PLURALGEN: EFFICIENT TEST-TIME SCALING BY DECOUPLING PARTICLE SEARCH FROM MODEL EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Particle search gives compact language models a way to improve reasoning at test time when deployment constraints rule out a larger model. Resampling directs a fixed population toward promising paths but often leaves many particles at the same complete state. Sequence runtimes nevertheless decode every particle separately. Prefix-aware KV sharing preserves inherited history but repeats the current Transformer pass and vocabulary projection, so execution remains proportional to particle count even as the search concentrates. We present PluralGen, which keeps particles as the unit of search while advancing each distinct complete state once. It stores each repeated state with its multiplicity, draws an independent continuation for every particle, and carries the grouping through model execution, resampling, and KV ownership. PluralGen reaches 3.2× end-to-end speedup, reduces KV allocation by 88.1%, supports 4× larger populations, and improves serving throughput by 2.9× on the evaluated workloads.

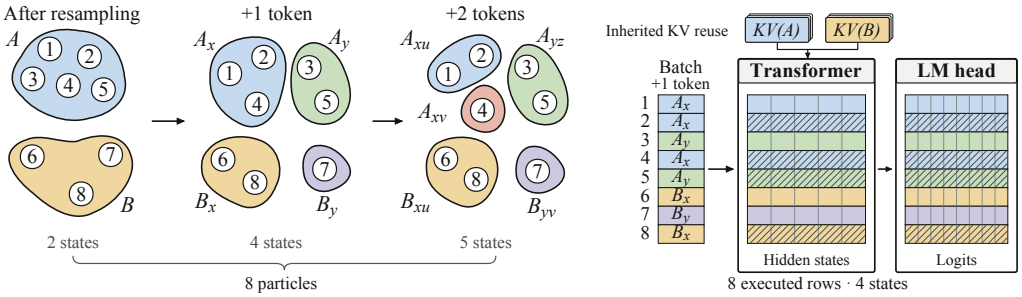
1 INTRODUCTION

Test-time scaling can improve reasoning without increasing model size by exploring alternative continuations and directing more work toward promising ones (Brown et al., 2024; Wu et al., 2024; Snell et al., 2025). This tradeoff is particularly useful for local and edge inference, where a larger model may not fit even though additional decoding time is available (Chen et al., 2026; Park et al., 2025). Particle methods make that extra computation adaptive: a proposal–weight–resample loop reallocates effort before a complete answer is available (Zhao et al., 2024; Emara et al., 2026). Because the population can concentrate and diverge during generation, its effective execution width changes with the evolving search rather than the nominal particle count.

However, resampling creates a mismatch between what the search knows and what the runtime schedules. A promising state may receive several descendants (Wang et al., 2026; Azizi et al., 2026). Their future draws must remain independent, yet the next step presents the same complete state, model input, next-token distribution, and inherited KV history to each descendant. Current runtimes still materialize every descendant as a separate row. Prefix-aware systems reuse the inherited prefix (Kwon et al., 2023; Zheng et al., 2024; Juravsky et al., 2024; Gim et al., 2024), but still schedule the next model step once per offspring. Figure 1 makes this mismatch visible: the particle population remains large even when far fewer complete states require model execution. Making that reuse safe without changing the particle process requires answering two related questions about representation and execution at every decoding step.

The first challenge is to determine when particles can share a model evaluation without changing the search. A common token history is not enough. Particle weights, accumulated likelihoods, stopping status, and method-specific variables can alter later transitions even when the visible prefix agrees (Wang et al., 2026; Azizi et al., 2026). At the same time, identical complete states still represent separate samples: each member needs an independent proposal and labeled ancestry for subsequent weighting and resampling. Existing cache-coherent resampling preserves shared history (Emara et al., 2026) but neither determines when the current model work is identical nor removes its repeated evaluation. A useful representation must share only the deterministic work implied by a common state while preserving every particle transition.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Repeated states with independent proposals **(b) Redundant model work**

Figure 1: Particle search and sequence execution diverge after resampling. Particles at the same complete state retain independent futures, while inherited KV sharing alone still leaves the current model work duplicated in the next model step.

The second challenge is to retain that sharing as particle states split and regroup during generation. Independent proposals divide one state into several continuations; resampling may gather many particles at the same state again; terminated particles leave active execution. Vocabulary projection, scheduling, and KV ownership, however, are organized around persistent sequence rows (Kwon et al., 2023; Zheng et al., 2024). If any stage expands a group back into one row per particle, the remaining path repeats the work that grouping removed. Static prefix sharing cannot follow membership that changes at token granularity. Multiplicity must remain visible through proposal and resampling, and a new model row should appear only when continuations actually diverge.

We propose PluralGen to keep particles distinct for the search while advancing each occupied complete state once. The particle process retains labels, weights, and independent random streams, while its multiplicity-aware density head (MDH) and grouped execution carry this separation through proposal, resampling, vocabulary projection, and KV ownership. Physical width therefore follows state occupancy rather than the nominal search budget. Across GSM8K, MATH500, GPQA Diamond, and AIME 2026, this design yields up to $3.2\times$ faster completion, 88.1% lower KV allocation, room for $4\times$ more particles, and up to $2.9\times$ higher serving throughput.

The main contributions are as follows:

- We identify the mismatch between particle population and model execution width and show that repeated rows persist after inherited KV sharing.
- We formulate counted particle inference, preserving independent randomness, ancestry, and sequence-wise transitions while evaluating each occupied state once.
- We develop a multiplicity-aware density head and grouped-state runtime that preserve this sharing throughout the decoding path.
- PluralGen reduces end-to-end latency by up to $3.2\times$ and KV allocation by 88.1%, enabling $4\times$ larger populations and up to $2.9\times$ higher serving throughput.

2 Background and Motivation

2.1 Particle search reallocates inference

Sequential Monte Carlo (SMC) represents a target distribution with weighted trajectories (Gordon et al., 1993; Kitagawa, 1996; Doucet et al., 2001). At each token, particles extend under proposal q_t, update their weights, and are systematically resampled when ESS falls (Douc & Cappe, 2005). A high-weight trajectory may then receive descendants that explore independent futures from the same state. This proposal–weight–resample structure also underlies particle MCMC and variational inference (Andrieu et al., 2010; Naesseth et al., 2018).

Recent LLM methods adapt this pattern with different target, proposal, and resampling designs and study its test-time tradeoffs (Wang et al., 2026; Azizi et al., 2026; Emara et al., 2026; Markovic-

Voronov et al., 2026; Golowich et al., 2026). ASMC mixes base and sharpened token distributions, whereas Power-SMC sharpens its proposal and target during generation. Despite their different schedules and estimators, both determine the next transition from the complete particle state. Resampling can therefore produce several particles with identical model inputs.

2.2 Distinct states after resampling

After resampling, consider a state with m descendants whose next tokens are independent draws from a categorical proposal q (Wang et al., 2026; Azizi et al., 2026). If K_m is the number of distinct continuations, the expected support is

$$\mathbb{E}[K_m] = \sum_v [1 - (1 - q(v))^m] . \tag{1}$$

This quantity grows sublinearly with m, so later offspring increasingly repeat a continuation. Resampling may gather them again, allowing equal states to persist across several decoding rounds.

Let N_t and U_t denote the active particles and distinct complete states. A sequence runtime schedules N_t rows although U_t suffice to advance the model (Wang et al., 2026; Azizi et al., 2026; Zheng et al., 2024). We call their ratio the distinct-state ratio,

$$\rho_t = U_t / N_t. \tag{2}$$

Equation 1 shows why this ratio falls as resampling gathers more particles at the same state, especially under concentrated proposals. A large population may therefore occupy far fewer model states. Whether that concentration becomes useful reuse depends on what the runtime shares after resampling.

2.3 Execution still follows particles

Current particle implementations already avoid part of the resampling cost. Cache-coherent resampling, unique-ancestor sharing, and device-resident updates reuse inherited KV state (Wang et al., 2026; Azizi et al., 2026; Emara et al., 2026); our SGLang baseline includes these optimizations. The next decoding step still resumes with one sequence row per offspring, including particles whose complete states remain equal.

