Accelerating Large Language Model Decoding with Speculative Sampling
&
Speculative Sampling for Temporal Point Processes

Week 5: June 27th to July 3rd

Problem

Sampling (i.e., decoding) from Large transformer models (500B+ parameters) is highly inefficient and costly:

- auto-regressive sampling: **sequential** and cannot benefit from parallel hardware
  - i.e., generating one token at a time, conditioned on all previous tokens
- sampling is **memory-bandwidth bound** , not compute-bound
  - KV (Key-Value) cache size; model params
- each token requires **model forwarding;** larger models worsen latency (How long does it take to get the output for one input?) due to **communication overhead** across accelerators
- **redundant calls** to large models:
  - even when the next token is obvious – waste compute

Related works & Limitations

Aim: **reduce sampling latency** for large transformers and auto-regressive models

- Quantization & Distillation
  - lower memory use and computation without degrading performance
- Cache & Attention Optimization
  - multi-query attention (Shazeer, 2019) reduce cache size by reusing the same keys/values across heads
- Hardware and Serving Efficiency
  - model parallelism and TPU-specific techniques (Pope et al., 2022)

Limitation of the above: do not address **auto-regressive latency bottlenecks directly**, especially in **large-scale, distributed** settings

- Parallel Sequence Generation
  - Greedy sampling: block parallel sampling (Stern et al., 2018) and aggressive decoding (Ge et al., 2022)
  - Other domains: images (e.g., Song et al., 2021) or autoregressive transformers (e.g., Wiggers & Hoogeboom, 2020)
  - limitation: either task-specific, greedy-only, or not yet scaled to massive distributed settings

Observation & its analysis

observation: Computing logits for a short continuation of K tokens in parallel takes roughly the same time as generating a single token

Reason:

- **Linear Layers**:
  - For *small batches*, matrix multiplications (e.g., attention projection) are memory-bound and not compute-bound
  - Increasing the number of tokens K slightly doesn't significantly increase latency
- **Attention Mechanism:**
  - KV-cache (key/value cache) stores intermediate attention values from previous tokens.
  - Since the cache size doesn't grow when scoring K tokens in parallel, hence the memory bandwidth cost stays almost the same
- **All-Reduce Operations:**
  - In distributed setups, each layer requires syncing activations across accelerators
  - For *small K*, this sync happens anyway, so transmitting more tokens doesn't increase latency much.

Proposed solution

Speculative Sampling (SpS): **speed up transformer decoding** while preserving the **distributional correctness** of the target model

Three main steps:

- Draft Generation (Fast Model)
  - Draft model: Generate a sequence of **K draft tokens** using a **smaller or faster model**
  - can be done either in **parallel** (e.g., block sampling) or using a faster **auto-regressive** model K times
- Scoring with Target Model (Accurate Model)
  - Target model: Score the draft sequence using a **larger, more accurate model**
  - target model provides **logits (probabilities;** Unnormalized scores over the vocabulary for a predicted token**)** for each drafted token
- Modified Rejection Sampling
  - Use a **rejection sampling scheme** to determine whether to accept each drafted token
  - Scoring: Tokens are accepted **from left to right** based on how closely the draft's predictions match those of the target
  - ensures that the final sampled output still follows the **distribution of the target model**
    - If accepted, the token is kept; if not, the next draft starts **after the last accepted or resampled token, then do scoring again**

Algorithm

Setting:

- x: a token proposed by the **draft model** (sampled from distribution p)
- q(x): probability of token x according to the **target model**
- p(x): probability of token x according to the **draft model**
- **Rejection sampling scheme**
  - Token acceptance mode:

$$\min\left(1, \frac{q(\tilde{x}_{n+1}|x_1, \ldots, x_n)}{p(\tilde{x}_{n+1}|x_1, \ldots, x_n)}\right)$$

  - If q(x)>p(x): target likes this token more than draft → always accept.
  - If q(x)<p(x): target is less confident → accept with probability q(x)/p(x).

