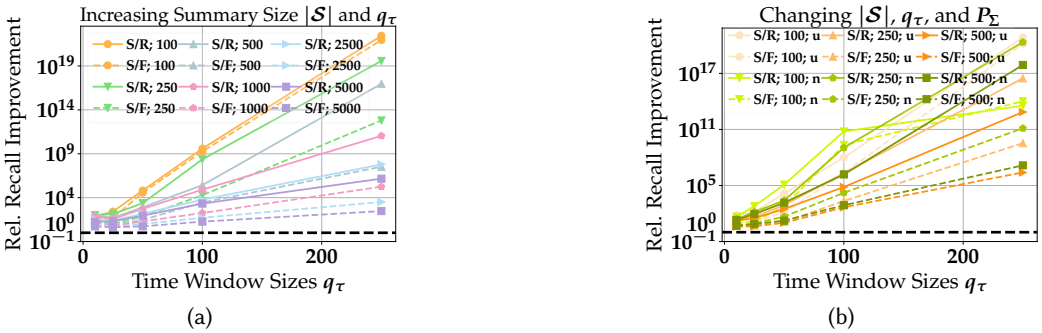


Fig. 6. Examining the impact on relative recall improvement (higher is better) by varying (a) summary and time window sizes and (b) the alphabet probability distribution P_Σ.

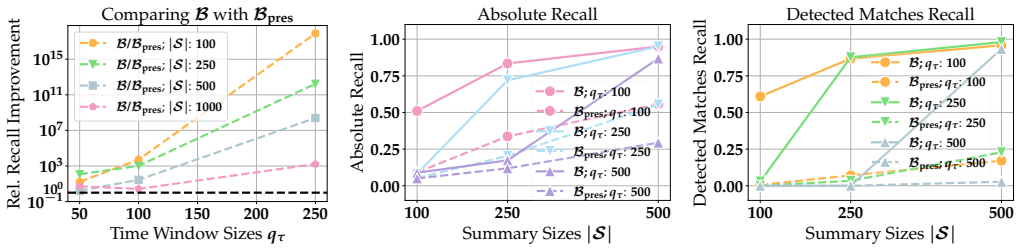


Fig. 7. Examining the impact of the benefit function $\mathcal{B}$ compared to $\mathcal{B}_{\text{pres}}$ (a) on the relative recall improvement, (b) proximity to lower bound in absolute recall, and (c) proximity to lower bound in detected matches recall (higher is better in all plots).

7.2 Overall Effectiveness

We first compare against a random baseline (SuSe/Random - S/R) and a FIFO baseline (SuSe/FIFO - S/F). In **??** - **??**, we assessed how variations in the summary size $|\mathcal{S}|$ (100 to 5000), time window size q_τ (10 to 250), and alphabet probability distribution P_Σ (Zipfian, uniform (u), normal (n)) affect the relative recall improvement, using a query with $q_\gamma = a(b^*c)^*d(e|f)g^*$.

SuSe outperforms its baselines by choosing summaries that generate up to 10^{20} more matches. There is no parameter combination, in which SuSe performs worse than a baseline (black dashed line). A clear trend emerges for both plots: the larger q_τ and the smaller $|\mathcal{S}|$, the more significant the advantage.

As the summary size increases, the baselines find more matches by chance; nevertheless, for these parameters, SuSe chooses stream subsequences that generate at least three orders of magnitude more matches on average.

Ablation Study. We compared SuSe's benefit function $\mathcal{B}$ against a setup using solely the present benefit $\mathcal{B}_{\text{pres}}$, i.e., denoted by $\mathcal{B}/\mathcal{B}_{\text{pres}}$, in terms of the relative recall improvement, the related absolute recall, and the detected matches recall. In **??**, a trend similar to that in **??** is observed: smaller $|\mathcal{S}|$ and larger q_τ lead to better subsequence selections. This is attributed to $\mathcal{B}$ providing a more accurate assessment of an element's future potential.

For $q_\tau = 250$, SuSe with $\mathcal{B}$ obtains at least three orders of magnitude more matches on average over solely using $\mathcal{B}_{\text{pres}}$ and up to 10^{18} more for smaller values of $|\mathcal{S}|$, showing the effectiveness of $\mathcal{B}$.