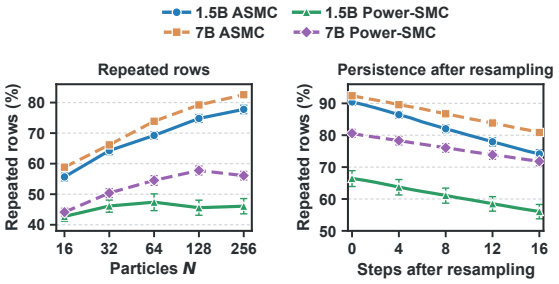


Figure 2: Measured share of repeated model rows after resampling across particle populations. Equal states remain together over subsequent decoding steps.

Figure 2 shows that duplicate execution persists beyond resampling. Even after KV sharing, 42.7–82.6% of Transformer rows repeat complete states, with the largest share under the more concentrated ASMC population. Many particles still share a state sixteen tokens later, so the reusable work extends beyond the inherited KV prefix to the current model input.

Once the inherited KV state has been shared, the remaining redundancy lies in the current model batch. This observation defines the boundary used by PluralGen. The particle process keeps all labeled trajectories, while the model advances one row per distinct complete state.

3 PluralGen

PluralGen realizes this execution model with a grouped particle representation that persists throughout decoding. Figure 3 follows one decode step from the current groups through counted proposals to the regrouped children that enter the next step.

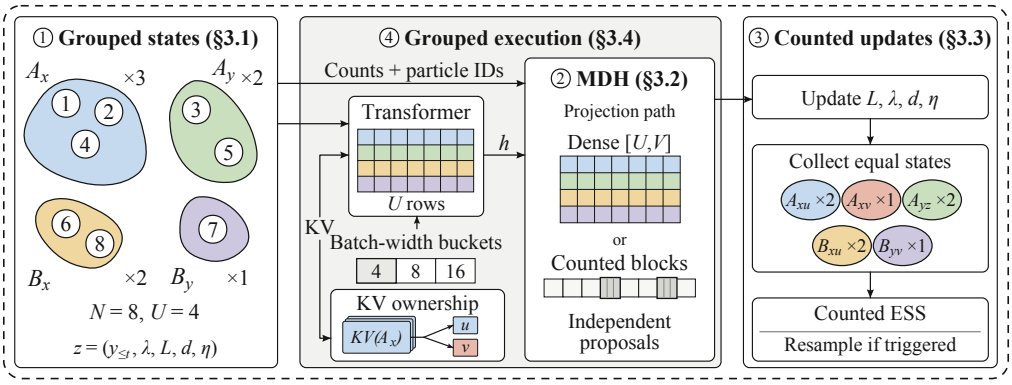


Figure 3: Overview of a decode step in PluralGen. Equal complete states share a Transformer row and KV history, while MDH draws independent proposals and counted updates form the groups used by the next decode step.

3.1 Grouped particle representation

At step t, particles whose complete states agree form one group. The grouped population can be written as the discrete measure

$$\mu_t = \sum_{i=1}^{U_t} m_i \, \delta_{z_i}, \qquad \sum_{i=1}^{U_t} m_i = N_t, \tag{3}$$

where U_t is the number of groups, m_i is the number of particles in group i, and N_t is the total particle count. For ASMC and Power-SMC, the complete state of a particle is

$$z_t = (y_{\le t}, \lambda_t, L_t, d_t, \eta_t), \tag{4}$$

where λ_t is the log importance weight, L_t is accumulated base-model log likelihood, d_t records termination, and η_t contains method variables for later transitions. Grouping by token prefix alone could merge particles with different weights or method variables and alter their subsequent search. Particles are therefore grouped only when the complete tuple agrees. Each group retains separate particle identifiers, which index independent counter-based random streams; equal states share a model evaluation without sharing a proposal draw. Terminal groups remain in the estimator but leave the active model batch for further decoding.

3.2 Multiplicity-aware proposals

MDH draws independent token proposals for all members of a group from one hidden state. Let ℓ_v be the base-model logit for legal token v. ASMC and Power-SMC use powered token mixtures (Wang et al., 2026; Azizi et al., 2026),

$$q_t(v \mid z_i) = \sum_{c=1}^{C_t} \pi_{t,c} \frac{\exp(\beta_{t,c}\ell_v)}{Z_{\beta_{t,c}}}, \qquad Z_\beta = \sum_u \exp(\beta \ell_u), \tag{5}$$

$$\Delta\lambda_t(v) = (a_t - a_{t-1})L_{t-1} + a_t \log p_t(v \mid z_i) - \log q_t(v \mid z_i), \tag{6}$$

where a_t is the target exponent. ASMC uses two proposal components and Power-SMC one.

Given hidden state h_i and multiplicity m_i, MDH returns

$$\mathcal{D}(h_i, m_i) = \{(v, k_v, \log p_t(v), \log q_t(v), \Delta\lambda_t(v), \mathcal{R}_v)\}, \tag{7}$$

where k_v particles select token v, $\sum_v k_v = m_i$, and $\mathcal{R}_v$ contains their identifiers. The returned counts follow the multinomial law of m_i independent draws from q_t; entries with $k_v = 0$ are omitted.

MDH avoids repeating a dense vocabulary projection for every particle. It chooses between a dense $[U_t, V]$ projection over groups and a counted block path that first computes probability mass by

vocabulary region, then evaluates only the regions selected by the particle draws. Each selected token retains the complete mixture density,

$$\log q_t(v) = \mathrm{LSE}_c\big(\log \pi_{t,c} + \beta_{t,c}\ell_v - \log Z_{\beta_{t,c}}\big)\,. \tag{8}$$

The runtime uses the dense path for low multiplicity or broad block coverage and the counted path when proposals concentrate on fewer regions. Both paths return Equation 7.

3.3 Particle updates and resampling

Counted proposals update the particle process without expanding model rows. Each child inherits its parent state, appends token v, and updates L_t and λ_t with the returned densities. Children with equal complete states are regrouped before the next model step, while terminal groups move to the estimator for final aggregation at the end of generation.

Because the m_i members of a group share weight e^{λ_i}, the multiplicity-weighted ESS is available without expanding the particle population:

$$\mathrm{ESS} = \frac{\big(\sum_i m_i e^{\lambda_i}\big)^2}{\sum_i m_i e^{2\lambda_i}}. \tag{9}$$

For systematic resampling (Douc & Cappe, 2005), groups are placed in a canonical order and their identifiers occupy contiguous probability intervals. Counting the systematic points in an interval gives its next multiplicity. The original terminal estimators are then evaluated with multiplicity-weighted masses (Wang et al., 2026; Azizi et al., 2026). After optional resampling, the surviving groups define the next model batch at the following step.

3.4 Persistent state execution

That batch contains one row and one paged-KV block table for every distinct state (Kwon et al., 2023; Zheng et al., 2024). Children share immutable prefix pages and acquire separate tails after their complete states diverge; changing a group multiplicity does not allocate another tail. Because proposal and resampling change the number of active states, the runtime batches requests by their current state count and reuses execution graphs across nearby widths. Particle weights and proposal statistics remain attached to their groups.

3.5 Particle semantics and model cost

The counted representation preserves the labeled particle process. Each label still receives an independent proposal draw, while weighting, systematic resampling, and terminal estimation account for group multiplicities. Expanding the groups after each transition recovers the same distribution over labeled particle states as sequence-wise ASMC or Power-SMC, including their terminal estimators. The correspondence carries through successive proposal and resampling steps. Resampling may change the multiplicity of a stored state without adding a model row; a proposal creates a new row only when its continuation is distinct. The model batch can therefore contract and expand while the labeled particle population follows its original transitions.