=> We *sample* from an easy distribution p(x) , then we accept or reject that sample so the resulting **final accepted distribution matches** the *true target distribution* q(x)

- What if rejected:

$$x_{n+1} \sim (q(x|x_1, \ldots, x_n) - p(x|x_1, \ldots, x_n))_+ \qquad (f(x))_+ = \frac{\max(0, f(x))}{\sum_x \max(0, f(x))}$$

  - Sample from the positive part of the **difference between the target and draft** distributions.
  - Then normalize it (so it's a valid distribution: sums to 1)

=> Only sample tokens that the **target model believes in more than the draft**, and do so proportionally to that surplus.

Algorithm 2 Speculative Sampling (SpS) with Auto-Regressive Target and Draft Models

Given lookahead K and minimum target sequence length T.
Given auto-regressive target model $q(.|.)$, and auto-regressive draft model $p(.|.)$, initial prompt sequence $x_0, \ldots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ **do**
  for $t = 1 : K$ **do**
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $K + 1$ sets of logits from drafts $\tilde{x}_1, \ldots, \tilde{x}_K$:

$$q(x|, x_1, \ldots, x_n),\ q(x|, x_1, \ldots, x_n, \tilde{x}_1),\ \ldots,\ q(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_K)$$

  for $t = 1 : K$ **do**
    Sample $r \sim U[0, 1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(x|x_1,\ldots,x_{n+t-1})}{p(x|x_1,\ldots,x_{n+t-1})}\right)$, **then**
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$.
    else
      sample $x_{n+t} \sim (q(x|x_1, \ldots, x_{n+t-1}) - p(x|x_1, \ldots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \ldots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \ldots, x_n, x_{n+K})$ and set $n \leftarrow n + 1$.
end while

e.g. say the first t−1 tokens are accepted, token x_{n+t} rejection happens: resample using the rejection mode to get x_{n+t}; Update n ← n + t; Then go **back to the top of the while loop:** A **new draft of K tokens** will be generated from the draft model p(·), **starting from the current full sequence context**: x0,⋯,x_{n+t}

Background & challenges & Main objective

- Background:
  - **Temporal Point Processes (TPPs)** are generative models used to represent **sequential events that occur irregularly over time**
  - generation **sequential: auto-regressive (AR) model**, which predicts the **next event** (time + type) based on the **entire event history**
- Challenges:
  - Sequential sampling limits efficiency:
    - especially for high-frequency event streams: **hundreds or thousands of events per second** needed
  - Existing solutions: **multi-step prediction** often require **changing or retraining the model architecture;** impractical
- Main objective
  - generate valid samples that are indistinguishable from those produced by conventional sampling
  - maintain efficiency on par with the conventional method; **Speed up TPP sampling** without changing or retraining the existing auto-regressive models

Proposed solution

A speculative sampling algorithm:

- Uses a **rejection sampling** framework
  - Maintains **exact distribution** of the original model
- **Does not alter or retrain** the underlying model: reusable, model-agnostic
- Utilizes **hardware parallelism** to increase efficiency
- Allows **parallel generation** of multiple future steps (e.g., event time and type) at once;
  - still use AR model in each batch generation
  - Parallel way of observation samples to speedup

Preliminaries

- TPP: generative models for timestamped sequences of events
  - Event type: x_i; time: t_i $\{(t_1, x_1), \ldots, (t_n, x_n)\}, \quad 0 < t_1 < \cdots < t_n < T$
  - Define TPPs:
    - A common way **intensity function** λ(t); models the rate of event occurrences over time
    - Alternative view: specify the **conditional density** $p(\tau_i, x_i | \mathcal{H}_i) \quad \mathcal{H}_i = \{(t_j, x_j) : t_j < t_i\} \quad \tau_i = t_i - t_{i-1},$
      - Relation between the two: $p(t_i \mid \mathcal{H}_{t_i}) = \lambda(t_i \mid \mathcal{H}_{t_i}) \cdot \exp\left(-\int_{t_{i-1}}^{t_i} \lambda(s \mid \mathcal{H}_s) ds\right)$
      - Commonly modeled using RNNs or Transformers to produce hidden state h_i
      - Be factorized as $p(\tau|\mathcal{H}_i)p(x|\mathcal{H}_i)$.
      - Can reuse any encoder (RNN, Transformer)
      - Enables **exact likelihood training** (no integral)

