Step Smarter, Not Harder: Queue-Aware Diffusion Sampling

Flaminia Trinca [*1] **Allizha Theiventhiram** [*2]

Abstract

Diffusion models can generate high-quality images, but inference is slow because sampling requires many sequential denoising steps. This makes interactive deployment difficult: when requests arrive over time, queueing delay can dominate end-to-end latency. We study a simple queue-aware early-exit policy that adapts the sampler family (DDPM vs. DDIM) and the denoising step budget using only the current queue length. We profile inference runtimes, fit a lightweight service-time model $S(n) \approx \alpha + \beta n$, and embed it in an event-driven single-server queue simulator with Poisson arrivals. To compare adaptive and fixed policies fairly, we report *effective* quality by averaging CLIPScore over the configurations executed during simulation. Overall, the queue-aware policy substantially reduces mean and tail latency under load compared to fixed high-quality sampling, while retaining higher effective quality than always using a short-step sampler.

1. Introduction

Diffusion models produce high-quality images, but inference is slow. Generating a single sample requires many sequential denoising steps, so latency grows almost linearly with the step budget. This makes diffusion models difficult to deploy in interactive settings where users expect fast and predictable responses.

The problem becomes more severe in *online serving*. Requests arrive over time and share a limited amount of compute. When arrivals temporarily exceed service capacity, a queue forms and each request experiences both *service time* (actual inference) and *waiting time* (time spent in the queue). Under moderate to high load, waiting time can dominate end-to-end latency. As a result, inference configurations that appear acceptable in offline benchmarks can lead to extremely poor responsiveness once deployed behind a realistic arrival process.

Most diffusion systems still rely on *fixed* inference configurations chosen mainly for offline quality, such as DDPM with a large number of steps. While this maximizes image quality, it can cause latency to explode under load. At the other extreme, always using a very fast sampler with few steps keeps latency low, but unnecessarily sacrifices quality when the system is lightly loaded. Neither approach adapts to changing demand.

In this project, we ask a simple systems question: *Can diffusion inference adapt its computation online to current congestion in order to improve the latency–quality trade-off?* Our key idea is to make inference *elastic*. When the queue is short, the system can afford high-quality sampling. When the queue grows, the system should reduce computation to prevent waiting times from spiraling. We implement this idea with a lightweight queue-aware policy that switches both the sampler family (DDPM vs. DDIM) and the number of denoising steps based only on the current queue length.

To study this mechanism end-to-end, we model diffusion serving as a single-server FIFO queue with Poisson arrivals. We first profile inference runtimes and fit a simple, interpretable service-time model, $S(n) \approx \alpha + \beta n$. We then embed this model into an event-driven queue simulator that explicitly tracks waiting time, service time, and total latency as load increases. To compare adaptive and fixed policies fairly, we link online behavior to offline image quality measurements by reporting an *effective* policy quality, defined as the average CLIPScore of the configurations actually used during simulation.

Contributions. This work makes the following contributions: (i) a simple three-level queue-aware diffusion policy that adapts sampler and step budget using only queue length; (ii) empirical validation that a linear service-time model is sufficient for system-level analysis of diffusion inference; (iii) an end-to-end queueing evaluation that separates waiting time from computation time under realistic load; and (iv) a latency–quality analysis, including sensitivity to thresholds, step budgets, and sampler switching. Together, these

[*]Equal contribution [1]University of Bern, Switzerland [2]University of Neuchâtel, Switzerland. Correspondence to: Flaminia Trinca <flaminia.trinca@students.unibe.ch>, Allizha Theiventhiram <allizha.theiventhiram@unine.ch>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

results show that even very simple queue-aware control can dramatically improve latency under load while preserving most of the achievable image quality.

2. Generative AI Application

We study image generation with diffusion models in an online serving setting, where users submit requests continuously and expect results within a short and predictable time. From a systems viewpoint, each request triggers a diffusion sampling process whose runtime is determined by the chosen inference configuration. Unlike offline batch generation, where throughput is the main concern, an online system must handle a stream of requests with varying load while keeping end-to-end latency under control.

We focus on two inference-time choices that largely determine compute cost: (i) the **sampler family** (DDPM vs. DDIM) and (ii) the **number of denoising steps** n. DDPM is the standard sampler and is commonly used as a high-quality baseline, but its many sequential steps make it computationally expensive. DDIM uses a modified update rule that often allows acceptable quality with fewer steps. Reducing n directly reduces the number of network evaluations and therefore shortens service time, but may degrade image quality when taken too far.

A central challenge is that these choices must be made *online*, without knowing future arrivals. Running an expensive configuration while the queue is long increases waiting time and can destabilize the system. Running a cheap configuration when the system is idle wastes available compute and unnecessarily sacrifices quality. This tension motivates queue-aware adaptation: use high-quality sampling when congestion is low, and progressively reduce computation as the queue grows.

To evaluate this idea across different compute regimes, we run experiments on three image datasets with pretrained diffusion models. As shown in Figure 1, these datasets span a wide range of inference costs. `cifar10` consists of small, low-resolution images and represents a lightweight workload. `flowers` contains higher-resolution natural images and leads to moderate inference cost. `faces` has the highest per-step compute cost, making it a challenging setting where queueing effects dominate latency even at modest arrival rates. This diversity allows us to study queue-aware inference in regimes where service time dominates as well as in regimes where waiting time dominates overall latency.

3. Statistical Analysis and Offline Characterization

Before introducing queue-aware early-exit, we first build a reliable offline measurement pipeline and extract the quantities needed to simulate online serving. Concretely, we (i) validate diffusion sampling for all datasets and samplers, (ii) define and sanity-check quality metrics (CLIPScore and FID), (iii) profile inference latency as a function of denoising steps, and (iv) fit a simple service-time model that becomes the input to the queue simulator.

Table 1. Sanity-check timing at 50 denoising steps (batch size $n\_imgs = 4$). DDPM and DDIM have similar per-step cost; dataset/model choice dominates runtime.

dataset	sampler	total (s)	per-step (ms)
cifar10	DDPM	1.018	20.36
cifar10	DDIM	0.986	19.71
flowers	DDPM	1.428	28.57
flowers	DDIM	1.408	28.15
faces	DDPM	9.569	191.38
faces	DDIM	9.630	192.61

3.1. Datasets and diffusion pipelines

We study three pretrained DDPM checkpoints that represent clearly different compute regimes: `cifar10` (lightweight), `flowers` (medium), and `faces` (heavy). Although images are resized consistently, the underlying UNet architectures and compute cost differ substantially across checkpoints, which strongly affects per-step runtime.

We support two samplers: **DDPM** (the standard stochastic reverse process) and **DDIM** (a deterministic/improved sampling variant). In our implementation, both samplers use the same UNet per denoising step; DDIM is not expected to be cheaper per step, but it can often reach acceptable quality with fewer steps.

