Parallel Token Generation for Language Models

Felix Draxler* Justus Will* Farrin Marouf Sofian
Department of Computer Science,
University of California, Irvine

Theofanis Karaletsos
Chan-Zuckerberg Initiative
& Pyramidal AI

Sameer Singh
Department of Computer Science,
University of California, Irvine

Stephan Mandt
Computer Science & AI in Science Institute,
University of California, Irvine

Abstract

We propose Parallel Token Prediction (PTP), a universal framework for parallel sequence generation in language models. PTP jointly predicts multiple dependent tokens in a single transformer call by incorporating the sampling procedure into the model. This reduces the latency bottleneck of autoregressive decoding, and avoids the restrictive independence assumptions common in existing multi-token prediction methods. We prove that PTP can represent arbitrary autoregressive sequence distributions. PTP is trained either by distilling an existing model or through inverse autoregressive training without a teacher. Experimentally, we achieve state-of-the-art speculative decoding performance on Vicuna-7B by accepting over four tokens per step on Spec-Bench. The universality of our framework indicates that parallel generation of long sequences is feasible without loss of modeling power.

1 Introduction

Autoregressive transformers (Vaswani et al., 2017) are the foundation of today's large language models (LLMs) (Brown et al., 2020). Their sequential generation process, however, remains a major bottleneck: each token depends on the full history, requiring one forward pass per token. For long outputs, this increases the inference latency significantly compared to what a single transformer call would achieve.

Many recent efforts aim to bypass this bottleneck by predicting multiple tokens at once. Broadly, they can be categorized into two lines of work: the first, speculative decoding, takes a systems approach, making predictions in a lightweight model that is verified by a large model (Leviathan et al., 2023; Chen et al., 2023; Sun et al., 2023; Zhong et al., 2025). The second line of work makes use of predicting several tokens independent of each other. This significantly reduces the search space for sequences and improves overall model quality (Qi et al., 2020; Gloeckle et al., 2024; DeepSeek-AI et al., 2025). Similarly, discrete diffusion iteratively refines generated sequences, again not modeling conditional dependencies between tokens in each denoising step (Hoogeboom et al., 2021; Austin et al., 2021). However, all of these methods still contain an irreducible sequential component to generate sequences.

Our work takes a step towards filling this gap. We propose a framework that, in theory, can generate arbitrary length sequences in parallel. This is enabled by a small but fundamental architectural change: instead of sampling from the distributions predicted by an autoregressive model in a post-processing step, we feed the involved random variables as an input to the model: the model learns to sample. This enables it to anticipate which tokens will be sampled and predict them jointly. Similar frameworks have been formulated in the normalizing flow literature: Inverse Autoregressive Flows (Kingma et al., 2016) generate samples

*Equal contribution. Corresponding authors: {fdraxler,jcwill,mandt}@uci.edu

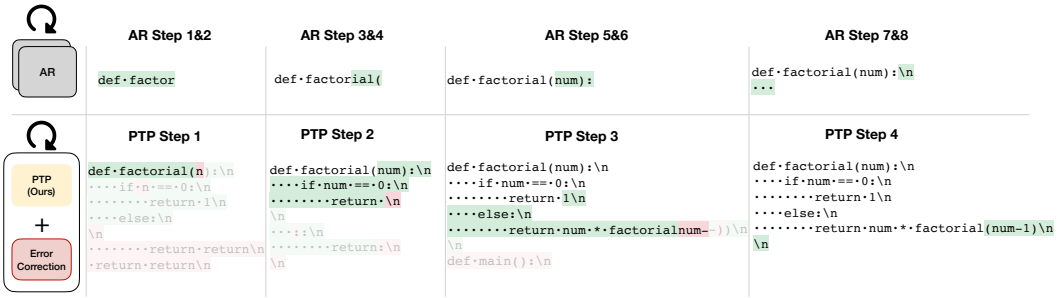

Figure 1: Our parallelized model generates the same text as its teacher in a fraction of the steps. By the time our model (bottom) has generated an entire function, an autoregressive model (top) only generates the method's signature. Prompt: Write a Python function that computes the factorial of a number. Green tokens are accepted tokens in that step, red tokens are incorrect. Semitransparent tokens are rejected after the first mistake.

of many continuous dimensions in parallel and Free-form Flows (Draxler et al., 2024) distill a fast generator network. We transfer these concepts to sampling discrete sequences from a continuous latent space.

Our contributions are as follows:

- We propose Parallel Token Prediction (PTP), a modeling approach for discrete data that generates multiple interdependent tokens in one model call (Section 2.1).
- We prove that PTP is as expressive as autoregressive models (Theorems 1 and 2).
- PTP can be trained to predict several tokens either by distilling an existing teacher (Section 2.2.1), effectively parallelizing it, or from scratch on training data (Section 2.2.2).
- Experimentally, we train real-world coding and natural language PTP models, achieving 7.0 respectively 4.18 accepted tokens per speculative decoding step (Section 3).

Together, our framework opens a design space to build models that accurately predict several tokens in parallel, reducing latency in language model output without limiting representational power.

2 Parallel Token Prediction

2.1 Parallel Sampling

To construct our Parallel Token Prediction framework, let us recap how a classical transformer decoder generates text. It iteratively predicts the categorical distribution of the next token $t_i \in \{1, \dots, V\}$ based on all previous tokens $t_{<i} = (t_1, \dots, t_{i-1})$,

$$P_i := P(t_i | t_{<i}) \tag{1}$$

For simplicity, we assume this distribution is the final distribution that is used to generate tokens, in that it already reflects temperature scaling (Guo et al., 2017), top-k and top-p sampling (Holtzman et al., 2020), or other approaches trading sample diversity with quality. To sample a token from this distribution, one draws an auxiliary random variable $u_i \sim \mathcal{U}[0, 1]$ and looks up the corresponding token from the cumulative distribution function as follows:

$$t_i = \text{Pick}(u_i, P_i) \equiv \min_{j \in \{1, \dots V\}} \{j : F_{ij} > u_i\}, \quad \text{where } F_{ij} = \sum_{l=1}^{j} P_{il}. \tag{2}$$

Here, j iterates possible token choices, P_{il} is the probability to sample $t_i = l$, and F_{ij} is the cumulative distribution to sample a token $t_i \in \{1, \dots j\}$.

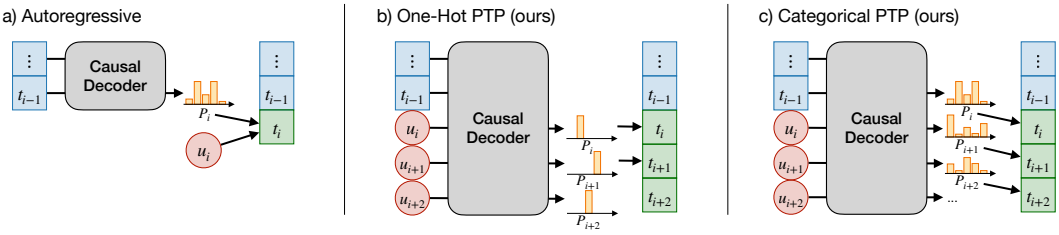

Figure 2: Parallel Token Prediction predicts several tokens in one model call. (a) An autoregressive model predicts the distribution of the single next token t_i. From the histogram, that token is chosen with the help of an auxiliary variable u_i that is chosen uniformly at random (see Fig. 3). (b) One-Hot Parallel Token Prediction merges the sampling into the model by feeding the auxiliaries directly into the model. This allows joint prediction of several tokens. (c) Categorical Parallel Token Prediction models the distribution of each token, but predicts them in parallel using the auxiliary variables.

Figure 2(a) illustrates how, in traditional autoregressive models, we first sample t_i from P_i before moving on to predicting the next token t_{i+1}, as the distribution P_{i+1} depends on the selected token t_i. Every new token involves another model call, increasing latency. To break this iterative nature, note that while Eq. (1) defines a distribution over possible next tokens, Eq. (2) is a deterministic rule once the auxiliary variable u_i is drawn. Thus, write this rule as an explicit deterministic function:

$$t_i = f_P(t_{<i}; u_i) = \text{Pick}(u_i, P(\cdot\,|t_{<i})). \tag{3}$$

Figure 3 illustrates how this function jumps from token to token as a function of u_i.

