Scheduler Design for Social Welfare on Solana

TLDR

- Scheduler design determines execution timing and ordering, not value creation. Welfare effects arise only once execution outcomes are evaluated through agent-specific losses.
- Concepts such as “better ordering” or “higher-value blocks” are ill defined without an explicit welfare model.
- Execution behaviour can be characterized empirically using an ordering observable (the scheduler footrule distance), allowing distinct scheduler regimes to be identified on chain.
- Under independent transaction arrivals, immediate execution with minimal buffering (Streaming) minimizes aggregate delay and Pareto-dominates batching-based schedulers across most preference profiles.
- Under systematic, time-coupled competition, buffering transactions over short windows reduces race volatility and can strictly improve welfare by stabilizing execution outcomes.
- Batching improves welfare only for specific preference structures, namely when losses are dominated by fee competition or by race outcomes under noisy arrivals.
- Batching is structurally misaligned with deadline-driven workloads: when welfare is dominated by absolute execution time, added latency directly increases loss and cannot be offset by smoother ordering or lower fees.
- No scheduler is universally optimal. Scheduler choice must be aligned with the arrival regime and with the dominant source of welfare loss faced by the agent population.
- Scheduler optimality lies on a continuum: batching converges to streaming as the batch window shrinks, suggesting that adaptive or data-driven batch sizing may better serve heterogeneous ecosystems.
- Execution scheduling is therefore an economic design problem, not a purely technical one.

1. Introduction

Transaction scheduling is a central design choice for high-throughput blockchains. Under congestion, schedulers determine which transactions execute first, which are delayed, and how conflicts are resolved. Despite this, discussions of scheduler quality are often framed in terms of loosely defined objectives such as higher value blocks, better ordering, or stronger priority enforcement. While intuitively appealing, these notions are ill defined in the absence of an explicit mapping to economic outcomes: properties of execution ordering are not intrinsically meaningful and can only be evaluated through the losses they induce on the agents submitting transactions.

This ambiguity is not merely theoretical. On Solana, the network does not operate under a single, uniform execution policy, but under a plurality of effective schedulers. Differences in client implementations, buffering strategies, conflict handling, and priority enforcement result in markedly different execution outcomes for users and applications, even when submitting economically similar transactions. In such an environment, claims about a “better” or “worse” scheduler cannot be grounded in execution mechanics alone. Absent a welfare-based interpretation, they reduce to statements about internal behaviour rather than system-level outcomes.

This paper therefore starts from the observation that the notion of a “best block” is not well defined unless it is grounded in agent welfare.

1.1 Why “Best Block” is ill Defined

Several commonly invoked criteria for scheduler quality illustrate this ambiguity:

- **Higher value blocks:** A block that extracts more fees or priority payments does not necessarily improve outcomes for users or applications. Increased fee revenue may reflect intensified competition or rent extraction rather than welfare gains.
- Better ordering: Ordering transactions “more cleanly” or “more consistently” has no intrinsic meaning unless one specifies which agents benefit from earlier execution and which incur losses from delay.
- **Stronger priority enforcement:** Enforcing priority more strictly improves the correlation between bids and execution rank, but this merely reallocates execution advantage toward agents with higher willingness or ability to pay. Whether this improves or worsens social outcomes depends on the underlying preference structure.
- In all cases, these criteria describe properties of execution, not outcomes. They become meaningful only once execution timing and ordering are evaluated through agent-specific loss functions.

  Consequently, scheduler design is unavoidably a welfare problem: schedulers do not create value, but allocate delay, ordering risk, and fee efficiency under contention.

1.2 Scheduler Design as a Welfare Problem

Transactions on a blockchain are submitted by heterogeneous agents with different objectives, latency profiles, and tolerance to execution delay. For some agents, absolute timeliness dominates; for others, relative ordering matters; for others still, fee expenditure is the primary concern. Identical execution outcomes can therefore improve welfare for some participants while worsening it for others.

A scheduler determines execution outcomes but does not assign value to them. Losses arise only once outcomes are evaluated through agent preferences. As a result, no scheduler can be evaluated in isolation from the losses it induces.

This observation has a methodological implication: any welfare analysis of scheduler design requires a way to translate execution behaviour into losses. To do so, execution ordering must first be represented in a quantifiable and comparable way.

Welfare modelling requires knowledge of when transactions execute and in what relative order. However, execution behaviour on a live network is shaped by multiple interacting factors, including stochastic arrival times, network propagation, conflict constraints, and scheduler logic. Aggregate block-level statistics—such as compute usage, block fullness, or fee revenue—do not isolate the contribution of the scheduler itself.

To connect scheduler design to welfare outcomes, we therefore require a block-level observable that:
1. Summarizes realized execution ordering
2. Is comparable across blocks of different sizes
3. Is interpretable independently of agent preferences
4. Can be measured empirically on chain.

In Sec. 2, we introduce such an observable and show how it can be used to map observed execution behaviour to a small set of scheduler primitives. In Sec. 3, we define welfare and introduce the evaluation criteria used throughout the paper. Subsequent sections study how welfare varies across scheduling regimes and agent configurations.

---

2. Execution Ordering as a Measurable Object

The welfare analysis developed in this paper requires a representation of execution behaviour that is both empirically observable and abstract enough to be evaluated independently of agent preferences. In this section, we introduce such a representation by focusing on realized execution

ordering within blocks. We construct a block-level observable that summarizes how transactions are ordered in practice and show how this observable can be used to map on-chain behaviour to a small set of scheduler primitives.

Throughout this section, we deliberately avoid normative interpretation. The goal is not to assess whether a given execution pattern is desirable, but to characterize how schedulers transform order flow into execution outcomes.

2.1 Execution Ordering and Scheduler Logic

The set of transactions executed in a block is the outcome of two intertwined mechanisms: order flow and scheduler logic. Order flow determines which transactions are visible to the leader and when, while scheduler logic governs how visible transactions are selected, ordered, and executed subject to conflict constraints.

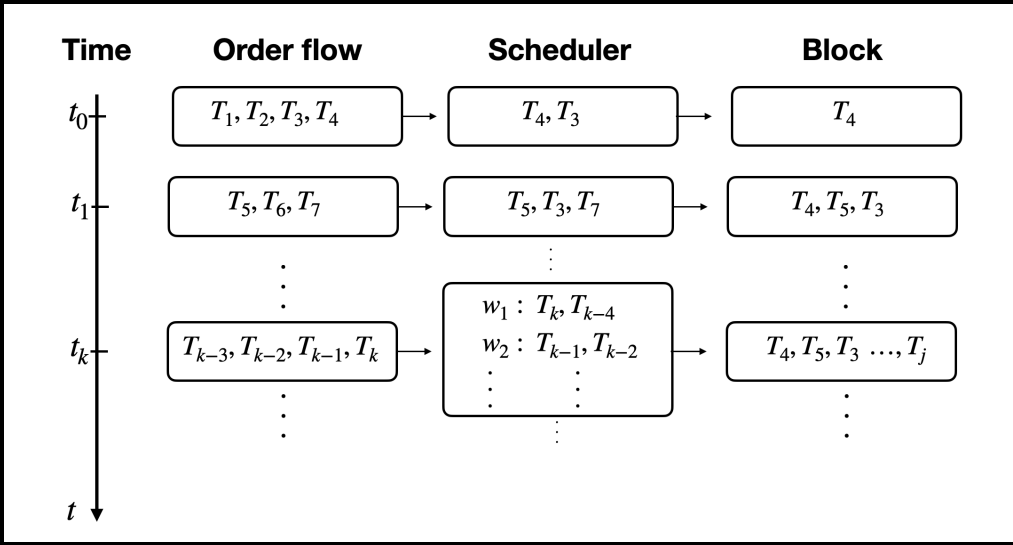

Fig.1: Txs execution pipeline

As transactions arrive at the validator, the scheduler incrementally decides which transactions are runnable and in what order they should be executed. Transactions that touch disjoint state can, in principle, execute concurrently, while fully conflicting transactions must be serialized. The execution order recorded on chain corresponds to a linearization of this partially ordered execution, shaped jointly by arrival times, conflicts, and scheduler rules.

In practice, order flow is influenced by multiple network-level mechanisms, including propagation latency, stake-weighted quality of service, client-side routing policies, and local buffering behaviour. As a result, different validators may observe systematically different transaction sets and arrival patterns even under otherwise comparable network conditions. To isolate scheduler behaviour from these effects, it is useful to reason in an idealized limit.

Consider a hypothetical regime in which all transactions visible to a validator are available simultaneously. In this limit, fully conflicting transactions must be executed in strict priority order, while non-conflicting transactions may execute in an arbitrary order due to parallelism. Deviations from this idealized priority ordering therefore arise from the interaction between arrival-time stochasticity, conflict structure, and scheduler logic.

This observation motivates a comparison between realized execution order and a reference ordering derived from transaction priority.

2.2 Scheduler Footrule Distance

To quantify deviations from priority ordering, we introduce a block-level observable based on the Spearman footrule distance between two permutations of the same transaction set.

For each block, we consider:

- rn_i: the realized execution order of transactions
- prio_i: the hypothetical order obtained by sorting the same transactions by priority per compute unit.

The footrule distance

$$D \sim \sum_{i=1}^{N} |\,\mathrm{prio}_i - \mathrm{rn}_i\,|\,,$$

measures the total displacement in rank between these two orderings. To enable meaningful comparison across blocks of different sizes, we normalize the distance by its maximum possible value, yielding a block-size–invariant quantity. We refer to this normalized metric as the scheduler footrule distance D, see Appendix A1.

By construction, the scheduler footrule distance summarizes how execution ordering reflects — or departs from — priority information once arrival-time randomness and conflict constraints are taken into account. Values close to zero indicate strong alignment with priority ordering, while larger values indicate increasing deviation.

Importantly, the scheduler footrule distance is not a welfare metric. It does not encode preferences, utilities, or losses. Rather, it is a behavioural observable that characterizes how a scheduler translates order flow into execution order, independently of how that order is valued by agents.

2.3 Interpretation and Null Models

To anchor interpretation, we consider a null model in which the realized execution order is independent of transaction priority. This corresponds to treating block ordering as a uniformly random permutation of the transaction set. While this null model is not intended to describe Solana's scheduler, it provides a useful reference for a regime in which arrival-time randomness dominates priority-based ordering. Under this random-permutation null, the expected scheduler footrule distance approaches $D \simeq 2/3$.

Accordingly, values of D significantly below 2/3 indicate that priority information is meaningfully reflected in execution, despite stochastic arrival effects. Conversely, values close to 2/3 are consistent with a regime dominated by arrival-time randomnsess. Value above 2/3 indicate that high-priority transactions are, on average, displaced farther from their priority ranks than would be expected under random ordering.

2.4 Empirical on-chain Distributions

Empirically, the distribution of the scheduler footrule distance D reveals that the network does not operate in a single homogeneous regime. At a coarse level, two broad behavioural classes are observed, see the bimodal distribution in Fig. 2.

