1.96	40.6	192.18
	32	0.0517	2.04	42.7	193.86

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	1	STP	0.0410	—	24.5	1.00
8	1	FF	0.0503	1.71	36.8	1.51
	32	HMM	0.0605	2.21	39.8	1.63
		CP	0.0517	2.04	42.7	1.75
		BTree	0.0554	2.19	42.9	1.76
16	1	FF	0.0506	1.72	36.8	1.50
	32	HMM	0.0703	2.25	34.8	1.42
		CP	0.0525	2.08	42.6	1.74
		BTree	0.0555	2.38	46.0	1.88

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	STP	—	0.0410	—	24.5	1.00
8	FF	0	0.0497	1.71	37.4	1.53
		1	0.0533	1.94	40.0	1.64
		2	0.0538	2.05	41.8	1.71
		4	0.0568	2.11	40.9	1.67
	BTree	0	0.0540	2.36	47.1	1.93
		1	0.0566	2.56	48.8	1.99
		2	0.0577	2.68	50.1	2.05
		4	0.0604	2.71	48.6	1.99
16	FF	0	0.0495	1.71	37.5	1.53
		1	0.0534	1.91	39.5	1.61
		2	0.0546	2.04	41.0	1.68
		4	0.0573	2.05	39.8	1.63
	BTree	0	0.0562	2.40	46.2	1.89
		1	0.0583	2.60	48.3	1.97
		2	0.0596	2.65	48.1	1.97
		4	0.0620	2.74	47.7	1.95

J.2.2. Greedy Speculative Decoding

		$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\max_{\text{tok/s}}$
FF	1	0.0448	2.69	61.7	195.66
CP	8	0.0485	3.23	70.0	193.55
	16	0.0496	3.41	72.5	194.39
	32	0.0496	3.45	73.6	193.76

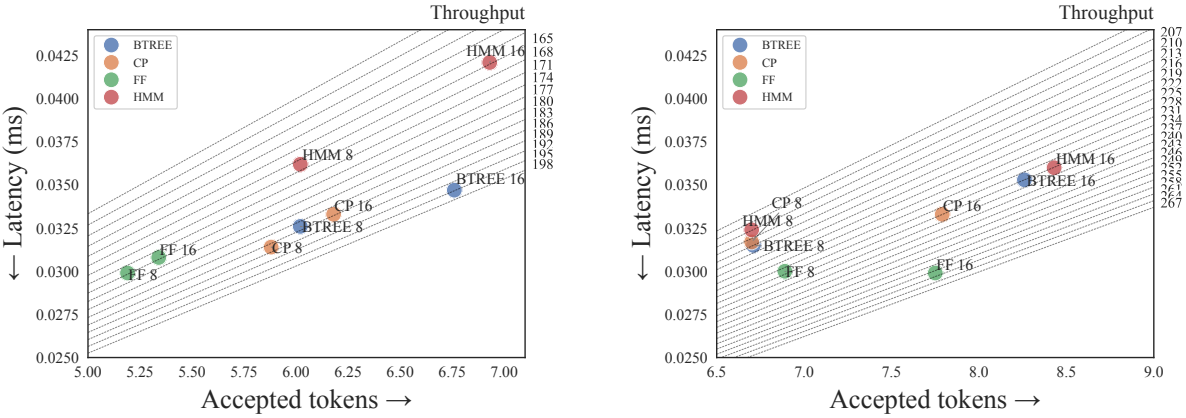
			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	1	STP	0.0413	—	24.3	1.00
8	1	FF	0.0448	2.69	61.7	2.54
	32	CP	0.0496	3.45	73.6	3.03
		BTree	0.0506	3.72	77.6	3.20
		HMM	0.0519	3.90	78.9	3.25
16	1	FF	0.0458	2.70	60.8	2.50
	32	CP	0.0500	3.20	68.0	2.80
		HMM	0.0559	4.10	77.1	3.17
		BTree	0.0518	4.00	81.7	3.36

			$\mu_{\text{lat}} \downarrow$	$\mu_{\text{acc}} \uparrow$	$\mu_{\text{tok/s}} \uparrow$	$\times_{\text{STP}}$
1	STP	—	0.0413	—	24.3	1.00
8	FF	0	0.0441	2.68	62.4	2.57
		1	0.0477	3.25	70.6	2.91
		2	0.0492	3.54	74.9	3.08
		4	0.0524	3.89	77.3	3.18
	BTree	0	0.0503	3.84	80.6	3.32
		1	0.0524	4.11	82.6	3.40
		2	0.0532	4.30	85.2	3.51
		4	0.0554	4.45	84.6	3.48
16	FF	0	0.0444	2.73	63.2	2.60
		1	0.0479	3.23	70.0	2.88
		2	0.0495	3.55	74.4	3.06
		4	0.0531	3.89	76.2	3.14
	BTree	0	0.0517	4.16	85.0	3.50
		1	0.0538	4.38	86.1	3.54
		2	0.0549	4.61	88.6	3.65
		4	0.0573	4.74	87.4	3.60