These auxiliary variables are all we need to perform parallel generation of text: all information about which token t_i we are going to select is available to the model if it has access to u_i as one of its inputs. By repeating the above argument and feeding all the auxiliary variables into the model, any subsequent token $t_{>i}$ can be predicted deterministically (proof in Appendix A.1):

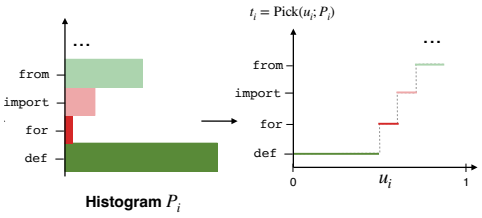

Figure 3: Sampling from a discrete distribution. Given a histogram P_i (left), compute the inverse cumulative distribution function (right) and look up the token at a random location $u_i \in \mathcal{U}[0,1]$. Our framework relies on considering both parts jointly.

Theorem 1. Let P denote a probability distribution for next token prediction. Then, the future token t_k can be selected as a deterministic function f_P of previous tokens $t_{<i}$ and auxiliary variables $u_i, \ldots, u_k \sim \mathcal{U}[0,1]$:

$$t_k = f_P(t_{<i}; u_i, \ldots u_k), \qquad \text{for all } k \geq i. \tag{4}$$

Theorem 1 shows a clear path to build a model that can sample many tokens in parallel: instead of learning the distribution $P(t_k|t_{<k})$, we propose to directly fit the function $f_P(t_{<i}; u_i, \ldots, u_k)$, which jointly predicts future tokens t_k.

Figure 2(b) visualizes how this path can be implemented with a standard transformer (Vaswani et al., 2017) backbone: alongside the previous tokens, simply feed the auxiliary random variables for the next N tokens into the model. Then predict $P(t_k|t_{<i}; u_i, \ldots, u_k)$ for each future token. Per Theorem 1, this prediction narrows down to a single token: $P(t_k|t_{<i}; u_i, \ldots, u_k) = \mathbf{1}(t_k = f_P(t_{<i}; u_i, \ldots, u_k))$. In practice, we take the argmax to extract that predicted token token:

$$t_k = f_P^{\text{O-PTP}}(t_{<i}; u_i, \ldots, u_k) = \text{argmax}(P(t_k \mid t_{<i}; u_i, \ldots, u_k)) \tag{5}$$

We refer to this model as a One-Hot Parallel Token Prediction Model (O-PTP). O-PTPs can be trained to replicate an existing autoregressive model P, as we discuss in Section 2.2.1.

O-PTPs converge to one-hot distributions, and therefore do not expose the underlying sampling distributions of the model. This prevents access to the original conditional probabilities $P(t_k|t_{<k})$, which are required for training without a teacher, adjusting temperature (Holtzman et al., 2020), and uncertainty quantification. To address this, we introduce Categorical Parallel Token Prediction (C-PTP), which recovers the full conditional distribution of each token. The key idea is to predict each token t_k while conditioning on all past auxiliary variables $u_i, \ldots, u_{k-1}$, but explicitly excluding its own auxiliary variable u_k. By Theorem 1, these past auxiliaries deterministically encode the sampled history $t_{<k}$. Withholding u_k preserves the uncertainty over t_k rather than collapsing it to a point mass. As a result, conditioning t_k on $(t_{<i}, u_i, \ldots, u_{k-1})$ exactly recovers the original autoregressive conditional $P(t_k \mid t_{<k})$, as formalized below.

Theorem 2. Let P denote a probability distribution for next token prediction. Then, the distribution of a token t_k is fully determined by context tokens $t_{<i}$ and the past auxiliary variables $u_i, \ldots, u_{k-1}$:

$$P(t_k|t_{<i}, u_i, \ldots, u_{k-1}) = P(t_k|t_{<k}), \qquad \text{for all } k \geq i. \tag{6}$$

Notably, Eq. (6) does not depend on u_k.

Figure 2(c) shows how Theorem 2 can be used to predict the distribution of the tokens $t_i, \ldots t_N$ in parallel. Just like for O-PTP, first sample all required auxiliary variables $u_i, \ldots u_N$, and then predict all $P_k = P(t_k|t_{<i}, u_i, \ldots, u_{k-1})$ in parallel. Sampling from these distributions is done just as in Eq. (3):

$$t_k = f_P^{\text{C-PTP}}(t_{<i}, u_i, \ldots, u_k) = \text{Pick}(u_k; P(t_{<i}, u_i, \ldots, u_{k-1})). \tag{7}$$

By using a causal decoder architecture, we can properly mask which token has access to which auxiliaries.

C-PTP can be trained without a teacher by iteratively solving Eq. (6) for the auxiliary u_k corresponding to a given token t_k, see Section 2.2.2. Like O-PTP, it can also be distilled from a teacher.

We provide a full proof for Theorem 2 in Appendix A.2 and give a short intuition here: For the first token, Eq. (6) is identical to the original autoregressive distribution in Eq. (1): $P_i = P(t_i|t_{<i}, \cancel{u_i}) = P(t_i|t_{<i})$. Moving to the next token t_{i+1}, we now do pass in the auxiliary variable u_i used to sample the first token t_i. Since P_i and u_i uniquely determine t_i, u_i and t_i contain the same information. By the law of total probability, this recovers Eq. (1) for t_{i+1}: $P_{i+1} = P(t_i|t_{<i}, u_i) = P(t_{i+1}|t_{<i}, t_i)$. Repeating this argument, we find that the distribution of every future tokens is available if we condition on all preceding auxiliary variables, concluding the proof.

Both One-Hot and Categorical Parallel Token Prediction allow coordinated token prediction in a single model call. By Theorems 1 and 2, there are no fundamental restrictions as to how many tokens can be jointly modeled apart from model capacity. In the next section, we propose two approaches to train from scratch (only C-PTP) or by distillation of an existing model (both O-PTP and C-PTP).

2.2 Training

Before deriving the training paradigms for Parallel Token Prediction Models, let us quickly recall that autoregressive models are trained by minimizing the cross-entropy between samples from the training data $t \sim P(t)$ and the model P_θ

$$\mathcal{L}(\theta) = \mathbb{E}_{t \sim P(t)} \left[-\sum_{i=1}^{N} \log P_\theta(t_i \mid t_{1\ldots i-1}) \right]. \tag{8}$$

Using a causal model such as a transformer (Vaswani et al., 2017) this loss can be evaluated on an entire sequence of tokens in a single model call (Radford et al., 2018). We first

present how to distill both One-Hot and Categorical Parallel Token Prediction Models from a trained autoregressive model. We then show how the latter can be self-distilled from data alone via Eq. (8).

2.2.1 Distillation

Both PTP variants can be trained to emulate the token-level predictions of an autoregressive teacher Q_φ, allowing for efficient, parallel generation of several tokens. We then call the PTP a student model P_θ. With enough data and model capacity, our algorithm leads to a student model that produces the same sequence of tokens in a single model call as the teacher does in high-latency autoregressive sampling (Theorems 1 and 2). We defer correcting errors arising from finite resources to the subsequent Section 2.3.

To train the student for a given training sequence $t_1, \ldots, t_T$, we reverse engineer the auxiliary variables $u_1, \ldots, u_N$ under which the teacher would have generated it, split the sequence into context and prediction sequences, and then evaluate a loss that leads the student towards the correct generation. This process is summarized in Algorithm 2 in Appendix H.1.

Auxiliary variables. First, we extract the auxiliary variables that the teacher model would use to generate the training sequence. We evaluate the teacher distributions of each training token to get the cumulative discrete distributions $F_1, \ldots, F_T$ for each token. Inverting Eq. (2), we find for every $k = 1, \ldots, T$:

$$u_k \in [F_{k,t_k-1}, F_{k,t_k}). \tag{9}$$

Since u_k is continuous, while t_k is discrete, we can randomly pick any compatible value. See Appendix D for details.

