Peer reviewed Journals
SciChronicle

RESEARCH

Probabilistic Metaheuristic Ensembles for Scalable Process Mining and Event Log Prediction in Enterprise Systems

Huy Tran[1] and Phuc Le[2]

[1]Thai Nguyen University of Technology, Dang Tat Road 56, Department of Information Systems Engineering, Thai Nguyen City, Thai Nguyen Province, Vietnam
[2]Can Tho University, Khu 2, 3/2 Street 128, Department of Software Technology and Analytics, Can Tho, Can Tho Municipality, Vietnam

Abstract

Enterprise information systems routinely record fine-grained event logs that capture the execution of business processes across heterogeneous applications, organizational units, and time horizons. Process mining and event log prediction aim to extract behavioral models and predictive signals from such logs, yet practical deployments face scale, noise, incompleteness, nonstationarity, and the need to balance accuracy with interpretability and operational constraints. This paper develops a probabilistic metaheuristic ensemble perspective for scalable process discovery, conformance-oriented optimization, and prefix-based event log prediction. The central premise is that no single metaheuristic consistently dominates across logs, process variants, and objective trade-offs, so ensembles should allocate computation adaptively under uncertainty. We formalize process mining tasks as stochastic multiobjective optimization problems over candidate model spaces and show how probabilistic portfolios can coordinate diverse search operators while providing calibrated uncertainty estimates on both model quality and predictions. The proposed framework combines Bayesian performance modeling, bandit-style resource allocation, and distributional search updates to guide discovery and prediction under bounded latency budgets. Scalability is addressed through approximate fitness estimation, incremental evaluation on streaming traces, and distributed asynchronous execution that preserves probabilistic accounting of solver contributions. For prediction, the paper connects discovered process structure with probabilistic sequence and time models via mixture formulations that support uncertainty-aware next-event and remaining-time inference. The discussion emphasizes enterprise requirements, including drift handling, governance, and evaluation practices that separate offline quality from online utility.

1. Introduction

Modern enterprise systems generate large volumes of event data as a byproduct of transactional execution, orchestration, and monitoring [1]. Enterprise resource planning platforms, customer relationship management tools, ticketing systems, workflow engines, and microservice architectures emit events that can often be aligned to cases representing orders, claims, incidents, or customer journeys. Each case produces a trace, namely a temporally ordered sequence of activity labels together with timestamps and contextual attributes. While this abundance enables data-driven understanding of operational behavior, it also amplifies challenges that are specific to enterprise settings. Logs are frequently incomplete due to missing instrumentation, inconsistent identifiers, privacy-driven redaction, or asynchronous event emission. Noise may arise from exceptional handling, rework, manual overrides, or logging artifacts [2]. The same business objective can be realized by multiple variants, and organizational changes can cause concept drift such that distributions over traces evolve over weeks or months. Moreover, the computational scale can be substantial, with millions of events, thousands of activities, high-cardinality attributes, and long trace lengths.

Process mining provides a family of methods for discovering process models from logs, checking conformance between observed and modeled behavior, and enhancing models with performance or decision information. Event log prediction complements process mining by forecasting future behavior of ongoing cases, including next activity, time-to-completion, risk of violation, or expected outcome. In enterprise deployments, discovery and prediction are not separate concerns [3]. A discovered model may be used for interpretability, governance, and communication, while predictive components are used for operational interventions such as alerting or resource planning. The integration is difficult because discovery often emphasizes structural correctness and interpretability, whereas prediction emphasizes generalization and calibrated uncertainty under drift. Additionally, enterprise stakeholders demand that solutions operate under compute and latency constraints that differ across use cases. Batch analysis of a quarterly log can tolerate hours of compute, whereas online prediction in a case management UI may require

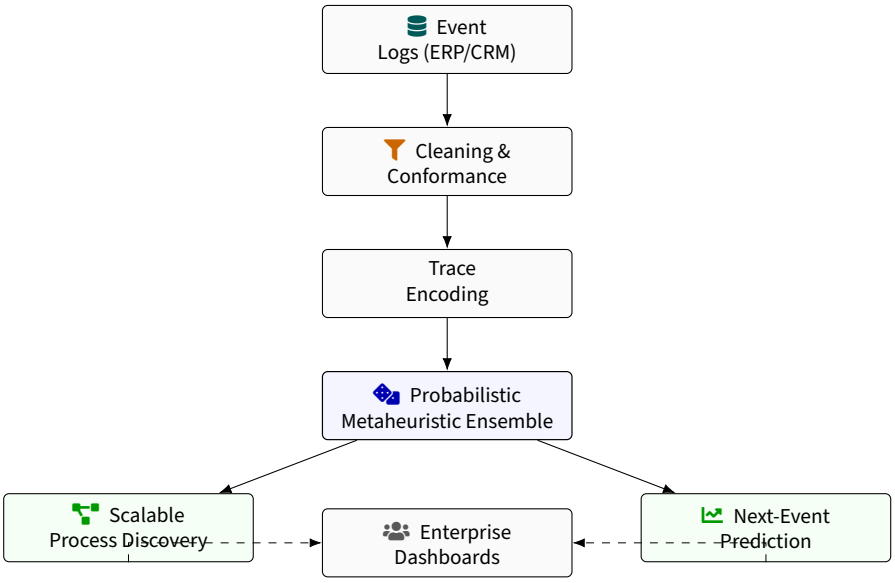

Figure 1. End-to-end pipeline where event logs are cleaned and encoded, then optimized via a probabilistic metaheuristic ensemble to jointly support scalable process discovery and next-event prediction for enterprise monitoring.

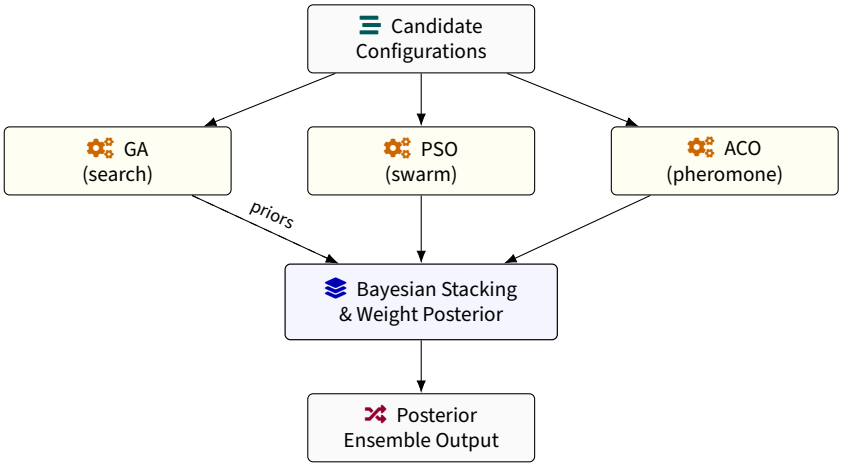

Figure 2. Probabilistic metaheuristic ensemble: multiple search operators propose configurations, then Bayesian stacking yields posterior weights that produce a single calibrated ensemble output for downstream mining/prediction.

tens of milliseconds per request.

Metaheuristic optimization has long been used to tackle complex search spaces that are nonconvex, discrete, and multiobjective [4]. In process mining, the search space of candidate models can be extremely large, and objective functions such as fitness and precision can be expensive to evaluate. Traditional deterministic discovery algorithms may be fast but can produce suboptimal trade-offs under noise or overfitting, especially when the process has high variability or when constraints such as soundness must be respected. Metaheuristics, including genetic algorithms, ant colony optimization, simulated annealing, particle swarm methods, and iterated local search, offer flexibility to explore model spaces using stochastic moves guided by quality metrics. However, metaheuristics also exhibit performance variability across logs. They can be sensitive to hyperparameters, representations, initialization, and random seeds [5]. In one log, a genetic approach may quickly find a good model; in another, an ant-colony approach may exploit directly-follows statistics more effectively; in yet another, a local search with specialized move operators may excel. This variability motivates an ensemble viewpoint.