Transformer evaluation, vocabulary projection, and KV allocation consequently follow U_t, the number of distinct states. Random draws and scalar particle updates still follow N_t. The attainable gain depends on the distinct-state ratio U_t/N_t and on how much time the baseline spends in row-proportional model work during each decode step.

4 Experiments

4.1 Experimental setup

System and models. Experiments use an NVIDIA GeForce RTX 4090 (24 GB), PyTorch 2.11, and CUDA 12.8. Both systems serve Qwen2.5-Math-1.5B-Instruct and Qwen2.5-Math-7B-Instruct (Yang et al., 2024). Model execution uses BF16; proposal densities, weights, ESS, and resampling remain in FP32 throughout the particle updates.

Baseline. We compare with **SGLang** (Zheng et al., 2024), which runs each particle as a separate sequence under the same proposal, weighting, resampling, and KV-sharing rules. Its particle updates remain device-resident, but every offspring resumes as a model row. PluralGen keeps equal states grouped until their continuations diverge.

Workloads and metrics. We evaluate ASMC and Power-SMC on GSM8K (Cobbe et al., 2021), MATH500 (MATH) (Hendrycks et al., 2021), GPQA Diamond (GPQA) (Rein et al., 2023), and AIME 2026 (AIME) from MathArena (Dekoninck et al., 2026). Their different proposal and target schedules expose different amounts of reusable state. We report end-to-end latency, particle throughput, model rows, KV allocation, search capacity, and serving throughput.

4.2 End-to-end efficiency

The reduction in execution width translates directly into lower latency and higher particle throughput in Table 1. Figure 4 traces this relationship across population sizes and relates the gains to the distinct-state ratio and KV demand.

At smaller populations, most particles occupy different states, so PluralGen remains close to SGLang. As repetitions accumulate, speedup reaches $3.2\times$, with up to 88.1% less KV allocation and 87.2% fewer model rows. SGLang shares inherited KV pages but resumes every offspring as a row, repeating the current Transformer and vocabulary work. PluralGen keeps one row per repeated state, shares its projection through MDH, and allocates KV tails only after continuations diverge.

ASMC gathers its population into fewer states, so its distinct-state ratio, batch width, and KV demand fall faster than under Power-SMC. Power-SMC yields less reuse, although its speedup grows as repeated states accumulate. Across tasks, the gains follow how strongly the search concentrates its batch into repeated states over successive decode steps.

4.3 Capacity and serving throughput

The memory released by grouping can support either a larger search within one request or more concurrent requests. Figure 5 measures the first effect, and Table 2 measures the second.

At 512 generated tokens, grouped execution adds 128 particles in most settings. At 1,024 tokens, the added capacity reaches 192 particles as repeated KV tails accumulate in SGLang. Counted states retain the common history once and allocate a new tail only after divergence, so the capacity advantage grows with the amount of generated state.

Table 2 shows the same effect in serving. Throughput rises with concurrency, and several SGLang configurations run out of memory before the larger batches. Because SGLang schedules every particle, repeated rows in one request consume capacity that could serve another. PluralGen instead schedules distinct states across requests. Across individual workloads, throughput reaches $2.8\times$ the baseline under fixed concurrency and $2.9\times$ under dynamic arrivals; the aggregated gain is larger for ASMC because its more concentrated populations release more capacity.

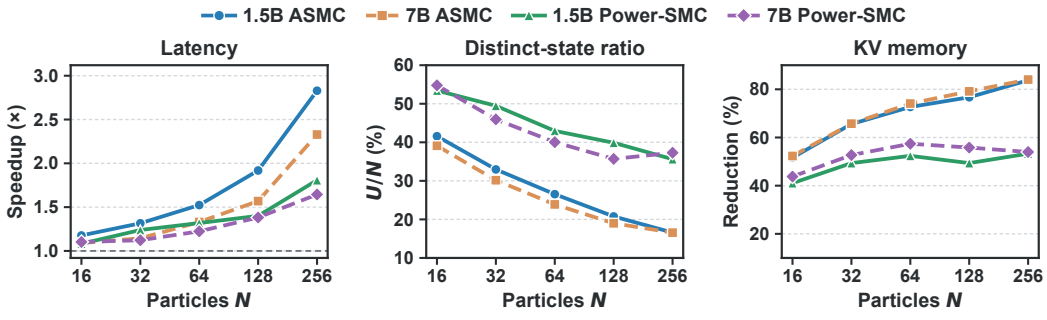


Figure 4: Latency, distinct-state ratio, and KV reduction as the particle population grows. ASMC yields more reusable state than Power-SMC because its resampling concentrates the population as particle counts increase during decoding.

Table 1: End-to-end performance at $N = 256$. Lower model-row and KV demand translates into shorter completion latency and higher particle throughput for the same search budget.

Task	Model	Method	Completion latency (s)			Particle throughput (10^3/s)			Reduction (%)	
			SGLang	PluralGen	Speedup	SGLang	PluralGen	Speedup	KV	Rows
GSM8K	1.5B	ASMC	14.87	4.88	**2.98**	5.10	14.87	**2.91**	85.5	84.3
	7B	ASMC	20.82	8.74	**2.39**	3.52	8.46	**2.40**	88.1	87.2
	1.5B	Power-SMC	12.28	7.22	**1.69**	4.24	6.93	**1.63**	50.8	51.7
	7B	Power-SMC	18.29	11.72	**1.53**	1.39	2.04	**1.46**	55.8	52.2
MATH	1.5B	ASMC	25.98	8.66	**2.97**	4.98	14.44	**2.90**	85.2	85.1
	7B	ASMC	26.45	10.78	**2.43**	3.62	8.85	**2.44**	87.4	86.7
	1.5B	Power-SMC	22.07	10.03	**1.97**	3.52	5.92	**1.68**	58.6	62.8
	7B	Power-SMC	23.13	13.28	**1.69**	2.07	3.54	**1.71**	63.1	59.6
GPQA	1.5B	ASMC	41.04	14.09	**2.28**	2.14	4.45	**2.07**	78.3	73.9
	7B	ASMC	29.46	13.79	**2.12**	3.34	6.96	**2.08**	77.4	76.3
	1.5B	Power-SMC	16.31	8.86	**1.38**	0.73	0.94	**1.28**	36.3	34.6
	7B	Power-SMC	25.93	15.52	**1.63**	2.09	3.20	**1.53**	42.0	53.6
AIME	1.5B	ASMC	66.41	20.13	**3.18**	4.48	13.52	**3.02**	86.0	84.6
	7B	ASMC	31.91	13.42	**2.38**	3.72	8.89	**2.39**	83.2	82.8
	1.5B	Power-SMC	51.25	19.52	**2.33**	2.29	4.11	**1.79**	67.5	70.5
	7B	Power-SMC	29.72	16.88	**1.74**	2.66	4.67	**1.75**	54.9	59.1

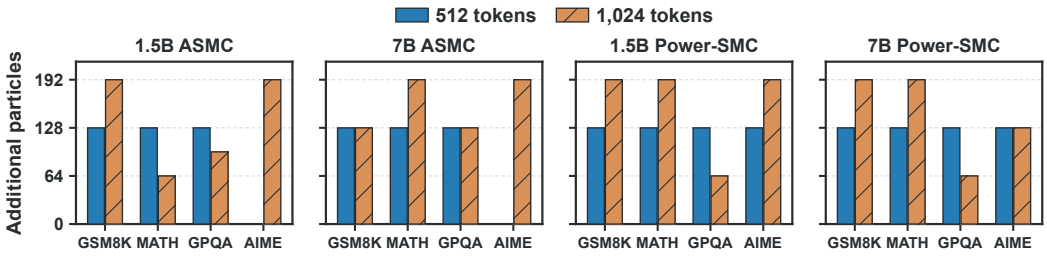


Figure 5: Additional particle capacity over SGLang at two generation lengths. The bars show the increase in the largest completed population and the KV headroom provided by grouped-state ownership within the serving runtime.