Absolute Recall. In **??**, we evaluated the loss function l_{COUNT} from **??**, showing how closely SuSe approaches the optimal recall value.

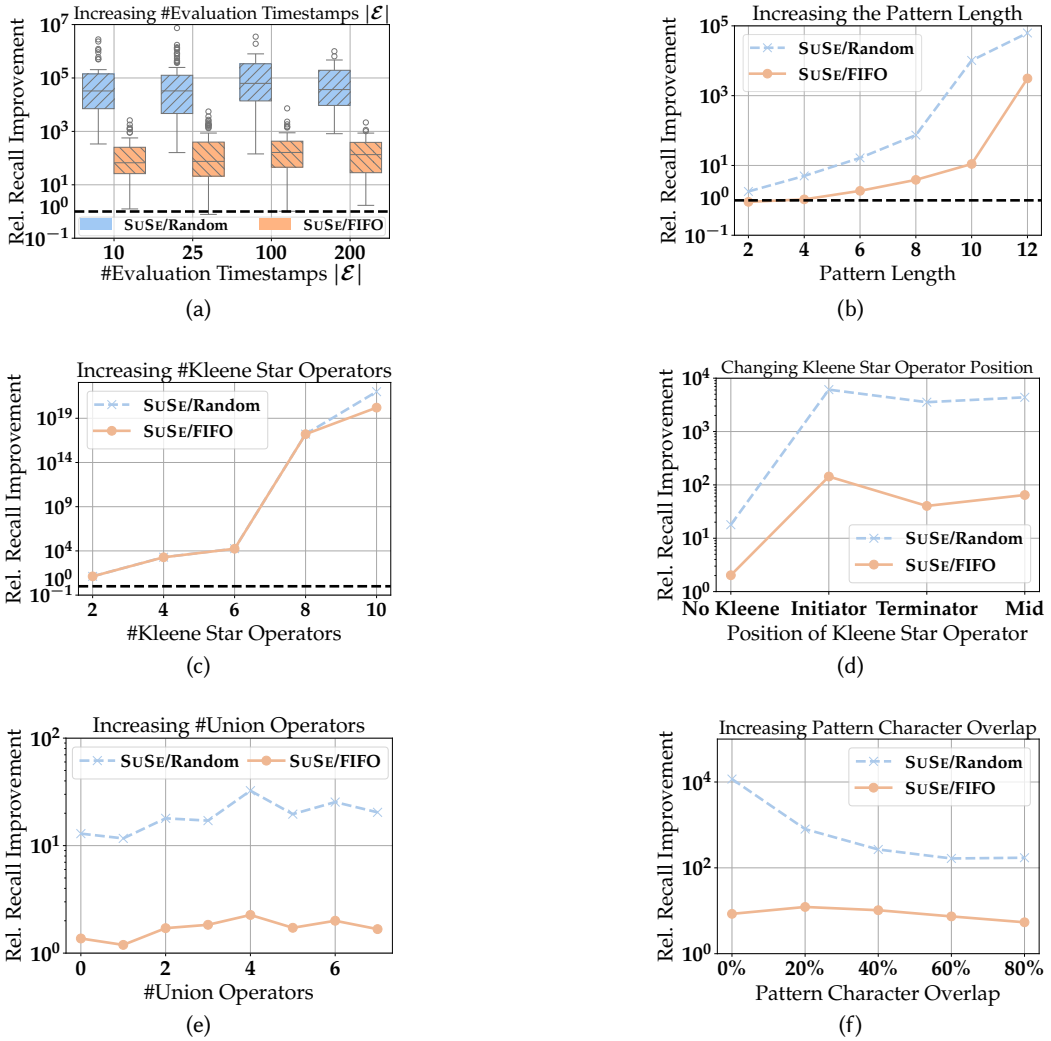


Fig. 8. Examining the impact on relative recall improvement (higher is better) by varying (a) the number of evaluation timestamps $|\mathcal{E}|$, (b) the pattern length, (c) the count of Kleene stars, (d) Kleene star position, (e) union operator count, and (f) query character overlap.