Traditional (Intensity-based) Likelihood: second integral **often intractable** or requires numerical approximation

$$\log p(\text{sequence}) = \sum_{i=1}^{n} \log \lambda(t_i|\mathcal{H}_{t_i}) - \int_0^T \lambda(t|\mathcal{H}_t)\, dt$$

Conditional Density Advantage: **no integral required**, because you're directly modeling the distribution from which you sample

$$\log p(\text{sequence}) = \sum_{i=1}^{n} \log p(\tau_i, x_i|\mathcal{H}_i)$$

Comparison of two perspective:

	Intensity-based	**Density-based**
Requires λ(t)?	Yes	No (can model density directly)
Involves integral in likelihood?	Yes	No
Recursion?	Can't avoid	Avoidable via precomputed embeddings
Easy sampling?	Needs thinning	Direct sampling from neural net
Parallelism?	Sequential	Can speculate + score in parallel
Causal modeling (e.g. Hawkes)	Explicitly possible: examine temporal influence	Harder (needs structured modeling)

Preliminaries

- Rejection sampling for TPP: generative models for timestamped sequences of events
  - Sampling from a density defined via λ(t) uses **thinning**:
    - Sample from a Poisson process with upper-bound rate $\lambda_{\max}$
    - Accept each point t_i with probability $\frac{\lambda(t_i)}{\lambda_{\max}}$

    => More generally, rejection sampling from **any proposal g(x)** to match **target f(x)**:
    - Accept sample with probability: $\frac{f(x)}{Mg(x)}$, where $M \geq \sup_x \frac{f(x)}{g(x)}$.
      - If target "likes" proposal sample: f(x)≫g(x) → low acceptance rate
      - If target ≈ proposal: f(x)≈g(x) → M≈1, high acceptance
      - If target disagrees with proposal: M≫1, high rejection, slow sampling
      - Ideal case: M=1 (target = proposal) → accept all samples
  - Requirements:
    - Must be able to evaluate both f(x) and g(x)
    - Must be able to sample from g(x)
    - Need to compute or bound M $M = \max_x \frac{f(x)}{g(x)}$

Method

- Motivation:
  - accelerates **long-horizon sampling in temporal point processes**
  - The key motivation: exploit **modern hardware parallelism**, which suffers if the model frequently switches between sampling and state updates
- Efficient Sampling:
  - Renewal Sampling
    - Uses an encoder (pretrained) to generate a **proposal distribution** of the next event p(τ,x ∣ H_i) from historical data.
    - Multiple proposed future events {(τ_{i+1}, x_{i+1}),...,(τ_{i+l}, x_{i+l})} are sampled **in parallel** (AR inside each event seq)
      - **Batching**: Parallel generation allows simultaneous scoring of multiple event candidates. This helps amortize cost and boosts throughput.

Method

- Efficient Sampling:
  - Sample Acceptance / Rejection
    - For each candidate sample inside batch:
    - computes **target distributions** $p^*(\tau, x|\mathcal{H})$ and rejection constants M
    - Each sample is accepted with probability:

$$P_{\text{accept}} = \frac{p^*(\tau_{i+j}, x_{i+j}|\mathcal{H}_{i+j})}{M_{i+j}p(\tau_{i+j}, x_{i+j}|\mathcal{H}_i)}$$

    - If a sample is rejected, **all future events after it are discarded**, and sampling restarts from that point using the updated history
      - **Exact sampling**: guarantees correctness despite rejection, using rejection sampling theory
- Computing M for **various distribution types**:

  when the **proposal and target distributions are known analytically:**
  - Categorical distributions

$$M = \max_i \frac{p_T^{(i)}}{p_P^{(i)}}$$

  - **Exponential distributions:**

$$\frac{f_T(x)}{f_P(x)} = \frac{\lambda_T}{\lambda_P} \exp\left((\lambda_T - \lambda_P)\, x\right). \qquad f(x) = \lambda \exp(-\lambda x).$$