Sequence splitting. Second, we split the training sequence into a context part $t_1, \ldots, t_{i-1}$ and a prediction part $t_i, \ldots, t_T$. We usually choose the split index i at random and then predict a fixed number of parallel tokens.

Loss evaluation. Both O-PTP and C-PTP can be trained with cross-entropy given an input sequence t and a split index i, with u_k extracted using Eq. (9):

$$\text{O-PTP:} \quad \mathcal{L}(\theta; t, i) = -\sum_{k=i}^{T} \log P_\theta(t_k | t_{<i}, u_i, \ldots, u_{k-1}, u_k), \tag{10}$$

$$\text{C-PTP:} \quad \mathcal{L}(\theta; t, i) = -\sum_{k=i}^{T} \log P_\theta(t_k | t_{<i}, u_i, \ldots, u_{k-1}). \tag{11}$$

C-PTP can also be trained similar to knowledge distillation (Hinton et al., 2015): to this end, we explicitly match the student's $P_{\theta,k}$ to the teacher distribution $Q_{\varphi,k}$. This works for any loss $d(Q, P)$ for the difference between categorical distributions, such as the Kullback–Leibler divergence $d = \mathrm{KL}(Q \,\|\, P)$ or its reverse variant $d = \mathrm{KL}(P \,\|\, Q)$:

$$\mathcal{L}(\theta; t, i) = \sum_{k=i}^{T} d(Q_\varphi(t_k | t_{<k}), P_\theta(t_k | t_{<i}, u_{i \ldots k-1})). \tag{12}$$

Note that while different losses d have different convergence properties, crucially, $d = 0$ implies identical conditional distributions and a perfectly distilled model.

Training data. We can train the student using sequences from any data source, even if the teacher assigns different probabilities to them. As long as the teacher assigns non-zero probability and the dataset is sufficiently large, querying the teacher for the corresponding auxiliary variables provides full supervision and allows the student to match the teacher everywhere.

This gives us a large design space and we compare several options empirically in Appendix C.2. If our goal is to deploy our parallelized student as a drop-in replacement of our teacher model, the lowest-variance option is to sample training sequences from the teacher. Another possibility is to directly sample training sequences from a dataset, such as

the one that was used to train the teacher model in the first place. This has the advantage that we can compute the teacher predictions $Q_\varphi(t_k \mid t_{<k})$ in parallel over a full sequence instead of iteratively having to generate it. Finally, we can sample sequences directly from the student model by first sampling auxiliary variables $u_i, \ldots, u_T \sim \mathcal{U}[0,1]$ and then using our student model at its current state to sample training sequences in parallel. As the student's prediction gets closer to that of the teacher during training, this approaches the same training sequence distribution as if we had sampled the teacher directly.

Conceptually, the latter is similar to the techniques used in distilling the autoregressive WaveNet into a parallel model (Oord et al., 2018). In practice, following their approach, which equates to using a reverse KL loss as noted above, is efficient but proved to be too unstable. Parallel WaveNet stabilizes training with several auxillary losses, like a perceptual loss, which we don't have in our setting, in general.

2.2.2 Inverse Autoregressive Training

Categorical Parallel Token Prediction Models can also be trained directly via Eq. (8), avoiding the need to have a teacher model as target. For a given training sequence $t_1, \ldots, t_T$, we again split it into the context $t_{<i}$ and the following prediction $t_{\geq i}$.

Exactly as for distillation, we have to find auxiliary variables that are compatible with every $t_k, k \geq i$. We can do this by selecting, randomly, any $u_k \in [F_{k,t_k-1}, F_{k,t_k})$, equivalently to Eq. (9), where F_k, t_k now is the cumulative probability under P_θ (instead of the teacher model) to choose t_k when predicting that token. As this probability depends on the previous auxiliary variables $u_i, \ldots, u_{k-1}$, we select them iteratively. Specifically, we can alternate between computing the logits of $P_\theta(t_k \mid t_{<i}, u_i, \ldots, u_{k-1})$, and drawing u_k using equation 9.

Finally, we can train our model using the cross-entropy loss

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim P(t), i\sim P(i|t)}\left[-\sum_{k=i}^{N} \log P_\theta(t_k | t_{<i}, u_i, \ldots, u_{k-1})\right]. \tag{13}$$

Algorithm 3 in Appendix H.1 summarizes the procedure. A similar approach of iteratively determining latent variables (our auxiliaries) was proposed by Inverse Autoregressive Flows (Kingma et al., 2016), although they considered continuous variables that are traced through an invertible neural network.

2.3 Error Correction

Theorems 1 and 2 say that with enough model capacity, O-PTP and C-PTP can correctly predict coherent sequences of arbitrary length in one model call. Practically, finite model capacity limits the length at which a single transformer pass can produce coordinated text. In this section, we present block-autoregressive approaches that speed up token generation through parallel prediction while maintaining a identical output. This combines ideas from speculative decoding (Leviathan et al., 2023) with the PTP setups described in Section 2.1.

Intuitively, we only accept a sequence of tokens if it could have been generated exactly by a high-quality autoregressive model. In distillation settings, the autoregressive teacher serves as a verifier. When no teacher is available, we instead verify the sequence using the model itself, while never using more than one auxiliary variable at a time.

We begin by fixing a sequence of auxiliary variables $u_i, \ldots, u_N$, which fully determine a sample. Conditioned on these auxiliaries, PTP P_θ generates an entire token sequence $t_i, \ldots, t_N$ from the context $t_{<i}$ in a single forward pass (cf. Eqs. (5) and (7)).

To verify that this sequence is consistent with the autoregressive reference model Q_φ, we run that model in parallel on the generated tokens. For each position k, we check whether sampling from the reference model $Q_\varphi(t_k \mid t_{<k})$ using the same auxiliary variable u_k (as in Eq. (2)) would produce the same token as PTP. We accept all leading tokens for which this condition holds, and additionally accept one more token from the teacher at the first position where the models disagree. This is made explicit in Algorithm 1 in Appendix H.1.

We define the number of correct tokens #correct by counting the identically predicted tokens. Similarly, the number of accepted tokens also includes the token corrected by the teacher, so usually #accepted = #correct +1. See Appendix B for formal definitions.

If both PTP and the verification model have the same size, and we execute the two models naively in sequence, this results in a wall-time speedup of: $\#\text{accepted}/2 = (\#\text{correct}+1)/2$. In Section 3.3.1 we will see that smaller model sizes further increase the speedup. We discuss how to leverage additional computing resources to further decrease the total number of sequential model calls and avoid the overhead of verification in Appendix E.

3 Experiments

We empirically evaluate Parallel Token Prediction (PTP) using the number of accepted tokens per step under teacher verification (#accepted, see Section 2.3) as this is directly proportional to the maximum speedup that can be obtained with efficient decoding (Samragh et al., 2025). We first show that C-PTP can be trained from data alone without access to a teacher model (Section 3.2). We then distill a 1.1B-parameter model on code generation and demonstrate that O-PTP achieves larger speedups than autoregressive speculative decoding by enabling the draft model to predict multiple tokens per call (Section 3.3.1). On the same task, we show that conditioning on auxiliary variables yields substantially more correct tokens than independently predicting multiple tokens (Section 3.3.2). Finally, on a speculative decoding benchmark spanning diverse language tasks, we finetune a 7B-parameter model and show that O-PTP consistently outperforms competitive baselines, achieving state-of-the-art parallel decoding performance (Section 3.4).

3.1 Design Choices

We implement PTP as a lightweight modification of a standard transformer architecture by adding a few parameters for the embedding of auxiliary variables. Specifically, we adopt the following approach:

$$\text{embed}(u) = W\,\text{bits}(u) + b, \quad \text{where } \text{bits}(u) \in \{0,1\}^{32}. \tag{14}$$

Here, $\text{bits}(u)$ maps the IEEE-754 bits of the float32 number u into a 32-dimensional vector, resembling an arithmetic coding scheme (Witten et al., 1987). The linear layer (W, b) maps this binary vector to the embedding space of the autoregressive transformer. This embedding is then added to the positional encoding of the underlying network. We present the reasoning behind this choice through an ablation in Appendix C. It is performed on a dataset that predicts pick-up locations for taxis in New York City (NYC TLC, 2017).