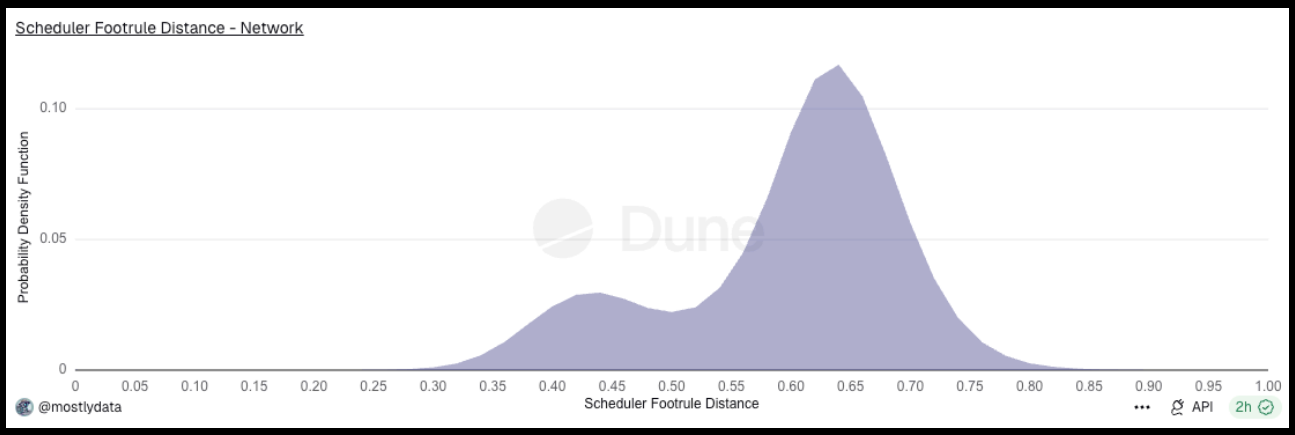

Fig.2: Distribution of the scheduler footrule distance for the entire Solana network. Source https://dune.com/queries/6364939/10123516

However, a more granular analysis shows that these can be further decomposed into four distinct scheduler behaviours, corresponding to variations within the same underlying design philosophy, cfr. Fig. 3.

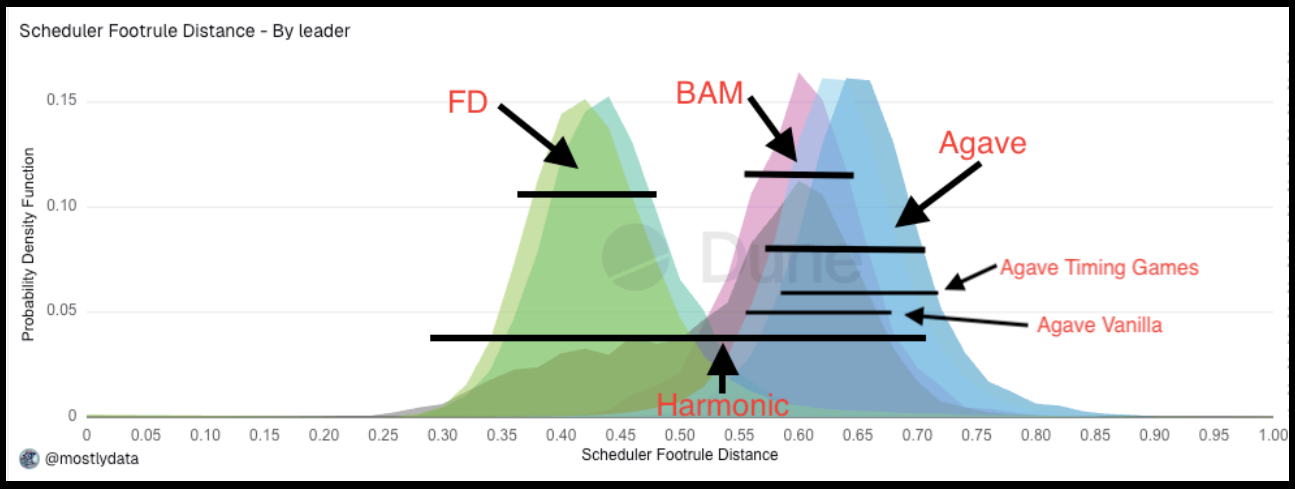

Fig.3: Distribution of the scheduler footrule distance across different validator implementations in the Solana network. Green-toned distributions correspond to Firedancer and Frankendancer, red to BAM, blu-toned distributions to Agave (with slot lagging — blue — and without — light blue), and grey to Harmonic .Source https://dune.com/queries/6365102/10123580.

These behaviours can be grouped as follows:

- **Mostly random (peak near 0.66) — arrival-dominated regime:** In this regime, the realized ordering is close to the random-permutation baseline $\mathbb{E}[D] \sim 2/3$. The scheduler appears to commit transactions quickly, with limited opportunity to reorder by priority. This is consistent with a small effective lookahead or batch size, where stochastic arrival effects dominate over explicit priority sorting. Validators running Agave predominantly fall into this category.
- **Slightly different from random (peak ~ 0.6):** Here, some priority information is reflected in the execution order, but only locally. This behaviour is consistent with limited reordering within small batches or shallow queues, where priority sorting is applied only over a restricted candidate set. Validators associated with BAM exhibit this behaviour.
- **Long tailed (peak ~ 0.6, with a pronounced left-tail):** These distributions indicate a mixture of two regimes within the same validator. In particular, they are characteristic of state-dependent schedulers: under certain conditions (e.g., low load or sufficient buffering), the scheduler effectively batches transactions and enforces priority ordering (yielding low D); under other conditions (e.g., high arrival rate, queue saturation, or compute pressure), it behaves in a more streaming-like fashion, producing higher$D$. Such behaviour can naturally arise from internal

thresholds, adaptive batching logic, or bursty order-flow. Harmonic validators fall into this category.
- **More aligned with priority (peak near 0.40-0.45):** In this regime, priority information is clearly and consistently reflected in the realized execution order. This suggests a larger effective holding window, stronger priority enforcement, or more aggressive reordering before commitment. Arrival-time randomness is still present, but it is substantially mitigated by scheduler logic. This behaviour is observed for Firedancer / Frankendancer validators operating with slot lagging enabled.

Taken together, these observations indicate that while the network exhibits two main qualitative approaches — arrival-dominated versus priority-enforcing scheduling — each admits multiple concrete implementations, resulting in a richer landscape of observable execution behaviours.

Finally, in Appendix B we show that the observed variation in D reflects scheduler-level effects, rather than block-level resource constraints such as compute usage or block fullness.

2.5 Mapping Execution Ordering to Scheduler Primitives

The empirical distributions presented in Section 2.4 establish that Solana does not operate under a single execution regime, but they do not, by themselves, identify the underlying scheduling mechanisms responsible for the observed patterns. To interpret these distributions, we complement the on-chain analysis with controlled simulations designed to reproduce and explain the scheduler regimes identified via the footrule distance D.

The objective of these simulations is not to model Solana's execution engine in full detail. Rather, the goal is to isolate a small set of scheduler primitives — minimal execution rules that differ only in buffering and reordering logic — and to determine whether these primitives induce systematically distinct distributions of D under identical order-flow conditions. This allows us to interpret empirical execution-ordering patterns in terms of underlying scheduling behaviour

Simulation Framework

Transaction arrivals are modeled as a homogeneous Poisson process with rate λ, reflecting decentralized and asynchronous submission by independent agents. For a block horizon of duration T, the number of transactions N is drawn from Poisson(λT), and conditional on N, arrival times are i.i.d. and uniformly distributed on $[0, T)$.

Each transaction is independently assigned a priority score $p \sim$ Unif$(0,1)$. This choice deliberately removes any correlation between arrival time and priority, ensuring that deviations from priority ordering arise solely from scheduler behaviour and arrival-time stochasticity, rather than from exogenous structure in the order flow.

For each simulated block, the priority-sorted order — obtained by sorting all transactions by decreasing priority — serves as the invariant reference against which realized execution order is compared via the scheduler footrule distance D.

Schedulers are assumed to be non-anticipative: at any point in time, they may only reorder transactions that have already arrived. This constraint captures the online nature of block production and is essential for generating deviations from the priority-sorted reference.

Scheduler primitives

We consider four scheduling policies, each corresponding to a distinct operational regime and mapping directly onto one of the empirical D-distributions observed on chain:

1. **Streaming (arrival-ordered execution with local tie-breaking via priority):** Transactions are executed as soon as they become runnable, with minimal buffering. When multiple runnable transactions are available simultaneously — due to discrete ingestion, queue flushes, or parallel arrival — priority is used only as a local tie-breaker. This policy represents an arrival-dominated regime with minimal lookahead and is expected to produce execution orders close to a random permutation relative to priority.
2. **Fixed time-batch scheduling:** Time is partitioned into fixed windows of width w. Transactions arriving within a window are buffered and executed at the end of the window in descending priority order, with batches processed sequentially. This policy enforces priority locally within batches but not across them. The batch width w controls the effective lookahead of the scheduler.
3. **Wait-up-to-W:** Upon the arrival of the first transaction, the scheduler buffers incoming transactions for up to a maximum duration W. When the waiting time elapses, a size threshold is reached, or the block horizon ends, buffered transactions are executed in descending priority order, after which execution proceeds in streaming mode. This policy represents a bounded-latency scheduler that dynamically trades off buffering against delay.
4. **Timing Game (strategic horizon extension):** In this regime, the scheduler deliberately extends the effective transaction collection horizon beyond the nominal block cutoff T, executing transactions arriving in a late window $[T, T + \Delta)$. The uniform-priority assumption is relaxed: transactions arriving in the late window are assigned priorities drawn from a distribution with higher expected value than those arriving earlier, modelling non-stationary order flow driven by market dynamics, congestion, or strategic submission behaviour. This modification is introduced solely to capture the defining feature of timing games and is not used elsewhere in the simulation framework.

Figure 4 shows the resulting distributions of D for these primitives in the absence of transaction conflicts.

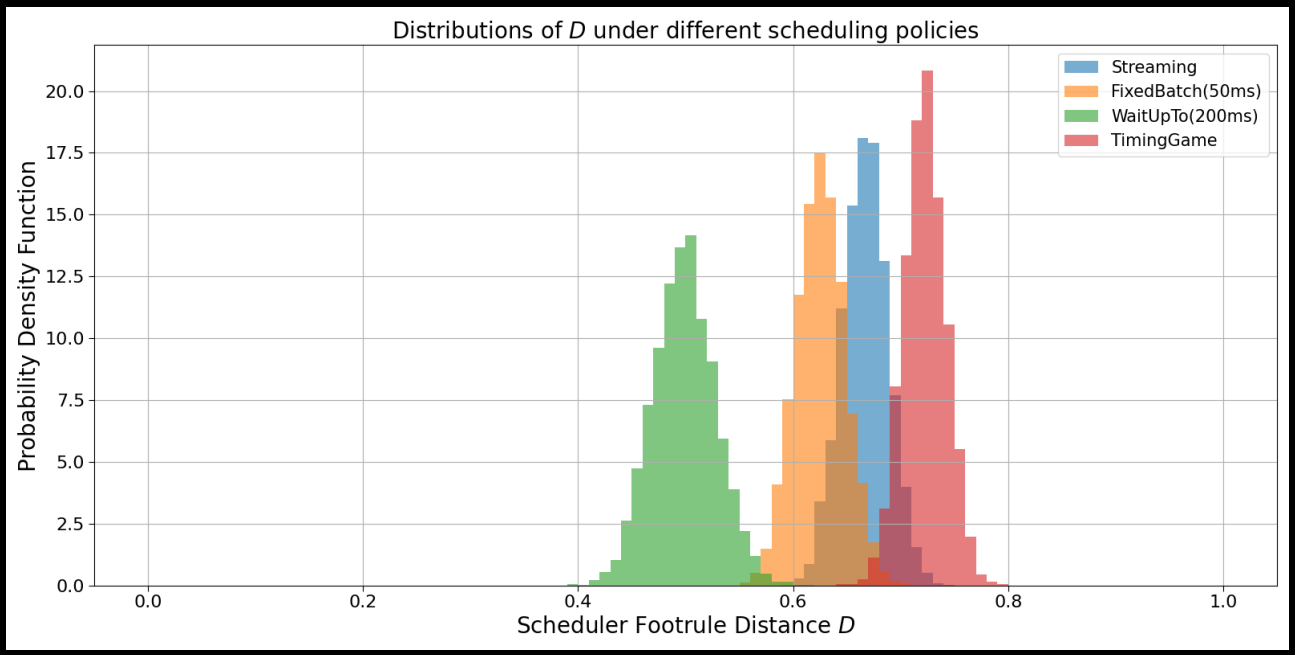

Fig. 4: Simulation of different scheduler behaviour and relative scheduler footrule distance distribution. Here conflicting transactions are not taken into account.