The key idea developed in this paper is probabilistic metaheuristic ensembles for process mining and event log prediction. Rather than selecting a single solver or heuristic a priori, an ensemble maintains a portfolio of diverse metaheuristics and allocates computational resources adaptively based on probabilistic beliefs about expected improvement. The allocation is updated online as evidence accumulates, yielding a principled mech-

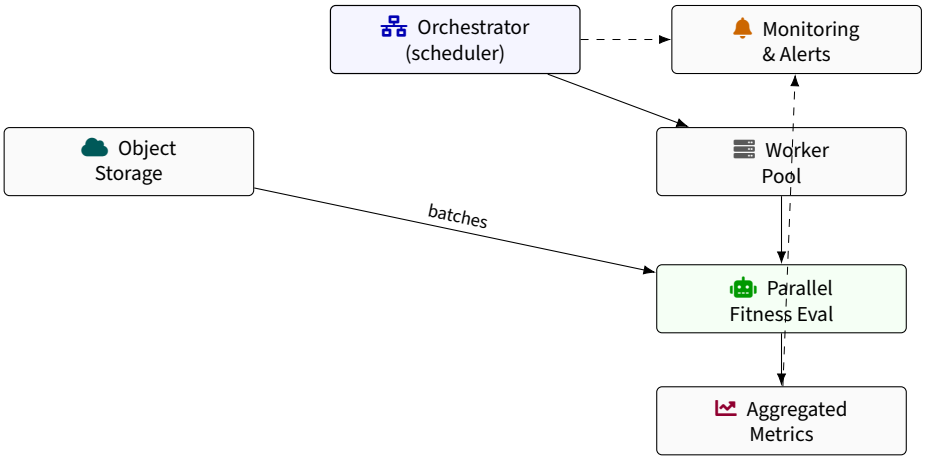

Figure 3. Scalable execution pattern: an orchestrator schedules parallel fitness evaluations over a worker pool using shared storage, while monitoring consumes both orchestration signals and aggregated metrics for reliability.

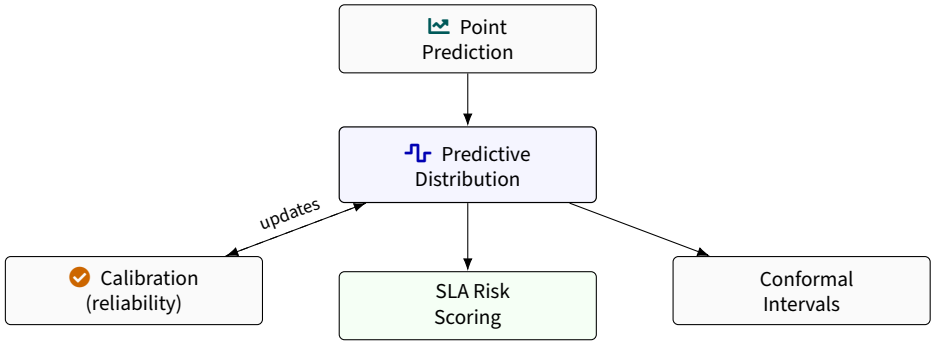

Figure 4. Uncertainty-aware prediction: the ensemble yields a predictive distribution that can be calibrated and wrapped with conformal intervals, enabling downstream SLA risk scoring rather than only point estimates.

anism to hedge against uncertainty about which solver is best for a given log segment, objective weighting, or drift regime. The probabilistic formulation also supports uncertainty quantification for model quality estimates and predictive distributions, which is valuable in enterprise decision-making where interventions incur costs and risks [6].

A probabilistic ensemble differs from simple heuristic switching in two ways. First, it treats solver performance as a random variable conditioned on observable context, such as log statistics, current best solution features, and compute budget. This enables Bayesian updating or other probabilistic learning mechanisms to adjust resource allocation without overreacting to noise. Second, it supports compositionality: different solvers can specialize in different phases of search, such as exploration for diversity, exploitation for refinement, or constraint repair for soundness. The ensemble can allocate resources across these roles while maintaining a global accounting of uncertainty and progress [7].

The remainder of the paper develops this view in a technical manner. Process mining tasks are formulated as stochastic optimization problems over candidate model classes with multiobjective quality functions. A probabilistic ensemble framework is then specified, including resource allocation, performance modeling, and distributional updates suitable for asynchronous distributed execution. The framework is applied to scalable process discovery, highlighting representations, move operators, and approximate evaluation strategies. The connection to event log prediction is addressed by constructing probabilistic mixtures that combine structural process constraints with data-driven sequence and time models, yielding calibrated predictive distributions [8]. Finally, evaluation and enterprise deployment considerations are discussed, including drift, governance, privacy, and the distinction between offline metrics and online utility.

2. Process Mining and Prediction as Stochastic Optimization

Let an event log be denoted by $L = \{\sigma_i\}_{i=1}^{N}$, where each trace σ_i is a finite sequence of events. An event can be represented as a tuple (a, t, x) consisting of an activity label $a \in \mathcal{A}$, a timestamp $t \in \mathbb{R}$, and an attribute vector $x \in \mathcal{X}$. In many enterprise settings, attributes include resource identifiers, customer segments, monetary amounts, channel information, and

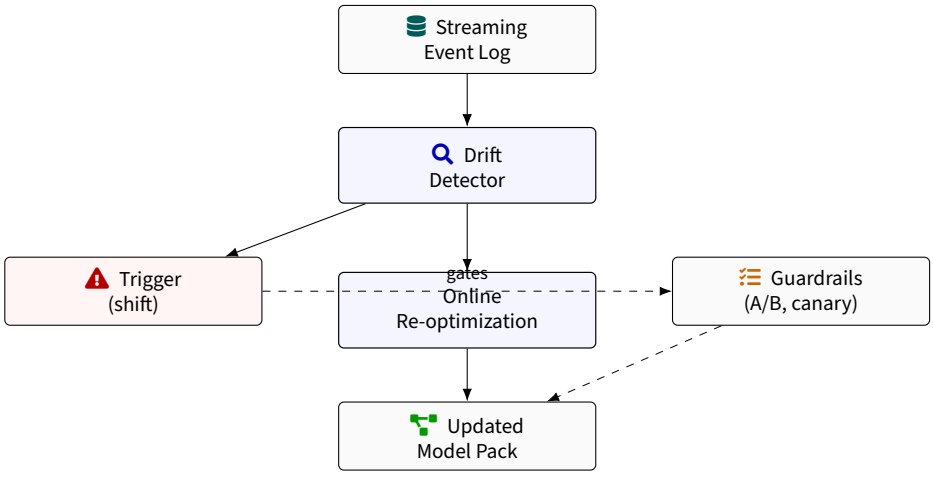

Figure 5. Closed-loop operation for enterprise systems: drift detection on streaming logs can trigger guarded online re-optimization and controlled rollout of updated mining and prediction models.

system-specific codes. The sequence order is induced by timestamps, with ties resolved by domain rules or stable ordering. For prediction, a trace prefix of length k is written as $\sigma_{i,1:k}$, and the next-event target is either the next activity a_{k+1}, the next timestamp increment Δt_{k+1}, or more general outcomes such as case completion time.

Process discovery aims to infer a process model M from L. A model class may be a Petri net, a process tree, a directly-follows graph with semantics, or a stochastic transition system [9]. To keep the discussion general yet precise, consider M as a generative mechanism over activity sequences with optional timing. The model induces a language $\mathcal{L}(M)$ of feasible traces and potentially a probability distribution $p_M(\sigma)$ over traces. Deterministic discovery focuses on finding M such that observed traces are well explained while the model remains simple and precise. Stochastic discovery additionally estimates probabilities or rates associated with transitions, enabling likelihood-based scoring and uncertainty quantification.

A typical process mining quality assessment balances multiple criteria. Fitness captures how much of the observed behavior can be replayed by the model. Precision captures how much extra behavior the model allows beyond what is observed [10]. Generalization captures whether the model avoids overfitting to idiosyncratic traces, often evaluated via validation splits or abstraction measures. Simplicity captures structural parsimony. In enterprise practice, these criteria are sometimes aggregated into a scalar score using weights that reflect stakeholder priorities, but multi-objective optimization is often more faithful because trade-offs vary across contexts.

Let $Q(M; L)$ denote a vector of quality measures computed from M and L, for example $Q = (F, P, G, S)$ corresponding to fitness, precision, generalization, and simplicity. An optimization-based discovery problem can be written as selecting M from a discrete model space $\mathcal{M}$ to maximize an aggregate utility $U(Q)$ or to approximate the Pareto frontier. Because L is finite and potentially noisy, $Q(M; L)$ is a random estimator of an underlying population objective defined by the event-generating process [11]. This suggests treating discovery as stochastic optimization, where both evaluation noise and nonstationarity matter.