For distilling a fast student model from a teacher, we train O-PTP with cross-entropy (Eq. (10)), sample training data from the teacher once and train multiple epochs on it. We again identify these choices through an ablation and practical considerations in Appendix C.

3.2 Inverse Autoregressive Training

As our first main experiment, we show that C-PTP can be trained directly from data. To this end, we follow the procedure in Section 2.2.2 and train a model from scratch on a dataset that predicts sequences of pick-up locations for taxis in New York City (NYC TLC, 2017).

Table 1 shows that the resulting model matches the performance of an autoregressive teacher. Based on the evidence for multi-token prediction (MTP) Gloeckle et al. (2024); DeepSeek-AI et al. (2025), we expect PTP our model to outperform autoregressive baselines at larger scales. We leave training larger models from scratch to future work. Next, we show how PTP already outperforms MTP frameworks in terms of inference speed.

3.3 Limitations of Competing Frameworks

Parallel Token Prediction overcomes important limitations in other frameworks to decrease latency: first, small surrogate models for speculative decoding do not leverage their parallel

Model	Perplexity (↓)
C-PTP (Ours)	19.88
Autoregressive	19.81

Table 1: C-PTP can be successfully trained from data alone. The table shows that the perplexity of Categorical Parallel Token Prediction (C-PTP) and an autoregressive model on taxi pickup location sequences (NYC TLC, 2017) are almost identical.

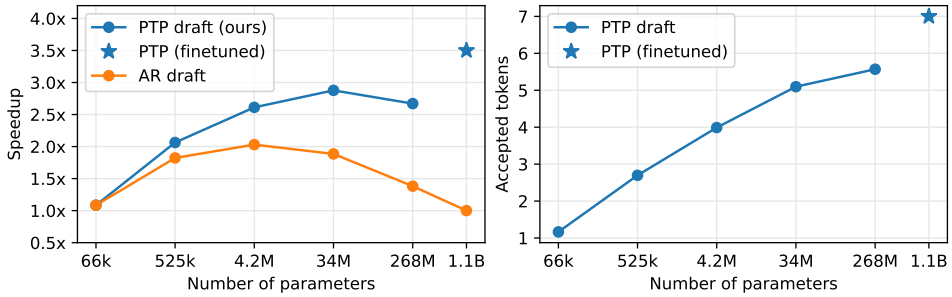

Figure 4: Speculative decoding with Parallel Token Prediction on code. The x-axis shows the draft-model parameter count. The left panel reports wall-clock speedup relative to standard autoregressive decoding, and the right panel shows the average number of tokens accepted per step under teacher verification. Curves compare autoregressive draft models to Parallel Token Prediction (PTP) drafts; (•) are models trained from scratch for a fixed number of epochs, and (⋆) is finetuned from the teacher. Across model sizes, PTP drafts achieve higher speedups by generating more than one correct token per step, whereas autoregressive drafts remain sequential. PTP allows parallelism in draft models, yielding larger speedups at equal model size.

multi-token potential, which we unlock with our framework in Section 3.3.1. Second, modeling tokens independently produces incoherent sequences (Section 3.3.2). This also limits the representational power of discrete diffusion models (Hoogeboom et al., 2021; Austin et al., 2021). We compare to speculative decoding and independent token prediction on CodeContests (Li et al., 2023), a dataset of coding challenges, by distilling TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2024).

3.3.1 Smaller Autoregressive Decoders

Speculative decoding distills small autoregressive drafting models to imitate the teacher with fewer parameters (Leviathan et al., 2023). Through their reduced size, these models require less compute to predict each token, and the teacher is used to verify them in parallel as in Section 2.3. This results in an overall speedup.

Figure 4 shows that training these smaller draft networks with our parallel token prediction framework further reduces latency, as it requires only a single call to the student model generating several tokens in parallel. The draft models for this experiment were trained to replicate TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2024) on CodeContests (Li et al., 2023). They are trained from scratch for a fixed number of epochs based on completions by that teacher. Details in Appendix H.

3.3.2 Independent Prediction

Many approaches to predicting multiple tokens in parallel, including multi-token prediction (Qi et al., 2020; Gloeckle et al., 2024) and discrete diffusion models (Hoogeboom et al., 2021; Austin et al., 2021), assume that future tokens are conditionally independent. As a result, later tokens are sampled from marginal distributions that average over incompatible earlier choices, making semantic and syntactic inconsistencies unavoidable even with infinite

Parallelization technique	#accepted (↑)
O-PTP (ours)	7.0 ± 0.1
Independent prediction	6.2 ± 0.1

Table 2: For code generation, speculative decoding with O-PTP accepts more tokens per step than independently predicting tokens. We distill TinyLlama-1.1B (Zhang et al., 2024) on coding problems (Li et al., 2022), once with meaningful auxiliaries that determine token dependence and once with a generic [MASK] token. Error is difference between two runs.

model capacity. In code generation, this manifests as spurious token combinations such as def numpy and import find. See Appendix G for a formal derivation.

To isolate this effect, we compare models trained with informative auxiliary variables u_i (our O-PTP) to models that have multiple independent heads for predicting multiple tokens (Qi et al., 2020; Gloeckle et al., 2024). As shown in Figure 5, O-PTP consistently produces meaningful token combinations by coordinating predictions through the auxiliary variables, while independent prediction frequently produces incompatible pairs. Quantitatively, Table 2 shows that auxiliary variables substantially increase the number of tokens that can be generated in a single model call. Figure 1 shows a qualitative sample of our model's predictions.

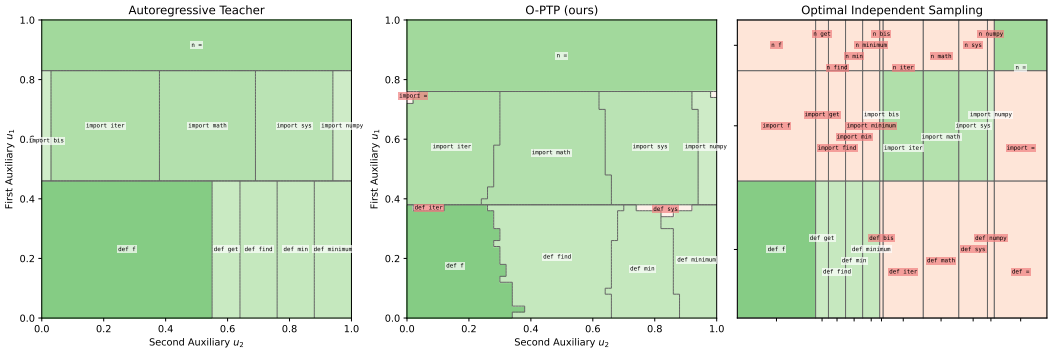

Figure 5: Parallel Token Prediction generates meaningful pairs of tokens. Each panel plots the first and second auxiliary variables $(u_1, u_2) \in [0, 1]^2$ on the axes; regions in this unit square correspond to the resulting two-token outputs (green: compatible, red: incompatible). (Left) In a coding problem, autoregressive sampling first selects one of def, import, or n, and then continues with meaningful predictions: a function name to declare, a package to import, or a variable assignment. (Center) Our code completion model produces similarly sensible token pairs, but in a single model call by coordinating predictions through the auxiliaries; only rarely ($< 1\%$) does it yield spurious combinations like def sys. (Right) A model that independently predicts future tokens is bound to fail: in about 60% of cases, it combines incompatible tokens because the second token is not informed about the first.

3.4 General-purpose Text Generation

In the following experiment, we confirm that a pretrained model can be finetuned to predict several tokens by training it with the PTP framework.

To this end, we distill a One-Hot Parallel Token Prediction model (O-PTP) student from a strong autoregressive teacher, following the distillation procedure described in Section 2.2.1. In our primary experiment, we use Vicuna-7B (Chiang et al., 2023) as the teacher and finetune the student on ShareGPT conversational data (Chen et al., 2024). Instead of fine-tuning a full model we train a gated LoRA (Samragh et al., 2025) adaptor, allowing us to make only the minimal changes that are necessary to correctly parse the auxiliary variables.