    - When λ_T < λ_P : A monotonically decreasing function, maximum ratio at x⋆ = 0.
    - When λ_T > λ_P: unbounded, restrict the domain to the *majority* of the target distribution, take 95th percentile of f_T; or mixing proposal density with a heavy tailed distribution
  - Log-normal distribution $x^* = \exp\left(\frac{\mu_P\sigma_T^2 - \mu_T\sigma_P^2}{\sigma_T^2 - \sigma_P^2}\right)$ $M = \frac{f_T(x^*)}{f_P(x^*)}$

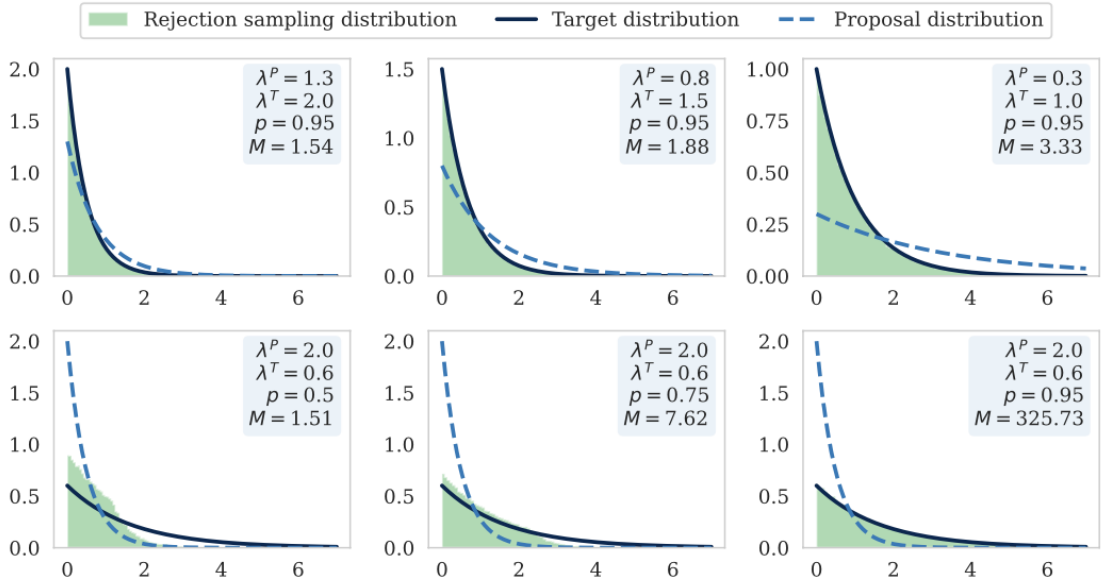
Algorithm 1 Efficient sampling of future events

Input: Historical data $\mathcal{H}_i$, encoder model Enc, decoder model Dec, rejection constant function RejectionConst, rejection function IsRejected
Output: Samples $\{(\tau_{i+1}, x_{i+1}), \ldots, (\tau_{i+l}, x_{i+l})\}$
$\mathcal{S} \leftarrow \{\}$ # Initialize set of samples
$\boldsymbol{h} \leftarrow \text{Enc}(\mathcal{H}_i)$ # History hidden state
$p \leftarrow \text{Dec}(\boldsymbol{h})$ # Proposal distribution
while $|\mathcal{S}| \leq l$ **do**
  $(\tau_{i+j}, x_{i+j}) \sim p(\tau_i, x_i|\boldsymbol{h})$ # Proposal events
  $\boldsymbol{h}_{i+j} \leftarrow \text{Enc}(\tau_{i+j}, x_{i+j})$ # New states
  $p^*_{i+j} \leftarrow \text{Dec}(\boldsymbol{h}_{i+j})$ # Target distributions
  $M_{i+j} \leftarrow \text{RejectionConst}(p, p^*_{i+j})$
  $u_{i+j} \leftarrow \text{IsRejected}(M_{i+j})$ # Boolean 0-1 output
  $k \leftarrow \arg\min_j u_{i+j}$ # Find first rejection
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\tau_{i+j}, x_{i+j})\}_{j=1}^{k-1}$ # Append samples
  $\boldsymbol{h} \leftarrow \boldsymbol{h}_{i+k-1}$ # Update state
  $p \leftarrow p^*_{i+k-1}$ # Update proposal
  $i \leftarrow k - 1$ # Update index
end while