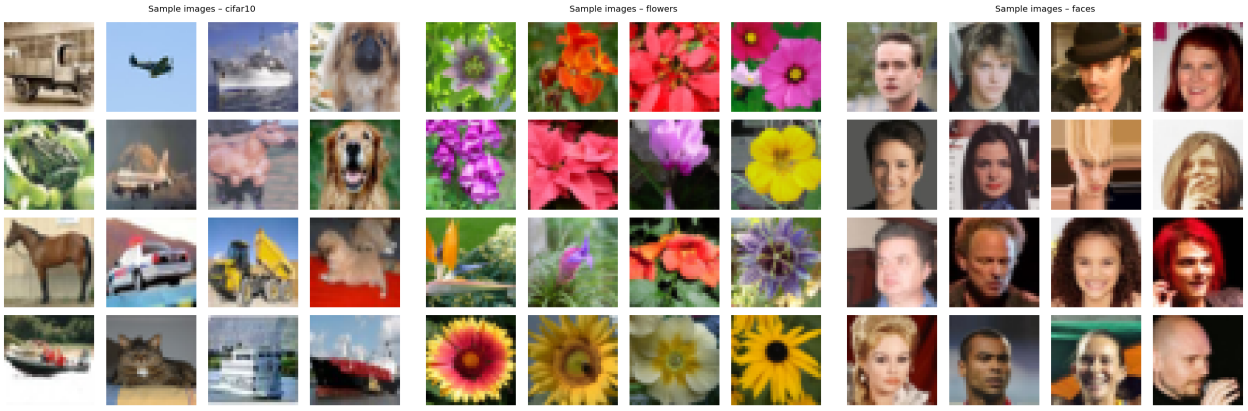
Table 1 reports a representative timing snapshot at 50 steps (batch size $n\_imgs = 4$). The dominant effect is the dataset/model: `faces` is roughly an order of magnitude more expensive per step than `cifar10` and `flowers`. Within each dataset, DDPM and DDIM have very similar per-step cost, supporting the key intuition that DDIM is mainly useful because it can run with fewer steps, not because each step is faster.

3.2. Quality metrics: CLIPScore and FID

Our serving policy changes the inference configuration $(\text{sampler}, n)$, which changes both service time and the distribution of generated images. To compare policies fairly, we need quality metrics that can detect the effect of fewer steps and sampler choices.

We use two complementary metrics:

- **CLIPScore (higher is better):** a fast semantic signal. In our setting it serves mainly as a *relative* indicator of how quality trends with step count and policy deci-



(a) `cifar10` (b) `flowers` (c) `faces`

Figure 1. Example samples from the three datasets used in our study. These datasets span low-resolution objects (`cifar10`), mid-resolution natural images (`flowers`), and higher-latency portrait-style images (`faces`), leading to very different service-time regimes.

Table 2. Sanity-check quality metrics at 50 steps (batch size 8). CLIPScore is stable but close across settings; FID is noisy at this small scale.

dataset	sampler	CLIP ↑	FID ↓	time (s)
cifar10	DDPM	0.224	402.64	1.04
cifar10	DDIM	0.222	342.34	1.33
flowers	DDPM	0.236	275.05	1.80
flowers	DDIM	0.229	277.13	2.03
faces	DDPM	0.234	364.78	19.04
faces	DDIM	0.228	369.28	19.19

sions.

- **FID (lower is better):** a distributional similarity metric that is more sensitive to perceptual fidelity, but statistically noisy at small sample sizes.

Table 2 shows a small sanity run at 50 steps (batch size 8). All values are finite and stable, confirming that our evaluation code works across datasets and samplers. As expected, CLIP scores are tightly clustered (generic prompt, small batch), while FID is large and noisy due to the small sample size. In later experiments, FID is computed on much larger batches to make it meaningful.

3.3. Baseline latency profiling over step count

To simulate online serving, we must know how long a single request takes as a function of the denoising steps. We therefore profile runtime for each dataset and each sampler over

$$n \in \{5, 10, 25, 50, 100, 250\},$$

using a small fixed batch size (default $n\_imgs = 4$) and fixed seeds for repeatability. Each run logs: (i) total wall-clock time $S(n)$ and (ii) per-step time $S(n)/n$.

The profiling results show a consistent pattern across datasets:

- total runtime grows approximately linearly in n,
- per-step cost depends strongly on the dataset/model,
- DDPM and DDIM have similar per-step cost at equal n.

This validates the core assumption used later: the *main lever* to reduce service time is reducing the number of steps.

3.4. Service-time model fit

To embed diffusion inference into a queueing model, we fit a simple linear service-time approximation:

$$S(n) \approx \alpha + \beta n, \tag{1}$$

where α captures fixed overheads (pipeline/setup costs) and β captures the marginal cost per denoising step (UNet forward pass). We fit (α, β) via least squares separately for each dataset and sampler using the profiling logs above.

Figure 2 visualizes the fitted curves across datasets and samplers. We plot the y-axis on a log scale because the slopes differ by more than an order of magnitude; otherwise the `faces` model would visually dominate the figure. The main takeaway is that the slope β is dataset-dependent (large for `faces`), while DDPM/DDIM differences are comparatively small.

3.5. Queueing model baseline validation (fixed policies only)

Up to this point, we measure inference in isolation. In an online system, however, requests arrive over time and

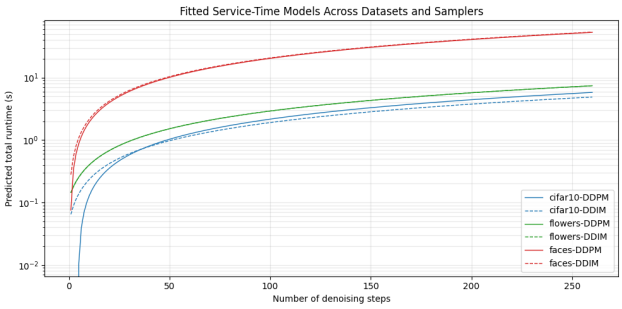


Figure 2. Fitted linear service-time models $S(n) \approx \alpha + \beta n$ for all datasets and samplers (log-scale y-axis). Dataset/model effects dominate the per-step slope β; DDPM and DDIM have similar per-step cost at equal step count.

compete for a single GPU. To validate our online setup before introducing adaptation, we simulate a FIFO single-server queue (M/G/1 abstraction):

- **M:** Poisson arrivals with rate λ (jobs/s), i.e., inter-arrival times $\sim \text{Exp}(\lambda)$,
- **G:** service times given by the fitted model $S(n)$ in Equation (1),
- **1:** one server (one GPU), FIFO scheduling.

For each job we record arrival time, service start time, and finish time, yielding the decomposition: waiting time W, service time S, and total latency $T = W + S$.