Here, increasing $|\mathcal{S}|$ positively affects recall. This is attributed to the reduced difference between the ground truth's summary size (equal to the stream size) and SuSe's summary size. SuSe clearly attains higher recall values when employing expected benefits $\mathcal{B}$, compared to using solely $\mathcal{B}_{\text{pres}}$. Also, when $|\mathcal{S}| \geq q_\tau$, with $\mathcal{B}$, SuSe selects substantially better subsequences, increasing recall by up to 80% – 95%. This trend can be attributed to our assumption in **??** regarding the expected benefit $\mathcal{B}$, where we implicitly assumed

that at least one time window size of elements fits into the summary. For values $|\mathcal{S}| \geq q_\tau$, this assumption is valid, leading to more accurate estimates and an enhanced recall.

Detected Matches Recall. For **??**, we examined how many of *all* possible matches were present in SuSe, again comparing $\mathcal{B}$ and $\mathcal{B}_{\text{pres}}$. The trends align closely with those observed for the absolute

recall. SuSe with $\mathcal{B}$ yields a recall of 90% − 98% for $|\mathcal{S}| = 500$, indicating that SuSe chooses rich stream subsequences.

7.3 Sensitivity Analysis

Next, we assess the effect of various parameters on the relative recall improvement. We fix $q_\tau = 100$ and $|\mathcal{S}| = 500$, and vary stream sizes (10000 - 25000), the numbers of evaluations (25 - 50), and P_Σ (Zipfian or uniform). We report averages of over 50 runs.

Number of Evaluation Timestamps. ?? examines the influence of increasing the number of evaluation timestamps $|\mathcal{E}|$ on the relative recall improvement. As the boxplots indicate, increasing the number of evaluations does not impact the results.

Pattern Length. ?? shows that as the pattern length grows, selecting subsequences that yield matches becomes increasingly challenging.

SuSe addresses this by quantifying the importance of individual elements, prioritizing, for instance, rare elements that are essential to keep to obtain matches.

While for pattern length two, FIFO performs slightly better, for pattern lengths above ten, SuSe is superior by up to almost five orders of magnitude.

Kleene Operator. In **??**, we employed a regular expression of length twelve and incrementally increased the number of Kleene operators, distributing them randomly. As SuSe is aware of characters with Kleene operators (by Count/Sum rules), it selects better summaries.

In **??**, we examined four regular expressions: one without a Kleene operator and three with the Kleene's position varied. Without Kleene, SuSe's advantages are minimal, while there is a large difference for the other cases, independent of the Kleene position.

Union Operator. We also explored the effect of the union operator in **??**. We varied the number of unions from zero to seven and generated and combined random sequences for q_γ's with union operators. Yet, the impact is negligible.

In **??**, we fixed a query of length ten and progressively increased the character overlap. We randomly selected two characters of the given query for each step, linking them to the query and a union.

While the baselines show different trends, in all configurations, SuSe yields significant improvements.

7.4 Efficiency

StateSummary State-of-the-art Comparison. We compared StateSummary's efficiency against the state-of-the-art RegEx engine REmatch and two CEP engines, FlinkCEP and CORE, in **??**. We summarize the idea of the experiment as follows: To assess the current state, i.e., the current number of matches and involvements of individual events, we would need to use a RegEx/CEP engine. However, such engines implement a traditional two-step approach: first building the state, then evaluating aggregate functions over it. Hence, these methods deliver matches and exact results. In contrast, our approach avoids the direct maintenance of (partial) matches and evaluates aggregate functions without match materialization. When the summary is smaller than the word to process, the resulting aggregate value is approximate; however, if the summary is greater than or equal to the word size, the results remain exact.

Given the inferior performance of other RegEx engines (see [**?**]) and their limitation in computing *all* subsequence matches, we focus on REmatch as a representative RegEx engine.

FlinkCEP is an industry-strength tool offering a robust framework for fault-tolerant and distributed stream processing. As such, it offers features that are not implemented in the other engines, which induces a non-negligible overhead during query evaluation that needs to be considered in any comparative analysis. We note for FlinkCEP, though, that parallelizing the workload is not