4.4 Mechanism analysis

Figure 6 compares full PluralGen with variants that remove one execution mechanism. Without persistent grouping, completion latency rises by 61–78% in the larger-population settings. Equal descendants often persist beyond resampling, so restoring one row per offspring discards several decoding steps of shared work.

The middle panel isolates vocabulary projection. Removing MDH expands the LM-head batch by 3.0–6.8× and adds 10–20% to completion latency. Once a group contains multiple particles, one shared projection replaces repeated work over the same hidden state.

The right panel captures the changing batch shape. Proposal and resampling continually alter the number of distinct states; without graph replay, each new shape returns to eager dispatch and adds about 75% to completion latency.

4.5 Sensitivity to available reuse

Available reuse depends on how long repeated states remain grouped and how much of a request is spent decoding. Table 3 varies the multiplicity cap and prompt length to separate these effects.

Table 2: Geometric-mean serving throughput relative to sequence-wise SGLang across tasks under fixed and dynamic arrivals. Dashes indicate cases where the baseline runs out of memory.

Model	Method	N	Fixed concurrency				Dynamic arrivals		
			2	4	8	16	Steady	Burst	Poisson
1.5B	ASMC	32	1.245	1.380	1.495	1.675	1.435	1.501	1.431
		128	2.012	2.274	–	–	2.122	2.306	2.109
7B	ASMC	32	1.185	1.310	1.542	1.751	1.429	1.470	1.433
		128	1.926	2.591	–	–	2.176	2.289	2.174
1.5B	Power-SMC	32	1.135	1.214	1.244	1.311	1.225	1.248	1.218
		128	1.396	1.452	–	–	1.405	1.452	1.412
7B	Power-SMC	32	1.098	1.183	1.261	1.296	1.190	1.209	1.190
		128	1.395	1.530	–	–	1.415	1.455	1.423

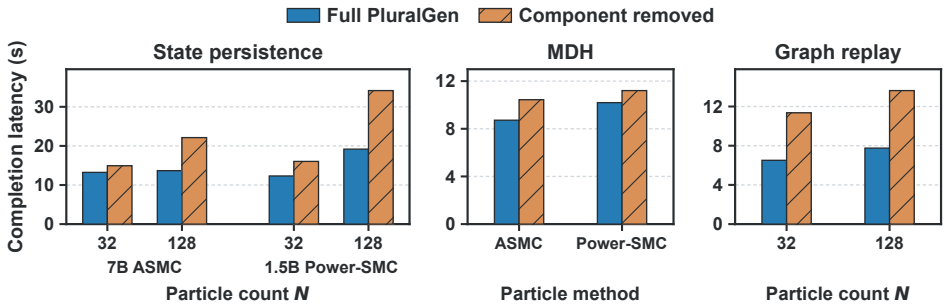


Figure 6: Absolute completion latency with each execution mechanism enabled or removed. State persistence, grouped vocabulary projection through MDH, and graph replay each preserve a different part of the end-to-end gain.

Table 3: Effect of group multiplicity and prompt length on available reuse. Keeping repeated states together preserves model and KV savings, whereas a longer shared prefill reduces their contribution to full-request completion latency as prompts grow in length.

Model	Method	N	Multiplicity cap 4		Unrestricted		Speedup by prompt length		
			Speedup	KV red. (%)	Speedup	KV red. (%)	Short	Medium	Long
1.5B	ASMC	64	1.247	39.4	1.528	72.7	1.641	1.475	1.454
		256	1.545	44.1	2.844	83.6	3.118	3.109	2.388
7B	ASMC	64	1.170	40.3	1.307	74.1	1.296	1.341	1.290
		256	1.443	41.1	2.260	83.8	2.302	2.311	2.170
1.5B	Power-SMC	64	1.198	31.2	1.324	52.4	1.254	1.421	1.311
		256	1.322	28.3	1.826	53.4	1.861	1.835	1.776
7B	Power-SMC	64	1.123	35.2	1.239	57.5	1.269	1.193	1.253
		256	1.326	32.9	1.748	59.3	1.699	1.752	1.782

Capping multiplicity lowers the speedup in every model and particle method, with the largest gap under ASMC. Its resampling forms larger groups, so splitting a group creates more model rows and separate KV tails. Power-SMC retains more distinct states, so less shared work is available to remove. Prompt length changes a different part of the request. The shared prefill costs the same in both systems, so longer prompts reduce the fraction of end-to-end latency affected by grouped decoding even when it removes the same repeated rows and KV histories. Across both model sizes, the cap also narrows ASMC's advantage over Power-SMC because it suppresses the larger groups from which ASMC benefits most. Prompt length changes how prominently the saved decode work appears in end-to-end latency, but it does not directly alter the groups formed after resampling.

5 Related Work

Test-time scaling. Repeated sampling and self-consistency compare complete solutions, while tree and graph search and process-guided reasoning direct computation during generation (Wang et al., 2023; Yao et al., 2023; Besta et al., 2024; Lightman et al., 2024; Wang et al., 2024; Brown et al., 2024; Wu et al., 2024; Snell et al., 2025). Particle methods repeatedly revise that allocation by weighting and resampling a live population before complete answers are available. Resampling can consequently place several labeled particles at one complete state even though their next proposals remain independent. PluralGen exposes this transient overlap to the runtime, reusing a shared model evaluation without changing the search procedure.

State reuse in LLM serving. Continuous batching, phase-aware scheduling, and model multiplexing organize work across requests (Yu et al., 2022; Agrawal et al., 2024; Zhong et al., 2024; Patel et al., 2024; Li et al., 2023; Sheng et al., 2023). Paged KV memory and prefix-aware runtimes reuse inherited histories (Kwon et al., 2023; Zheng et al., 2024; Juravsky et al., 2024; Gim et al., 2024); InferCept, Pensieve, and CacheBlend extend reuse across interrupted or composed requests (Abhyankar et al., 2024; Yu et al., 2025; Yao et al., 2025). FlashInfer optimizes the underlying operators (Ye et al., 2025), while HeraSys shares work across complete workflows (Li et al., 2026). These systems exploit reuse at the request, prefix, or workflow level. Particle search instead creates groups that split and reform within one probabilistic computation. PluralGen shares the current model evaluation inside each group while keeping proposals, weights, and ancestry separate.

Efficient generation under constrained resources. IO-aware attention and KV-cache compression reduce the cost of retaining and attending to each model state (Dao et al., 2022; Dao, 2024; Zhang et al., 2023; Liu et al., 2024; Hooper et al., 2024; Tang et al., 2024); weight quantization and sparse activation reduce model-side demand (Lin et al., 2024; Song et al., 2024). Speculative decoding instead proposes and verifies future tokens in parallel (Chen et al., 2023; Leviathan et al., 2023; Miao et al., 2024; Cai et al., 2024; Li et al., 2024; Zhou et al., 2024; Chen et al., 2024). FastTTS and SpecEdge adapt efficient reasoning to constrained devices (Chen et al., 2026; Park et al., 2025), while FlashSampling and SonicSampler reduce sampling overhead (Ruiz et al., 2026; Ponnusamy et al., 2026). These methods reduce the cost of advancing a sequence. PluralGen addresses a different source of work by advancing fewer model rows when resampling concentrates the particle population.

6 Discussion

Execution width is not fixed by the search budget. Proposals and resampling continually change how many complete states are active, so a scheduler can reclaim model rows and KV space as groups form and spend that capacity on deeper search within a request or on additional concurrent requests. The useful width is a property of the evolving search, not a fixed request attribute.

This boundary extends beyond SMC. A population method can remain defined over labeled members while exposing only the deterministic states they occupy to the runtime. When members share a model state but retain distinct randomness, weights, or ancestry, the runtime can advance the occupied states without redefining the algorithm.

More generally, the runtime exposes each complete state, its multiplicity, and transition bookkeeping without coupling execution to a proposal or resampling rule. This interface lets future particle methods change their proposal or resampling policy while reusing the same execution path.