To sanity-check the simulator behavior, we run three fixed baselines: **DDPM-100** (slow/high quality), **DDIM-50** (medium), and **DDIM-20** (fast/low compute) under three loads $\lambda \in \{0.3, 0.6, 0.9\}$. Table 4 (appendix) reports mean waiting, service, and total latency.

Two patterns appear immediately. First, *waiting time grows rapidly with load* for slow policies, even though their service time is constant. Second, *small reductions in service time lead to very large reductions in waiting time*, especially for the expensive `faces` workload. This establishes the central motivation for queue-aware early-exit: under congestion, controlling S is the most effective way to control end-to-end latency T.

Summary of insights. This offline-to-online pipeline yields three core insights that drive the remainder of the project: (i) service time scales roughly linearly with the number of denoising steps, (ii) per-step cost is highly dataset/model dependent (especially severe for `faces`), and (iii) under load, waiting time dominates total latency, so even modest reductions in service time can dramatically stabilize end-to-end performance. In the next section, we exploit these observations with a queue-aware early-exit policy that adapts $(\text{sampler}, n)$ based on current congestion.

4. Online Optimization: Queue-Aware Sampling

The previous section established three key facts: (i) diffusion service time grows approximately linearly with the number of denoising steps, (ii) per-step cost varies strongly across datasets but only weakly across samplers, and (iii) under load, queueing delay quickly dominates end-to-end latency. In particular, it showed that even small reductions in service time can lead to disproportionately large reductions in waiting time once the system approaches saturation.

In this section, we turn these observations into a concrete online optimization mechanism. Rather than treating diffusion inference as a fixed-cost operation, we allow the serving system to adapt its computation dynamically based on instantaneous congestion. This queue-aware early-exit policy is the central contribution of the project.

4.1. From measurement to mechanism

Before, we deliberately evaluated only fixed policies (e.g., DDPM-100, DDIM-50, DDIM-20) in order to answer a baseline question: *what happens if computation is held constant while load increases?* The answer was clear: long service times push the system toward high utilization, causing waiting time to explode and making high-quality sampling impractical under sustained load.

The next question is therefore natural: *given real-time congestion, how much computation can the system afford for the next request?* Instead of committing to a single operating point, we allow the system to choose the sampler and number of denoising steps on a per-job basis, using only the current queue state.

4.2. Why queue length as a control signal

Queue length is a simple but powerful indicator of system congestion. It is locally observable, requires no prediction of future arrivals, and directly reflects whether jobs are waiting. Empirically, we showed that: (i) when the queue is short, waiting time is negligible and latency is dominated by service time; (ii) once the queue grows, waiting time increases rapidly; (iii) reducing service time under congestion yields large latency gains.

This motivates a tiered response to load:

Queue length therefore provides exactly the signal needed to decide how much computation the system can afford at a given moment.

Table 3. Queue-aware intuition: match compute to congestion.

Queue state	Dominant cost	Best action
Low load	Quality	Use slow, high-quality sampling
Medium load	Service time	Reduce steps moderately
High load	Waiting time	Aggressively shorten computation

4.3. Three-level early-exit policy

We operationalize this intuition using a simple three-level queue-aware policy. We introduce two queue-length thresholds, T_1 and T_2 with $T_2 > T_1$, which partition system operation into low, medium, and high congestion regimes. In all experiments, we start with $T_1 = 2$ and $T_2 = 5$, corresponding to the onset of queueing and clear congestion, respectively. The sensitivity of these choices is analyzed later via ablations.

We define three computation levels:

$$k_{\text{full}} = 100, \quad k_{\text{med}} = 50, \quad k_{\text{short}} = 20.$$

These are paired with samplers as follows:

- DDPM-100 at low load for maximum quality,
- DDIM-50 at medium load for reduced service time with modest quality loss,
- DDIM-20 at high load for aggressive latency control.

This design is grounded in the profiling results: DDPM and DDIM have similar per-step cost, but DDIM enables acceptable quality at much smaller step counts.

Formally, the policy maps the current queue length q to a configuration:

$$(\text{algo}, k) = \begin{cases} (\text{DDPM}, k_{\text{full}}), & q < T_1, \\ (\text{DDIM}, k_{\text{med}}), & T_1 \le q < T_2, \\ (\text{DDIM}, k_{\text{short}}), & q \ge T_2. \end{cases}$$

The policy is intentionally simple: it uses only queue length, applies per job at arrival time, and integrates directly with the M/G/1 simulator by selecting the service time via the fitted model $S(n) = \alpha + \beta n$.

4.4. Experimental evaluation

We now evaluate whether this adaptive mechanism improves system behavior in practice. Experiments are designed to answer three questions: (i) does the policy stabilize latency under load, (ii) how does quality change as computation is reduced, and (iii) what quality do users effectively experience once adaptation is taken into account.

Experiment 1: Latency vs. load. We stress-test the simulator by increasing the arrival rate $\lambda \in \{0.3, 0.6, 0.9\}$ and measuring mean waiting, service, and total latency. For each configuration we run multiple independent simulations and report means with 95% confidence intervals.

Figure 3 shows mean response time as a function of λ across datasets. Fixed DDPM-100 exhibits rapidly growing latency as load increases, reflecting queue instability caused by long service times. Fixed DDIM-50 and DDIM-20 are more stable but always operate at reduced quality. The queue-aware policy consistently lies between these extremes: under low load it incurs higher latency because it favors quality, but as λ increases it adapts and prevents latency from exploding. This effect is especially pronounced for the `faces` dataset, where adaptation reduces latency by several hundreds of seconds compared to DDPM-100 at high load.

Figure 4 decomposes latency into waiting and service components. As load increases, waiting time quickly dominates for slow fixed policies. The adaptive policy suppresses this growth by reducing service time under congestion, illustrating the nonlinear queueing effect: small reductions in service time yield large reductions in waiting time.

Experiment 2: Offline quality grid. Queue simulations do not generate images, so we separately measure the quality of all sampler configurations used by the policies. For each dataset and configuration (DDPM-100, DDIM-50, DDIM-20), we generate 512 images and compute CLIPScore and FID. The resulting quality grid serves as a lookup table that maps $(\text{sampler}, k)$ to expected quality.

Results show that reducing the number of steps does not catastrophically degrade quality for any dataset. For CIFAR-10, DDIM configurations outperform DDPM even at moderate step counts. For Flowers, DDPM achieves the best FID but DDIM remains competitive. For Faces, quality differences across configurations are minimal compared to the enormous latency differences observed earlier. These measurements isolate the intrinsic quality–compute trade-off, independent of queueing effects.

Experiment 3: Effective quality under load. Finally, we combine queue dynamics with the offline quality grid to estimate the quality users effectively experience. For each policy and arrival rate, we record how often each configuration is selected and compute a weighted average of CLIPScore and FID.