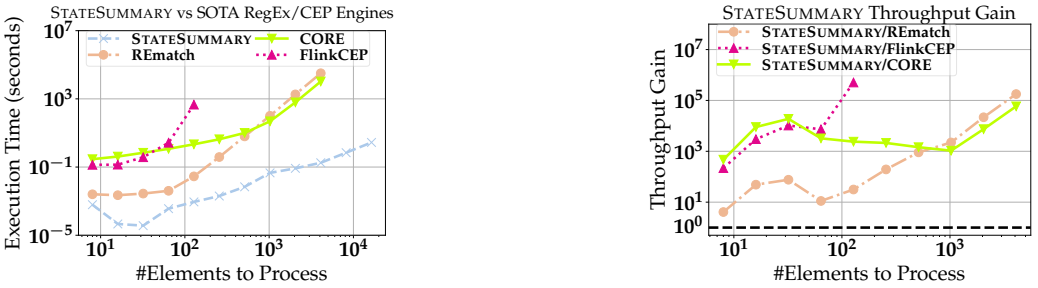


Fig. 9. Comparison against the state of the art, focusing on (a) execution time (lower is better) and (b) throughput gain (higher is better), varying the number of processed elements.

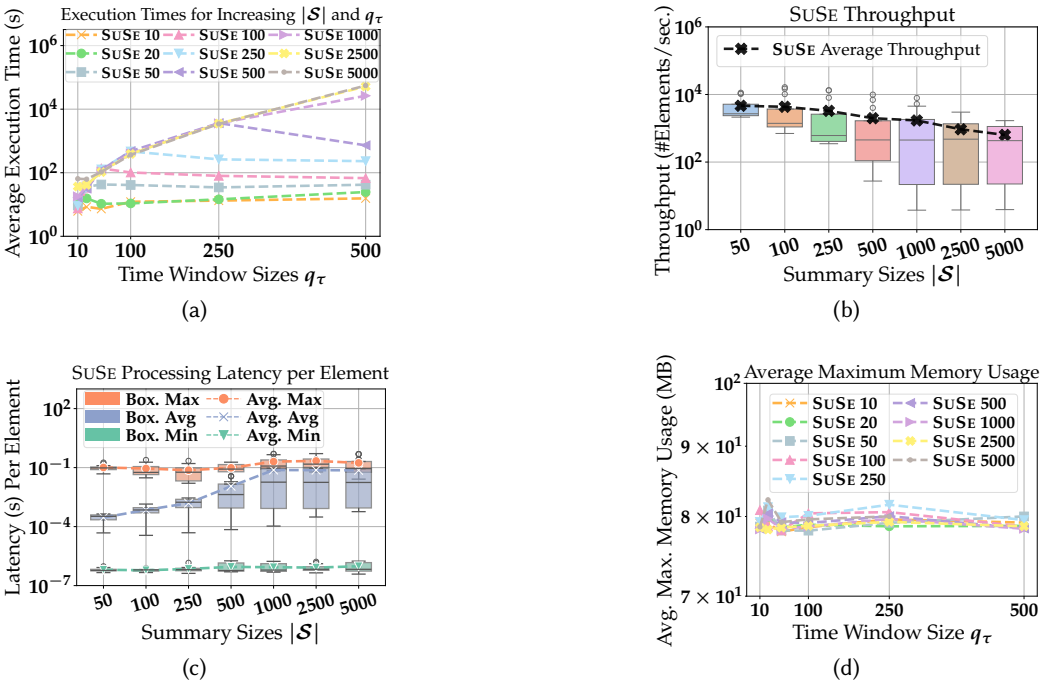
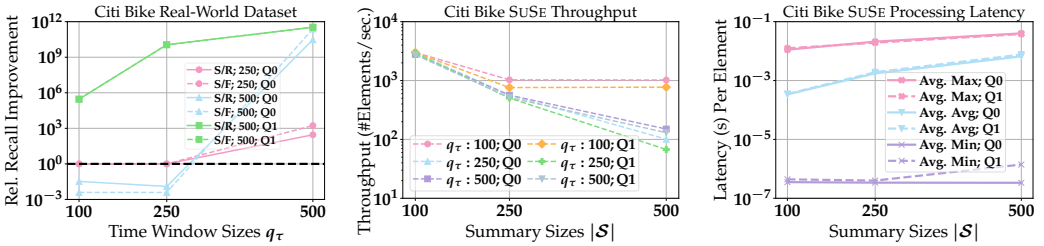


Fig. 10. (a) Execution time (lower is better) for processing 10^5 elements across different summary and time window sizes, (b) throughput (higher is better), (c) average minimum, average maximum, and average processing latency per element (lower is better), and (d) average maximum memory usage (lower is better).

straightforward, since the input stream must be keyed and partitioned while ensuring that *all* matches are found, which is further complicated by operators like Kleene star. While FlinkCEP employs a traditional automata-based evaluation, CORE employs compact match representations, avoiding exponential state growth, and provides match enumeration with output-linear delay.