Interpretation and role of conflicts

Despite the simplified setting, the simulations reproduce the qualitative ordering signatures observed on chain. Each scheduler primitive isolates a fundamental mechanism — immediate commitment, periodic batching, bounded waiting, or strategic delay — that can be directly identified in empirical D-distributions.

In this sense, the simulation framework should be interpreted not as a literal model of Solana's scheduler, but as a basis decomposition: a minimal set of behavioural components whose superposition reproduces the observed landscape of execution behaviour.

The baseline simulations abstract away conflicts between transactions. When conflicts are explicitly incorporated, two effects emerge. First, even under identical scheduling rules, the distribution of D becomes broader and more heavy-tailed, reflecting the additional constraints imposed by conflict resolution. Second, this increased dispersion closely matches empirical on-chain observations, where high-contention accounts and bursty order flow routinely induce large deviations from idealized priority ordering (Fig. 5).

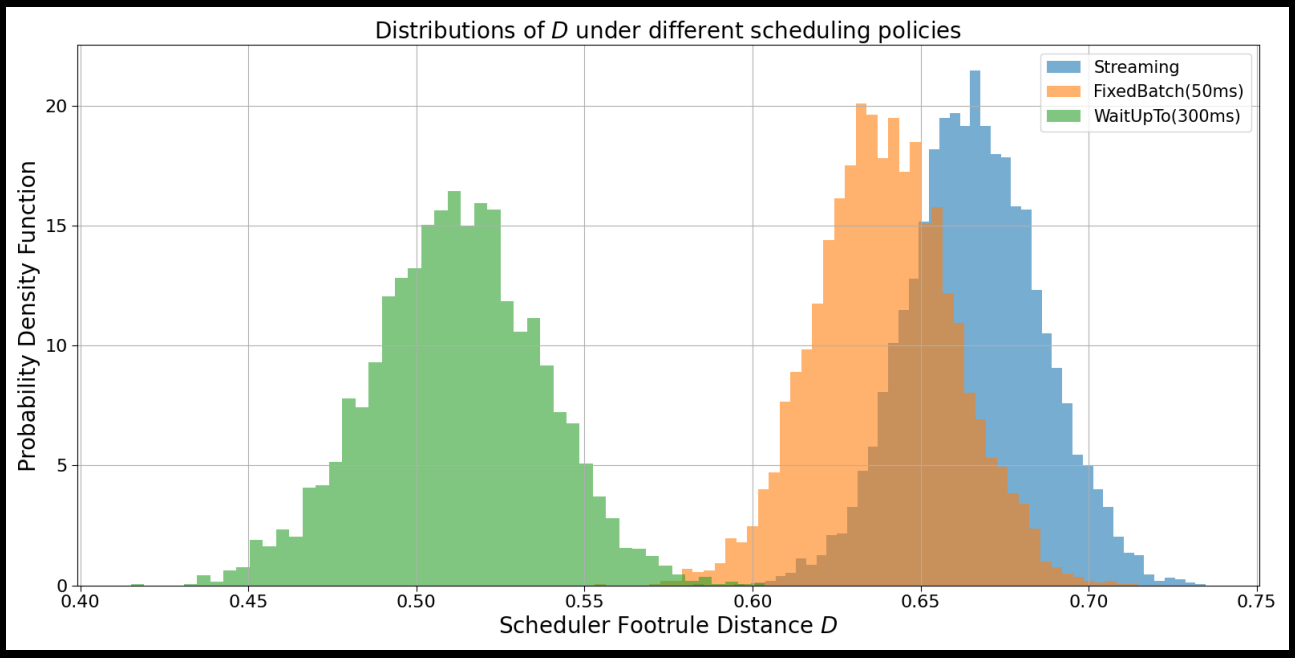

Fig. 5: Simulation of different scheduler behaviour and relative scheduler footrule distance distribution. Here we take into account also conflicting transactions.

Conflicts are introduced through a reduced-form execution model that approximates parallel execution across multiple workers. Time is discretized into PoH ticks, and only a limited number of transactions touching a given conflict account may contribute to the block's linearized execution order per tick. This representation captures the essential feature that conflicting transactions must be serialized, while non-conflicting transactions may execute concurrently.

These results indicate that a substantial fraction of the observed variability in D arises from the interaction between scheduler logic and conflict resolution, rather than from arrival-time stochasticity alone. Incorporating conflicts therefore brings the observable closer to the operational reality of Solana blocks, while preserving its role as a compact, block-level summary of execution ordering.

Finally, we exclude timing-game regimes from the welfare analysis that follows. Timing games represent explicitly extractive behaviour achieved by extending slot duration on top of a given scheduling rule, rather than a distinct execution primitive. Since our objective is to study welfare implications of scheduler design under a fixed block horizon, these regimes fall outside the scope of the analysis in subsequent sections.

---

3. Social Welfare under Contention

The analysis so far has characterized how schedulers transform order flow into execution ordering. We now address the complementary question: **how do these execution outcomes affect the welfare of network participants?**

Scheduler design does not directly generate value. Instead, it allocates execution timing and ordering under contention. Welfare effects arise only once these execution outcomes are evaluated through agent-specific preferences. As a result, identical execution orders may improve welfare for some agents while worsening it for others.

To study this interaction in a tractable and interpretable way, we introduce a minimal, reduced-form welfare model that maps execution outcomes to agent-level losses. The loss functions are not derived from an explicit optimization or equilibrium model; instead, their parameters should be interpreted as sensitivity weights, and the analysis as a comparative-static exploration of how different scheduling rules redistribute welfare across preference regimes, without modelling informational rents or payoff changes arising from faster execution.

This abstraction allows schedulers to be compared on welfare grounds while isolating the effects of execution mechanisms from strategic bidding behaviour or informational advantages.

3.2 A Minimal Two-Agent Welfare Model

We model the network as populated by two representative agents submitting fully conflicting transactions, so that execution is necessarily serialized. This abstraction captures the smallest unit of execution externality and applies equally to user–application interactions, bot–bot races, or competition between applications. The labels “user” and “application” should therefore be understood as roles with potentially different preferences, not as distinct economic categories.
Each agent $i \in \{U, A\}$ is assigned a loss function

$$L_i = L_i^{\text{prio}} + L_i^{\text{delay}} + L_i^{\text{order}},$$

which maps execution outcomes into disutility, see Appendix D1 for a detailed description. The scheduler determines execution timing and ordering; losses arise only after these outcomes are evaluated through preferences.

- Priority cost captures disutility from fee expenditure incurred to bid for earlier execution.
- Delay cost captures sensitivity to execution latency, including deadlines and tail-risk penalties.
- Ordering cost captures sensitivity to relative execution order when agents compete for the same state update.

This formulation is deliberately flexible: by varying preference parameters, it spans environments dominated by fee sensitivity, absolute timeliness, or race outcomes.

3.2 Latency Advantage as a Redistribution

When transactions are fully conflicting, execution is rivalrous: executing one transaction necessarily delays the execution of others touching the same state. In such settings, differences in arrival time can affect execution order even when transaction priorities are comparable.

Under arrival-dominated schedulers, transactions observed earlier by the leader are more likely to be executed first whenever contention is present. A systematic latency advantage for one participant therefore shifts early execution opportunities in its favour, increasing the waiting time of competing transactions. This effect arises mechanically from queue composition and scheduler logic and does not require adversarial behaviour or strategic intent.

A key modelling assumption of this paper is that latency does not confer informational advantage. Being faster affects when a transaction executes, but not what economic state it observes or what payoff it realizes conditional on execution. Formally, we assume that asset values follow a martingale with respect to public information, so that execution timing alone does not generate expected profit or loss beyond its impact on ordering and delay.

Under this assumption, latency advantages do not create surplus at the system level. Instead, they reallocate execution priority among competing agents: improvements in one participant's execution timing necessarily come at the expense of others competing for the same execution resource. In the loss model introduced in this section, this assumption is reflected by the absence of any term directly linking latency to asset value or informational rents.

Accordingly, we treat latency asymmetry as a pure redistribution channel. Scheduler design matters because it determines how strongly arrival-time differences are translated into execution advantages, thereby shaping welfare outcomes through redistribution rather than value creation. This assumption is consistent with recent theoretical results showing that, in the absence of informational advantages, speed alone reallocates surplus without increasing aggregate welfare. While this abstraction excludes settings such as price discovery or informational arbitrage — where execution speed may affect beliefs or payoffs — it allows us to isolate the welfare effects of scheduling rules under contention without conflating them with information-driven mechanisms.

3.3 Welfare Comparison across Scheduling Regimes

We compare scheduling regimes by evaluating the loss vectors (L_U, L_A) induced by each scheduler under heterogeneous agent preferences. To avoid imposing an arbitrary social welfare function, we adopt a Pareto-dominance criterion. For a fixed preference profile θ, a scheduler π is said to Pareto-dominate another scheduler π' if it delivers weakly lower expected loss for both agents and strictly lower loss for at least one of them, up to a small tolerance τ. Concretely, π Pareto-dominates π' if

$$(i) \qquad \mathbb{E}[L_U(\pi;\theta)] \le \mathbb{E}[L_U(\pi';\theta)] + \tau_U(\theta)$$
$$(ii) \qquad \mathbb{E}[L_A(\pi;\theta)] \le \mathbb{E}[L_A(\pi';\theta)] + \tau_A(\theta)\,,$$

with at least one inequality holding strictly. The tolerances $\tau_i(\theta)$ are defined as a fixed fraction of the corresponding loss scale. Differences within this band are treated as economically insignificant and classified as ties.

We evaluate Pareto dominance across a broad, non-restrictive parameter space spanning fee sensitivity, delay sensitivity, deadline penalties, tail-risk aversion, and ordering sensitivity. Preference profiles are sampled uniformly over this space; for each profile, expected losses are computed by averaging over many simulated slots to account for stochastic execution dynamics

(arrival times, latency noise, and conflict resolution). Aggregating results across sampled profiles yields a Pareto win-rate matrix, reported in Figures 6-8.

Across the full parameter space and in the absence of systematic latency asymmetries, streaming execution emerges as the welfare-dominant regime, see Fig. 6. In the majority of preference profiles, streaming Pareto-dominates both fixed-window batching and bounded waiting policies. Where dominance does not obtain, outcomes are typically Pareto-incomparable: batching reduces losses for one agent class while increasing losses for the other. These cases correspond to redistribution, not welfare improvement.