7 Conclusion

PluralGen treats the concentration created by particle resampling as reusable execution state. Its counted process and MDH preserve proposals, weights, and resampling while scaling Transformer, vocabulary, and KV work with occupied states, yielding up to $3.2\times$ faster generation, 88.1% less KV allocation, $4\times$ larger particle populations, and $2.9\times$ higher throughput. This separates reasoning budget from physical batch width: recovered capacity can support deeper search or concurrent requests. Exposing occupancy without changing particle semantics keeps that tradeoff available as proposal rules and serving workloads evolve.

AI USE STATEMENT

Generative AI tools assisted with code generation and review, developing and checking proofs, searching the related literature, and suggesting edits to improve the clarity of the manuscript. The authors reviewed and verified all AI-assisted material and take responsibility for the final manuscript, code, results, and claims presented in this submission.

ETHICS STATEMENT

This work studies infrastructure for serving LLMs using publicly available models and benchmarks. It involves neither human participants nor private user data.

REPRODUCIBILITY STATEMENT

Code for reproducing the reported results is available in the anonymous repository.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 2024.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B*, 2010.

Seyedarmin Azizi, Erfan Baghaei Potraghloo, Minoo Ahmadi, Souvik Kundu, and Massoud Pedram. Power-SMC: Low-latency sequence-level power sampling for training-free LLM reasoning. *arXiv preprint arXiv:2602.10273*, 2026.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

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. In *Proceedings of the 41st International Conference on Machine Learning*, 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.

Hao Mark Chen, Zhiwen Mo, Guanxi Lu, Shuang Liang, Lingxiao Ma, Wayne Luk, and Hongxiang Fan. FastTTS: Accelerating test-time scaling for edge LLM reasoning. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2026.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with LLMs. *arXiv preprint arXiv:2605.00674*, 2026.

R. Douc and O. Cappe. Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis*, 2005.

Arnaud Doucet, Nando de Freitas, and Neil Gordon (eds.). *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

Yahya Emara, Mauricio Barba da Costa, Chi-Chih Chang, Cameron Freer, Tim Vieira, Ryan Cotterell, and Mohamed S. Abdelfattah. Faster LLM inference via sequential Monte Carlo. *arXiv preprint arXiv:2604.15672*, 2026.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, 2024.

Noah Golowich, Fan Chen, Dhruv Rohatgi, Raghav Singhal, Carles Domingo-Enrich, Dylan J. Foster, and Akshay Krishnamurthy. Reject, resample, repeat: Understanding parallel reasoning in language model inference. *arXiv preprint arXiv:2603.07887*, 2026.

Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. Novel approach to nonlinear/Non-Gaussian Bayesian state estimation. *IEE Proceedings F: Radar and Signal Processing*, 1993.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, 2021.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, 2024.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput LLM inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

Genshiro Kitagawa. Monte Carlo filter and smoother for Non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 1996.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Size Li, Zhiqing Tang, Hongrui Liang, Jianxiong Guo, Jiong Lou, Tian Wang, and Weijia Jia. HeraSys: Collaborative serving of multiple LLM workflows via fine-grained end-to-end optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation*, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, 2024.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Jelena Markovic-Voronov, Wenhui Zhu, Bo Long, Zhipeng Wang, Suyash Gupta, Kayhan Behdin, Bee-Chung Chen, and Deepak Agarwal. Sampling for quality: Training-free reward-guided llm decoding via sequential monte carlo. *arXiv preprint arXiv:2604.16453*, 2026.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024.

Christian A. Naesseth, Scott W. Linderman, Rajesh Ranganath, and David M. Blei. Variational sequential Monte Carlo. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, 2018.

Jinwoo Park, Seunggeun Cho, and Dongsu Han. SpecEdge: Scalable edge-assisted serving framework for interactive LLMs. In *Advances in Neural Information Processing Systems*, 2025.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture*, 2024.

Pragaash Ponnusamy, Shivam Sahni, Jue Wang, and Tri Dao. SonicSampler: Unified tile-aware kernels for LLM sampling and speculative verification. *arXiv preprint arXiv:2607.20475*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Tomas Ruiz, Zhen Qin, Yifan Zhang, Xuyang Shen, Yiran Zhong, and Mengdi Wang. FlashSampling: Fast and memory-efficient exact sampling. *arXiv preprint arXiv:2603.15854*, 2026.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations*, 2025.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. PowerInfer: Fast large language model serving with a consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Ke Wang, Zehao Yu, Luwei Wang, and Yongchao Huang. Cache coherent resampling for efficient test time scaling in LLM reasoning via adaptive sequential Monte Carlo. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, 2025.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, 2022.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 2024.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, 2024.

A Formal Analysis of Counted Particle Inference

The analysis follows labeled particles through grouping, proposal, resampling, and model execution. Each label indexes a random stream and records ancestry, while particles with identical complete states are stored together. The grouped representation can therefore be related directly to the labeled process and to Transformer, vocabulary, and KV-cache cost.

A.1 Population representation

Associate each particle in a request with an independent random stream. The labeled population at step t is

$$\mathcal{X}_t = \big((z_t^{(1)}, r_t^{(1)}), \dots, (z_t^{(N)}, r_t^{(N)})\big), \tag{10}$$

where z is the complete state in Equation 4, and r identifies the random stream assigned to that particle. Grouping equal states gives

$$\mathcal{F}_t = \{(z_{t,i}, \mathcal{R}_{t,i})\}_{i=1}^{U_t}, \qquad m_{t,i} = |\mathcal{R}_{t,i}|. \tag{11}$$

The sets $\mathcal{R}_{t,i}$ partition the particle identifiers. Expanding $\mathcal{F}_t$ assigns $z_{t,i}$ to every identifier in $\mathcal{R}_{t,i}$ and recovers $\mathcal{X}_t$. All particles in a group have the same conditional transition, while their labels select separate random coordinates within that transition.

Formally, define $j \sim_t k$ when $z_t^{(j)} = z_t^{(k)}$. This is an equivalence relation on the labeled population, and the sets $\mathcal{R}_{t,i}$ are its equivalence classes. Let Count replace each class by its common state and identifiers, and let Expand restore one labeled state per identifier.

Proposition 1 (Counted quotient)**.** *For every labeled population $\mathcal{X}$,* $\mathsf{Expand}(\mathsf{Count}(\mathcal{X})) = \mathcal{X}$. *For every valid counted population $\mathcal{F}$,* $\mathsf{Count}(\mathsf{Expand}(\mathcal{F})) = \mathcal{F}$ *up to the order of its groups.*

Proof. Counting changes neither a state nor its particle identifier; it collects the identifiers whose complete states agree. Expansion assigns the stored state back to every identifier in its class, which recovers each pair in $\mathcal{X}$. Conversely, expanding a valid counted population creates exactly the equivalence classes from which it was formed. Counting those classes therefore restores the same states and identifier sets.

Both representations use the same canonical order before systematic resampling: groups are ordered by complete state and identifiers are ordered within each group. Equal states occupy adjacent probability intervals, which can be counted as one interval without changing the systematic points.

A.2 Incremental importance weights

Following the annealed particle targets used by ASMC and Power-SMC (Wang et al., 2026; Azizi et al., 2026), let the unnormalized target at step t be $\gamma_t(y_{\le t}) = p(y_{\le t} \mid x)^{a_t}$. Extending a trajectory under proposal q_t changes its importance weight by

$$\frac{w_t}{w_{t-1}} = \frac{\gamma_t(y_{\le t})}{\gamma_{t-1}(y_{<t}) q_t(y_t \mid x, y_{<t})} \tag{12}$$

$$= p(y_{<t} \mid x)^{a_t - a_{t-1}} \frac{p(y_t \mid x, y_{<t})^{a_t}}{q_t(y_t \mid x, y_{<t})}. \tag{13}$$

Taking logarithms yields Equation 6. The first factor is one when the target exponent is fixed and accounts for the changing target when it is annealed. Since members of a counted group have the same parent state, the parent-dependent term is evaluated once; token-dependent terms attach to the counted children.