The evaluated procedures utilized one to two cores on average (i.e., FlinkCEP did not parallelize processing as the input stream is not keyed). The maximum resident set size was ≈ 500 MB for FlinkCEP (increasing to over 1 GB for $x = 256$, and not terminating after several hours), ≈ 409 MB for CORE, ≈ 3.3 MB for REmatch, and ≈ 81 MB for SuSe.

We set the summary size to the corresponding x-value, ensuring that the processed word fits entirely within the summary. Thus, we focused exclusively on StateSummary operations without



(a) (b) (c)

Fig. 11. Citi Bike: (a) relative recall improvement (higher is better), (b) throughput (higher is better), and (c) latency (lower is better).

employing summary selection, computing exact values instead of approximations. We set the time window size such that all elements fit into a single window, maximizing the number of matches to be detected. Also, we ensured that all methods produced the same result when evaluating the aggregate function. We tested with $q_\gamma = ABCD$ (the REmatch syntax being $!x\{A\}.*!y\{B\}.*!z\{C\}.*!w\{D\}$) for the required processing time of input words of varying sizes. As input, we choose words of the form $A^iB^iC^iD^i$ with $i \in \mathbb{N}^+$, e.g., $A^8B^8C^8D^8$, ensuring an increase in matches for rising x-values.

StateSummary consistently processes words at least an order of magnitude faster than REmatch, and at least three orders of magnitude faster than CORE and Flink. Particularly for words longer than 512 characters, the performance gap between StateSummary and both REmatch and CORE significantly widens, while Flink did not terminate for these values due to overload.

The performance differences induce throughput gains, as shown in **??**. The ratio indicates how much more quickly StateSummary processed a word than REmatch, CORE, or Flink. The throughput gain always exceeds one, indicating StateSummary's superiority; for longer words, this gain increases significantly.

SuSe's Overall Efficiency. In **??**, we examined how varying $|\mathcal{S}|$ (see legend) and q_τ affects the execution time, throughput, processing latency per element, and memory usage of SuSe (employing summary selection) for processing a stream of 10^5 elements. **??** shows that

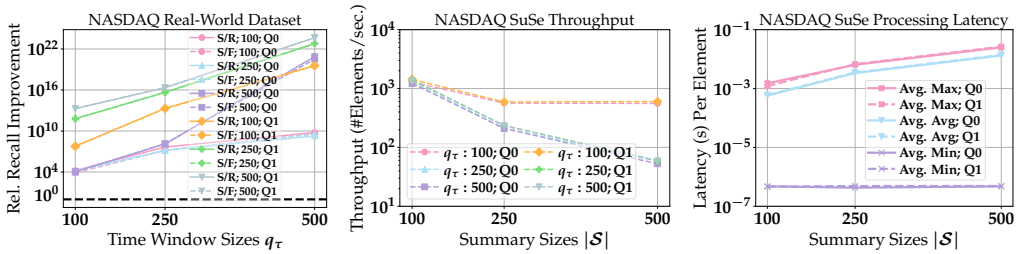
the larger $|\mathcal{S}|$ and the larger q_τ, the longer the execution time. Also, a time window size exists for all procedures up to a summary size of 500, where execution time stabilizes or slightly decreases. The reasoning being that the Remove operation is the most expensive one (see **??**), with cost $min(q_\tau, |\mathcal{S}|)$. Once either $|\mathcal{S}| > q_\tau$ or $q_\tau > |\mathcal{S}|$, the smaller value of the two parameters becomes the limiting factor for runtime, rendering the influence of the other parameters marginal. With larger q_τ, SuSe chooses better subsequences, which reduces the number of Remove operations and explains the drop in runtime for $q_\tau \geq |\mathcal{S}|$.

In **??**, we examined the resulting throughput values for a fixed $q_\tau = 500$.

For the same reasons as discussed above, there is a decline in throughput as the summary size expands.