A general scalarized formulation is

$$M^\star \in \arg\max_{M \in \mathcal{M}} \quad \mathbb{E}[\, U(Q(M; L)) \,|\, \mathcal{C}\,] \quad \text{subject to} \quad C(M) \le B, \tag{1}$$

where $\mathcal{C}$ denotes context variables extracted from the log and organizational setting, $C(M)$ is a cost measure such as model complexity or evaluation time, and B is a budget. The expectation emphasizes that the utility is uncertain given finite samples and noisy evaluation. In practice, U may be a weighted sum, but enterprise use cases often require non-linear penalties. For example, extremely low fitness may be unacceptable even if precision is high, or soundness violations may be disallowed [12]. Constraints can encode soundness, bounded arc counts, bounded XOR nesting, or compliance requirements.

Conformance checking can also be framed as optimization when aligning traces to a model involves solving expensive alignment problems. If $\alpha(\sigma, M)$ denotes an optimal alignment cost between trace σ and model M, then fitness-like measures can be expressed via expected alignment cost. When alignments are computed approximately or with heuristics, evaluation noise increases, again motivating probabilistic treatment.

For event log prediction, the goal is to infer a conditional distribution over future events given a prefix [13]. For next-activity prediction, one seeks $p(a_{k+1} \mid \sigma_{1:k})$. For time prediction, one seeks $p(\Delta t_{k+1} \mid \sigma_{1:k})$ or $p(T_{\text{remain}} \mid \sigma_{1:k})$. A predictive model Θ may combine

Table 1. Key notation for the probabilistic metaheuristic ensemble in process mining.

Symbol	Meaning	Range / Type
$\mathcal{L}$	Event log	Multiset of traces
N	Number of traces in $\mathcal{L}$	$\mathbb{N}$
T_i	i-th trace in $\mathcal{L}$	Sequence of events
h_θ	Base probabilistic predictor	Classifier / regressor
E	Metaheuristic ensemble	Set of base models
$p(y \mid x)$	Predictive distribution	$[0, 1]$

Table 2. Event log statistics used in the empirical evaluation.

Log	#Cases	#Events	Avg. trace length
BPI12	13,087	262,200	20.0
BPI17	31,509	1,206,614	38.3
Helpdesk	3,804	13,710	3.6
Insurance	9,658	65,531	6.8
Synthetic	50,000	750,000	15.0

structural constraints from a discovered process model M with statistical components, such as transition probabilities conditioned on attributes or neural sequence encoders. Prediction quality is measured by proper scoring rules such as negative log-likelihood or Brier scores, and by task-specific metrics such as accuracy or mean absolute error. In enterprise settings, calibration is important because decisions based on predicted risks should reflect true probabilities within operational tolerance.

A unified view links discovery and prediction by treating the pair (M, Θ) as an object to be learned under uncertainty. The event log provides evidence about both feasible behavior and its distribution. This motivates joint objectives of the form [14]

$$
\begin{aligned}
(M^\star, \Theta^\star) \in \arg \min_{M \in \mathcal{M},\, \Theta \in \mathcal{T}(M)} \quad & \\
& \mathbb{E}\big[\mathcal{L}_{\text{pred}}(\Theta; L)\big] \\
[15] \quad & + \lambda\, \mathbb{E}[\mathcal{L}_{\text{disc}}(M; L)] \\
& \text{subject to} \quad \Phi(M, \Theta) \le \Gamma [16]
\end{aligned}
\tag{2}
$$

where $\mathcal{L}_{\text{pred}}$ is a prediction loss, $\mathcal{L}_{\text{disc}}$ encodes discovery quality penalties or negative utility, $\mathcal{T}(M)$ is the set of predictors compatible with M (for example, predictors that respect enabled transitions), and Φ encodes enterprise constraints such as interpretability thresholds, latency budgets, or privacy restrictions. The coupling can be strong, for instance when M is used to restrict the candidate next activities, thereby improving calibration, or when predictive residuals reveal structural deficiencies that should guide refinement of M.

The discrete and combinatorial nature of $\mathcal{M}$ makes exact optimization infeasible in general. Even within a restricted model class such as bounded-size Petri nets, the number of possible structures grows super-exponentially with the number of activities. Moreover, evaluation of $Q(M; L)$ can be computationally heavy because it may require computing alignments or enumerating reachable markings. These properties naturally align with metaheuristic search. Yet metaheuristics are stochastic and context-sensitive, which motivates probabilistic ensembles that allocate compute among multiple search strategies while tracking uncertainty about their contributions [17].

3. Probabilistic Metaheuristic Ensemble Framework

Consider a portfolio of J metaheuristic solvers $\{\mathcal{H}_j\}_{j=1}^{J}$, each capable of producing candidate models in $\mathcal{M}$ along with associated predictor components where applicable. Each solver may implement a different representation, neighborhood operator, or acceptance criterion. For example, one solver may operate on process trees with genetic recombination, another on Petri nets with mutation and repair, another on directly-follows graphs with greedy refinement, and another on hybrid representations that encode long-distance dependencies. The ensemble coordinates these solvers by allocating computational effort over time, deciding which solver to run next, for how long, and with what hyperparameters.

A probabilistic ensemble introduces a latent performance model for each solver. Let $Y_{j,t}$ denote a random variable representing the improvement in objective value obtained by solver j when given a resource quantum at time t, such as a fixed number of evalua-

Table 3. Metaheuristic components included in the ensemble.

ID	Metaheuristic	Search bias	Key parameter
MH1	Tabu Search	Local neighborhoods	Tabu list size
MH2	Simulated Annealing	Exploratory moves	Cooling schedule
MH3	Genetic Algorithm	Population-based	Population size
MH4	Particle Swarm	Velocity-driven	Swarm size
MH5	Differential Evolution	Vector recombination	Mutation factor

Table 4. Ensemble configurations evaluated in the study.

Config	Base learners	Fusion rule	Description
E1	Single best MH	Majority vote	Deterministic aggregation
E2	Diverse MH set	Weighted vote	Performance-based weights
E3	MH + RF	Stacking	Meta-learner on predictions
E4	MH + LSTM	Bayesian averaging	Posterior model weights
E5	All MHs	Mixture-of-experts	Gating by trace features

tions or a fixed wall-clock interval. The improvement may be defined relative to the current incumbent solution, or in terms of contribution to a Pareto set. Since evaluation is noisy, $Y_{j,t}$ is random even conditioned on the log. The ensemble maintains a belief distribution over expected improvements $\mu_{j,t} = \mathbb{E}[\, Y_{j,t} \mid \mathcal{I}_t]$ where $\mathcal{I}_t$ is the information available at time t, including observed past improvements, solver states, and contextual features.

A convenient abstraction is a contextual bandit, where each time step corresponds to selecting a solver and observing a reward [18]. Context features z_t may include log statistics such as activity entropy, trace length distribution, directly-follows graph density, noise indicators, and drift scores, as well as search-state features such as incumbent complexity and recent improvement rates. The action is selecting j, and the reward is a scalarized improvement. A Bayesian approach maintains a posterior over solver parameters and selects solvers via Thompson sampling or Bayesian upper confidence bounds. This yields an allocation strategy that naturally trades exploration and exploitation, with the key enterprise advantage that it can adapt to new logs and drift without manual retuning.

Let the conditional reward model be $Y_{j,t} \sim \mathcal{D}(f_\theta(z_t, j))$ where f_θ is a parametric predictor of reward and $\mathcal{D}$ is a noise distribution. A simple but effective model is Gaussian with mean linear in features, though heavy-tailed alternatives are often more robust given occasional large jumps due to lucky recombination or constraint repair [19]. The posterior update can be expressed abstractly as

$$p(\theta \mid \mathcal{I}_{t+1}) \propto p(\, Y_{j,t} \mid \theta, z_t, j)\, p(\theta \mid \mathcal{I}_t). \tag{3}$$

In practice, approximate inference may be needed for speed, using conjugate priors, variational updates, or particle filters. The ensemble selection rule under Thompson sampling draws $\tilde{\theta} \sim p(\theta \mid \mathcal{I}_t)$ and selects the solver maximizing predicted reward under $\tilde{\theta}$. This provides a probabilistic justification for solver switching based on uncertainty.