Figure 5 visualizes the resulting latency–quality trade-off. Fixed policies appear as nearly horizontal lines because their quality is load-independent, while the queue-aware policy traces a load-dependent trajectory. Across all datasets, the adaptive policy remains close to the desirable low-latency, high-quality region as λ increases. In latency-dominated

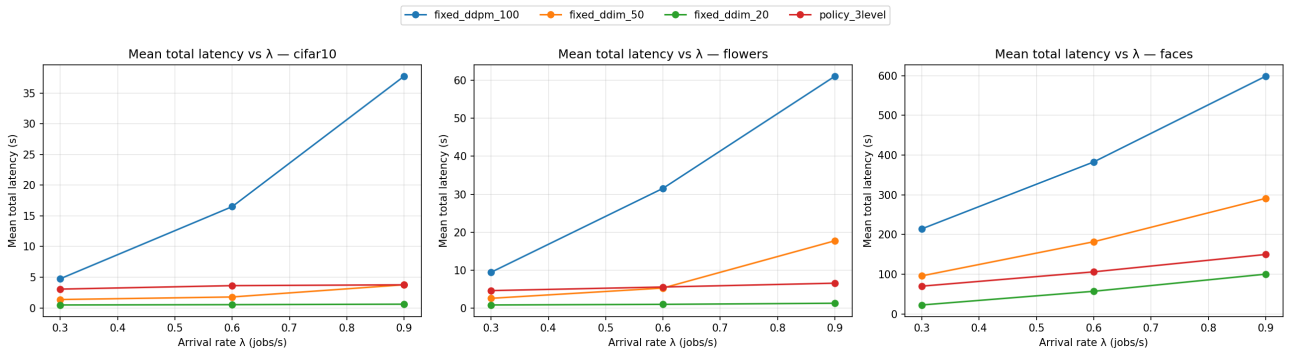


Figure 3. Mean total latency vs. arrival rate λ across datasets. Fixed DDPM-100 becomes increasingly dominated by queueing delay as load increases, while the queue-aware 3-level policy keeps end-to-end latency bounded by switching to cheaper configurations under congestion.

Figure 4. Latency decomposition into average waiting time (queueing) and average service time (inference) under increasing load for `cifar10` and `flowers`. Fixed DDPM-100 accumulates large waiting times at higher λ, while the 3-level policy reduces service time under congestion, limiting queue growth.

regimes such as Faces, adaptation yields massive responsiveness gains with almost no measurable quality loss.

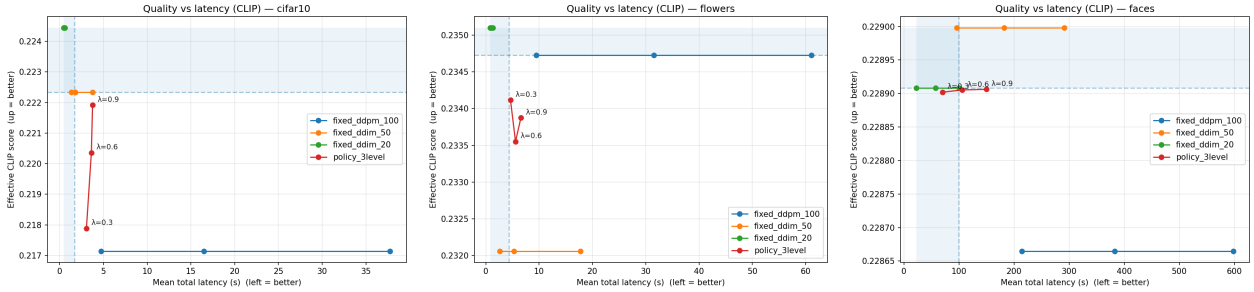
Summary. Together, these experiments show that queue-aware early exit fundamentally changes how diffusion models behave under load. By adapting computation to instantaneous congestion, the system avoids queue collapse, achieves orders-of-magnitude latency reductions compared to fixed high-quality sampling, and preserves most of the achievable image quality. No single fixed configuration can achieve this behavior across all load regimes.

5. Ablations and Sensitivity Analysis

We now ask a more practical question: *which parts of the early-exit mechanism actually matter, and how stable is the policy if we change them?*

Our queue-aware diffusion policy has three main **knobs**: (i) the queue-length **thresholds** (T_1, T_2) that determine *when* we switch to cheaper computation, (ii) the **step budgets** $(k_{\text{full}}, k_{\text{med}}, k_{\text{short}})$ that determine *how much* computation we reduce at each tier, and (iii) **sampler switching** (DDPM $\rightarrow$ DDIM), i.e., whether mixing samplers is necessary beyond changing step counts.

Across all ablations we use the same simulation setup as before: the three datasets, arrival rates $\lambda \in \{0.3, 0.6, 0.9\}$ jobs/s, horizon $H = 60$s, and service times from the fitted linear model $S(n) \approx \alpha + \beta n$. To connect latency and quality, we reuse the offline quality grids from the previous section, and compute effective policy quality by averaging the CLIP/FID values of the configurations actually used by jobs in each run.



(a) `cifar10` (b) `flowers` (c) `faces`

Figure 5. Quality–latency trade-off using effective CLIP score. Each point corresponds to one policy under a given arrival rate. The shaded quadrant highlights configurations that are both faster (left) and higher quality (up) relative to a reference point.

5.1. Ablation 1: Sensitivity to queue thresholds (T_1, T_2)

The thresholds control how early the mechanism reacts to congestion. If T_1 is too small, the policy switches to DDIM too early and may reduce quality even when the system is lightly loaded. If T_1 and T_2 are too large, the policy reacts too late: queues form, waiting time dominates, and total latency can blow up. We evaluate $T_1 \in \{1, 2, 4\}$ and $T_2 \in \{3, 5, 8\}$ with $T_2 > T_1$, while keeping step budgets fixed to the default $(k_{\text{full}}, k_{\text{med}}, k_{\text{short}}) = (100, 50, 20)$.

The results *(Appendix: Table 5)* show that threshold choice primarily affects **waiting time growth**, not computation: service time stays within a narrow range because the same three tiers exist, but *when* we enter cheaper tiers strongly determines whether the queue stays stable. The Pareto plots (Table 7) at $\lambda = 0.6$ reveal a clean latency–quality frontier: aggressive thresholds reduce latency but slightly reduce effective CLIP, while conservative thresholds preserve quality but can incur large queueing delay. The latency heatmaps (Table 6) make this sensitivity very clear: latency increases as thresholds become more conservative, with especially bad outcomes when both T_1 and T_2 are large. Finally, the robustness curves (Table 8) (best vs. worst threshold choices) show that the gap between good and bad threshold settings widens as load increases, and is most pronounced for more expensive workloads (notably `faces`). Overall, this ablation confirms that threshold tuning is a first-order control lever because it decides *how early* we cut service time to avoid nonlinear queueing amplification.