A.3 Distinct descendants and collisions

Resampling creates groups of repeated particles; the proposal determines how quickly their continuations separate again. The occupancy results below relate these two effects to the batch width seen by the model.

For m independent draws $V_1, \dots, V_m$ from categorical distribution q, write K_m for the number of distinct values and C_m for the number of equal-value pairs.

Lemma 1 (Expected support and collisions)**.**

$$\mathbb{E}[K_m] = \sum_v \left[1 - (1 - q(v))^m\right], \qquad \mathbb{E}[C_m] = \binom{m}{2} \sum_v q(v)^2. \tag{14}$$

Proof. Let I_v indicate whether token v occurs. Then $K_m = \sum_v I_v$ and $\Pr(I_v = 1) = 1 - (1 - q(v))^m$, giving the first identity. Any pair of draws agrees with probability $\sum_v q(v)^2$; summing over the $\binom{m}{2}$ pairs gives the second.

Let $s_2(q) = \sum_v q(v)^2$ denote the collision probability of two proposal draws.

Proposition 2 (Collision-based support bounds)**.** *For $m \geq 1$,*

$$\mathbb{E}[K_m] \geq \max\left\{ \frac{m}{1 + (m-1)s_2(q)}, m - \binom{m}{2} s_2(q) \right\}. \tag{15}$$

Proof. Let n_v be the realized occupancy of token v. Cauchy–Schwarz gives $m^2 = (\sum_v n_v)^2 \leq K_m \sum_v n_v^2$, where the sums may be restricted to occupied tokens. Since $x \mapsto 1/x$ is convex,

$$\mathbb{E}[K_m] \geq \frac{m^2}{\mathbb{E}[\sum_v n_v^2]} = \frac{m}{1 + (m-1)s_2(q)}. \tag{16}$$

For the second bound, every repeated occurrence contributes one to $m - K_m$, while a cell of size n_v contains $\binom{n_v}{2} \geq n_v - 1$ equal pairs. Hence $m - K_m \leq C_m$ pointwise. Taking expectations and using Equation 14 proves the result.

Proposition 3 (Sublinear support growth)**.** $\mathbb{E}[K_m]$ *is increasing and concave in m, and*

$$\mathbb{E}[K_{m+1}] - \mathbb{E}[K_m] = \sum_v q(v)(1 - q(v))^m, \tag{17}$$

$$1 \leq \mathbb{E}[K_m] \leq \sum_v \min\{1, mq(v)\} \leq \min\{m, |\operatorname{supp}(q)|\}. \tag{18}$$

The ratio $\mathbb{E}[K_m]/m$ is non-increasing in m.

Proof. Subtracting consecutive instances of Equation 14 gives the increment. Each term is nonnegative and decreases with m, proving monotonicity and concavity. The upper bounds follow from $1 - (1 - q)^m \leq \min\{1, mq\}$. A nonnegative concave sequence with value zero at $m = 0$ has non-increasing secant slope, hence $\mathbb{E}[K_m]/m$ is non-increasing.

Proposition 4 (Concentration order)**.** *Let q' and q be categorical distributions on the same support. If q' majorizes q, then for every integer $m \geq 1$,*

$$\mathbb{E}_{q'}[K_m] \leq \mathbb{E}_q[K_m]. \tag{19}$$

Proof. For fixed m, $f_m(x) = 1 - (1 - x)^m$ is concave on $[0, 1]$. The expected support $\sum_v f_m(q(v))$ is therefore a symmetric concave function of the probability vector and hence Schur-concave. Majorization gives Equation 19.

Proposition 5 (Power sharpening)**.** *For $q_\beta(v) = p(v)^\beta / Z(\beta)$ on a fixed positive support and $\beta \geq 0$, $\mathbb{E}_{q_\beta}[K_m]$ is non-increasing and the collision probability $\sum_v q_\beta(v)^2$ is non-decreasing in β.*

Proof. Order the support so that $p(1) \geq \cdots \geq p(V)$. For every $k < V$, the derivative of the leading cumulative mass is

$$\frac{d}{d\beta} \sum_{v \leq k} q_\beta(v) = \operatorname{Cov}_{q_\beta}(\mathbf{1}\{v \leq k\}, \log p(v)) \geq 0. \tag{20}$$

Thus q_{β_2} majorizes q_{β_1} whenever $\beta_2 \geq \beta_1$, and Equation 19 proves the support statement. The collision probability is $Z(2\beta)/Z(\beta)^2$. Differentiating its logarithm gives

$$\frac{d}{d\beta} \log \sum_v q_\beta(v)^2 = 2 \left(\mathbb{E}_{q_{2\beta}}[\log p] - \mathbb{E}_{q_\beta}[\log p] \right) \geq 0, \tag{21}$$

because $d\mathbb{E}_{q_\gamma}[\log p]/d\gamma = \mathrm{Var}_{q_\gamma}(\log p) \geq 0$.

Equation 18 and Proposition 5 explain the reuse observed in the experiments. The fraction $\mathbb{E}[K_m]/m$ falls as resampling places more particles in one group, while a sharper proposal produces fewer distinct descendants at a fixed multiplicity. ASMC consequently exposes more reusable state than Power-SMC, and the difference grows with the particle count.

A.4 Proposals from a grouped state

For a group of m particles in state z, let K_v count the identifiers that draw token v. The count vector follows the multinomial law

$$\Pr((K_v)_v = (k_v)_v) = \frac{m!}{\prod_v k_v!} \prod_v q_t(v \mid z)^{k_v}, \qquad \sum_v k_v = m. \tag{22}$$

Lemma 2 (Multiplicity-aware transition)**.** *For a group of m particles in state z,* MDH *produces independent draws from $q_t(\cdot \mid z)$. It returns each selected token with its count, identifiers, proposal density, target density, and the weight increment in Equation 6.*

Proof. For particle j, draw component $C_j \sim \mathrm{Cat}(\pi_1, \ldots, \pi_C)$. Let $\mathcal{V}_r$ be a vocabulary block and define

$$B_{c,r} = \sum_{v \in \mathcal{V}_r \cap \mathcal{A}} \exp(\beta_c \ell_v). \tag{23}$$

Conditional on component c, block r is selected with probability $B_{c,r}/Z_{\beta_c}$; conditional on that block, token v has probability $\exp(\beta_c \ell_v)/B_{c,r}$. Hence

$$\Pr(V_j = v) = \sum_c \pi_c \frac{B_{c,r(v)}}{Z_{\beta_c}} \frac{\exp(\beta_c \ell_v)}{B_{c,r(v)}} \tag{24}$$

$$= q_t(v \mid z). \tag{25}$$

Different identifiers use independent coordinates, so the labeled draws are independent. Grouping equal tokens yields the count law in Equation 22. Finally, Equation 8 evaluates every selected token using all component normalizers and supplies the proposal density required by the importance weight.

Corollary 1 (Additive statistics)**.** *For any function f of a parent state and sampled token,*

$$\sum_{j=1}^{m} f(z, V_j) = \sum_{v : k_v > 0} k_v f(z, v). \tag{26}$$

Thus likelihood sums, answer masses, and other additive particle statistics can be evaluated directly on counted children.

Proof. Partition the labeled draws by token value. The class for token v has cardinality k_v, and $f(z, V_j) = f(z, v)$ for every member of that class.

The one-step model batch width follows by summing the occupancy result over active parent groups. Distinct parent states contain different complete histories, so appending a token cannot make children from two parents equal. Conditioned on $\mathcal{F}_t$,

$$\mathbb{E}[U_{t+1} \mid \mathcal{F}_t] = \sum_{i=1}^{U_t} \sum_v \left[1 - (1 - q_t(v \mid z_i))^{m_i} \right]. \tag{27}$$

In Equation 27, the current groups determine the multiplicities m_i, and the proposal determines how many distinct children each group produces.

A.5 Resampling and terminal estimators

The expanded population contains m_i copies of weight $w_i = e^{\lambda_i}$.