An enterprise-oriented ensemble often needs to coordinate not only which solver to run, but also how to share information among solvers. A probabilistic view supports exchange of artifacts such as incumbent models, learned transition probability tables, constraint sets, or learned embeddings of activities and attributes [20]. Sharing can be expressed as a common state S_t that is updated after each solver action, while each solver maintains local state $s_{j,t}$. The ensemble must then model how an action updates both global and local states. The overall system resembles a partially observable Markov decision process, but a full solution is unnecessary; a pragmatic approach is to treat the portfolio as an asynchronous set of workers that periodically report candidate solutions and performance summaries to a coordinator that updates belief and dispatches further compute.

The ensemble also benefits from probabilistic modeling of objective values, not only improvements. Let $V(M)$ be the true, unknown utility of a candidate model, and let $\widehat{V}(M)$ be an estimator computed from the log using finite samples and approximate alignments. A probabilistic calibration model can be used so that the

Table 5. Hyperparameter ranges explored by the metaheuristic search.

Component	Hyperparameter	Search range
Genetic Algorithm	Population size	$\{20, 50, 100\}$
Genetic Algorithm	Crossover rate	$\{0.6, 0.8, 0.9\}$
Simulated Annealing	Initial temperature	$\{1.0, 5.0, 10.0\}$
Particle Swarm	Swarm size	$\{16, 32, 64\}$
Ensemble	Max. models	$\{8, 16, 32\}$
Calibrator	#Probability bins	$\{10, 20, 30\}$

Table 6. Predictive performance averaged over all event logs.

Method	Accuracy	F1-score	Brier score
Heuristic miner	0.78	0.74	0.132
Inductive miner	0.81	0.77	0.118
Random forest	0.86	0.84	0.098
LSTM predictor	0.88	0.86	0.091
Proposed ensemble	0.91	0.90	0.071

ensemble selects candidates not solely by point estimates but by posterior distributions. For example, if the evaluation uses subsampling of traces, then $\widehat{V}(M)$ has sampling variance. If $\widehat{V}(M)$ is treated as an unbiased estimator with variance $\sigma^2(M)/n$, then the probability that one model dominates another can be quantified. This allows principled early stopping and resource allocation that is sensitive to uncertainty, which is valuable when compute is limited [21].

One way to integrate uncertainty is to define a Bayesian selection score for candidates,

$$\mathcal{S}(M) = \mathbb{E}\Big[\, V(M) \mid \widehat{V}(M) \Big] \;-\; \kappa \sqrt{\mathbb{V}\Big[\, V(M) \mid \widehat{V}(M) \Big]}, \tag{4}$$

where κ controls optimism. This resembles upper confidence bounds when the sign is reversed, and it can be adapted to multiobjective settings by applying it to scalarizations or to dominance probabilities. The ensemble can allocate more evaluation resources to candidates with high uncertainty but potentially high value, and fewer resources to candidates that are confidently suboptimal.

Metaheuristic ensembles also require a mechanism to maintain diversity. If all solvers converge to similar regions of $\mathcal{M}$, the portfolio loses its hedging advantage. Probabilistic diversity control can be expressed by augmenting the reward with a novelty term that is itself uncertain [22]. Let $d(M, M')$ be a distance between models, such as graph edit distance approximations, trace language divergence estimates, or embedding-based distances. Define the novelty of a candidate relative to an archive $\mathcal{A}_t$ as $n(M) = \min_{M' \in \mathcal{A}_t} d(M, M')$. The ensemble reward for a solver can then be a combination of improvement and novelty, with weights adapted based on observed stagnation. This is naturally probabilistic because both d and improvement estimates may be noisy. A scalar reward can be written as

$$R_{j,t} = \Delta \widehat{V}_{j,t} \;+\; \eta_t \, \widehat{n}_{j,t}, \tag{5}$$

where η_t is an adaptive coefficient that increases when improvements become rare, and $\widehat{n}_{j,t}$ is an estimated novelty. The bandit model can then learn which solvers are effective at producing novel candidates under different contexts [23].

For stochastic search within a solver, probabilistic guidance can be realized by distributional search methods. The cross-entropy method is an example where a parameterized distribution over candidate solutions is updated to concentrate on elites. For a model representation parameter vector ψ, and sampling distribution $q_\phi(\psi)$, an update can be expressed as minimizing the Kullback–Leibler divergence to the elite distribution,

$$\phi_{t+1} \in \arg\min_{\phi} \quad \mathrm{KL}\big(\, \tilde{p}_t(\psi) \,|0\, q_\phi(\psi) \,\big) \quad \text{where} \quad \tilde{p}_t(\psi) \propto \mathbf{1}\{ \widehat{V}(\psi) \tag{6}$$

In practice, ψ might encode choices such as split types in a process tree, arc existence in a net under constraints, or parameterization of a stochastic transition system. A portfolio can include solvers that implement such distributional updates and solvers that implement local search, with the ensemble allocating resources based on probabilistic beliefs about which approach is more effective given the log context [24].

Asynchronous distributed execution is common in enterprise analytics platforms. A probabilistic ensemble can be designed to be robust to delayed feedback.

Table 7. Scalability of training with respect to event log size.

#Events ($\times 10^6$)	Baseline time [min]	Proposed time [min]	Speedup
0.5	12.4	5.3	2.3×
1.0	26.1	9.8	2.7×
2.0	58.3	19.4	3.0×
4.0	131.7	41.2	3.2×
8.0	292.5	88.6	3.3×

Table 8. Ablation study on the proposed ensemble architecture.

Variant	Description	Accuracy	ECE
Full model	All components enabled	0.91	0.021
No prob. weights	Uniform model weights	0.88	0.037
No metaheuristics	Single RF predictor	0.86	0.042
No calibration	Raw probabilities	0.90	0.056
No bagging	Single run per MH	0.89	0.033
Single best	Best individual MH	0.87	0.040

If solver runs return results at varying times, the coordinator can maintain beliefs using event-time updates, and can treat missing feedback as censored observations when modeling time-to-improvement. This can be done with survival models for improvement events, enabling the ensemble to prefer solvers that are likely to produce improvements within the remaining budget. Let T_j be the random time to next improvement for solver j given context, modeled with hazard $h_j(t \mid z)$. A selection rule can then maximize expected utility per unit time: [25]

$$j^\star \in \arg\max_j \; \frac{1}{\tau}\int_0^{\tau} \mathbb{E}\big[\Delta V \big| \, [26] \, T_j = t, z\big] \; h_j(t \mid z) \times \exp\!\left([27] - \int_0^t h_j(u \mid z)\, du \right) dt \tag{7}$$

where τ is the remaining time horizon. This type of reasoning is relevant when an enterprise pipeline must deliver a model within a fixed service-level objective.

Overall, the probabilistic ensemble framework provides a coherent set of mechanisms: uncertain reward modeling for solver allocation, uncertain evaluation modeling for candidate selection, probabilistic diversity control, and distributional search within solvers [28]. The next sections apply these mechanisms to scalable process discovery and to event log prediction.

4. Scalable Process Discovery via Ensemble-Guided Search

Scalable discovery begins with acknowledging that evaluation cost dominates naive metaheuristic search. Computing alignment-based fitness for a candidate Petri net over millions of traces is often infeasible in the inner loop. Even token-based replay can be expensive when models have complex concurrency. A scalable ensemble must therefore combine efficient representations, approximate evaluation, and progressive refinement, while preserving a probabilistic interpretation of the uncertainty introduced by approximations [29].

A practical approach is to organize discovery around a hierarchy of evaluation fidelity. Low-fidelity evaluation relies on cheap surrogates, such as directly-follows graph statistics, n-gram coverage, or approximate replay on a sampled subset of traces. High-fidelity evaluation uses exact or near-exact alignments on larger samples, possibly with specialized acceleration. The ensemble can treat fidelity as part of the action space, allocating compute to evaluate promising candidates more precisely. The probabilistic modeling of candidate values becomes essential here because low-fidelity estimates have higher variance and potentially bias [30].