Algorithm 1 shows the proposed method. For clarity, wherever $*_{i+j}$ is used, it is implied that all values $\{*_{i+j}\}_{j=1}^{l}$ are computed in parallel. We also split the model into an en-

Method

- Computing M for **various distribution types**:
  For **non-analytic or complex distributions:**
  - replacing f_P (x) with an under-approximation $\tilde{f}$ (x)≤ f (x) for all x
  - replacing f_T (x) with an over-approximation $\tilde{f}$ (x) ≥ f (x)
- Mixture distributions and general cases
  - Envelope-based over-approximation:
    - Divide the support into segments using quantiles (e.g., 5th, 95th percentiles)
    - For each segment, compute upper bounds using tangent lines or slopes
    - Construct a **piecewise upper envelope** of f_T/f_P $M = \max_{x \in \text{grid}} \frac{f_T(x)}{f_P(x)}$
  - Under-approximation (for proposal):
    - Similar idea: bound the proposal from below with a piecewise function



- Top row: When f_T≈f_P, or f_T ≪ f_P in most regions, M remains small → **efficient sampling**
- **Bottom row:** The further into the **tail** we push our bound for x^*, the more we capture regions where the **proposal underestimates** the target → the ratio f_T/f_P grows → M explodes → sampling becomes **inefficient**

Algorithm 2 RejectionConst for any pair of densities

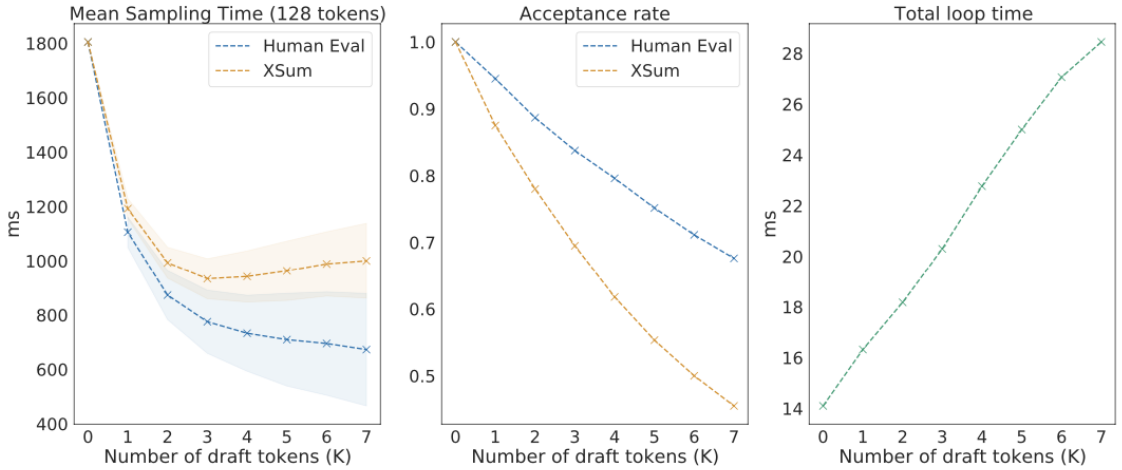
Input: Proposal distribution q, target distribution p, target percentile α, number of grid points n, grid generator GetGridPoints, GetBounds (Algorithm 3)
Output: Rejection constant M
$\mathcal{X}_{\text{left}}, \mathcal{X}_{\text{right}} \leftarrow \text{GetGridPoints}(q, p, \alpha, n)$
$\mathcal{P}_{\text{left}}, \mathcal{P}_{\text{right}} \leftarrow \text{GetBounds}(p, \mathcal{X}_{\text{left}}, \mathcal{X}_{\text{right}}, \text{True})$
$\mathcal{Q}_{\text{left}}, \mathcal{Q}_{\text{right}} \leftarrow \text{GetBounds}(q, \mathcal{X}_{\text{left}}, \mathcal{X}_{\text{right}}, \text{False})$
$\mathcal{R} \leftarrow [\mathcal{P}_{\text{left}}/\mathcal{Q}_{\text{left}}, \mathcal{P}_{\text{right}}/\mathcal{Q}_{\text{right}}]$
$M \leftarrow \max(\mathcal{R})$
Return M