In **??**, we investigated the resulting average min, max, and average processing latency per element per second. A boxplot shows the latencies for increasing q_τ, while the corresponding line plots denote the averages. A larger $|\mathcal{S}|$ leads to elevated processing latencies. However, the median of the average processing latencies consistently remains below $10^{-1}s$, showing streaming feasibility.

Finally, the induced memory footprint turned out to be negligible and constant for various $|\mathcal{S}|$ and q_τ, see **??**.



(a) (b) (c)

Fig. 12. NASDAQ: (a) relative recall improvement (higher is better), (b) throughput (higher is better), and (c) latency (lower is better).

7.5 Case Study: Citi Bike

Character Types and Queries. ?? denotes our results for the Citi Bike real-world dataset. The character types encompass rides on frequented routes, pinpoint brief rides at busy stations, rides at central stations, and member rides at quieter stations, giving a thorough understanding of the Citi Bike dynamics. We defined a query with $Q0_\gamma = E(H|K)^*E$, recognizing patterns where a ride starts at a busy station, followed by a combination of rides at central and quieter stations, ending with a ride at a busy station, suggesting probable areas for station enhancements or upkeep. The second query, $Q1_\gamma = (E|B|F)^+CEHF$, captures sequences of rides that start at busy stations, popular routes, or extended rides at central spots, followed by short and consecutive rides at busy stations, a trip at a central location, and concluding with a prolonged ride there.

Effectiveness. To examine the relative recall improvement, we varied $|\mathcal{S}|$ and q_τ in ??. For both queries, not all summary sizes yielded matches. Generally, there is an upward trend in the results with a larger q_τ; SuSe identifies subsequences that are between three and twelve orders of magnitude superior for $q_\tau = 500$.

Throughput and Latency. Throughput and latency of SuSe are shown in ?? and ??. For smaller values of $|\mathcal{S}|$ and q_τ (refer to legend), SuSe's throughput increases. However, with larger values for $|\mathcal{S}|$ and q_τ, the lower throughput is

attributed to the Remove operation. ?? shows the avg min, avg max, and avg latency of SuSe, with $q_\tau = 500$. An increase in summary size leads to higher latencies. Yet, the average max latency hovers around 10^{-2} seconds, while the average processing latency remains under 10^{-2} seconds for $|\mathcal{S}| = 500$, which meets real-time demands.

7.6 Case Study: NASDAQ

Character Types and Queries. ?? represents our results for the NASDAQ real-world dataset. We derived character types based on stock market activities, e.g., significant price changes, trade volumes, daily peak or lowest prices, periods of consistent behaviour or fluctuations, and market uncertainty. We formulated a query with $Q0_\gamma = A(B|G)^*A$, detecting an initial price increase, succeeded by any number of price reductions or uncertain market periods, and ending with a subsequent price surge, potentially signalling a fluctuating stock price ascent. The second query, $Q1_\gamma = A^*G(A|B)^*G^*A$, captures zero or more price rises, followed by market uncertainty, zero or more occurrences of either significant price rises or drops, a potential period of market uncertainty, and ends with a price rise.

Effectiveness. In ??, we varied $|\mathcal{S}|$ and q_τ between $100 - 500$ using the $Q0$ and $Q1$ above. Here, for all $|\mathcal{S}|$, matches were obtained. Generally, larger $|\mathcal{S}|$ and q_τ led SuSe to derive superior stream subsequences, resulting in up to 10^{23} additional matches. However, even at smaller parameter

values, SuSe's subsequences surpassed both baselines by magnitudes ranging from 10^4 to 10^{13}. Notably, for $Q1$, SuSe identified better subsequences due to the increased count of Kleene operators.

Throughput and Latency. In our examination of throughput and processing latency of SuSe, as shown in **??** and **??**, a similar trend as for Citi Bike emerges: when decreasing the summary and time window size (see legend), SuSe's throughput enhances, reaching peaks of up to 1500 elements per second. However, when the $|\mathcal{S}|$ and q_τ increase, the throughput decreases to around $70 - 80$ elements.

Turning to **??**, we fixed $q_\tau = 500$ and increased the summary size, resulting in an upward trend of the latencies. Nevertheless, while the average maximum latency approaches 10^{-2} seconds, the average processing latency for $\mathcal{S} = 500$ stays just below 10^{-2} seconds, again denoting latencies suitable for real-time computing.

Received October 2024; revised January 2025; accepted February 2025