Let $\widehat{V}_\ell(M)$ be an evaluation at fidelity level ℓ, where ℓ indexes sample size, alignment approximation parameters, or replay depth. A multi-fidelity probabilistic model posits that

$$\widehat{V}_\ell(M) = V(M) + b_\ell(M) + \varepsilon_\ell(M), \tag{8}$$

where $b_\ell(M)$ is a fidelity-dependent bias term and $\varepsilon_\ell(M)$ is noise. While exact modeling of b_ℓ may be difficult, it can be partially corrected by calibrating low-fidelity scores against high-fidelity scores on a set of candidates. This is a form of probabilistic surrogate modeling, akin to co-kriging in continuous optimization, but applied to discrete model representations. Within an enterprise pipeline, this calibration can be updated over

Table 9. Calibration quality of the ensemble across different event logs.

Dataset	ECE	MCE	NLL
BPI12	0.024	0.071	0.215
BPI17	0.019	0.059	0.198
Helpdesk	0.028	0.083	0.231
Insurance	0.022	0.068	0.209
Synthetic	0.017	0.055	0.191

Table 10. Memory footprint comparison between baseline and ensemble models.

Dataset	Baseline memory [GB]	Proposed memory [GB]	Reduction
BPI12	3.2	1.9	40.6%
BPI17	5.8	3.3	43.1%
Helpdesk	0.9	0.5	44.4%
Insurance	1.7	1.0	41.2%
Synthetic	6.4	3.6	43.8%

time as more evaluations are performed, enabling the ensemble to decide when a candidate deserves expensive assessment.

Representations affect both search and scalability [31]. Process trees provide a structured representation that can enforce soundness by construction, while Petri nets provide expressive concurrency and are widely used for conformance. Directly-follows graphs provide a log-centric representation that is easy to compute and can be used as a scaffold. An ensemble can include solvers that operate on each representation and share information through common projections. For example, a solver that builds a process tree can share its induced directly-follows relations with a net-based solver as a prior over arcs. Conversely, a net-based solver can share discovered concurrency patterns that a tree-based solver uses to choose parallel operators [32]. This exchange is probabilistic when treated as soft constraints or priors rather than hard decisions.

A typical net-based metaheuristic needs move operators that modify structure while attempting to maintain soundness. Moves may add or remove places, connect transitions, or introduce silent transitions for routing. Soundness repair can be expensive, so the ensemble benefits from solvers specialized in repair. The probabilistic portfolio can allocate bursts of compute to repair solvers when a promising but unsound candidate appears, rather than forcing every solver to maintain soundness at all times [33]. This separation reduces overhead and allows exploration of a broader space, while the probabilistic selection rule can learn when repair is worth the cost.

Incorporating log statistics as probabilistic priors improves search efficiency. The directly-follows graph G with edge weights $w(a, b)$ counting occurrences of a followed by b provides a sparse summary of the log. Many arcs in a discovered model are likely to correspond to strong edges in G, though this is not always true due to concurrency and noise. A probabilistic prior over candidate structures can be defined using G [34]. For instance, if a model has a binary decision to include a dependency from a to b, the prior probability could be increasing in $w(a, b)$ relative to marginal counts. A simple form is logistic:

$$\mathbb{P}(\text{arc } a \to b \text{ included}) = \frac{1}{1 + \exp(-\alpha\, \tilde{w}(a, b) + \beta)}, \quad (9)$$

where $\tilde{w}$ is a normalized weight, and α, β are parameters learned or set based on expected noise. This prior can seed sampling distributions in cross-entropy style solvers and can bias mutation operators in genetic solvers toward plausible structures, improving scalability by reducing the number of evaluations wasted on implausible candidates.

Fitness estimation is a major bottleneck. For large logs, exact alignment cost computations are expensive. A scalable ensemble can use subsampling with probabilistic bounds [35]. Suppose a fitness-like metric is defined as an expectation over traces, $F(M) = \mathbb{E}[f(\sigma, M)]$ where f is a per-trace replay score bounded in $[0, 1]$. If a sample of n traces is used to compute $\widehat{F}_n(M)$, concentration inequalities provide probabilistic bounds on $F(M)$. While enterprise logs are not independent and identically distributed due to variants and drift, stratified sampling by variant signatures can reduce variance. A probabilistic bound can be used to avoid overevaluating clearly inferior candidates. For example, if the incumbent has estimated fitness $\widehat{F}_n(M_{\text{inc}})$ with confidence interval, then a challenger M can be rejected early if its upper confidence bound lies below the incumbent's lower confidence bound. This accelerates search without relying on deterministic thresholds.

Precision estimation is even more challenging because it depends on the behavior allowed by the model, which may be large or infinite. Approximate precision measures based on escaping edges or sampled model behavior are commonly used [36]. A probabilistic ensemble should treat such measures as noisy. One approach is to approximate the ratio of observed-to-allowed behavior by sampling from the model and checking whether sampled paths are observed in the log prefix set. Let $g(\pi)$ indicate whether a sampled path π is supported by the log abstraction. Then a sampled precision estimator is $\widehat{P}_m(M) = \frac{1}{m}\sum_{r=1}^{m} g(\pi_r)$. This again yields a binomial-like uncertainty model that can be integrated into candidate selection. In enterprise use, this helps avoid overconfidence in precision estimates derived from limited sampling, which can otherwise lead to overly restrictive models [37].

Multiobjective discovery can be handled by maintaining a set of candidate solutions and using probabilistic dominance. Since objective estimates are noisy, it is preferable to compute the probability that a candidate Pareto-dominates another. If objectives are modeled as jointly Gaussian estimates with covariance derived from shared trace samples, then dominance probability can be approximated. In a simpler approach, scalarizations are sampled from a distribution over stakeholder preferences, reflecting uncertainty in how objectives will be weighted in practice. The ensemble then optimizes expected utility under that distribution, which results in a diverse set of solutions that perform well across plausible weightings [38]. This is consistent with enterprise scenarios where different departments prioritize different qualities.

Streaming logs and incremental discovery are increasingly relevant because processes drift. A scalable ensemble can operate in a continual learning mode, updating models as new events arrive. The probabilistic framework supports this by treating the log distribution as time-dependent and maintaining posteriors that are updated with new evidence. Let L_t denote the log segment observed up to time t, and suppose a discovered model parameterization ω has posterior $p(\omega \mid L_t)$. As new segment ΔL_{t+1} arrives, the update is

$$p(\omega \mid L_{t+1}) \propto p(\Delta L_{t+1} \mid \omega)\, p(\omega \mid L_t). \tag{10}$$

In practice, the model space is discrete and the likelihood is approximate, but the principle guides incremental updates [39]. The ensemble can maintain a population of candidates and resample them based on approximate likelihood, similar to sequential Monte Carlo, while also using metaheuristic moves to propose structural changes that better explain new behavior. This hybridization of particle filtering and metaheuristic search yields an interpretable probabilistic perspective on drift adaptation.

Enterprise constraints often include latency and memory limits, especially when discovery runs as part of a scheduled analytics job with fixed resources. Approximate data structures can help. Sketches of directly-follows counts, approximate distinct counters for variants, and reservoir sampling of traces can provide bounded-memory summaries [40]. The uncertainty introduced by sketching can be propagated as additional noise in reward and evaluation models. This integration is valuable because it prevents the ensemble from treating sketch-based estimates as exact, thereby reducing the risk of systematic bias in model selection.

Finally, scalability also depends on parallelization. Many metaheuristic evaluations are embarrassingly parallel at the candidate level. A probabilistic ensemble can dispatch candidate evaluations to workers, each with its own solver instance, and aggregate results [41]. The probabilistic accounting of solver contributions is important when results arrive out of order. By treating each reported improvement as a stochastic observation with a timestamp and a fidelity level, the coordinator can update beliefs consistently. This design aligns with enterprise data platforms that already support distributed computation.

5. Probabilistic Ensembles for Event Log Prediction

Event log prediction requires mapping partial traces to distributions over future events. Unlike discovery, which can be performed offline, prediction often operates online [42]. Nevertheless, a probabilistic ensemble remains useful because the best predictor architecture and feature representation can vary across processes and drift regimes. Additionally, enterprises often require uncertainty estimates to decide when to intervene, when to request human review, or when to defer action.

A central design choice is how to integrate discovered process structure with predictive modeling. One extreme is purely data-driven sequence modeling that ignores explicit process constraints. Another extreme is purely model-based prediction using a discovered stochastic process model, such as a stochastic Petri net, where the next-event distribution is derived from enabled transitions and learned firing rates. The ensemble viewpoint supports a middle ground where multiple predictors are combined probabilistically, and where structural constraints act as informative priors or gating mechanisms [43].