Lemma 3 (Counted ESS)**.** *The ESS of the expanded population equals the multiplicity-weighted ESS of the counted population.*

Proof. Collecting equal terms in the expanded ESS gives

$$\frac{(\sum_j w_j)^2}{\sum_j w_j^2} = \frac{(\sum_i m_i w_i)^2}{\sum_i m_i w_i^2}, \tag{28}$$

which is Equation 9.

For systematic resampling (Douc & Cappe, 2005), define cumulative group masses

$$G_i = \frac{\sum_{k \le i} m_k w_k}{\sum_k m_k w_k}, \qquad G_0 = 0, \tag{29}$$

and draw $u \sim \mathrm{Unif}[0, 1/N)$. The offspring multiplicity of group i is

$$m'_i = \sum_{j=0}^{N-1} \mathbf{1}\left\{ G_{i-1} \le u + \frac{j}{N} < G_i \right\}. \tag{30}$$

Lemma 4 (Counted systematic resampling)**.** *Under the canonical order, Equation 30 produces the same labeled offspring as systematic resampling of the expanded population.*

Proof. In canonical order, the m_i labeled copies occupy adjacent intervals whose union is $[G_{i-1}, G_i)$. Counting systematic points in the individual intervals or in their union gives the same m'_i. The canonical identifier order assigns those offspring to the same labeled parents.

If $g(z)$ extracts the answer of terminal state z, the terminal answer mass under ASMC (Wang et al., 2026) is

$$\widehat{P}(a) = \frac{\sum_i m_i w_i \mathbf{1}\{g(z_i) = a\}}{\sum_i m_i w_i}. \tag{31}$$

Power-SMC chooses group i with probability $m_i w_i / \sum_k m_k w_k$ and then returns its trajectory (Azizi et al., 2026). These expressions are the corresponding labeled estimators with equal terms collected.

A.6 Equivalence to sequence-wise transitions

Let P_t be the labeled transition kernel for one token step, including a possible resampling decision, and let $\widetilde{P}_t$ be its counted counterpart. For an event $\mathcal{A}$ over labeled populations, $\mathsf{Expand}^{-1}(\mathcal{A})$ denotes the counted populations whose expansions lie in $\mathcal{A}$.

Lemma 5 (Commuting transition)**.** *For every valid counted population $\mathcal{F}$ and every event $\mathcal{A}$,*

$$\widetilde{P}_t\big(\mathcal{F}, \mathsf{Expand}^{-1}(\mathcal{A})\big) = P_t(\mathsf{Expand}(\mathcal{F}), \mathcal{A}). \tag{32}$$

Proof. Lemma 2 couples the proposal draws identifier by identifier, so expansion gives the same labeled tokens, densities, and weight updates. The counted ESS gives the same resampling decision. When resampling occurs, Lemma 4 gives the same offspring under the shared systematic point. Collecting equal children changes only the representation and is undone by Expand according to Proposition 1.

Theorem 1 (Counted particle process)**.** *At every token and resampling boundary, expanding the multiplicities produced by* PluralGen *gives the same distribution over labeled particle states as the corresponding ASMC or Power-SMC process. Their terminal estimators therefore have the same distribution.*

Proof of Theorem 1. Initially, the single group of multiplicity N expands to the labeled initial population. Apply Equation 32 inductively at every token boundary. Absorbed trajectories follow the same stopping rule and therefore remain coupled after leaving the active population. At termination, Equation 31 and the grouped trajectory probabilities are the labeled estimators with equal terms collected. The terminal outputs thus have the same distribution.

A.7 Compute and memory analysis

A.7.1 Selected vocabulary blocks

Let $Q_r = \sum_{v \in \mathcal{V}_r} q(v)$ be the proposal mass of vocabulary block r. If R_m blocks are selected by m independent draws, the same occupancy argument as Equation 1 gives

$$\mathbb{E}[R_m] = \sum_r \left[1 - (1 - Q_r)^m\right]. \tag{33}$$

The expected number of refined blocks is therefore concave in multiplicity. A concentrated proposal produces both fewer distinct descendants and fewer blocks requiring token-level projection.

If block r contains V_r token rows, the expected token-level projection volume is

$$\mathbb{E}[W_m] = \sum_r V_r \left[1 - (1 - Q_r)^m\right]. \tag{34}$$

Equation 34 makes the path choice shape-dependent: a few selected large blocks can still favor dense projection, while repeated draws concentrated in small blocks favor the counted path.

A.7.2 Per-step compute

Let $C_{\mathrm{tr}}(b)$ be Transformer cost at batch width b, $C_{\mathrm{head}}(b, V)$ a dense vocabulary projection, and $C_{\mathrm{part}}(N_t)$ particle bookkeeping. The two execution paths can be written as

$$C_{\mathrm{base}}(t) = C_{\mathrm{tr}}(N_t) + C_{\mathrm{head}}(N_t, V) + C_{\mathrm{part}}(N_t), \tag{35}$$

$$C_{\mathrm{PG}}(t) = C_{\mathrm{tr}}(U_t) + \min\{C_{\mathrm{head}}(U_t, V), C_{\mathrm{count}}(U_t, (R_i)_i)\} + C_{\mathrm{part}}(N_t). \tag{36}$$

The counted path retains $O(N_t)$ random draws and scalar particle updates, but the Transformer cost follows U_t. Its vocabulary work depends on the selected blocks R_i from Equation 33; the minimum is the dense-versus-counted choice made by MDH.

A per-step model relates the state ratio to end-to-end latency. Let ϕ_t be the fraction of baseline time spent in row-proportional model work, $\rho_t = U_t/N_t$, and ω_t the counted-transition overhead normalized by baseline step time. Then

$$S_t \approx \frac{1}{(1 - \phi_t) + \phi_t \rho_t + \omega_t}. \tag{37}$$

As resampling lowers ρ_t, a larger fraction of row-proportional work is removed. When ρ_t is near one, the dense head avoids the block-selection overhead and the Transformer has little repeated work to remove. Equation 37 also explains why model-row reduction can be larger than the measured end-to-end gain.

A.7.3 KV growth and capacity

For parent group i, the baseline allocates one child tail per particle draw. PluralGen allocates one tail per distinct outcome. Expected tail counts are

$$T_{\mathrm{base}}(t) = \sum_i m_i = N_t, \qquad \mathbb{E}[T_{\mathrm{PG}}(t)] = \sum_i \sum_v \left[1 - (1 - q_i(v))^{m_i}\right]. \tag{38}$$

If one generated-token KV state occupies b_{KV} bytes and fixed model state occupies M_0, the memory after T decode steps is approximated by

$$M_{\mathrm{base}}(T) = M_0 + b_{\mathrm{KV}} \sum_{t=1}^{T} N_t, \tag{39}$$

$$\mathbb{E}[M_{\mathrm{PG}}(T)] = M_0 + b_{\mathrm{KV}} \sum_{t=1}^{T} \sum_i \mathbb{E}[K_{m_i,t}]. \tag{40}$$

Multiplicities can change during resampling without adding a KV tail, so the difference in Equation 40 accumulates across decoding. That cumulative difference is what shifts the capacity boundary in Figure 5.

For a realized coupled execution ω, define the counted KV envelope

$$M^{\omega}_{\mathrm{PG}}(T) = M_0 + b_{\mathrm{KV}} \sum_{t=1}^{T} U^{\omega}_t. \tag{41}$$

Corollary 2 (Cumulative KV dominance)**.** *For every coupled execution ω,*

$$M_{\mathrm{base}}(T) - M^{\omega}_{\mathrm{PG}}(T) = b_{\mathrm{KV}} \sum_{t=1}^{T} (N_t - U^{\omega}_t) \geq 0. \tag{42}$$

Equality holds exactly when the active population has no repeated complete state at any decode step.

Proof. Every distinct state carries at least one particle, hence $U^{\omega}_t \leq N_t$. Subtracting the two realized memory envelopes gives Equation 42. A strict gap appears at the first step with $U^{\omega}_t < N_t$ and remains in the cumulative allocation.