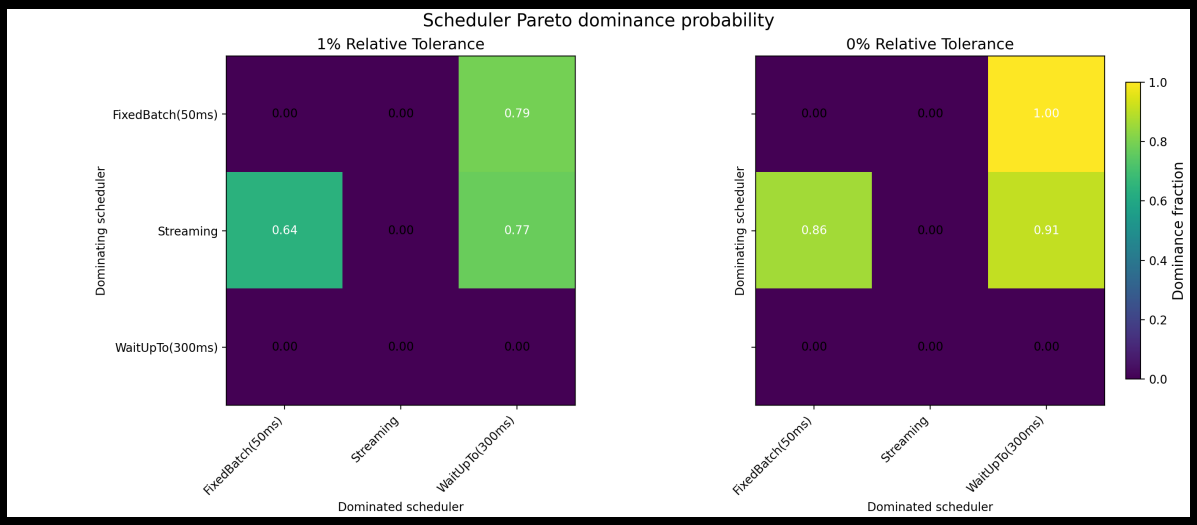

Fig. 6: Pareto-dominance probability heatmaps for different schedulers. The left panel is computed using a 1% relative tolerance in the Pareto-dominance criterion. The right panel is computed without applying any relative tolerance threshold. Each cell reports the fraction of sampled preference profiles for which the scheduler on the row Pareto-dominates the scheduler on the column. Values near one indicate frequent Pareto dominance; intermediate values correspond to Pareto-incomparable outcomes.

Introducing user-side latency noise weakens but does not overturn this conclusion. Batching reduces variance in execution timing and partially shields latency-disadvantaged users, but does so by shifting delay and ordering costs onto applications. As a result, batching rarely yields Pareto improvements and primarily alters the distribution of welfare rather than increasing it, see Fig. 7.

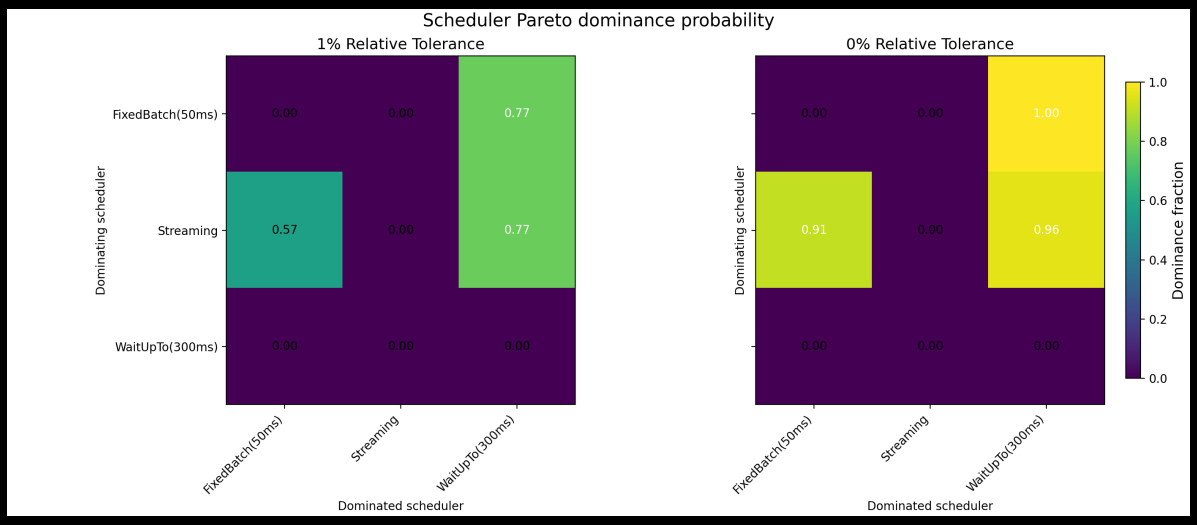

Fig. 7: Pareto-dominance probability heatmaps for different schedulers assuming user-side latency noise (400ms jitter). The left panel is computed using a 1% relative tolerance in the Pareto-dominance criterion. The right panel is computed without applying any relative tolerance threshold. Each cell reports the fraction of sampled preference profiles for which the scheduler on the row Pareto-dominates the scheduler on the column. Values near one indicate frequent Pareto dominance; intermediate values correspond to Pareto-incomparable outcomes.

Restricting preferences to race- and fee-dominated losses clarifies the underlying mechanism. In these environments, batching dampens sensitivity to micro-timing differences and reduces losses for agents exposed to arrival-time noise. However, these gains are offset by increased losses for competing agents whose execution is delayed by buffering. No scheduler uniformly dominates in this regime, and Pareto incomparability becomes prevalent, see Fig. 8.

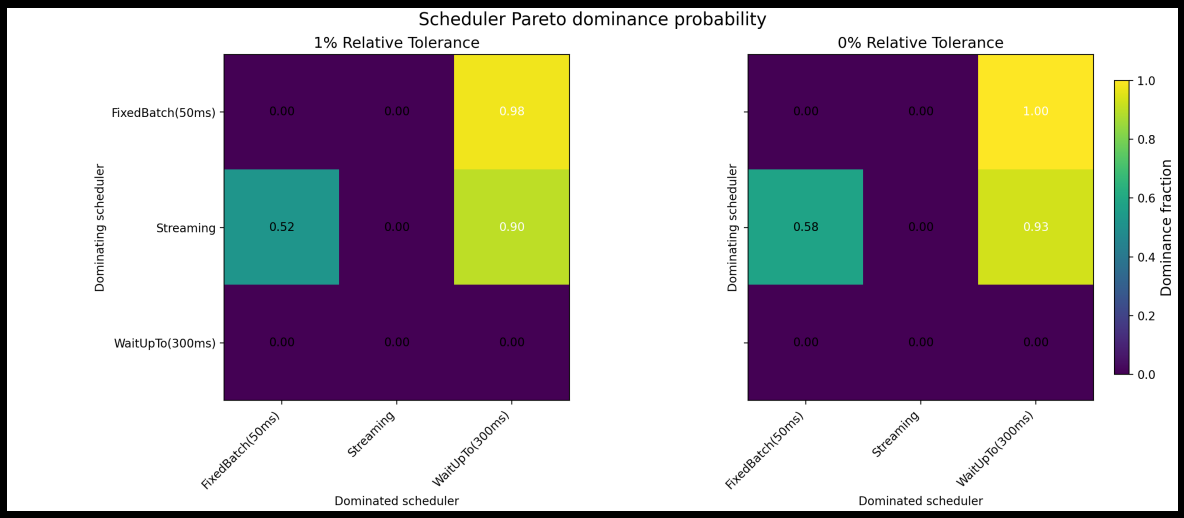

Fig. 8: Pareto-dominance probability heatmaps for different schedulers restricting the parameter space to pure race- and fee-based losses. The left panel is computed using a 1% relative tolerance in the Pareto-dominance criterion. The right panel is computed without applying any relative tolerance threshold. Each cell reports the fraction of sampled preference profiles for which the scheduler on the row Pareto-dominates the scheduler on the column. Values near one indicate frequent Pareto dominance; intermediate values correspond to Pareto-incomparable outcomes.

More generally, the welfare ranking of schedulers depends on the composition of agent preferences:

- In balanced environments, where no agent class exhibits systematically stronger time sensitivity, immediate execution with minimal buffering minimizes aggregate delay and yields the highest social welfare.
- In latency-race environments, batching acts as a buffering mechanism that redistributes execution advantages by reducing the impact of arrival-time asymmetries, but does not generate Pareto improvements.
- No scheduler is universally optimal: scheduling rules implement different trade-offs between aggregate delay minimization and redistribution of execution priority under contention.

Importantly, these results are obtained without introducing informational asymmetries or extractive behaviour. The observed welfare trade-offs arise from structural properties of execution under contention, rather than from fine-tuned assumptions or strategic exploitation.

3.4 Systematic Competition and Time-Coupled Arrivals

The welfare analysis in Section 3.3 assumes that transactions arrive according to independent stochastic processes, so competitive interactions arise only incidentally through congestion. In many real-world settings, however, competing agents respond to the same external signal — such as a price movement, liquidation opportunity, or on-chain event — leading to time-coupled transaction submission.

To capture this regime, we extend the arrival model to allow transactions from competing agents

to be generated in response to a common trigger, subject to reaction delays and bounded response windows. This induces systematic and repeated competition between agents. All other components of the model — scheduler definitions, execution mechanics, and loss functions — are held fixed, allowing us to isolate the effect of correlated arrivals on welfare outcomes.

Under time-coupled arrivals, the role of the scheduler changes qualitatively. Execution policies no longer merely resolve incidental congestion, but repeatedly shape the outcomes of execution races under timing uncertainty. As a result, the welfare ranking of schedulers differs fundamentally from the independent-arrival regime.

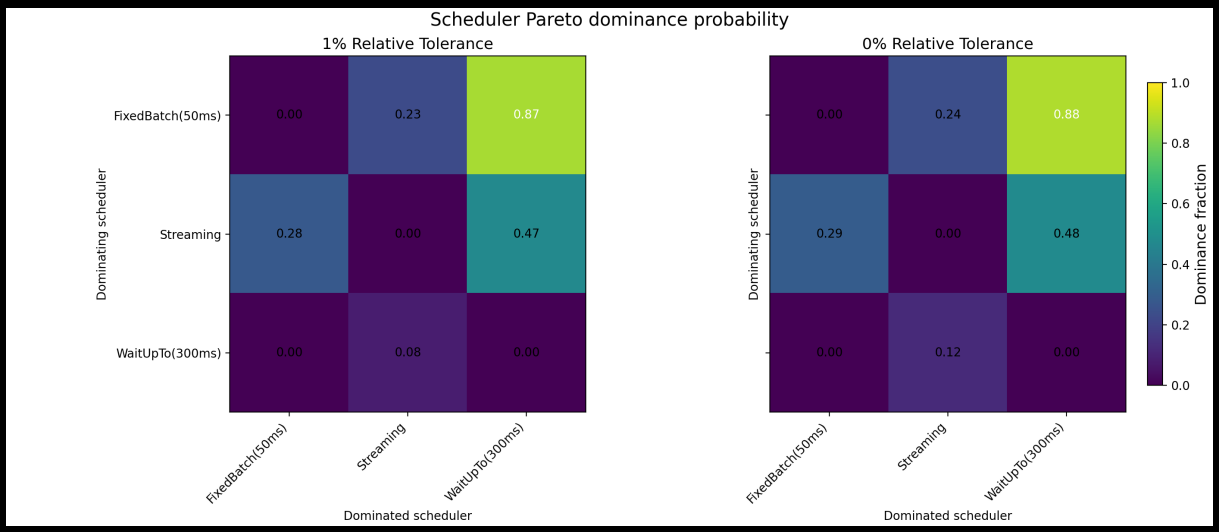

Fig. 9: Pareto-dominance probability heatmaps for different schedulers in the case of systematic competition between agents. The left panel is obtained using the whole parameter space, representing heterogeneous agent composition. The right panel is obtained considering only time-sensitive agents. Each cell reports the fraction of sampled preference profiles for which the scheduler on the row Pareto-dominates the scheduler on the column. Values near one indicate frequent Pareto dominance; intermediate values correspond to Pareto-incomparable outcomes.