Parallelization	MTC	TL	SUM	QA	Math	RAG	Task-Average
O-PTP (ours)	4.52	3.73	4.54	3.42	4.62	4.29	4.18
SAMD (Hu et al., 2025)	4.62	3.12	4.18	3.38	4.16	3.95	3.90
Eagle-2 (Li et al., 2024a)	4.54	3.19	3.81	3.41	4.38	3.80	3.86
Hydra (Ankner et al., 2024)	3.90	2.88	2.95	3.21	3.90	3.37	3.37
Eagle (Li et al., 2024b)	3.66	2.74	3.15	2.81	3.57	3.21	3.19
Medusa (Cai et al., 2024)	2.70	2.18	2.12	2.24	2.67	2.26	2.36
SpS (Chen et al., 2023)	2.12	1.39	2.16	1.80	1.92	2.26	1.94
Recycling (Luo et al., 2025)	2.74	2.48	2.68	2.59	3.05	2.63	2.70
PLD (Saxena, 2023)	1.67	1.10	2.73	1.37	1.80	1.71	1.73
Lookahead (Fu et al., 2024)	1.69	1.24	1.54	1.56	1.92	1.42	1.56
None	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 3: Average number of accepted tokens under teacher verification (↑). For each method and task, we list the average length of the longest prefix matching the teacher's output, which directly determines decoding speedup. We finetune a One-Hot Parallel Token Prediction (O-PTP) model by distilling Vicuna-7B (Chiang et al., 2023) on ShareGPT (Chen et al., 2024) and compare it to concurrent parallelization techniques on the diverse tasks of SpecBench (Xia et al., 2024), including Multi-turn Conversation (MTC), Translation (TL), Summarization (SUM), Question Answering (QA), Mathematical Reasoning (Math, GSM8K), and Retrieval-augmented Generation (RAG). Errors indicate the standard deviation over three runs. Temperature is set to 0.7, and values are rounded to reflect statistical certainty. Across nearly all tasks, O-PTP achieves the highest number of accepted tokens.

To reflect a practical deployment scenario for a chat-oriented large language model, we evaluate the performance on SpecBench (Xia et al., 2024), containing a diverse set of tasks such as multi-turn conversation, translation, summarization, question answering, mathematical reasoning (GSM8K), and retrieval-augmented generation. Performance is measured using the average number of accepted tokens, as this is directly proportional to the maximum speedup that can be obtained with efficient decoding schemes.

As shown in Table 3, O-PTP consistently predicts significantly more tokens identical to its teacher than existing parallelization baselines across most tasks. These results establish a new state of the art on SpecBench, demonstrating that PTP remains effective when scaled to larger models and heterogeneous real-world datasets.

4 Related Work

Speeding up the generation of autoregressive models and discrete sequence models in particular has been the focus on a broad body of work, see (Khoshnoodi et al., 2024) for an overview.

Our framework combines two ideas from the Normalizing Flow literature and imports them to modeling discrete data: Inverse Autoregressive Flows (IAF) are trained with fast prediction in mind (Kingma et al., 2016) by iteratively identifying latent variables (our auxiliary variables) that generate a particular continuous one-dimensional value, and Free-Form Flows (FFF) train a generating function when a fast parallel sampler is not available (Draxler et al., 2024).

In the LLM literature, speeding up generation has been approached from various angles. Speculative decoding takes a system perspective, using a small draft model to propose multiple tokens and a large target model to verify them (Leviathan et al., 2023; Chen et al., 2023). Variants verify entire sequences (Sun et al., 2023) or use a smaller verifier network Zhong et al. (2025) to improve quality and speed. Latent variable methods first sample latent codes from the prompt so that the distribution of subsequent tokens factorizes given latent codes (Gu et al., 2018; Ma et al., 2019). Diffusion language models leave autoregressive sampling behind by iteratively refining the text starting from a noisy or masked variant

(Hoogeboom et al., 2021; Austin et al., 2021). Multi-head output models predict several next tokens independent of each other (Qi et al., 2020; Gloeckle et al., 2024; DeepSeek-AI et al., 2025), narrowing down on the possible set of next tokens. Both discrete diffusion and multi-head models assume independence of tokens, which is fundamentally limited in modeling capacity, compare Section 3.3.2 and Feng et al. (2025). Recent work on discrete diffusion models leverage copula models to capture dependencies in an additional model (Liu et al., 2025). We model dependencies between several tokens without an additional model at inference time.

In contrast to the above, our work introduces a new class of fast language models that are universal in the sense that can approximate arbitrary dependence between several tokens in a single model call. Our new method is complementary to existing approaches, and we leave exploring these combinations open for future research.

5 Conclusion

In this paper, we introduce Parallel Token Prediction, a framework that permits consistent generation of several tokens in a single autoregressive model call. It eliminates the independence assumptions that limited prior approaches, allowing to model multiple tokens with arbitrary dependency between them. Empirically, we show that existing models can be distilled into efficient parallel samplers. With error correction, these models produce identical output as a teacher while significantly increasing how many tokens are obtained per model call.

This speedup makes language models more practical for real-time applications. Future work includes extending our framework to large scale models, multimodal generation, combining it with complementary acceleration strategies, and exploring theoretical limits on parallelization.

Overall, our results suggest that the sequential bottleneck in autoregressive transformers is not inherent, and that universal, efficient parallel generation is within reach.

We think that our experiments in Section 3 only scratch the surface of the possibilities enabled by our Theorems 1 and 2. Replicating a teacher model limits the performance achievable by the student. We envision future work to train large models from scratch that think in long sequences, and conjecture that this will enable new reasoning capabilities.

Acknowledgments

Sameer Singh acknowledges funding from the National Science Foundation (NSF) through an NSF CAREER award IIS-2046873. Stephan Mandt acknowledges funding from the National Science Foundation (NSF) through an NSF CAREER Award IIS-2047418, IIS-2007719, the NSF LEAP Center, and the Hasso Plattner Research Center at UCI, and the Chan Zuckerberg Initiative. This project was supported by the Chan Zuckerberg Initiative, and the Hasso Plattner Research Center at UCI.

Ethics Statement

Our work focuses on reducing the inference time of Large Language Models, enabling more computations per unit time and supporting large-scale or real-time applications. While this can improve responsiveness and resource efficiency, it may also increase the potential for misuse, such as generating misinformation or automated spam at higher volumes. Faster inference does not mitigate underlying model biases, so responsible deployment, monitoring, and safeguards are critical to balance performance gains with societal risks.

Reproducibility Statement

We include proofs for all theoretical results introduced in the main text in Appendix A. We include further experimental and implementation details (including model architectures and other hyperparameter choices) in Section 3.1 and Appendix H. Our code will be made available by the time of publication.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. arXiv preprint arXiv:2402.05109, 2024.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. Advances in neural information processing systems, 34:17981–17993, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), Advances in neural information processing systems, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. arXiv preprint arXiv:2401.10774, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. arXiv preprint arXiv:2302.01318, 2023.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In European Conference on Computer Vision, pp. 370–387. Springer, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q.

Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. DeepSeek-V3 Technical Report, February 2025. URL http://arxiv.org/abs/2412.19437.

Felix Draxler, Peter Sorrenson, Lea Zimmermann, Armand Rousselot, and Ullrich Köthe. Free-form Flows: Make Any Architecture a Normalizing Flow. In Artificial Intelligence and Statistics, 2024.

Felix Draxler, Yang Meng, Kai Nelson, Lukas Laskowski, Yibo Yang, Theofanis Karaletsos, and Stephan Mandt. Transformers for mixed-type event sequences. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025. URL https://openreview.net/forum?id=MtwsRjPZhf.

William Falcon and The PyTorch Lightning team. PyTorch lightning, March 2019.

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025. URL https://openreview.net/forum?id=fGBCRZQVse.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. arXiv preprint arXiv:2402.02057, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In Proceedings of the 41st international conference on machine learning, pp. 15706–15734, 2024.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In International conference on learning representations, 2018. URL https://openreview.net/forum?id=B1l8BtlCb.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In International conference on machine learning, pp. 1321–1330. PMLR, 2017.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825): 357–362, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In International conference on learning representations, 2020.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), Advances in Neural Information Processing Systems, 2021. URL https://openreview.net/forum?id=6nbpPqUCIi7.