For concurrent requests $r \in \mathcal{Q}$, let $M^{\mathcal{Q}}_0$ collect the model state and request-specific fixed allocations. The same argument applies after summing their envelopes:

$$M^{\mathcal{Q}}_{\mathrm{PG}}(T) = M^{\mathcal{Q}}_0 + b_{\mathrm{KV}} \sum_{r \in \mathcal{Q}} \sum_{t=1}^{T} U_{r,t} \leq M^{\mathcal{Q}}_0 + b_{\mathrm{KV}} \sum_{r \in \mathcal{Q}} \sum_{t=1}^{T} N_{r,t}. \tag{43}$$

The inequality shows how falling distinct-state counts release KV capacity across concurrent requests.

B Execution Mechanisms

The grouped representation spans the complete decoding path, from model evaluation to the groups used at the next step. This section follows that path through proposal generation, particle updates, KV ownership, and batching.

B.1 Counted decoding

Algorithm 1 gives the particle loop over groups. Between proposal and resampling, the runtime carries each complete state with its multiplicity rather than materializing one sequence row per particle.

Density path selection. Dense and counted projection suit different group structures. Dense projection evaluates the vocabulary once for each of the U states; counted blocks refine the regions selected by the particle draws. The runtime chooses the path from the current state count, group multiplicities, and proposal type. Both paths produce the transition in Equation 7.

For the counted path, component c first forms the normalized block distribution

$$P_c(r) = \frac{B_{c,r}}{Z_{\beta_c}}, \qquad B_{c,r} = \sum_{v \in \mathcal{V}_r \cap \mathcal{A}} \exp(\beta_c \ell_v). \tag{44}$$

Each particle identifier independently selects its component and block. The selected blocks are compacted into a work list, projected at token granularity, and sampled from their conditional distributions. Sorting by parent state and token yields the tuples in Equation 7. The dense path forms one $[U, V]$ matrix and applies the same count reduction to its draws.

Algorithm 1 Multiplicity-aware particle decoding for one request. Particle identities remain distinct while model execution follows complete-state groups throughout the decoding loop.

Require: prompt x, particle count N, particle method A
Initialize one group at x with multiplicity N and identifiers $0{:}N-1$
while an unfinished group remains **do**
    Evaluate the Transformer for active distinct states
    Construct the proposal and target transforms for A
    Draw counted children with MDH
    Update likelihoods, weights, stopping state, and child counts
    Merge equal complete children and update KV ownership
    if A reaches a resampling boundary **then**
        Compute the multiplicity-weighted ESS
        if ESS triggers resampling **then**
            Canonically order groups and count systematic points
            Replace offspring counts by multiplicities and reset weights
return the terminal estimator specified by A

State and KV ownership. After every token and resampling step, the group multiplicities sum to the particle count. Each active complete state owns one model row, so the physical batch width is the number of distinct states in μ_t.

Suppose parent i owns page table H_i, and its distinct child tokens form $\mathcal{K}_i$. Child tables are

$$H_{i,v} = H_i \,\|\, \mathrm{page}(v), \qquad v \in \mathcal{K}_i, \tag{45}$$

where the prefix pages in H_i are reference counted. Multiplicity changes do not alter $H_{i,v}$; a tail is created only when the complete child state is new. Pages are released when a group becomes empty or terminal.

Batching by distinct states. The scheduler batches requests by model rows and page demand. Execution graphs cover a small set of state-count buckets, while multiplicities and particle identifiers remain tensor inputs. A request moves between buckets as its distinct-state count contracts or expands. Terminal groups leave the model batch but retain their count and weight for answer aggregation or trajectory selection.

B.2 Persistence, projection, and replay

The three mechanisms act on different parts of the same decoding loop. State persistence avoids repeated Transformer and KV work, MDH ties vocabulary work to U_t, and graph replay accommodates the changing state count between steps. Figure 6 separates their effects.

Persistence across decode steps. Suppose one stored state may represent at most c particles. A group of multiplicity m_i then occupies $\lceil m_i/c \rceil$ model rows, so the capped batch width is

$$U_t^{(c)} = \sum_{i=1}^{U_t} \left\lceil \frac{m_i}{c} \right\rceil, \qquad U_t \le U_t^{(c)} \le N_t. \tag{46}$$

Keeping each group intact gives the left endpoint; materializing every particle gives the right. Regrouping equal descendants after each transition carries this difference into the next Transformer step and KV allocation. The multiplicity-cap and persistence ablations compare the two cases.

Proposition 6 (Cumulative model-row reduction)**.** *For a realized generation of T steps, let $R_{\mathrm{base}}(T) = \sum_{t=1}^{T} N_t$ and $R^{(c)}(T) = \sum_{t=1}^{T} U_t^{(c)}$. Then*

$$R_{\mathrm{base}}(T) - R^{(c)}(T) = \sum_{t=1}^{T} \sum_{i=1}^{U_t} \left(m_{t,i} - \left\lceil \frac{m_{t,i}}{c} \right\rceil \right) \ge 0. \tag{47}$$

With unrestricted grouping, the reduction becomes $\sum_{t=1}^{T} (N_t - U_t)$.

Proof. At each step, $N_t = \sum_i m_{t,i}$ and the capped execution uses the row count in Equation 46. Substitution gives Equation 47; each summand is nonnegative for $c \geq 1$. When c is at least every realized multiplicity, each group occupies one row.

Prompt length and decode reuse. Let $P(L)$ denote the shared prefill cost for a prompt of length L, and let $D_{\mathrm{base}}(t)$ and $D_{\mathrm{count}}(t)$ denote the two decode-step costs. For a generation of T steps, the observed speedup is

$$S(L,T) = \frac{P(L) + \sum_{t=1}^{T} D_{\mathrm{base}}(t)}{P(L) + \sum_{t=1}^{T} D_{\mathrm{count}}(t)}. \tag{48}$$

Increasing L enlarges the prefill term shared by both systems, whereas the repeated states arise later during resampling. Prompt length therefore changes the fraction of end-to-end latency attributable to decoding; model-row and KV savings continue to accumulate over the generated tokens.

Projection and replay. For state count U_t and vocabulary size V, MDH chooses between dense projection and counted blocks:

$$C_{\mathrm{head}}(t) = \min\{C_{\mathrm{dense}}(U_t, V), C_{\mathrm{block}}(U_t, (m_i)_i)\}. \tag{49}$$

Dense projection is preferable while most multiplicities are one. As resampling concentrates particles into groups, counted blocks amortize vocabulary work over more draws. Expanding the head to N_t rows repeats the projection, whereas disabling graph replay restores the dispatch cost of successive batch shapes. The corresponding ablations isolate these costs.

Batch-width buckets. Graph replay uses a geometric family of batch widths. For $\gamma > 1$, let $b_k = \lceil \gamma^k \rceil$ and select the smallest indexed width

$$b(U) = \min\{b_k : b_k \geq U\}. \tag{50}$$

If $b(U) = b_k$ with $k > 0$, minimality gives $\lceil \gamma^{k-1} \rceil < U$. Hence

$$U \leq b(U) < \gamma U + 1, \qquad 0 \leq b(U) - U < (\gamma - 1)U + 1. \tag{51}$$

The same bound holds for $k = 0$. Thus replay accommodates a changing state count with bounded padding. Up to a maximum width $U_{\max}$, at most $1 + \lceil \log_\gamma U_{\max} \rceil$ indexed graphs are needed. The active states occupy the first U_t rows of the selected graph, while multiplicities, particle identifiers, and KV page tables remain runtime inputs. Moving between buckets changes the executed shape but not the particle transition.

Concurrent serving. For active requests $r \in \mathcal{Q}$, batching uses $\sum_r U_{r,t}$ model rows, while the particle algorithms retain $\sum_r N_{r,t}$ particles. Scheduling by the distinct-state count shortens iterations and leaves KV capacity for other requests, which produces the serving behavior in Table 2.