Figure 9 reports Pareto-dominance probabilities across the preference space under systematic competition. Unlike the independent-arrival case, Streaming no longer Pareto-dominates alternative schedulers across the majority of profiles. While it continues to outperform longer waiting policies, FixedBatch strictly dominates Streaming over a non-negligible region of the parameter space, particularly when agents exhibit strong sensitivity to relative execution order.

This dominance reversal is driven by the structure of competition rather than by latency asymmetry. Immediate arrival-ordered execution exposes agents to repeated race losses when arrivals are time-coupled, translating micro-timing noise directly into welfare outcomes. By contrast, batching partially decouples execution outcomes from fine-grained arrival differences, reducing the frequency and severity of execution races. In environments where losses are dominated by ordering rather than absolute delay, this coordination effect outweighs the latency cost introduced by buffering.

Importantly, introducing user-side latency noise does not qualitatively alter this conclusion. While latency uncertainty affects the absolute level of losses, the regions of dominance observed in Fig. 9 are primarily shaped by systematic competition itself, not by protecting a latency-disadvantaged class of participants.

3.5 Regime Dependence of Scheduler Optimality

Taken together, the results of Sections 3.3 and 3.4 demonstrate that the Pareto-optimality of scheduling policies is fundamentally regime-dependent.

When transaction arrivals are independent and competition arises only incidentally through congestion, immediate execution with minimal buffering minimizes aggregate delay and Pareto-dominates alternative scheduling rules across most of the preference space. In this regime, batching primarily redistributes welfare between agents without generating Pareto improvements.

Under systematic, time-coupled competition, the dominance landscape changes qualitatively. Repeated execution races make relative ordering a first-order determinant of welfare, and schedulers that buffer transactions over short windows act as coordination mechanisms that stabilize competitive outcomes. In these environments, batching transitions from a redistributive policy to a welfare-improving execution rule, capable of strictly Pareto-dominating streaming execution over large regions of the preference space.

No single scheduler is therefore welfare-optimal across all environments. Scheduler design cannot be evaluated independently of the arrival regime and the dominant sources of loss faced by participants. Immediate execution is optimal when minimizing delay dominates welfare considerations; buffering becomes optimal when repeated competition makes ordering risk the primary welfare driver.

This conclusion closes the welfare analysis: scheduler choice is not a question of “better ordering” in the abstract, but of which execution externalities dominate under the prevailing competitive structure.

---

4. Parameter-Space Regions of FixedBatch Dominance

The welfare analysis in Section 3 established that scheduler optimality depends on the arrival regime. Under independent arrivals, streaming execution minimizes aggregate welfare loss, while under systematic, time-coupled competition, batching can strictly dominate streaming by stabilizing execution races. However, these results do not by themselves answer a more granular design question: for which classes of agent preferences does batching become strictly welfare-improving?

To address this, we perform a parameter-space dominance analysis. Rather than averaging welfare outcomes across heterogeneous environments, we ask whether FixedBatch Pareto-dominates alternative schedulers for specific preference profiles, and how the size and structure of this dominance region depend on the batch window.

We fix the execution environment and arrival process as in Section 3.4, focusing on the triggered-competition regime that induces systematic contention between agents. For each batch window size, we sample a high-dimensional space of agent preference profiles, varying sensitivities to:

- execution delay
- transaction fees (priority)
- execution deadlines
- tail risk
- and relative ordering (race outcomes).

For each sampled profile, we compute per-agent welfare under Streaming, FixedBatch, and WaitUpTo, and evaluate Pareto dominance using the criteria defined in Section 3.

The first structural result is that the dominance region of FixedBatch expands monotonically with the batch window. For very small windows (on the order of 10–20 ms), FixedBatch Pareto-dominates alternative schedulers only for a negligible fraction of preference profiles. As the batch window increases to intermediate values (50–100 ms), the fraction of profiles for which FixedBatch strictly dominates grows substantially. Beyond this range, further increases in the

batch window yield diminishing returns: the dominance region does not expand materially when moving from 100 ms to 200 ms, see from Fig. E1 to Fig. E5 in Appendix E.

Crucially, even at large batch sizes, dominance never becomes universal. FixedBatch improves welfare only on specific regions of the preference space, while other regions remain either streaming-dominant or Pareto-incomparable. This confirms that batching is not a generic improvement, but a regime- and preference-dependent design choice.

4.1 Structure of Dominant Preference Profiles

Figure 10 should be read as a preference-space diagnostic, rather than as a classification of concrete agent types. Each point corresponds to a single sampled preference profile, defined by a specific combination of sensitivities to execution delay, transaction fees, deadlines, tail risk, and relative ordering. For each profile, we compute the induced loss under Streaming, FixedBatch, and WaitUpTo scheduling, and evaluate whether FixedBatch Pareto-dominates the alternatives under the criteria defined in Section 3.

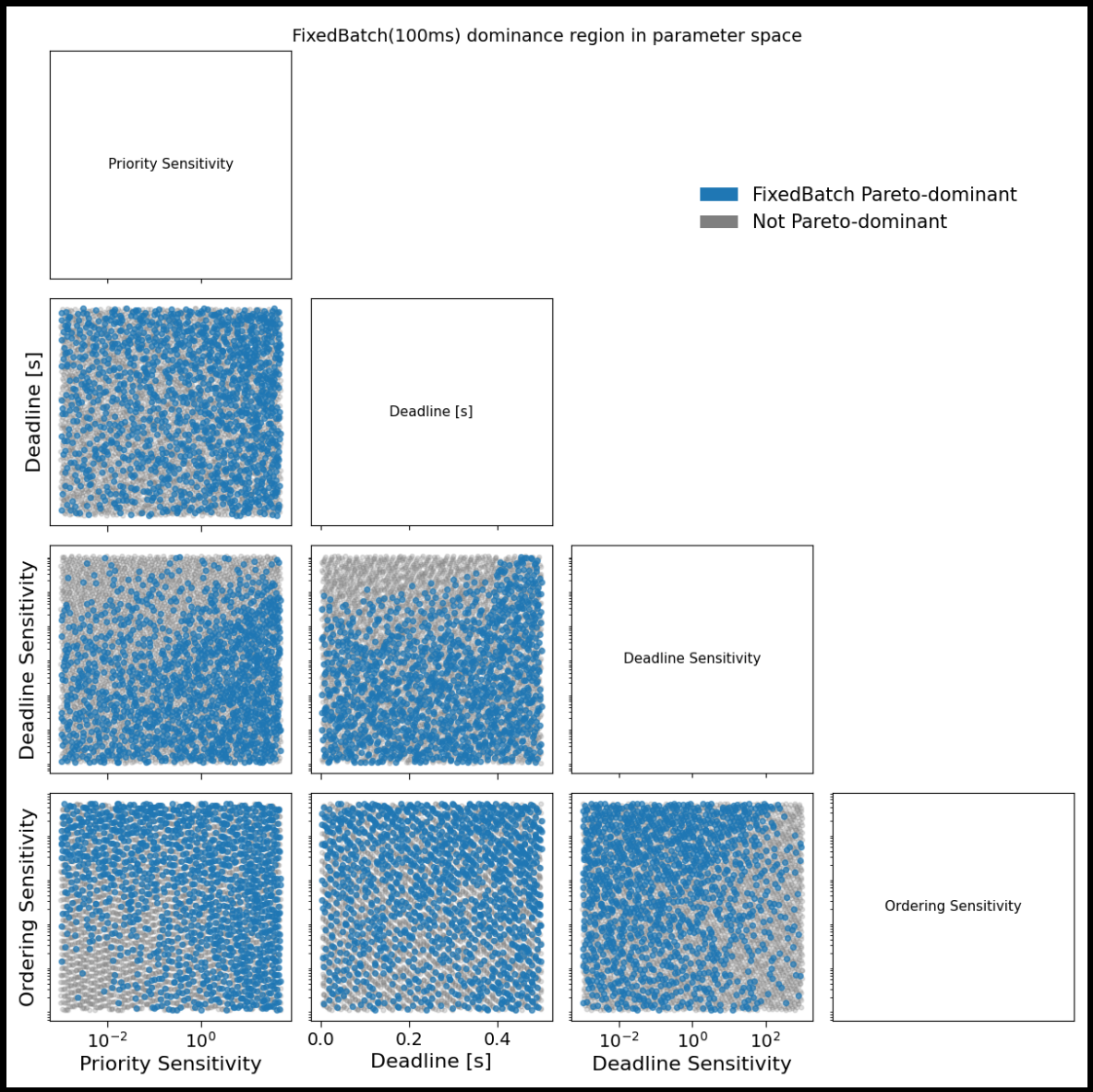

Fig. 10: Pairwise projections of sampled loss parameters for user agents comparing FixedBatch(100ms) with Streaming and WaitUpto(300ms). Blue points denote agent preference profiles where FixedBatch(100ms) dominates alternative schedulers in social welfare; grey points denote non-dominant profiles.

A blue point therefore indicates a preference profile for which FixedBatch yields strictly lower welfare loss than both Streaming and WaitUpTo under the same execution environment. Grey points indicate profiles for which FixedBatch is either Pareto-dominated or Pareto-incomparable. The figure does not encode the magnitude of welfare improvement, only whether a strict improvement occurs.

The purpose of the figure is to identify regions of the preference space for which batching improves welfare under the assumed loss model. Interpretation proceeds by examining where blue points concentrate and which combinations of sensitivities are associated with FixedBatch dominance.

Two robust patterns emerge.

First, FixedBatch dominates for profiles with low sensitivity to relative execution order and high sensitivity to transaction fees. In the corresponding panels of Fig. 10, blue points appear predominantly when priority sensitivity is large, even when ordering sensitivity is close to zero. This indicates that batching improves welfare for agents whose losses are driven primarily by fee expenditure rather than by execution order. For these profiles, the additional latency introduced by batching is economically insignificant, while the reduction in fee competition yields a net welfare gain.

Second, FixedBatch also dominates for profiles with high sensitivity to relative execution order and low sensitivity to transaction fees. In this region, welfare losses are dominated by execution races rather than by fees. Small, noisy arrival-time differences translate into large losses under streaming execution. By aggregating transactions over short windows, batching dampens this sensitivity to micro-timing noise, stabilizing execution outcomes and yielding strict Pareto improvements.

These two regions correspond to distinct mechanisms through which batching improves welfare: fee smoothing in the first case, and race stabilization in the second. Importantly, they do not define exclusive agent categories. Rather, they identify directions in preference space along which batching mitigates the dominant source of loss.

In contrast, preference profiles characterized by high deadline sensitivity and tight execution deadlines rarely fall within the FixedBatch dominance region. For such profiles, absolute timeliness is binding: execution delay cannot be traded off against lower fees or reduced ordering noise. In these regimes, the buffering intrinsic to batching directly translates into welfare loss, overwhelming any coordination benefits. As a result, deadline-driven workloads remain structurally misaligned with batching, even under systematic competition.

4.2 What Batching Changes (and What It Cannot)

Batching improves welfare only when it reduces the main source of loss experienced by agents:

- If agents mainly lose welfare because of fee competition or execution races, batching helps by making execution less sensitive to small differences in priority bids or arrival times
- If agents mainly lose welfare because of delay itself, batching hurts by adding latency that cannot be compensated elsewhere.