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

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. Sam decoding: Speculative decoding via suffix automaton. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 12187–12204, 2025.

J. D. Hunter. Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007.

Mahsa Khoshnoodi, Vinija Jain, Mingye Gao, Malavika Srikanth, and Aman Chadha. A Comprehensive Survey of Accelerated Generation Techniques in Large Language Models, May 2024. URL http://arxiv.org/abs/2405.13019. arXiv:2405.13019 [cs].

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR), 2015.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), Advances in neural information processing systems, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/ddeebdeefdb7e7e7a697e1c3e3d8ef54-Paper.pdf.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Proceedings of the 40th international conference on machine learning, volume 202 of Proceedings of machine learning research, pp. 19274–19286. PMLR, July 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, J Li, J Chim, and others. StarCoder: May the source be with you! Transactions on machine learning research, 2023. Publisher: OpenReview.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. arXiv preprint arXiv:2406.16858, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. arXiv preprint arXiv:2401.15077, 2024b.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. Science, 378(6624): 1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Feng Lin, Hanling Yi, Yifan Yang, Hongbin Li, Xiaotian Yu, Guangming Lu, and Rong Xiao. Bita: Bi-directional tuning for lossless acceleration in large language models. Expert Systems with Applications, 279:127305, 2025.

Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=FXw0okNcOb.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations (ICLR), 2019.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. Turning trash into treasure: Accelerating inference of large language models with token recycling. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 6816–6831, 2025.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In Proceedings of the 2019 conference on empirical methods in natural language processing, Hong Kong, November 2019.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman (eds.), 9th Python in Science Conference, 2010.

New York City Taxi and Limousine Commission. 2016 yellow taxi trip data, 2017. City of New York, OpenData portal.

Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. In International conference on machine learning, pp. 3918–3926. PMLR, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, 2019.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. ProphetNet: Predicting future n-gram for sequence-to-SequencePre-training. In Trevor Cohn, Yulan He, and Yang Liu (eds.), Findings of the association for computational linguistics: EMNLP 2020, pp. 2401–2410, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.217. URL https://aclanthology.org/2020.findings-emnlp.217/.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. OpenAI, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI Technical Report, 2019.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. arXiv preprint arXiv:2507.11851, 2025.

Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. SpecTr: Fast speculative decoding via optimal transport. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), Advances in neural information processing systems, volume 36, pp. 30222–30242. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6034a661584af6c28fd97a6f23e56c0a-Paper-Conference.pdf.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel Bikel, Lukas Blecher, Nikolay Bogoychev, William Brannon, Anthony Brohan, Humberto Caballero, Andy Chadwick, Jenny Lee, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.

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

Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic Coding for Data Compression, volume 30. Communications of the ACM, 1987.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771, 2019.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. arXiv preprint arXiv:2401.07851, 2024.

Siqiao Xue, Xiaoming Shi, Zhixuan Chu, Yan Wang, Hongyan Hao, Fan Zhou, Caigao Jiang, Chen Pan, James Y. Zhang, Qingsong Wen, Jun Zhou, and Hongyuan Mei. EasyTPP: Towards open benchmarking temporal point processes. In International conference on learning representations (ICLR), 2024. URL https://arxiv.org/abs/2307.08097.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model, 2024. arXiv: 2401.02385 [cs.CL].

Meiyu Zhong, Noel Teku, and Ravi Tandon. Speeding up speculative decoding via sequential approximate verification. In ES-FoMo III: 3rd workshop on efficient systems for foundation models, 2025. URL https://openreview.net/forum?id=Y4KcfotBkf.

A Proofs

A.1 Proof of Theorem 1

Proof. By Theorem 2, it holds that the distribution of token $t_k, k \geq i$ is fully determined by $t_1, \ldots, t_{k-1}$ and $u_i, \ldots, u_{k-1}$, showing that the categorical distribution P_k of token t_k is fully determined.

Thus, the function to compute token t_k is given by Eq. (2):

$$t_k = f_P(t_1, \ldots, t_{i-1}; u_i, \ldots u_k) = \text{Pick}(u_k; P_k). \tag{15}$$

□

A.2 Proof of Theorem 2

Proof. We prove by induction over $k, k \geq i$.

For $k = i$, there is nothing to show, since there are no auxiliaries involved in the statement.

For $k \mapsto k + 1$, assume the statement holds for k. This gives us access to the distribution P_k of the token t_k. Since token t_k is uniquely determined from P_k and u_k via Eq. (3), any distribution conditioning on P_k, t_k can instead condition on P_k, u_k via the law of total probability. □

B Average number of correct tokens

To quantify the effectiveness of parallel token generation, we measure the number of correctly predicted tokens. Given a student output sequence $t_1^{\text{S}}, t_2^{\text{S}}, \ldots, t_N^{\text{S}}$ and a teacher output sequence $t_1^{\text{T}}, t_2^{\text{T}}, \ldots, t_N^{\text{T}}$ of the same length, this metric is defined as

$$\#\text{correct} = \max\left\{k \;\middle|\; t_j^{\text{S}} = t_j^{\text{T}} \;\; \forall\, j \leq k\right\}. \tag{16}$$

$$\#\text{accepted} = \min\{\#\text{correct} + 1, N\}. \tag{17}$$

That is, #correct equals the length of the longest prefix of the student's output that exactly matches the teacher's output. Intuitively, this measures how many tokens the student can generate in parallel before the first disagreement, and directly corresponds to the number of tokens that can be safely accepted without error correction.

C Additional Ablation Results

Based on a dataset (NYC TLC,) that contains latitudes and longitudes for pick-up locations for all taxi rides in 2016, we divide the city into 25 neighborhoods via k-Means clustering to obtain a discrete-valued time-series that we can split into overlapping chunks of length N. This is a common benchmark dataset in the literature of marked temporal point processes (Xue et al., 2024), and autoregressive transformers are a common architecture (Draxler et al., 2025).

We now provide some of the specific choices we made when implementing the general framework of Parallel Token Prediction. Specifically, we discuss the empirical difference between O-PTP and C-PTP and which specific loss to choose. We will specify our model architecture and how to embed both tokens and auxiliary variables in the same embedding space, and lastly compare the proposal distributions our training sequences can be sampled from.

We test our framework by training a model that predicts pick-up locations for taxis in New York City. Based on a dataset (NYC TLC, 2017) that contains latitudes and longitudes for pick-up locations for all taxi rides in 2016, we divide the city into 25 neighborhoods via k-Means clustering to obtain a discrete-valued time-series that we can split into overlapping chunks of length N. This is a common benchmark dataset in the literature of marked temporal point processes (Xue et al., 2024).

Proposal Distribution $P(t)$	kl	kl-rev	bce	ce	MTP
Teacher	40	41	**45**	44	10.1
Student	44	39	**45**	**45**	10.1
Dataset	29	36	44	43	10.1

Table 4: Our framework is compatible with several losses. Average number of correct tokens (↑) on the taxi dataset, evaluated on 16000 samples. O-PTP are distilled with KL or reverse KL loss (kl, kl-rev), C-PTP with binary or categorical cross entropy loss (bce, ce). Independent prediction (MTP) (Gloeckle et al., 2024) achieves 10.1. Numbers rounded to reflect level of statistical certainty.

As a teacher model, we pretrain a 29M-parameter autoregressive causal transformer based on the architecture of GPT-2 (Radford et al., 2019), using the cross-entropy loss in Eq. (8). For our PLM we choose the same GPT-style transformer architecture as the teacher. This allows us to use the teacher's parameters as a warm-start. We evaluate all our parallel models in terms of the average number of leading tokens predicted by our student model that are identical to the teacher. In the end, this is the quantity that limits the maximum latency reduction that can be achieved, see Section 2.3.