K. Further Plots on Extending the MTP Token Window to $n = 16$

(a) EvaByte Speculative Sampling

(b) EvaByte Greedy Speculative Decoding



(c) Llama Speculative Sampling

(d) Llama Greedy Speculative Decoding

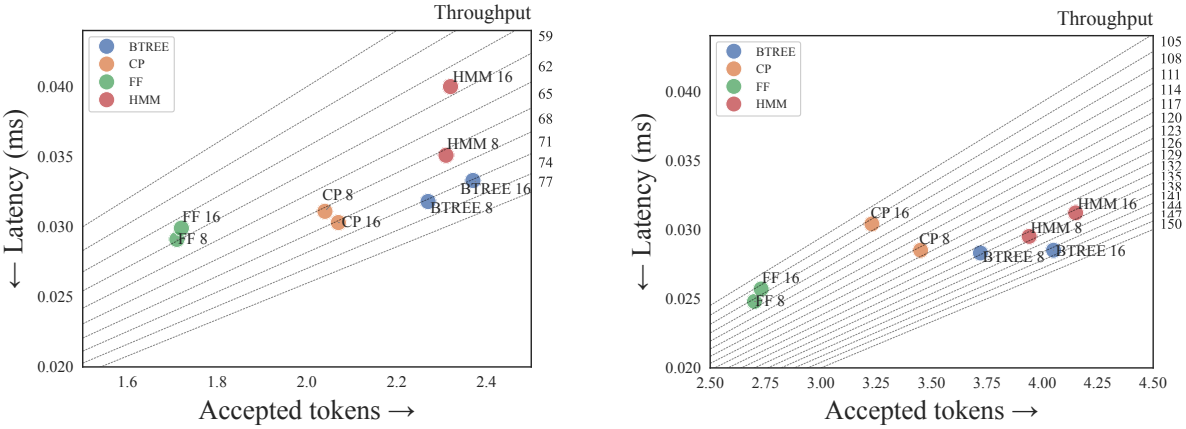


Figure 6. Extending the MTP window of expressive PCs such as MTPC-HMM and MTPC-BTREE to $n = 16$ boosts their ☑μ_{acc} in all settings, as can be verified by checking that HMM 16 is to the right of HMM 8 and BTREE 16 is to the right of BTREE 8 in all four subplots. Importantly, MTPC-BTREE improves $\mu_{\text{tok/s}}$ for all cases apart from Llama with speculative sampling (bottom left), highlighting its strong expressiveness–latency trade-off. In contrast, MTPC-HMM lags behind due to high latency from its AR nature (HMM 16 is much higher in the plots than HMM 8).

L. Further Plots on Adding more LoRA Layers to the Draft Model

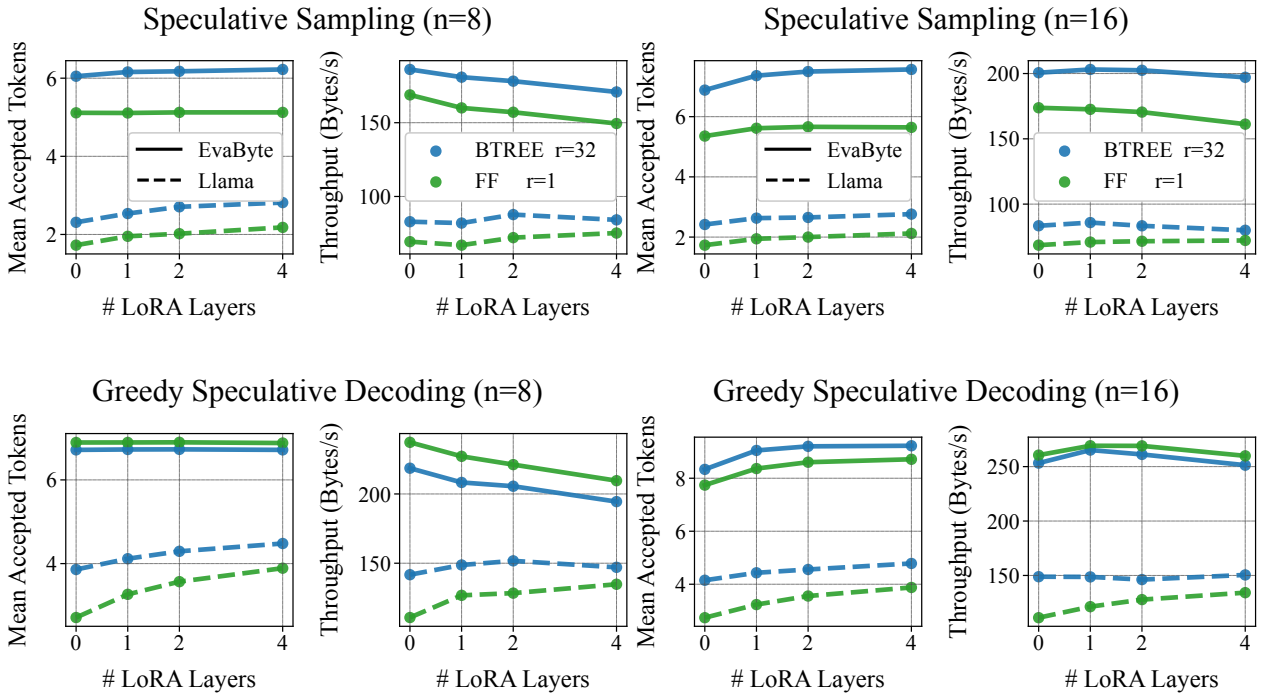


Figure 7. Increasing the expressiveness of our draft model by adding LoRA layers. For the $n = 8$ case, we get very little improvement for sampling with EvaByte and no improvement for greedy speculative decoding. We believe this is because EvaByte has been pretrained as a MTP model with $n = 8$ and as such the draft backbone has already converged to MTP representations which cannot be improved further. When we move to $n = 16$, EvaByte acceptance rates are boosted significantly both for speculative sampling and greedy speculative decoding and the optimal throughput is obtained with 1 LoRA layer. On the other hand, the Llama acceptance rates and throughput are boosted in all cases.