5.2. Ablation 2: Sensitivity to step budgets $(k_{\text{full}}, k_{\text{med}}, k_{\text{short}})$

Next, we keep thresholds fixed (default $T_1 = 2, T_2 = 5$) and vary the step budgets to test how strongly compute scaling affects latency and quality. We compare three step-budget variants: **conservative** $(100, 50, 20)$ (default), **aggressive** $(80, 40, 10)$ (faster, riskier), and **quality-biased** $(150, 80, 40)$ (more compute, potentially higher quality, but higher service time).

The results *(Appendix: Table 6)* show a consistent pattern across datasets: changing step budgets shifts the system along a clear latency–quality spectrum. The heatmaps over $(\lambda, \text{variant})$ (Table 9) show monotonic behavior: higher load and larger step budgets both increase latency, and the quality-biased variant is the least robust under congestion because higher service time amplifies waiting time. The Pareto plots at $\lambda = 0.6$ (Table 10) highlight that step budgets alone can span a wide range of operating points: aggressive settings achieve the lowest latency but generally lower effective CLIP, while quality-biased settings move toward higher quality but at substantially higher latency. The robustness curves (Table 11) reinforce that step-budget choice directly affects stability under load: aggressive settings scale best, quality-biased settings scale worst. Overall, this ablation confirms that step budgets are a powerful (but blunt) lever: they directly control service time, and therefore indirectly control queueing delay.

5.3. Ablation 3: Role of sampler switching (DDPM → DDIM)

Finally, we isolate whether sampler switching is necessary, beyond adapting step counts. The default mechanism uses DDPM in the full tier (to prioritize quality when uncongested) and DDIM in cheaper tiers (to enable low-step sampling under load). To test this, we compare: **ddpm_only** (all tiers use DDPM), **ddim_only** (all tiers use DDIM), and **mixed** (DDPM at the full tier, DDIM otherwise; our default design).

The results *(Appendix: Table 7)* suggest that switching samplers is a *secondary* effect compared to step adaptation. The latency heatmaps (Table 12) show that differences between ddpm_only, ddim_only, and mixed are relatively small compared to the dominant effect of arrival rate and workload scale; this matches our earlier profiling observation that DDPM and DDIM have similar per-step costs at equal step

counts. The Pareto plots at $\lambda = 0.6$ (Table 13) show that the three strategies are often close: in some datasets DDIM-only is slightly faster, while DDPM-only can be slightly higher-quality, and the mixed strategy tends to sit between them rather than consistently dominating both. The robustness curves (Table 14) confirm that all three switching strategies scale similarly with λ and do not introduce instability. Overall, this ablation supports a key design insight: **most of the system-level gain comes from reducing steps under congestion**. Sampler switching is safe and can be helpful to support low-step operation, but it is not the primary driver of queue stabilization.

Takeaway. Across all ablations, the policy's robustness mainly comes from adapting compute *early enough* (thresholds) and *strongly enough* (step budgets) to prevent waiting-time blow-up. Sampler switching is comparatively less important: it may fine-tune the operating point, but the dominant mechanism is still early exit via step reduction under congestion.

6. Conclusion

In this project, we studied diffusion model inference as an online queueing problem and asked whether inference computation can be adapted dynamically to reduce end-to-end latency under load. By combining empirical profiling, simple service-time modeling, and queueing-theoretic simulation, we evaluated how diffusion sampling behaves when placed behind a realistic stream of requests rather than executed in isolation.

Our results strongly support the central hypothesis of the project. Across all datasets and arrival rates, fixed high-quality sampling (e.g., DDPM with a large step budget) quickly becomes impractical under moderate to high load: service times remain constant, but waiting time grows rapidly and dominates total latency. Conversely, always using a fast low-step sampler stabilizes latency but sacrifices quality even when the system is lightly loaded. The proposed queue-aware three-level policy consistently strikes a better balance by reducing computation only when congestion arises. This adaptive behavior yields large reductions in both mean and tail latency while preserving substantially higher effective image quality than aggressive fixed baselines.

Beyond demonstrating the effectiveness of queue-aware early exit, the project provides several system-level insights. First, a simple linear service-time model $S(n) \approx \alpha + \beta n$ is sufficient to capture the dominant behavior of diffusion inference and to predict queueing effects accurately. Second, multi-level adaptation leads to smoother and more stable behavior than binary switching, avoiding abrupt quality drops and latency oscillations. Third, quality degradation introduced by moderate early exit remains controlled for the evaluated models, revealing a practical "sweet spot" in the latency–quality trade-off. Finally, our ablation studies show that the mechanism is robust: reasonable threshold choices, step-budget variants, and sampler switching strategies all preserve the core benefits of queue-aware adaptation.

Overall, this work demonstrates that system-level control is a powerful complement to model-level optimization. Rather than treating diffusion inference as a fixed-cost operation, making computation elastic with respect to queue state can dramatically improve scalability and responsiveness in online serving scenarios. These results suggest that queue-aware adaptive inference is a promising direction for deploying diffusion models in interactive and real-time systems.

Limitations and future work. Our simulator models a single-GPU FIFO system with Poisson arrivals and deterministic service times derived from fitted models. Future work could explore multi-server deployments, bursty or correlated arrival processes, stochastic service-time variability, and learned or continuously adaptive policies. On the quality side, expanding the offline quality grid and incorporating additional perceptual metrics would enable finer-grained adaptation and stronger quality guarantees. Despite these limitations, the project shows that even simple, interpretable queue-aware mechanisms can yield large system-level gains for generative AI serving.

Code. All code, experiments, and plots are available in our public repository (Trinca & Theiventhiram, 2025).

References

Trinca, F. and Theiventhiram, A. Step smarter, not harder: Queue-aware diffusion sampling. https://github.com/ftrinca/queue-aware-diffusion-sampling, 2025. Course project, Swiss Joint Master in Computer Science.

A. Appendix

Table 4. Baseline queue simulator validation (means over a 60s horizon). Queueing delay dominates for slow policies as λ increases; reducing service time sharply reduces waiting time.

dataset	policy	λ	mean wait (s)	mean service (s)	mean total (s)
cifar10	DDPM-100	0.30	4.01	2.16	6.17
	DDIM-50	0.30	0.35	0.98	1.33
	DDIM-20	0.30	0.06	0.42	0.48
	DDPM-100	0.60	17.40	2.16	19.56
	DDIM-50	0.60	1.00	0.98	1.97
	DDIM-20	0.60	0.10	0.42	0.52
	DDPM-100	0.90	38.67	2.16	40.84
	DDIM-50	0.90	3.34	0.98	4.32
	DDIM-20	0.90	0.26	0.42	0.68
flowers	DDPM-100	0.30	9.44	2.92	12.35
	DDIM-50	0.30	1.30	1.52	2.81
	DDIM-20	0.30	0.16	0.68	0.84
	DDPM-100	0.60	31.74	2.92	34.66
	DDIM-50	0.60	5.12	1.52	6.63
	DDIM-20	0.60	0.32	0.68	0.99
	DDPM-100	0.90	61.69	2.92	64.61
	DDIM-50	0.90	18.92	1.52	20.43
	DDIM-20	0.90	1.02	0.68	1.70
faces	DDPM-100	0.30	218.03	20.31	238.34
	DDIM-50	0.30	98.68	10.36	109.05
	DDIM-20	0.30	24.59	4.19	28.78
	DDPM-100	0.60	362.15	20.31	382.46
	DDIM-50	0.60	173.19	10.36	183.55
	DDIM-20	0.60	55.88	4.19	60.07
	DDPM-100	0.90	592.09	20.31	612.40
	DDIM-50	0.90	288.75	10.36	299.12
	DDIM-20	0.90	100.44	4.19	104.63