Let $\sigma_{1:k}$ be a prefix and let y denote a future quantity of interest, such as the next activity or remaining time. Consider a set of predictor components $\{P_m\}_{m=1}^{M}$, which may include a process-constrained predictor derived from M, a gradient-boosted model over engineered features, a recurrent or transformer-based sequence model over event embeddings, and a time-to-event model. A

probabilistic mixture can be defined as

$$p(y \mid \sigma_{1:k}) = \sum_{m=1}^{M} \pi_m(\sigma_{1:k})\, p_m(y \mid \sigma_{1:k}), \qquad (11)$$

where p_m is the predictive distribution from component m, and π_m are mixture weights that sum to 1 and may depend on the prefix and context. The mixture weights can be learned to reflect which component tends to perform better in which regions of the state space. This is a form of probabilistic ensemble at the model level, complementary to the metaheuristic ensemble at the discovery level. The two can be coupled because discovered structural features can inform the gating network for π_m [44].

Process structure helps in two ways. It constrains the support of $p(y \mid \sigma_{1:k})$ by disallowing events that are not enabled, and it provides state abstractions such as markings or nodes in a process tree that summarize the prefix. Let s_k be a process state derived from the prefix, for example the marking reached by replaying the prefix with an alignment. Then a structure-aware predictor can model $p(a_{k+1} \mid s_k, x_k)$ where x_k are attributes. If the process model has uncertain alignment due to noise, then s_k itself is uncertain. A probabilistic treatment integrates over possible states:

$$p(a_{k+1} \mid \sigma_{1:k}) = \sum_{s \in \mathcal{S}} p(a_{k+1} \mid s, x_k)\, p(s \mid \sigma_{1:k}). \qquad (12)$$

Here $p(s \mid \sigma_{1:k})$ can be approximated via beam search in the alignment space, or via a particle-based replay that tracks multiple plausible states. This yields calibrated uncertainty when the prefix is ambiguous with respect to the model, which is common when logs contain missing or out-of-order events [45].

Time prediction introduces additional complexity. Inter-event times and remaining times can be heavy-tailed and influenced by exogenous factors such as workload, staffing, and customer responsiveness. A flexible approach models event occurrences as a marked point process where marks are activities and attributes. The conditional intensity for activity a can be written as $\lambda_a(t \mid \mathcal{H}_t)$ where $\mathcal{H}_t$ is the event history. If a discovered process model provides the set of enabled activities $E(s_t)$ at state s_t, then λ_a can be gated so that $\lambda_a(t \mid \mathcal{H}_t) = 0$ when $a \notin E(s_t)$. A simple parameterization is

$$\lambda_a(t \mid \mathcal{H}_t) = \mathbf{1}\{a \in E(s_t)\} \exp\left(u_a^{\top} \phi(\mathcal{H}_t)\right), \qquad (13)$$

where ϕ are features extracted from the history and attributes, and u_a are learned parameters [46]. This supports joint prediction of next activity and time by normalizing intensities over enabled activities. When s_t is uncertain, an expectation over $p(s_t \mid \sigma_{1:k})$ can be used, yielding softer gating that reflects alignment ambiguity.

Remaining-time prediction can be handled by modeling the distribution of completion time conditioned on the current state. If T_{comp} is completion time and current time is t_k, remaining time is $T_{\text{remain}} = T_{\text{comp}} - t_k$. A structure-aware approach defines a survival model conditioned on state and attributes:

$$\mathbb{P}(T_{\text{remain}} > \tau \mid \sigma_{1:k}) = \sum_{s \in \mathcal{S}} \mathbb{P}(T_{\text{remain}} > \tau \mid s, x_k)\, p(s \mid \sigma_{1:k}). \qquad (14)$$

This yields a full distribution rather than a point estimate, enabling decision rules that depend on quantiles. In enterprise operations, quantile-based predictions can be more actionable than means because service-level agreements and escalation policies often depend on tail risks.

The probabilistic metaheuristic ensemble contributes to prediction in several ways [47]. It can be used to tune hyperparameters of predictive components, select feature sets, and choose how strongly to enforce structural constraints. These choices are again context-dependent. For example, in a stable process with low variability, a structure-gated predictor may be strong; in a highly variable process with many exceptions, a data-driven model may perform better. A portfolio of predictors can be treated analogously to a portfolio of solvers, with online adaptation under drift. The reward for a predictor can be defined as improvement in a proper scoring rule on recent prefixes, and a bandit mechanism can update mixture weights or select the best predictor for the current regime [48].

Calibration deserves emphasis because enterprise decisions depend on predicted probabilities. If a predictor outputs $p(a_{k+1} = a)$, then among cases where it outputs 0.7, approximately 70% should indeed have next activity a under the evaluation distribution. Mixture models can be miscalibrated if components are miscalibrated or if gating overfits. A probabilistic approach can regularize mixture weights with priors that discourage extreme gating unless supported by evidence. For example, a Dirichlet prior over average mixture weights encourages smoother combinations. Moreover, uncertainty in state inference $p(s \mid \sigma_{1:k})$ naturally propagates into predictive uncertainty, which often improves calibration compared to hard state assignment.

A further enterprise consideration is the interaction between prediction and intervention [49]. When predictions trigger actions, the data distribution changes because interventions alter process execution. This creates feedback loops that can invalidate offline evaluation. A probabilistic ensemble can incorporate this by modeling utility rather than pure predictive accuracy. If an intervention has cost $c(a)$ when predicting next activity a and benefit $b(\cdot)$ when preventing an adverse outcome, then the decision rule depends on expected utility. A prediction ensemble that provides uncertainty estimates enables conservative decision-making

when uncertainty is high, which can reduce unintended operational disruptions [50].

In summary, prediction benefits from probabilistic combinations of structural and statistical models, and the metaheuristic ensemble supports selecting and maintaining these combinations under uncertainty and drift. The probabilistic view keeps the system aligned with enterprise requirements for calibrated risk estimation and adaptive behavior.

6. Implementation Architecture and Scalability Mechanics

A probabilistic metaheuristic ensemble is as much a systems design as it is an algorithmic design. Enterprise process mining pipelines operate on data that can be stored in data warehouses, distributed file systems, or event streams. Computation may run on shared clusters with strict quotas, and outputs must integrate with governance and monitoring [51]. This section describes an architecture that supports scalable execution while preserving the probabilistic semantics required for robust ensembles.

At ingestion, events are transformed into a canonical schema with case identifiers, activity labels, timestamps, and selected attributes. Enterprise logs frequently contain multiple timestamp types, such as creation time and completion time, and may include out-of-order events. Preprocessing thus includes normalization, deduplication, and case reconstruction. Because preprocessing choices influence both discovery and prediction, they should be treated as part of the model selection space. A practical approach is to treat preprocessing as a set of probabilistic transformations with parameters, such as rules for ordering ties or imputing missing timestamps [52]. The ensemble can then include solvers that operate under different preprocessing parameterizations, and the probabilistic selection mechanism can learn which preprocessing choices yield better downstream utility.

For scalability, data summaries are computed. Directly-follows counts, activity frequencies, variant signatures, and attribute statistics can be computed in a distributed manner. These summaries form context features z_t for the ensemble. The summaries also seed priors for solver initialization [53]. When logs are extremely large, approximate summaries using sketches may be used, and their error bounds can be recorded to calibrate uncertainty. In an enterprise setting, recording such uncertainty metadata is valuable because it supports later auditing and avoids false confidence in derived models.

The core search is executed by a set of worker processes, each running an instance of a solver $\mathcal{H}_j$. Workers maintain local state, including populations of candidates, caches of evaluation results, and solver-specific hyperparameters. The coordinator maintains global artifacts such as the incumbent set, an archive for diversity measurement, a posterior over solver performance, and calibration models for multi-fidelity evaluation. Communication occurs through message passing, where workers send candidate summaries and evaluation results to the coordinator and receive dispatch instructions [54]. Asynchrony is important because evaluation times vary widely across candidates, especially when some candidates trigger expensive alignment computations.

To preserve probabilistic correctness under asynchrony, the coordinator timestamps observations and uses event-driven updates. If a worker returns a reward observation R_j derived from improvement over the incumbent, the coordinator records the incumbent at the time the job was launched, not necessarily the current incumbent. This avoids attributing improvements to the wrong baseline. When needed, rewards can be re-expressed relative to the current incumbent using stored candidate values and probabilistic uncertainty models. This accounting reduces bias in performance estimates, which otherwise could cause the ensemble to over-allocate to solvers that happened to run during periods of easy improvement [55].