In other words, batching helps when welfare is dominated by relative effects (who goes first, how much is paid), and fails when welfare is dominated by absolute timing (how fast execution happens).

This is why batching can strictly improve welfare in race-driven or fee-sensitive environments, but is structurally incompatible with deadline-driven workloads, where any additional delay directly translates into loss.

Batching is therefore not a general improvement to execution quality, but a targeted mechanism that is beneficial only when it suppresses the dominant source of welfare loss.

Appendix

A1. Scheduler Footrule Distance

For a block containing N non-vote transactions, define:

- prio_i: the rank of transaction i when transactions are sorted by priority per compute unit,
- rn_i: the rank of the same transaction in the actual execution order.

We define the Spearman footrule distance between these two permutations:

$$S = \sum_{i=1}^{N} |\,\text{prio}_i - \text{rn}_i\,| \,.$$

This quantity measures the total displacement, in rank space, between the realized execution order and the ideal priority-sorted order.

Since the raw footrule distance scales as $O(N^2)$, it is not directly comparable across blocks of different sizes. Its maximum value over all permutations is:

$$S_{max} = \begin{cases} \dfrac{N^2}{2} & N \text{ even}, \\ \dfrac{N^2-1}{2} & N \text{ odd}\,. \end{cases}$$

and we therefore introduce the normalized scheduler footrule distance

$$D = \frac{S}{S_{max}}\,,$$

which constitutes a block-size invariant with

- $D = 0$: realized order equals priority order exactly
- $D = 1$: realized order is maximally anti-correlated with priority (reverse order)
- intermediate values quantify partial deviation.

A1.1. Expected value under a random-permutation null

In our framework, if $i \in \mathcal{I} = \{1,\ldots,n\}$ is the index of the transaction ordered by priority and $\sigma(i) \in \mathcal{I} = \{1,\ldots,n\}$ is its permutation to reflect execution, we can write the Spearman's Footrule metric as

$$S(i,\sigma(i)) = \sum_{i=1}^{n} |\,i - \sigma(i)\,| \,.$$

Given the fact that for each i, $\sigma(i)$ can be any of the $\{1,\dots,n\}$ with probability $1/n$ - given the random permutation assumption - we have

$$\mathbb{E}[S(i,\sigma(i))] = \frac{1}{n}\mathbb{E}[S(i,j)] = \frac{1}{n}\sum_{i,j=1}^{n} |\,i-j\,|\,.$$

At this point, since when $i=j$ the sum is null and when $i<j$ it is equal to the sum when $j<i$ (due to the absolute value) we have that

$$\mathbb{E}[S(i,j)] = 2\times\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i-1}(i-j)\,.$$

From the equality

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}\,,$$

it follows

$$\mathbb{E}[S(i,j)] = \frac{2}{n}\sum_{i=1}^{n}\left[\sum_{j=1}^{i-1}(i-j)\right] = \frac{2}{n}\sum_{i=1}^{n}\left[\sum_{k=1}^{i-1}k\right] = \frac{2}{n}\sum_{i=1}^{n}\left[\frac{i(i-1)}{2}\right]\,.$$

At this point, breaking the sum we get

$$\mathbb{E}[S(i,j)] = \frac{1}{n}\left[\frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2}\right] = \frac{n^2-1}{3}\,.$$

Dividing by S_{max} and sending $N\to\infty$ we get the expected value of the Scheduler Footrule Distance in the case of random permutations

$$\mathbb{E}[D] = \frac{\mathbb{E}[S]}{S_{max}} \xrightarrow[N\to\infty]{} \frac{2}{3}\,.$$

B1. The Scheduler Footrule Distance under High workload

During the March–April 2025 period, blocks on the Solana network were, on average, consistently full and exhibited limited variability in total compute usage, as documented in the analysis by Chorus One.

Focusing on the window from March 23rd to April 3rd, we observe that the distribution of the scheduler footrule distance D remains centered close to the random-permutation baseline ($D \simeq 2/3$), indicating that, at a network-wide level, transaction ordering was still largely dominated by arrival-time effects.

At the same time, the distribution exhibits a pronounced left tail, corresponding to blocks in which the realized execution order is substantially closer to priority ordering. Unlike later periods, this left tail does not develop into a secondary peak, which is consistent with the relatively small fraction of stake running Firedancer during this time window.

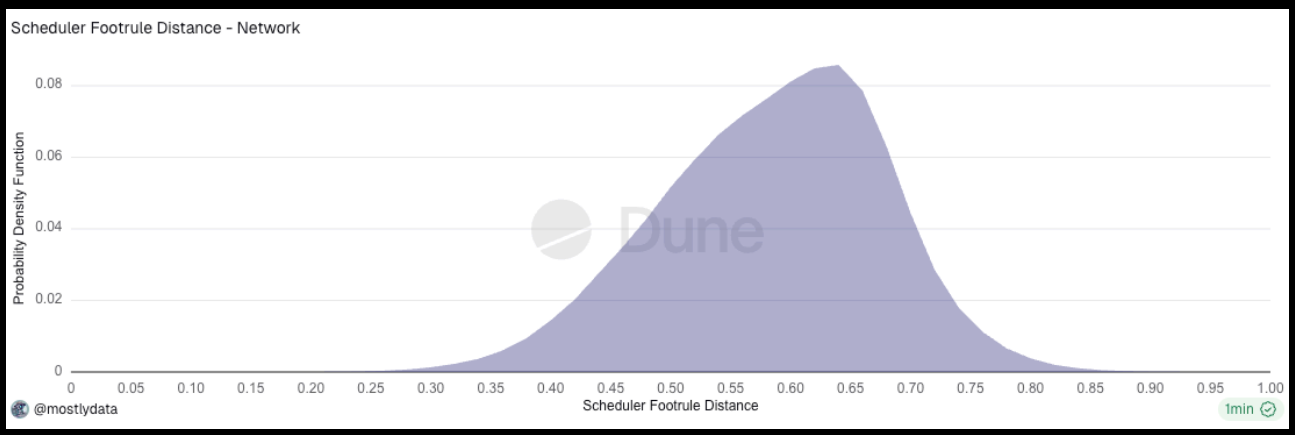

Fig.B1: Distribution of the scheduler footrule distance for the entire Solana network from March 23rd to April 3rd 2025. Source https://dune.com/queries/6364939/10123516

Importantly, this behaviour emerges despite the fact that blocks were already saturated in terms of compute units, ruling out explanations based on variable block fullness or unused capacity.

This interpretation is further supported by the absence of any statistically meaningful correlation between the scheduler footrule distance D and total compute units consumed per block. As shown in Fig. B2, blocks spanning the full range of compute usage exhibit similar values of D, indicating that deviations from random ordering are not explained by variations in CU load.

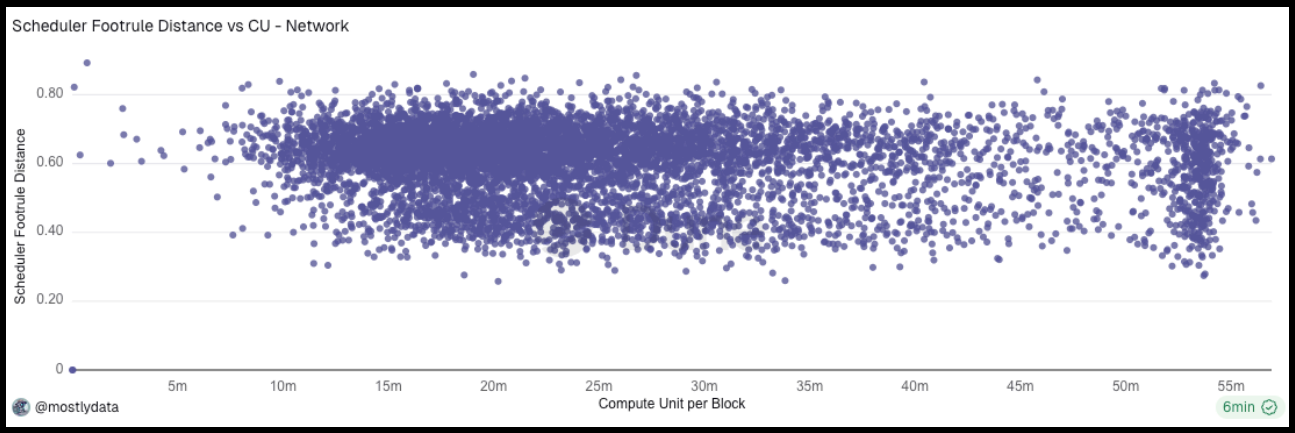

Fig.B2: Distribution of the scheduler footrule distance for the entire Solana network from March 23rd to April 3rd 2025 vs CU per block. Source https://dune.com/queries/6380791/10145030?n_days_n26d66=1

Taken together, these observations suggest that the long-left tail in the D distribution reflects scheduler-level effects, rather than block-level resource constraints.

C1. Relation to PoH-tick–level metrics

An important aspect of the scheduler footrule distance D is that it implicitly integrates information contained in lower-level timing metrics, such as transactions per PoH tick, while remaining directly grounded in on-chain execution order.

Schedulers that operate on rigid tick-based execution models — such as those that collect transactions externally (e.g., from a block builder) and execute them wholesale within predefined PoH ticks — may appear highly structured when viewed through tick-level metrics alone.

However, such metrics do not capture how priority information is reflected in the realized execution order once order-flow variability and conflict constraints are taken into account.

By construction, D measures the deviation between priority ordering and actual execution, independently of how transactions are grouped temporally at the PoH level. As a result, schedulers that execute all transactions within a fixed tick (as in harmonic or builder-driven designs) can still exhibit D distributions that resemble those of more adaptive or state-dependent schedulers, depending on how priority and conflicts are resolved in practice.

In this sense, the scheduler footrule distance captures the effective on-chain behaviour arising from the interaction of order-flow and scheduler logic, rather than the nominal structure imposed by tick-level execution. This makes D particularly suitable for comparing heterogeneous scheduler designs on equal footing and for identifying adaptive behaviours that are not directly visible from timing metrics alone.

D1. Agent Loss Function and Preference Heterogeneity

This appendix provides the formal specification of the agent-level loss functions used in Section 3. The purpose of this construction is to map execution timing and ordering outcomes into agent-specific disutility in a flexible but tractable way.

Throughout, the scheduler is treated as value-neutral: it determines when transactions execute and in what order. Losses arise only once these outcomes are evaluated through agent preferences.

Each agent i is assigned a per-block loss function composed of three classes of disutility:

$$L_i = L_i^{\text{prio}} + L_i^{\text{delay}} + L_i^{\text{race}}.$$

- **Priority cost:** Execution timing is not exogenous. Agents can attempt to obtain earlier execution by bidding higher priority, which directly increases their fees. In our model, a transaction submitted by agent i carries a priority bid $b_i > 0$, and the agent incurs a fee disutility proportional to the bid,

  $$L_i^{\text{prio}} = \alpha_i b_i,$$

  where $\alpha_i \geq 0$ represents the agent's marginal sensitivity to fee expenditure. The scheduler affects welfare because it governs the extent to which higher bids actually translate into improved execution: under strong priority enforcement, increasing b_i reliably advances execution rank, whereas under arrival-dominated execution the same payment may yield little or no advantage.
  - A retail user submitting occasional swaps may have a large α_i, reflecting higher sensitivity fees.
  - A professional application (e.g., a market maker or oracle) may have a small α_i, treating fees as an operational cost necessary to maintain execution quality.
- **Delay-related losses:** Execution delay induces disutility through three channels: baseline latency sensitivity, deadline penalties, and tail-risk aversion. We group these components into a single delay-related loss term

  $$L_i^{\text{delay}} = L_i^{\text{pure delay}} + L_i^{\text{ddl}} + L_i^{\text{tail}}.$$

- **Baseline delay sensitivity:** The simplest source of disutility is delay. Some agents are relatively insensitive to execution time within a slot, while others experience rapidly increasing loss as execution is delayed. We model this through a linear delay term,

  $$L_i^{\text{pure delay}} = \beta_i \Delta t_i \,,$$

  where $\beta_i \geq 0$ controls the agent's sensitivity to execution latency, and Δt_i represents the difference between execution time and transaction generation time.