C.1 Auxiliary Variable Embeddings

In our experiments we use transformers that embedded tokens into a higher-dimensional embedding space via a learned embedding before adding a positional embedding. This doesn't work out-of-the box for our auxiliary variables since they are one-dimensional continuous variables. Thus we learn a separate embedding. We combine two components, for each of which we test several variants: (1) A learned affine linear transform [lin] or a fully connected neural network [NN]. (2) Feed either the scalar u [fl], a n-dimensional threshold-embedding $e_i = 1\{u \leq i/n\}$ [th], or an n-dimensional embedding $e_i = 1\{u2^{i-1} \bmod 1 \leq 0.5\}$ [ar] inspired by arithmetic coding (Witten et al., 1987).

Empirically, all methods work reasonably well, but a structured embedding leads to faster and more stable training convergence. This is similar to the transformer's positional embedding were both learned and fixed embeddings work well but the later is preferred in practice (Vaswani et al., 2017). For further experiments we use the [ar + lin] embedding. Table 5 shows the detailed effect of different embedding strategies.

C.2 Distillation Losses and Proposal Distributions

As our framework is deliberately general, it is compatible with a wide selection of losses. We here compare the distillation losses (Section 2.2.1), focusing on KL and cross-entropy losses in Eqs. (10) and (12). Specifically the KL loss (kl), reverse KL loss (kl-rev), binary cross-entropy loss (bce), and categorical cross-entropy loss (ce). During training we sample training sequences from a dataset and continuations $t_{\geq i}$ either from the teacher model Q_φ, the student model P_θ, or directly from a dataset. Table 4 shows the results for different losses. Empirically we note, that O-PTPs are easier to train than C-PTPs and achieve a higher number of average correct tokens. This is most likely due to the fact that O-PTPs do not have to predict the full token distribution accurately, which includes tail behavior, as long as they learn which token is the most likely given the auxiliary variable. In the following, we choose to sample training sequences from the teacher model for best results.

D Sampling of Auxiliary Variables

Our framework conditions, for a prompt $t_{<i}$, not on token t_k directly but on the auxiliary variable $u_k \in [F_{k,t_k-1}, F_{k,t_k})$ that contains the same information. During inference we sample $u_k \sim \mathcal{U}[0,1]$ as to not bias our predictions. During training on the other hand, we have more flexibility and can sample the permissible interval using $u_k = F_{k,t_k-1} + \tilde{u}_k\,[F_{k,t_k} - F_{k,t_k-1}]$, where $\tilde{u}_k \sim \mathrm{Beta}(b,b)$. For $b = 1$ this simplifies to a uniform distribution

Model	fl + NN	th + lin	th + NN	ar + lin	ar + NN	MTP
O-PTP	35.9	40.9	39.1	45.4	**46.1**	10.1
C-PTP	28.8	36.8	36.6	**40.4**	35.7	10.1

Table 5: Structured embeddings of auxiliary variables u_k are more stable than fully-learned embeddings. Average number of correct tokens (↑) on the taxi dataset, evaluated on 16000 samples. Trained using the KL loss (C-PTP) and binary cross-entropy loss (O-PTP), respectively. Independent prediction (MTP) (Gloeckle et al., 2024) achieves 10.1. Numbers rounded to reflect level of statistical certainty.

b	2	1	0.5	MTP
O-PTP	12.9	**13.9**	13.8	8.4
C-PTP	13.6	**13.8**	13.5	8.4

Table 6: Different sampling strategies for u_k are available. Average number of correct tokens (↑) for $\tilde{u}_k \sim \text{Beta}(b, b)$ on the taxi dataset, evaluated on 16000 samples, with $N = 16$. Trained using the KL loss (C-PTP) and binary cross-entropy loss (O-PTP), respectively. Independent prediction (MTP) (Gloeckle et al., 2024) achieves 8.4. Numbers rounded to reflect level of statistical certainty.

while $b \neq 1$ puts more or less weight on predictions that land closer to the border of the permissible interval and thus are more difficult to predict. Training results for different values of b can be found in Table 6. Empirically, we find that while the choice of b does not seem to effect the final average number of correct samples, a larger b might speeds up the earlier stages of training while a smaller b might yield slightly better sample quality during inference, as measured by model perplexity.

E Abundant Computational Resources

If reducing latency is more important than total compute, we can already start predicting more tokens by another PTP call, for example, by prematurely accepting the first c tokens, while the teacher has not yet verified the predicted sequence. If the first c tokens are confirmed to be correct, we select the produced sequence and repeat the process. If we do this with many different offsets in parallel, we minimize the chances that none of the generated sequences will be selected, and can completely avoid the overhead of error correction. In practice, this may be implemented using multiple GPUs or in one GPU with appropriately constructed attention masks (Samragh et al., 2025; Li et al., 2024a; Lin et al., 2025).

Another way to leverage several models run at once is to use them to improve the expected number of correct tokens directly. Specifically, for a fixed context we can let M PTPs compute M independent predictions using independently drawn auxiliary variables $u_{i,m}, \ldots, u_{i+N,m}$. By choosing the best prediction, i.e. the one that gives us the best chance of a higher number of correct tokens, we can improve latency further.

Crucially, we have to choose the best prediction in a way that doesn't bias the marginal distribution over future tokens. If we, for example, naively choose the sequence that is correct for the most amount of tokens, we will bias our prediction towards sequences that are easier to predict. On way to achieve bias-free improvements is to pick the set of auxiliary variables that lands, on average, closest to the center of a token's valid interval $I_k(t_k) = [F_{k,t_k}, F_{k,t_k-1})$ where F_{k,t_k} is the cumulative probability under Q_φ to choose t_k when predicting that token. Specifically, choose

$$\text{argmax}_m \sum_{k=i}^{i+N} \left| \frac{u_{k,m} - F_{k,t_{k,m}}}{F_{k,t_{k.m}-1} - F_{k,t_{k,m}}} - \frac{1}{2} \right|. \tag{18}$$

This does not bias the marginal distribution but does bias the distribution of the selected u_k to be closer to the center of its interval $I_k(t_k)$. making the prediction less prone to small

M	1	4	16	64	256	1024	10^6	∞
Avg. correct tokens	45.36	49.67	54.76	55.74	56.91	59.79	65.42	90.17

Table 7: Additional compute increases correctness. Average number of correct tokens (↑) for M O-PTPs running in parallel on the taxi dataset, with sequence length $N = 100$.

differences in the teacher's and student's logits. In the limit $M \to \infty$ we always select the middle point of $I_k(t_k)$, yielding an upper bound to the possible improvement. Table 7 shows the performance gains on the taxi dataset.

We can combine both techniques, avoiding the additional latency of verification while still keeping the higher expected number of correct tokens. Because the selection in Eq. (18) relies on the teacher logits it can only be made after the verification step. To avoid waiting for the verification we assume, as before, that after a model call one of $n = 1 \dots S$ tokens are correct and pre-compute the future tokens based on this assumption. Instead of one call as before, we now have to make M-many calls for each n. After the verification step we discard all but the best call from the correct n^*. As we have to repeat this for all M viable calls, that are yet to being verified, in parallel this approach benefits from M^2S PTPs running in parallel.

F Restricted Computational Resources

Limiting the number N of token's our PTP predicts at once to a smaller number will reduce the total number of floating point operations, increasing energy efficiency. This, of course, negatively effects the possible latency gains, especially since N is an upper bound on the average number of correct tokens. Table 8 shows the result for different values of N on the taxi dataset.

N	1	2	4	8	16	64	100	∞
Avg. correct tokens	1.00	1.99	3.93	7.59	13.90	36.60	45.36	48.76
MTP	1.00	1.91	3.53	5.86	8.40	10.08	10.07	10.20

Table 8: Less compute decreases correctness. Average number of correct tokens (↑) for limited number of predicted tokens N per O-PTP call on the taxi dataset, $M = 1$.