Caching and memoization are essential. Many candidates share substructures, especially in process trees or when local search modifies only a small part of a net. Incremental evaluation techniques can reuse replay results for unchanged fragments. Probabilistically, caching affects the noise model because repeated evaluations of similar candidates are not independent. The ensemble can account for this by estimating effective sample sizes or by explicitly modeling correlations. While exact correlation modeling can be complex, a pragmatic approach is to downweight repeated evaluations that share a high similarity score, thereby preventing the reward model from becoming overconfident due to correlated observations [56].

For streaming and drift, the architecture supports windowed evaluation. A sliding or exponential window over recent traces defines the operational distribution for prediction and for incremental discovery. The ensemble can maintain multiple incumbents corresponding to different windows, enabling detection of drift when incumbents diverge. A probabilistic drift score can be computed from changes in variant frequencies, changes in directly-follows distributions, or changes in predictive residuals. When drift is detected, the ensemble can increase exploration weights, allocate more resources to solvers that are known to adapt quickly, or spawn new solver instances with different initialization priors [57].

Privacy and governance constraints are first-class. In many enterprises, raw events cannot be freely shared across teams. The architecture can separate sensitive attribute processing from model search by computing privacy-preserving features or by restricting solvers to

operate on anonymized representations. The probabilistic framework helps here because it supports modeling the information loss introduced by anonymization as increased uncertainty. When sensitive attributes are removed, predictive uncertainty should increase accordingly, and decision thresholds can adapt [58]. This aligns with responsible enterprise use where models should not silently become overconfident after privacy-preserving transformations.

Latency constraints vary across tasks. Discovery might run offline, but prediction requires fast inference. The ensemble can incorporate latency as a constraint in optimization. Candidate predictors can be evaluated not only by accuracy but also by inference cost [59]. A combined utility can be written as

$$U_{\text{online}}(\Theta) = \mathbb{E}\left[-\mathcal{L}_{\text{pred}}(\Theta)\right] - \rho\,\mathbb{E}[\text{latency}(\Theta)], \quad (15)$$

where ρ is a trade-off coefficient representing the enterprise cost of latency. The metaheuristic ensemble can search over model architectures and compression methods, such as distillation into smaller models or pruning of features, while tracking uncertainty in latency estimates that may depend on deployment hardware.

A final systems issue is reproducibility. Metaheuristic search is stochastic, which complicates audit trails. The architecture should record random seeds, solver configurations, evaluation samples, and posterior states used for selection [60]. This does not eliminate stochasticity, but it provides a path to reproduce outcomes or to at least explain why a given model was selected. In regulated environments, such traces are often necessary. The probabilistic ensemble framework is compatible with reproducibility because it explicitly represents uncertainty and can record the sequence of probabilistic decisions.

7. Evaluation Methodology and Enterprise Utility

Evaluating probabilistic metaheuristic ensembles requires careful separation of offline metrics, computational efficiency, and online utility. In enterprise contexts, a model that is marginally better on an offline score may be worse in practice if it is unstable under drift, difficult to interpret, or expensive to maintain [61]. A neutral evaluation therefore examines multiple dimensions, including the quality of discovered models, predictive performance, calibration, resource usage, and robustness to nonstationarity.

For discovery, evaluation should consider both point estimates and uncertainty. When fitness and precision are estimated via sampling, confidence intervals should be used when comparing candidates. A probabilistic ensemble should be evaluated on its ability to reach high-quality models within a fixed compute budget, and on the stability of its selections across repeated runs. Stability matters because enterprises need consistent outputs for governance and communication [62]. One measure of stability is the variability of selected model characteristics, such as the number of nodes, concurrency patterns, or dominant paths, across runs. Another is the agreement in replay outcomes across cross-validation splits. These stability measures should be interpreted alongside uncertainty, because high stability may reflect over-regularization and underfitting in highly variable processes.

For prediction, proper scoring rules provide a principled basis for comparing probabilistic forecasts. Negative log-likelihood rewards both accuracy and calibration, whereas Brier scores provide an interpretable squared-error measure for probabilities [63]. Time predictions benefit from evaluating full predictive distributions, for example via continuous ranked probability scores, rather than only point errors. Enterprises often also report familiar metrics such as accuracy or mean absolute error; these can be included, but they do not capture calibration. A probabilistic ensemble should be assessed on calibration, for example by reliability curves or by comparing predicted quantiles to empirical frequencies, because miscalibration can lead to costly misallocation of resources.

Compute efficiency is central to scalability claims. Evaluation should report wall-clock time, number of evaluations, and resource utilization. Because enterprise platforms differ, it is useful to normalize by compute budget, such as CPU-seconds or GPU-seconds, and to evaluate how performance scales with increasing budget [64]. A probabilistic ensemble may have overhead due to coordination and posterior updates, but it can compensate by better allocation. The evaluation should clarify whether improvements come from better search or simply from higher compute. In distributed settings, the effect of asynchrony and straggler tasks should be considered because they can change effective allocation.

Robustness to drift can be evaluated by splitting logs temporally. A discovery model learned on early data can be tested on later data for conformance degradation, and prediction models can be trained on earlier prefixes and tested on later prefixes [65]. A probabilistic ensemble can be evaluated on its adaptation speed, measured by how quickly predictive loss returns to baseline after a drift event. Because drift events can be subtle, evaluations should consider gradual drift as well as abrupt drift. In enterprise contexts, drift may correspond to policy changes, new product introductions, or system migrations. While synthetic drift injections can be used for controlled experiments, temporal splits on real logs are often more representative.

An enterprise evaluation should also consider interpretability and actionability [66]. A discovered model that is too complex may be accurate but unusable for stakeholders. Similarly, a predictor that is accurate but opaque may face resistance in regulated workflows. Interpretability can be proxied by structural complex-

ity metrics and by the ability to map model elements to business concepts. Actionability can be assessed by simulating interventions using predicted risks and measuring expected utility. If a predicted risk triggers an escalation with cost c and benefit b when preventing a negative outcome, then expected utility depends on calibration and on the chosen threshold [67]. A probabilistic predictor supports computing expected utility directly:

$$\mathbb{E}[\text{utility} \mid \sigma_{1:k}] = b\, p(\text{negative} \mid \sigma_{1:k})\, \mathbf{1}\{\text{intervene}\} - c\, \mathbf{1}\{\text{intervene}\} \tag{16}$$

and integrating this over cases yields an operational objective. While enterprises may not know b precisely, sensitivity analysis over plausible b/c ratios can reveal whether a predictive improvement translates into meaningful operational gains. This avoids overstating the value of small metric differences that do not change decisions [68].

Another enterprise issue is fairness and compliance in predictions. Even when sensitive attributes are removed, proxies may remain. A probabilistic ensemble can incorporate constraints on disparate impact or on error rates across groups, but evaluating such constraints requires careful definition of groups and outcomes. In many enterprise process logs, group definitions are tied to organizational units or customer segments. Evaluations should check whether predictive uncertainties differ systematically across groups, which could indicate data quality differences or structural bias in the process [69]. Addressing such issues often requires improving logging or adjusting interventions rather than solely adjusting models.

Finally, evaluation should examine failure modes. In process discovery, failure modes include producing models that are unsound, overly permissive, or overly restrictive under noise. In prediction, failure modes include confident wrong predictions on rare but important cases, and instability under drift. The probabilistic framework helps identify these failures because it surfaces uncertainty, but it does not eliminate them [70]. An enterprise evaluation should therefore report not only average performance but also tail behavior, such as worst-case loss over variants or over time windows. This tail analysis is particularly relevant when service-level violations are driven by rare cases.

Overall, a careful evaluation frames probabilistic metaheuristic ensembles as tools for managing uncertainty and variability in enterprise process mining and prediction, rather than as guaranteed improvements in all settings. The empirical value depends on data quality, process characteristics, and operational constraints.