Table 5. Ablation 1 (Threshold Sensitivity). Summary at medium load ($\lambda = 0.6$). Each row reports mean total latency and effective quality for a threshold pair (T_1, T_2). Lower latency and higher CLIP are better.

Dataset	(T_1, T_2)	Mean wait (s)	Mean service (s)	Mean total (s)	Eff. CLIP
cifar10	(1,3)	1.22	1.20	2.42	$\uparrow$
cifar10	(2,5)	2.29	1.44	3.73	$\approx$
cifar10	(4,8)	4.84	1.58	6.42	$\downarrow$
flowers	(1,3)	2.44	1.39	3.82	$\uparrow$
flowers	(2,5)	3.86	1.53	5.39	$\approx$
flowers	(4,8)	8.12	1.78	9.90	$\downarrow$
faces	(1,3)	83.14	4.92	88.06	$\approx$
faces	(2,5)	103.50	5.49	108.99	$\approx$
faces	(4,8)	136.80	6.48	143.28	$\approx$

Table 6. Ablation 2 (Step Budgets). Summary at medium load ($\lambda = 0.6$). Aggressive budgets reduce latency and tail latency, while quality-biased budgets increase queueing delay.

Dataset	Step budget ($k_{\text{full}}, k_{\text{med}}, k_{\text{short}}$)	Mean wait (s)	Mean service (s)	Mean total (s)	P95 wait (s)	P95 total (s)
cifar10	Aggressive (80,40,10)	1.40	1.29	2.69	3.50	4.01
cifar10	Conservative (100,50,20)	2.29	1.44	3.73	4.83	5.75
cifar10	Quality-biased (150,80,40)	4.72	1.58	6.29	7.82	8.69
flowers	Aggressive (80,40,10)	2.75	1.52	4.27	5.48	6.20
flowers	Conservative (100,50,20)	3.86	1.53	5.39	6.80	7.62
flowers	Quality-biased (150,80,40)	11.01	1.71	12.73	15.90	17.14
faces	Aggressive (80,40,10)	60.50	3.33	63.82	72.23	74.36
faces	Conservative (100,50,20)	103.50	5.49	108.99	148.23	152.42
faces	Quality-biased (150,80,40)	198.98	10.08	209.06	315.20	323.50

Table 7. Ablation 3 (Sampler Switching). Summary at medium load ($\lambda = 0.6$). Differences between DDPM-only, DDIM-only, and mixed policies are small relative to queueing effects.

Dataset	Policy	Mean wait (s)	Mean service (s)	Mean total (s)	P95 wait (s)	P95 total (s)
cifar10	DDPM-only	2.33	1.45	3.78	4.95	5.91
cifar10	DDIM-only	1.93	1.34	3.27	4.31	5.23
cifar10	Mixed	2.29	1.44	3.73	4.83	5.75
flowers	DDPM-only	3.88	1.53	5.41	6.81	7.64
flowers	DDIM-only	3.86	1.53	5.39	6.80	7.62
flowers	Mixed	3.86	1.53	5.39	6.80	7.62
faces	DDPM-only	99.44	5.27	104.71	140.22	144.18
faces	DDIM-only	104.16	5.51	109.67	148.92	153.11
faces	Mixed	103.50	5.49	108.99	148.23	152.42

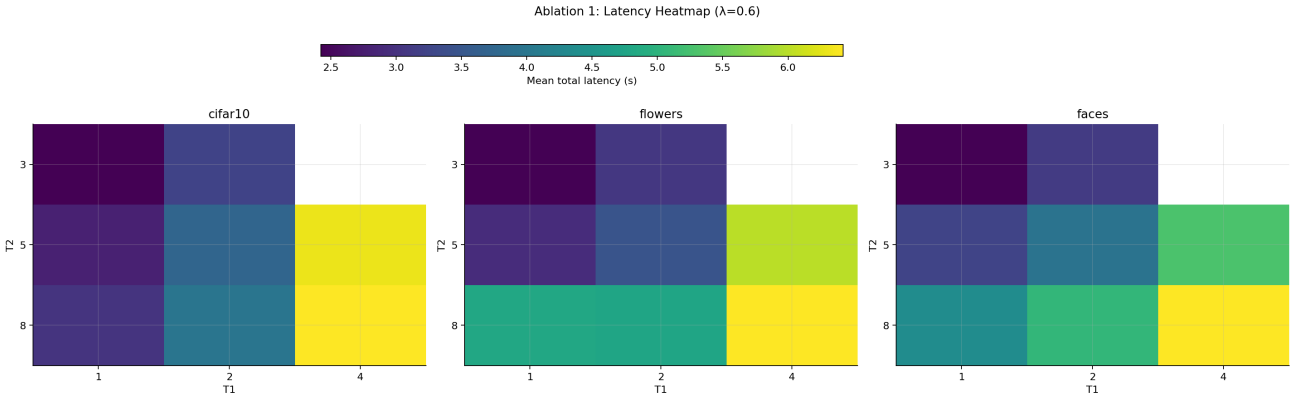


Figure 6. Ablation 1 (thresholds): Mean total latency at $\lambda = 0.6$ across threshold pairs (T_1, T_2) for each dataset. Conservative thresholds (larger T_1, T_2) delay switching and lead to higher latency due to queue buildup, especially for heavier workloads.