G Formal limitations of independent prediction

Without the auxiliary variables, the distribution of all tokens $k > i + 1$ is not informed about the choice we make for tokens $t_{i+1}, \dots t_{k-1}$. Instead, they are marginalized out, which explains the spurious code snippets:

$$\text{Independent: } P(t_k|t_{<i}) = \sum_{t_i,\dots t_{k-1}} P(t_k|t_{<k})P(t_i,\dots,t_{k-1}|t_{<i}) \neq P(t_k|t_{<k}). \tag{19}$$

$$\text{Dependent (ours): } P(t_k|t_{<i}, u_i, \dots u_k) \overset{\text{thm. 2}}{=} P(t_k|t_{<k}). \tag{20}$$

This limits how many tokens can be sampled in parallel without auxiliary variables, even for infinite model capacity. Our Theorems 1 and 2 allow for coordinating tokens, making model capacity the only restriction.

H Experimental Details

H.1 Algorithms

Algorithm 2 shows how to distill a PTP from a teacher, Algorithm 3 shows how to train directly from data.

Algorithm 1 Sampling with error correction

Require: Prompt $t_1, \ldots, t_i$, one-hot or categorical PTP P_θ, verification model $Q_\varphi(t)$.
  Sample $u_k \sim \mathcal{U}[0,1]$ for all $k \geq i$.
  while $i < T$ do
    if P_θ is one-hot PTP then
      $P_k \leftarrow P_\theta(t_k | t_{<i}, u_i, \ldots, u_k)$, jointly for all $k \geq i$. ▷ Student one-hot distributions
      $t_k = \text{argmax}_l P_{kl}$, for all $k \geq i$. ▷ Eq. (5)
    else
      $P_k \leftarrow P_\theta(t_k | t_{<i}, u_i, \ldots, u_{k-1})$, jointly for all $k \geq i$. ▷ Student categorical distributions
      $t_k = \text{Pick}(u_k, P_k)$, for all $k \geq i$. ▷ Eq. (7)
    end if
    $Q_k \leftarrow Q_\varphi(t_k | t_{<k})$, jointly for all $k \geq i$. ▷ Teacher categorical distributions
    $\tilde{t}_k \leftarrow \text{Pick}(u_k, Q_k)$, for all $k \geq i$. ▷ Teacher tokens
    $i \leftarrow \min_{k>i}\{k : t_k \neq \tilde{t}_k\} - 1$, or N if no mistake was made. ▷ Compare cf. Eq. (16)
    if #correct $< N$ then
      $t_i \leftarrow \tilde{t}_i$. ▷ All together: #accepted = #correct + 1
      $i \leftarrow i + 1$
    end if
  end while

Algorithm 2 Training PTP (distillation)

Require: Sequence proposal distribution $P(t)$ (teacher, student, dataset, or combination), cutoff distribution $P(i|t)$, teacher model $Q_\varphi(t)$, one-hot or categorical PTP P_θ.
  while not converged do
    Sample $t \sim P(t)$
    $P_k = Q_\varphi(t_k \mid t_{<k})$ in single model call.
    Sample $u_k \in [F_{k,t_k-1}, F_{k,t_k})$.
    Sample $i \sim P(i|t)$.
    Compute $\nabla_\theta \mathcal{L}(\theta, t)$ using one of Eqs. (10) to (12).
    Gradient step.
  end while

H.2 Training details

The model used in Section 3.2 and Appendix C is a GPT-2–style transformer language model with 4 transformer layers, a hidden size of 1536, and approximately 29 million trainable parameters. Each layer follows the standard GPT-2 architecture, consisting of multi-head self-attention and position-wise feedforward sublayers, combined with residual connections and layer normalization. The vocabulary size is set 25. Unless otherwise noted, all other hyperparameters and initialization schemes follow the original GPT-2 specification (Radford et al., 2019). During training and inference of our student model we don't provide any context and evaluate the correctness of the next $N = 100$ tokens, by comparing $Q_\varphi(t_k \mid t_{<k})$ and $P_\theta(t_k)$. For results on a smaller $N = 16$, see Appendix F. We train every model for 150k steps with a batch size of 32 with the Adam optimizer (Kingma & Ba, 2015) and learning rate 0.0001.

The teacher model used in Section 3.3 is a dialogue-tuned variant of the TinyLlama (Zhang et al., 2024) 1.1 billion parameter model, adopting the same architecture and tokenizer as LLaMA 2 (Touvron et al., 2023): TinyLlama-1.1B-Chat-v1.0. The model uses a transformer architecture comprising 22 transformer layers, each with standard multi-head self-attention, SwiGLU feedforward blocks, residual connections, and layer normalization. The embedding and hidden dimension is 2048, and the intermediate (feedforward) dimension is 5632, consistent with a LLaMA-style scaling. The vocabulary size is $32,000$. The parameters are available via https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0.

Algorithm 3 Training PTP (inverse autoregressive)

Require: Dataset $P(t)$, cutoff distribution $P(i|t)$, categorical PTP P_θ.
  while not converged do
    Sample training sequence $t_1, \ldots, t_N$.
    Sample split position $i \in \{1, \ldots, N\}$.
    for $k = i, \ldots, N$ do
      $P_k = P_\theta(t_k \mid t_{<i}, u_i, \ldots, u_{k-1})$, with auxiliary available form previous iterations.
      Sample $u_k \in [F_{k,t_k-1}, F_{k,t_k})$.
    end for
    Compute $\nabla_\theta \mathcal{L}(\theta, t)$ using Eq. (13).
    Gradient step.
  end while

We train an O-PTP on predicting 16 additional tokens. We adopt the strategy of finetuning using gated low-rank adaptation, where changed parameters are only applied for completion tokens (Samragh et al., 2025). We choose a LoRA (Hu et al., 2022) rank of $r = 256$, although comparable speedups can be achieved already with $r = 64$. We train every model for steps with a batch size of 56 with the AdamW optimizer (Loshchilov & Hutter, 2019) on Eq. (10) and learning rate 0.0001 for 1.3M steps. We generate training and validation data by generating Python code completions on CodeContests (Li et al., 2022) from the teacher, splitting generated sequences between randomly into input and completion. We use a teacher sampling temperature of 0.7, top-$k = 50$ and top-$p = 0.9$, as is recommended for this model.

For the MTP baseline, we use Eq. (11) with uninformative us in otherwise identical code for a fair comparison.

For the scaling experiments in Section 3.3.1, we parameterize models of different scales as in Table 9. We adopt the same architecture choices as TinyLlama.

For the natural language tasks on SpecBench (Xia et al., 2024), we finetune Vicuna-7B-v1.5 (Chiang et al., 2023) on ShareGPT (Chen et al., 2024). We train it with the same gated $r = 128$ LoRA setup as for TinyLlama, but with batch size 64 and for 1M steps.

Model Size	Hidden Size	Hidden Layers	Attention Heads
4k	16	1	1
66k	64	1	1
525k	128	2	2
4.2M	256	4	4
34M	512	8	8
268M	1024	16	16
1.1B	2048	22	32

Table 9: Model configuration by parameter scale for scaling experiment

We base our code on transformers (Wolf et al., 2019), PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (McKinney, 2010) for data evaluation.

H.3 Prompt for Figure 5

```
You are given a permutation p_1, p_2, ..., p_n.

In one move you can swap two adjacent values.

You want to perform a minimum number of moves, such that in the end there
     will exist a subsegment 1,2,..., k, in other words in the end there
    should be an integer i, 1 <= i <= n-k+1 such that p_i = 1, p_{i+1} =
    2, ..., p_{i+k-1}=k.
```

```
6
7  Let f(k) be the minimum number of moves that you need to make a
       subsegment with values 1,2,...,k appear in the permutation.
8
9  You need to find f(1), f(2), ..., f(n).
10
11 Input
12
13 The first line of input contains one integer n (1 <= n <= 200 000): the
       number of elements in the permutation.
14
15 The next line of input contains n integers p_1, p_2, ..., p_n: given
       permutation (1 <= p_i <= n).
16
17 Output
18
19 Print n integers, the minimum number of moves that you need to make a
       subsegment with values 1,2,...,k appear in the permutation, for k=1,
       2, ..., n.
20
21 Examples
22
23 Input
24
25
26 5
27 5 4 3 2 1
28
29
30 Output
31
32
33 0 1 3 6 10
34
35
36 Input
37
38
39 3
40 1 2 3
41
42
43 Output
44
45
46 0 0 0
```