Algorithm 3 GetBounds function for bounding density

Input: Distribution p, left segment bounds $\mathcal{X}_{\text{left}}$, right segment bounds $\mathcal{X}_{\text{right}}$, boolean upperBound
Output: Left and right segment values which define the linear segments which bound the density
$\bar{\mathcal{X}} \leftarrow (\mathcal{X}_{\text{left}} + \mathcal{X}_{\text{right}})/2$ # Mid points
$\bar{\mathcal{P}} \leftarrow p(\bar{\mathcal{X}}),\ \mathcal{P}_{\text{left}} \leftarrow p(\mathcal{X}_{\text{left}}),\ \mathcal{P}_{\text{right}} \leftarrow p(\mathcal{X}_{\text{right}})$
$\bar{\mathcal{P}}' \leftarrow p'(\bar{\mathcal{X}})$ # Derivative in mid points
$\mathcal{Z}_{\text{left}} \leftarrow \bar{\mathcal{P}}'(\mathcal{X}_{\text{left}} - \bar{\mathcal{X}}) + \bar{\mathcal{P}}$ # Tangent values in edges
$\mathcal{Z}_{\text{right}} \leftarrow \bar{\mathcal{P}}'(\mathcal{X}_{\text{right}} - \bar{\mathcal{X}}) + \bar{\mathcal{P}}$
if upperBound **then**
  $\mathcal{Y}_{\text{left}}, \mathcal{Y}_{\text{right}} \leftarrow \max(\mathcal{P}_{\text{left}}, \mathcal{Z}_{\text{left}}), \max(\mathcal{P}_{\text{right}}, \mathcal{Z}_{\text{right}})$
else
  $\mathcal{Y}_{\text{left}}, \mathcal{Y}_{\text{right}} \leftarrow \min(\mathcal{P}_{\text{left}}, \mathcal{Z}_{\text{left}}), \min(\mathcal{P}_{\text{right}}, \mathcal{Z}_{\text{right}})$
end if
Return $\mathcal{Y}_{\text{left}}, \mathcal{Y}_{\text{right}}$

Discussion

- Takeways:
  - LLM paper:
    - Works well when draft and target agree (e.g., predictable next tokens).
    - Must tune draft length K
    - Reduces the number of calls to the slow target model
  - TPP paper:
    - Enables parallel event sampling without retraining the model
    - Applies rejection sampling theory to exact sampling from target TPPs
    - Proposes multiple strategies for computing the constant M
    - Maintains output quality while improving sampling speed significantly
- Limitation:
  - Cannot get rid of AR mode of sequence generation
  - Rejection sampling scheme introduces inherent randomness, and that can affect time efficiency; bring variance; Stochastic accept/reject decisions



1st paper: Trade-off Analysis: Longer Drafts vs Frequent Scoring

•**Increasing K** reduces how often the **large target model** is called → potential speedup.

•However, this comes at a cost:

- **More time per iteration** (due to more draft tokens being scored).
- **Lower acceptance** of later tokens → more frequent fallback to slow sampling.