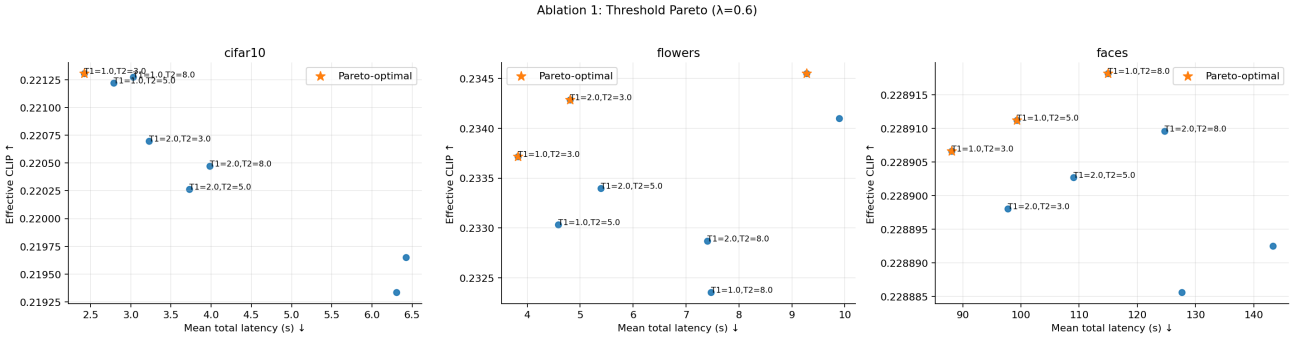


Figure 7. Ablation 1 (thresholds): Latency–quality Pareto trade-off at $\lambda = 0.6$. Each point is one (T_1, T_2) pair. Aggressive thresholds reduce latency but may slightly reduce effective CLIP; conservative thresholds preserve quality but can incur large queueing delay.

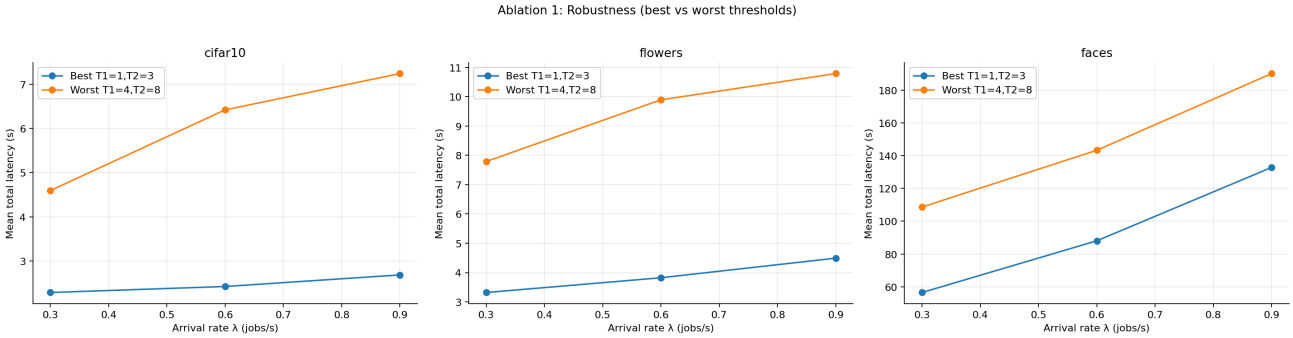


Figure 8. Ablation 1 (thresholds): Robustness across arrival rates λ comparing best vs. worst threshold choices. Poor threshold placement causes superlinear latency growth under load, while well-chosen thresholds keep latency much more stable.

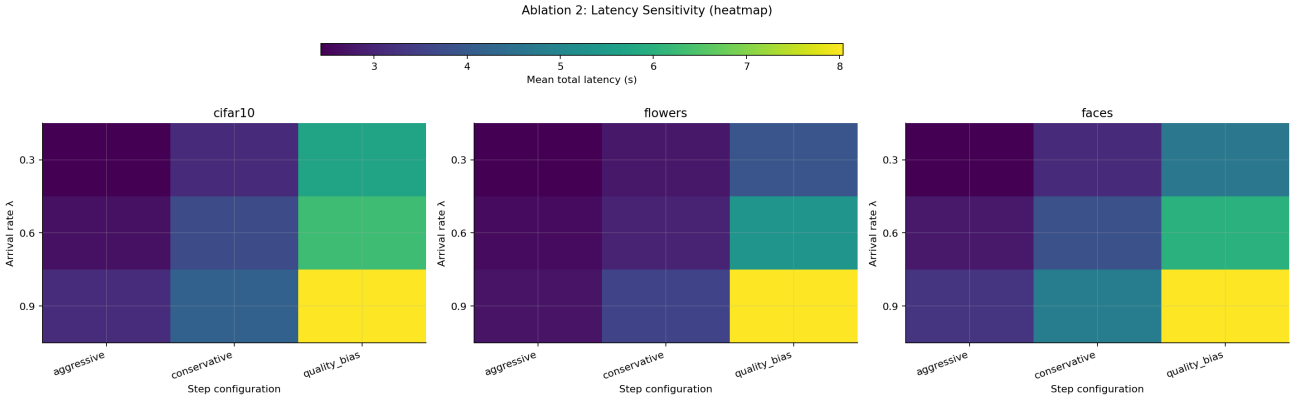


Figure 9. Ablation 2 (step budgets): Mean total latency across arrival rates λ for three step-budget variants. Larger step budgets increase service time and amplify queueing delay under load; aggressive budgets remain most stable at high λ.

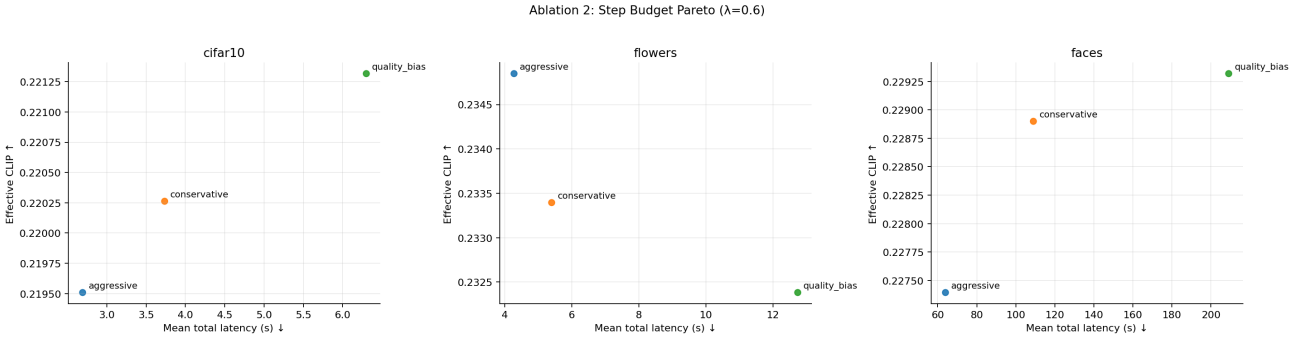


Figure 10. Ablation 2 (step budgets): Latency–quality trade-off at $\lambda = 0.6$ across step-budget variants. Aggressive budgets favor responsiveness; quality-biased budgets favor quality but incur much higher latency.