  - A retail user submitting a swap with wide slippage tolerance may have a small β_i.
  - A latency-sensitive application, such as a pricing oracle or routing service, may have a much larger β_i.

- **Deadline sensitivity (soft deadlines):** Many agents face timing constraints beyond which execution remains possible but increasingly undesirable (e.g., stale prices or violated risk bounds). We model this with a one-sided penalty,

  $$L_i^{\text{ddl}} = \gamma_i \max(0, t_i^{\text{exec}} - d_i) \,,$$

  where d_i is the agent's deadline and $\gamma_i \geq 0$ controls the severity of lateness.

  - A casual user may have a small γ_i, tolerating late execution with modest degradation.
  - A risk-sensitive application may have a large γ_i approximating "execute or fail" behaviour.

- **Tail-risk aversion:** Two schedulers may yield similar average delays but differ substantially in their tail behaviour. Some agents are particularly sensitive to rare but severe delays. We capture this via an excess-delay penalty applied beyond a high quantile τ,

  $$L_i^{\text{tail}} = \delta_i \sum_k \max(0, \Delta t_{i,k} - q_i(\tau)) \,.$$

  - An infrastructure application that must maintain predictable performance may have a large δ_i.
  - A non-urgent user may have $\delta_i \sim 0$, caring little about rare execution outliers.

- Relative ordering (execution race): Because user and application transactions touch the same state account, welfare may depend on which executes first when both are simultaneously runnable. We model this with a simple ordering penalty,

  $$L_i^{\text{race}} = k_i \mathbf{1}\{j \prec i\} \,,$$

  where $j \prec i$ means agent j transaction comes before agent i transaction.

  - A user trading against on-chain liquidity may prefer the application to execute first (small k_i).
  - A liquidation bot incur loss if competitors act first (large k_i).

E1. Parameter-Space Regions of FixedBatch Dominance

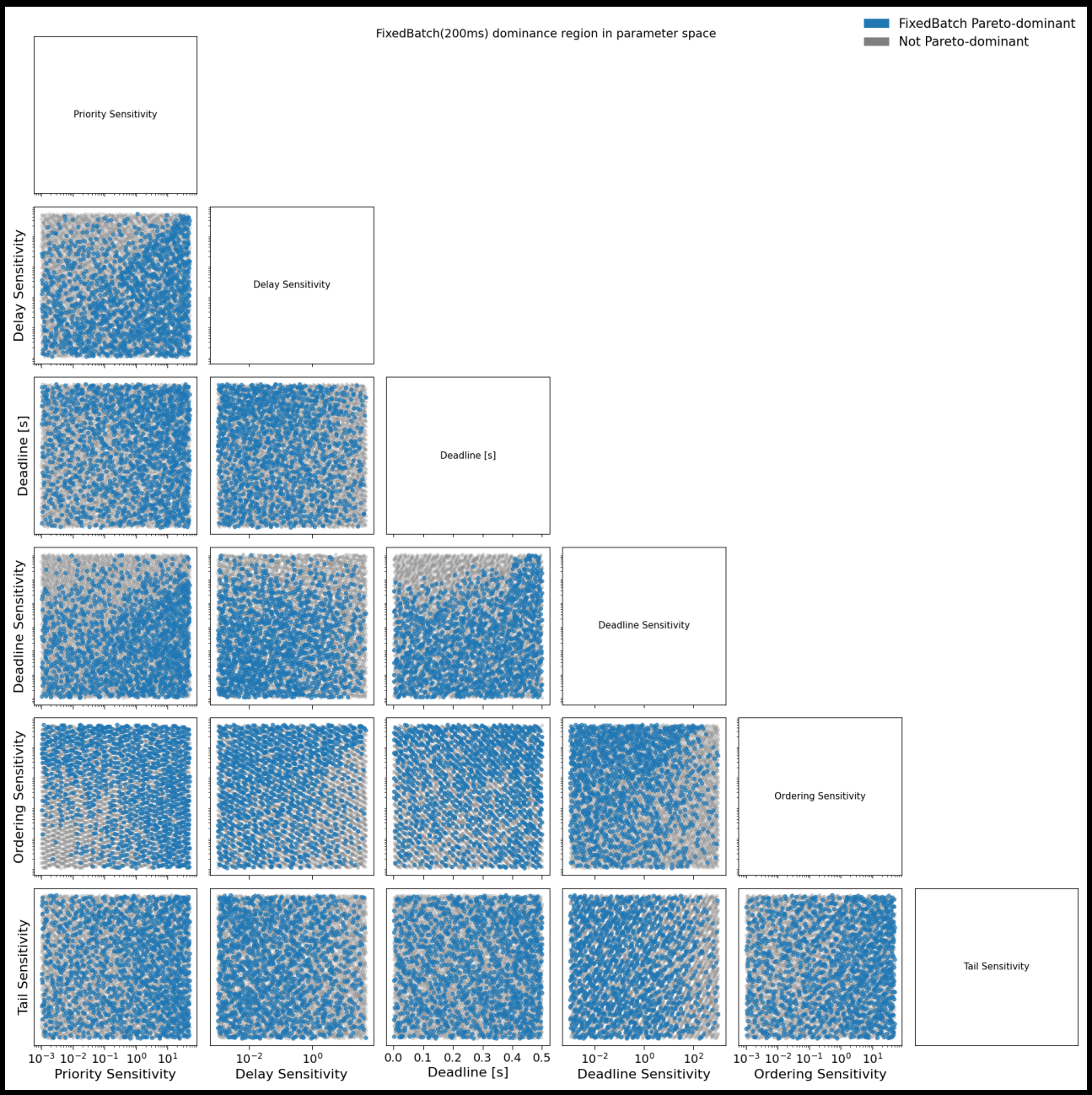

Fig. E1: Pairwise projections of sampled loss parameters for user agents comparing FixedBatch(200ms) with Streaming and WaitUpto(300ms). Blue points denote agent preference profiles where FixedBatch(200ms) dominates alternative schedulers in social welfare; grey points denote non-dominant profiles.

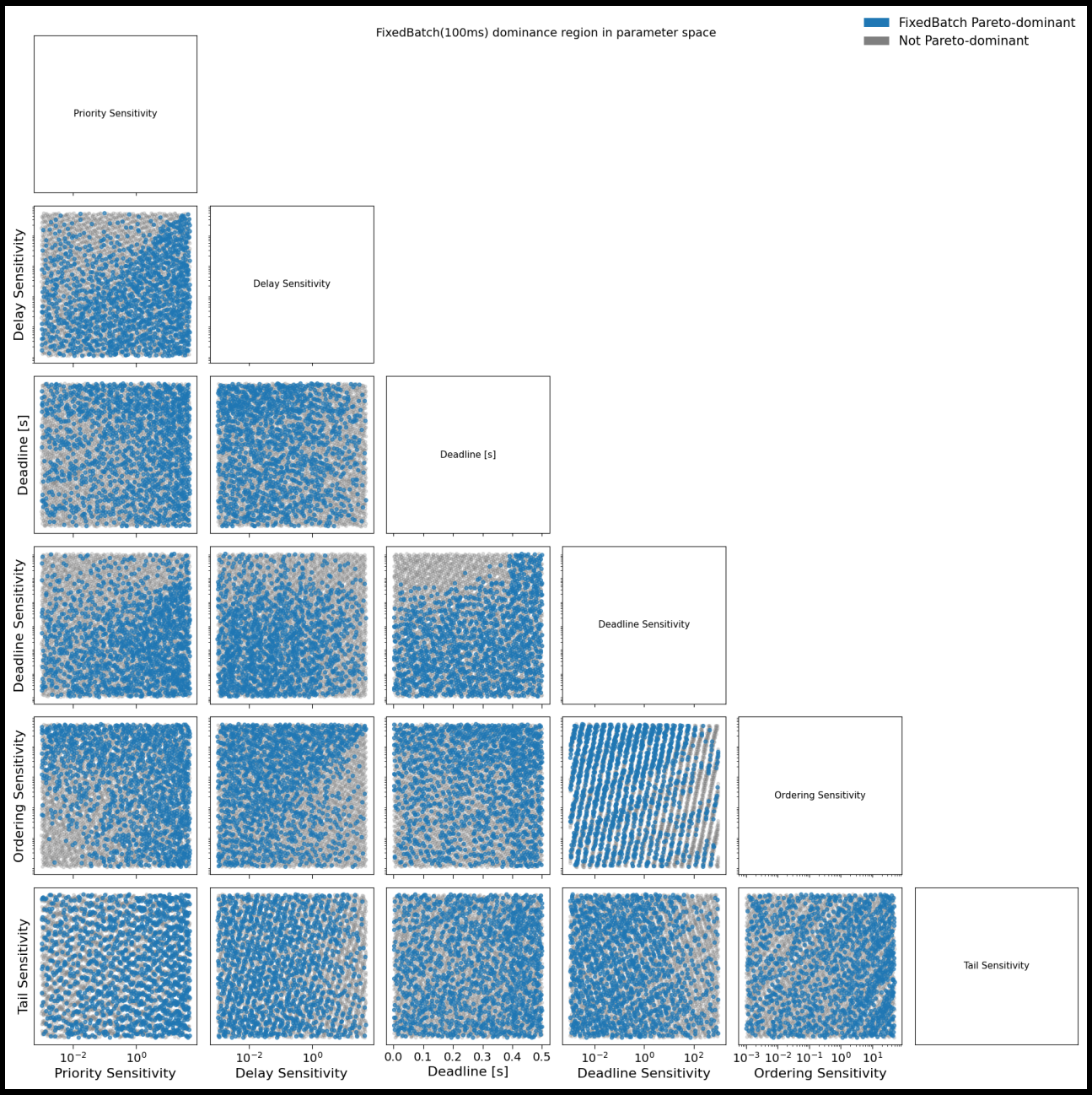

Fig. E2: Pairwise projections of sampled loss parameters for user agents comparing FixedBatch(100ms) with Streaming and WaitUpto(300ms). Blue points denote agent preference profiles where FixedBatch(100ms) dominates alternative schedulers in social welfare; grey points denote non-dominant profiles.