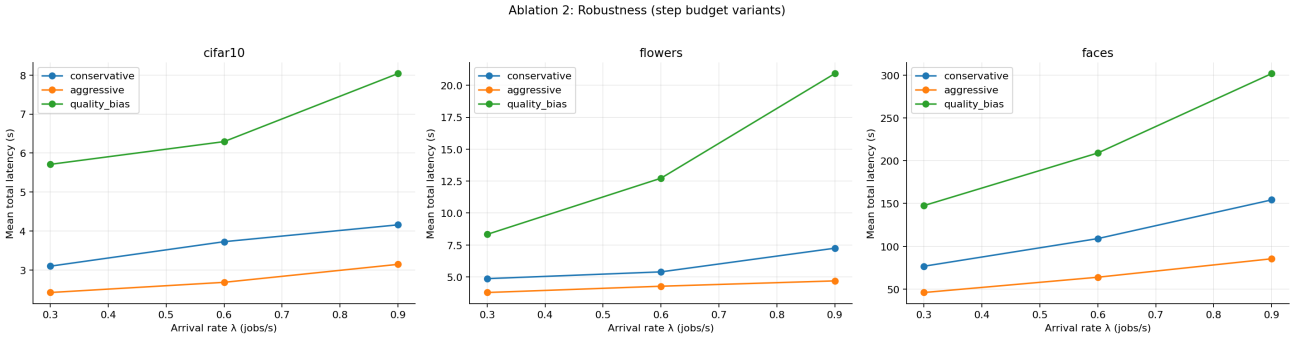


Figure 11. Ablation 2 (step budgets): Robustness across λ. Aggressive step budgets scale best (lower service time $\rightarrow$ less waiting amplification), while quality-biased budgets become increasingly costly under congestion.

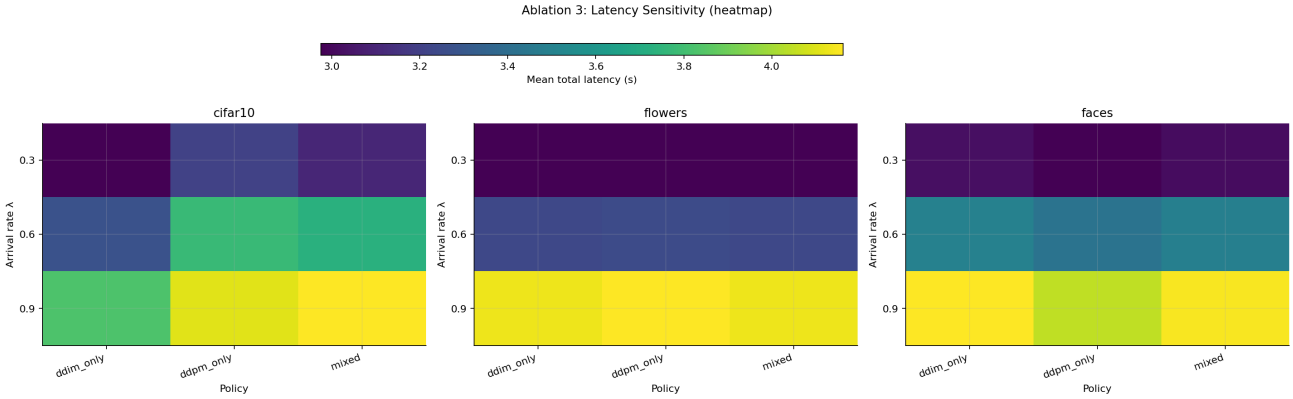


Figure 12. Ablation 3 (sampler switching): Mean total latency across arrival rates λ for ddpm_only, ddim_only, and mixed switching. Differences are generally small relative to the effect of load and dataset scale.

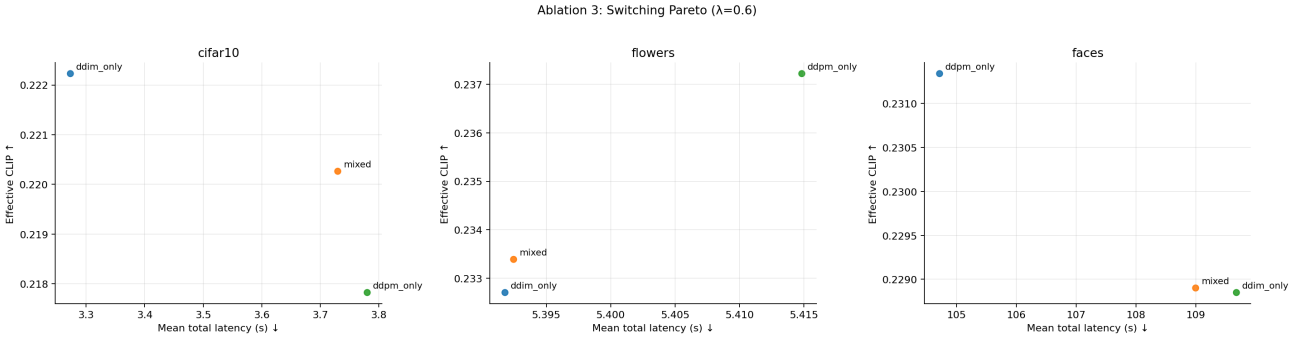


Figure 13. Ablation 3 (sampler switching): Latency–quality trade-off at $\lambda = 0.6$ for ddpm_only, ddim_only, and mixed. Strategies are often tightly clustered, indicating switching is not a dominant lever compared to step reduction.

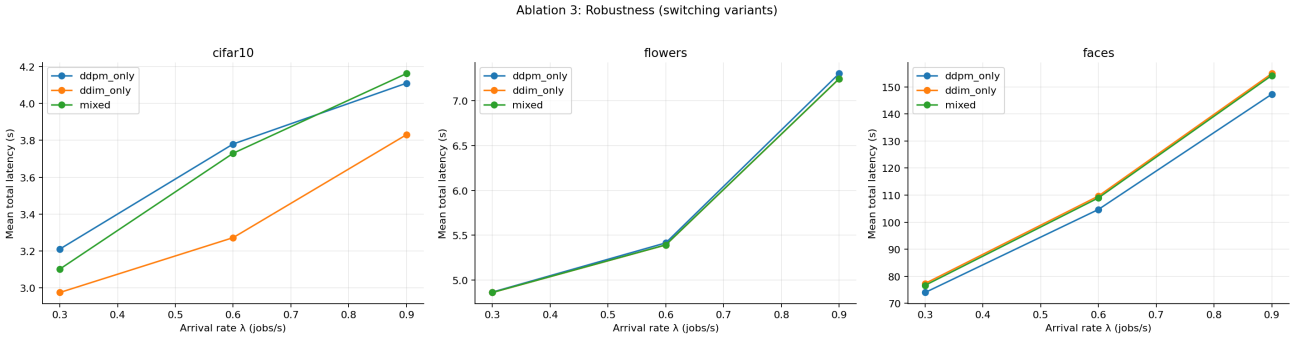


Figure 14. Ablation 3 (sampler switching): Robustness across λ for switching strategies. All three scale similarly with load; sampler switching alone does not substantially change stability compared to step adaptation.