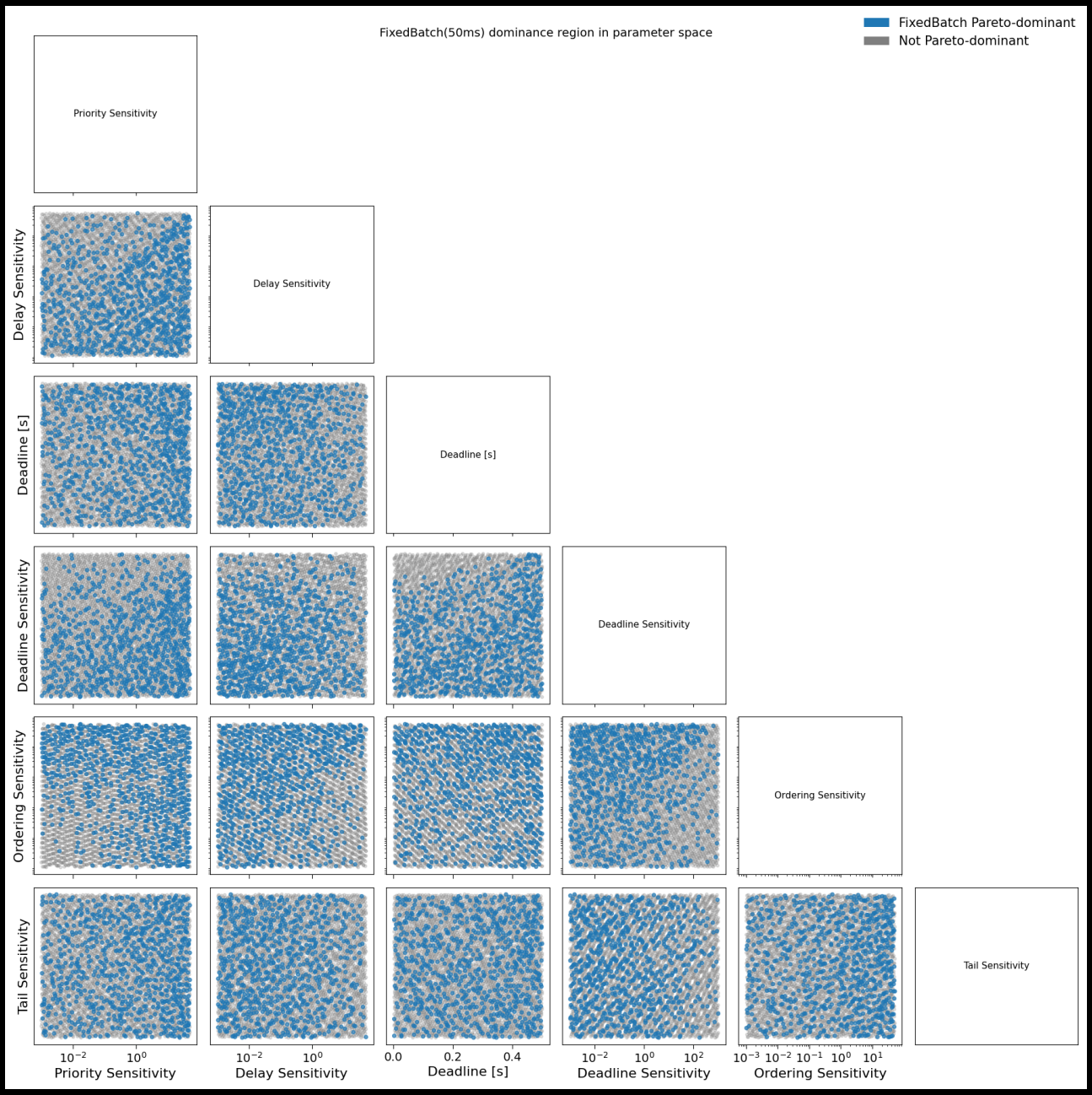

Fig. E3: Pairwise projections of sampled loss parameters for user agents comparing FixedBatch(50ms) with Streaming and WaitUpto(300ms). Blue points denote agent preference profiles where FixedBatch(50ms) dominates alternative schedulers in social welfare; grey points denote non-dominant profiles.

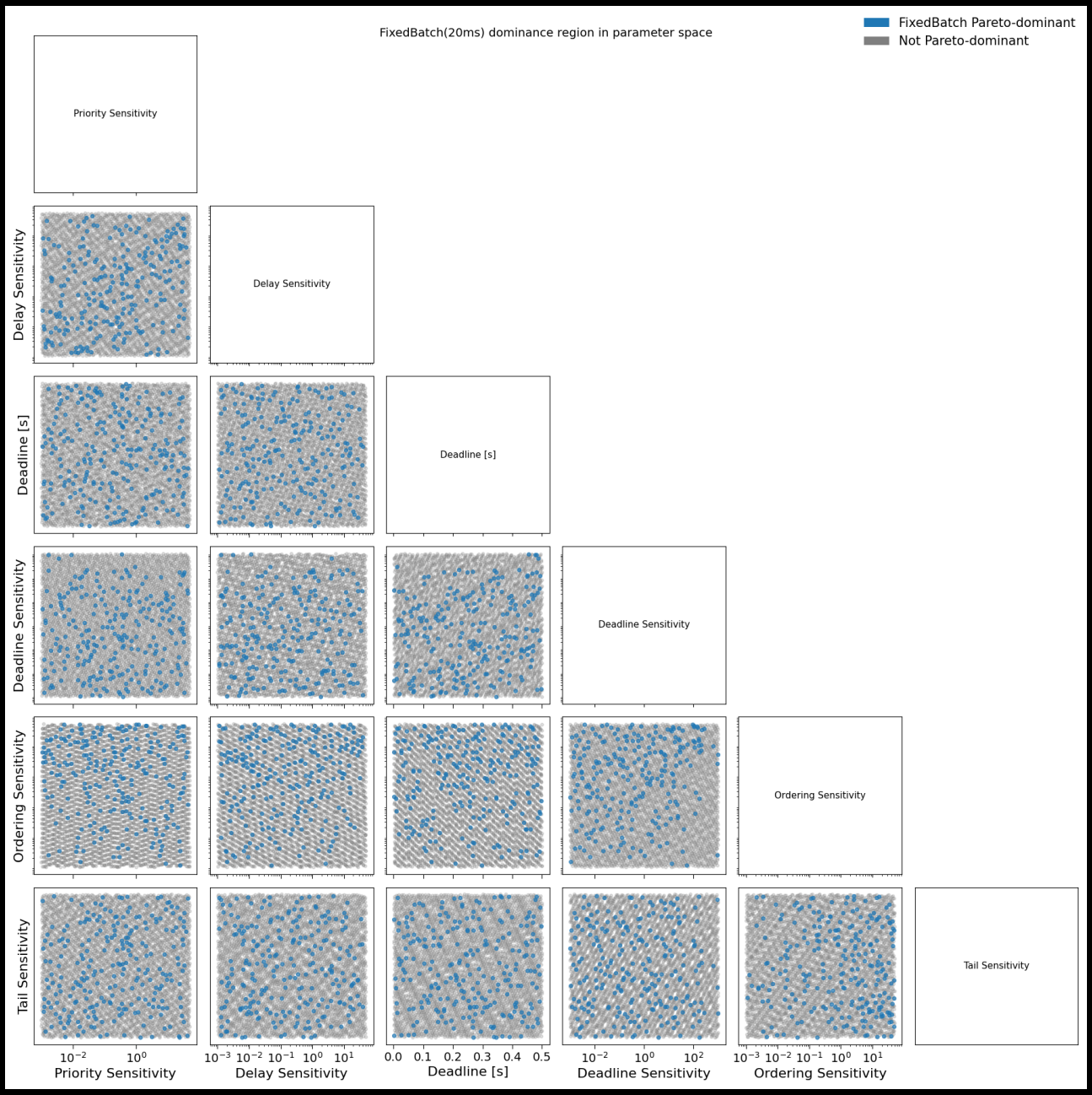

Fig. E4: Pairwise projections of sampled loss parameters for user agents comparing FixedBatch(20ms) with Streaming and WaitUpto(300ms). Blue points denote agent preference profiles where FixedBatch(20ms) dominates alternative schedulers in social welfare; grey points denote non-dominant profiles.

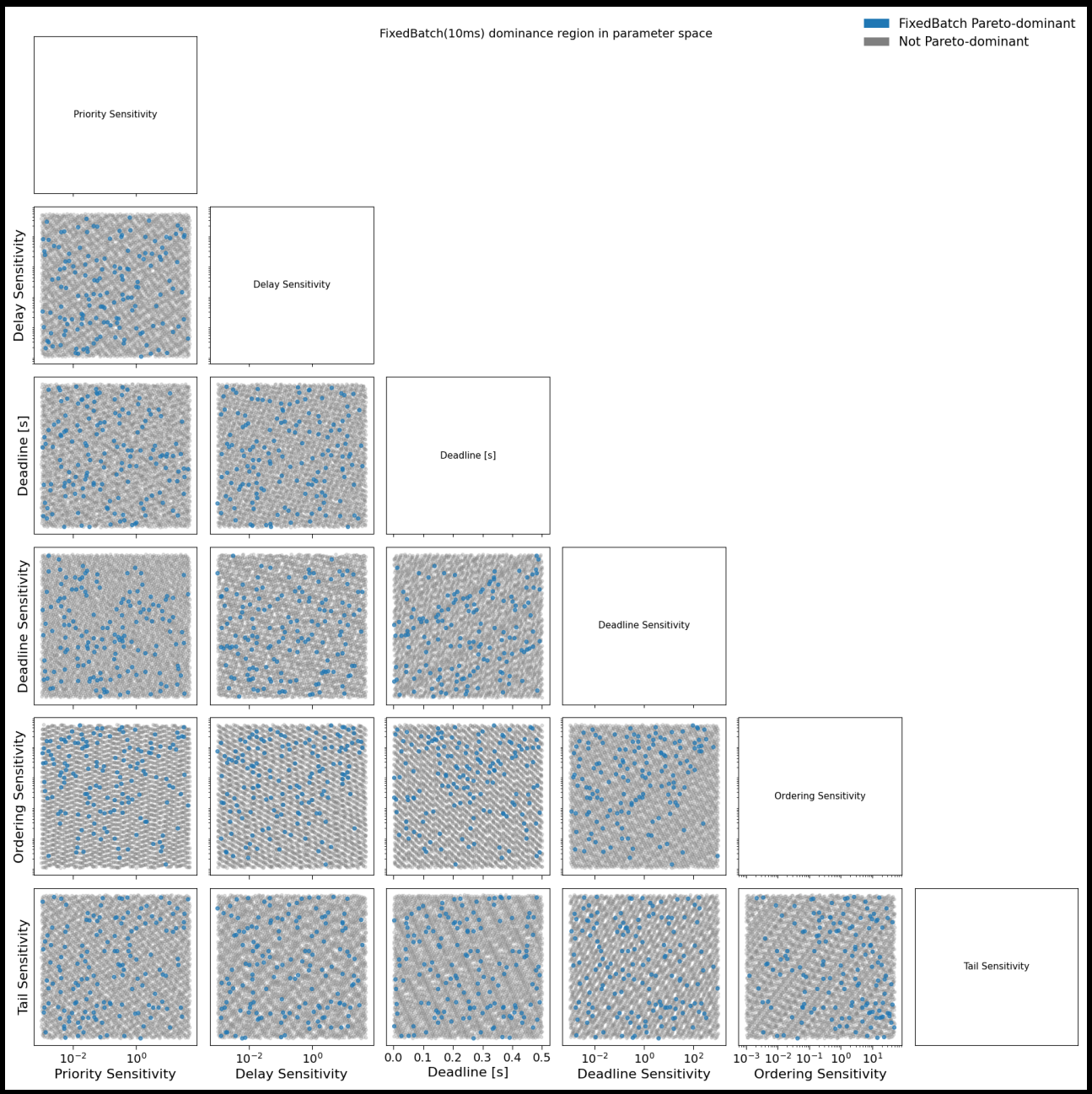

Fig. E5: Pairwise projections of sampled loss parameters for user agents comparing FixedBatch(10ms) with Streaming and WaitUpto(300ms). Blue points denote agent preference profiles where FixedBatch(10ms) dominates alternative schedulers in social welfare; grey points denote non-dominant profiles.