8. Conclusion

This paper presented a probabilistic metaheuristic ensemble perspective on scalable process mining and event log prediction in enterprise systems [71]. The core argument is that enterprise logs exhibit heterogeneity, noise, and drift that make solver performance uncertain, and that probabilistic portfolios provide a principled mechanism to allocate compute across diverse metaheuristics while tracking uncertainty in both search progress and candidate evaluation. Process discovery and conformance were framed as stochastic multiobjective optimization problems over large discrete model spaces, motivating ensembles that combine multi-fidelity evaluation, diversity-aware reward modeling, and distributional search updates. Scalability considerations were addressed through approximate metrics, incremental and streaming operation, and asynchronous distributed execution with probabilistic accounting. For prediction, the paper connected discovered process structure with probabilistic sequence and time models using mixture formulations and state-uncertainty integration, emphasizing calibrated predictive distributions and enterprise-relevant utility. The discussion highlighted that practical value depends on careful evaluation across quality, calibration, compute cost, robustness to drift, and governance constraints. Probabilistic ensembles do not remove the need for domain-informed preprocessing, constraint specification, and monitoring, but they provide a coherent framework for adapting to uncertainty and variability that are common in enterprise process analytics [72].

References

[1] T. Liu, Y. Tang, J. Wang, Y. Li, N. Yang, and Y. Liu, "Drive system failure control for distributed drive electric vehicles," IOP Conference Series: Materials Science and Engineering, vol. 231, pp. 012149–, 9 2017.

[2] L. Shi, H. Qiao, C. Yang, Y. Jiang, and B. Long, "Research on highly reliable distributed message queue system," in Seventh International Conference on Advanced Electronic Materials, Computers, and Software Engineering (AEMCSE 2024), pp. 36–36, SPIE, 8 2024.

[3] D. M. Walls, "Sigdoc - distributed value system matrix: a new use for distributed usability testing," in Proceedings of the 25th annual ACM international conference on Design of communication, pp. 256–262, ACM, 10 2007.

[4] M. Valera and S. A. Velastin, A review of the state-of-the-art in distributed surveillance systems, pp. 1–30. Institution of Engineering and Technology, 1 2006.

[5] S. N. Talukdar and E. Cardozo, "Patchwork synthesis and distributed processing for power system diagnosis," in 22nd Intersociety Energy Conversion Engineering Conference, American Institute of Aeronautics and Astronautics, 6 1987.

[6] T. L. Huntsberger and M. L. Hilton, "Distributed multisensor integration in a cooperative multirobot system," SPIE Proceedings, vol. 2589, pp. 134–141, 9 1995.

[7] A. Ma, D. Li, and Y. Xi, "Event-triggered distributed model predictive control for pwa systems," International Journal of Robust and Nonlinear Control, vol. 35, pp. 435–451, 10 2024.

[8] K. J. Lynne, "Distributing the server function in a multiring pac system," in SPIE Proceedings, vol. 1446, pp. 177–187, SPIE, 7 1991.

[9] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in International Conference on Scientific and Statistical Database Management, pp. 167–184, Springer, 2009.

[10] Z. Gao, L. Shi, and P. Reviriego, "Enhancing data protection with a distributed storage system based on the redundant residue number system," Wireless Networks, vol. 30, pp. 5601–5612, 3 2023.

[11] P. Das, A. Xhebraj, and T. Rompf, "Specializing data access in a distributed file system (generative pearl)," in Proceedings of the 23rd ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, pp. 44–52, ACM, 10 2024.

[12] A. Filaseta, D. Pianini, and A. Cortecchia, "An architecture and prototype for monitoring distributed simulations of distributed systems," in 2024 28th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 158–165, IEEE, 10 2024.

[13] O. Ulusoy and G. Belford, "A simulation model for distributed real-time database systems," in Proceedings. 25th Annual Simulation Symposium, pp. 232–240, IEEE Comput. Soc. Press, 1 2003.

[14] S. P. Nagarkatti, C. O. Rahn, D. M. Dawson, E. Zergeroglu, and A. A. Malatpure, "Modal control of flexible systems using distributed sensing," in Vibration and Control of Continuous Systems, pp. 11–21, American Society of Mechanical Engineers, 11 2000.

[15] N. Arshad, D. Heimbigner, and A. L. Wolf, "Dealing with failures during failure recovery of distributed systems," ACM SIGSOFT Software Engineering Notes, vol. 30, pp. 1–6, 5 2005.

[16] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in 2006 International Conference on Wireless and Mobile Communications (ICWMC'06), pp. 44–44, IEEE, 2006.

[17] H. Mendes, C. Tasson, and M. Herlihy, "Stoc - distributed computability in byzantine asynchronous systems," in Proceedings of the forty-sixth annual ACM symposium on Theory of computing, pp. 704–713, ACM, 5 2014.

[18] K. Yuan, K. Zhang, and J. Cao, "Pinning control of the coupled distributed parameter system with time delay," Asian Journal of Control, vol. 21, pp. 1250–1259, 5 2018.

[19] N. W. Bergmann, Y. Y. Chung, X. Yang, Z. Chen, W.-C. Yeh, X. He, and R. Jurdak, "Using swarm intelligence to optimize the energy consumption for distributed systems," Modern Applied Science, vol. 7, pp. 59–66, 5 2013.

[20] H. Hu, D. Wang, and C. Wu, "Aaai - distributed machine learning through heterogeneous edge systems," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 7179–7186, 4 2020.

[21] P. Heredia, E. Garcia, and S. Mou, "Distributed state estimation for nonlinear systems with unknown parameters," in 2022 American Control Conference (ACC), pp. 96–101, IEEE, 6 2022.

[22] F. Hu, K. Mehta, S. Mishra, and M. AlMutawa, "Distributed edge ai systems," in Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing, vol. 2, pp. 1–6, ACM, 12 2023.

[23] L. Fasanotti, S. Cavalieri, E. Dovere, P. Gaiardelli, and C. E. Pereira, "An artificial immune intelligent maintenance system for distributed industrial environments," Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, vol. 232, pp. 401–414, 4 2018.

[24] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in 2006 International Conference on Advanced Computing and Communications, pp. 628–629, IEEE, 2006.

[25] X. An, X. Yu, W. Song, L. Han, T. Yang, Z. Li, and Z. Su, "A software-defined distributed architecture for controlling unmanned swarm systems," Electronics, vol. 12, pp. 3739–3739, 9 2023.

[26] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Task scheduling in distributed real-time systems," in SPIE Proceedings, vol. 857, pp. 909–917, SPIE, 10 1987.

[27] F. Kart, A Distributed E-Healthcare System. IGI Global, 1 2011.

[28] I. P.-Vaisband, R. Jakushokas, M. Popovich, A. V. Mezhiba, S. Kose, and E. G. Friedman, Stability in Distributed Power Delivery Systems, pp. 397–411. Springer International Publishing, 4 2016.

[29] C. Sirocchi and A. Bogliolo, Distributed Averaging for Accuracy Prediction in Networked Systems, pp. 130–145. Germany: Springer Nature Switzerland, 4 2024.

[30] M. E. Douglas, M. K. Sahm, and W. J. Wepfer, "Economic optimization of a distributed generation system for an orifice building," in Advanced Energy Systems, pp. 415–420, ASMEDC, 1 2006.

[31] X. Zhang, X. Zhang, H. Li, Y. Bi, H. Hu, and H. Wang, "Distributed data management system at ihep," in Proceedings of International Symposium on Grids & Clouds (ISGC) 2023 in conjunction with HEPiX Spring 2023 Workshop — PoS(ISGC&HEPiX2023), pp. 27–027, Sissa Medialab, 10 2023.

[32] A. D. George, R. B. Fogarty, J. S. Markwell, and M. D. Miars, "An integrated simulation environment for parallel and distributed system prototying," SIMULATION, vol. 72, pp. 283–294, 5 1999.

[33] E. Almberg, M. P. Twedt, C. Gerometta, and S. P. Gent, "Economic feasibility of corn stover torrefaction for distributed processing systems," in Volume 1: Biofuels, Hydrogen, Syngas, and Alternate Fuels; CHP and Hybrid Power and Energy Systems; Concentrating Solar Power; Energy Storage; Environmental, Economic, and Policy Considerations of Advanced Energy Systems; Geothermal, Ocean, and Emerging Energy Technologies; Photovoltaics; Posters; Solar Chemistry; Sustainable Building Energy Systems; Sustainable Infrastructure and Transportation; Thermodynamic Analysis of Energy Systems; Wind Energy Systems and Technologies, American Society of Mechanical Engineers, 6 2016.

[34] A. H. F. Chow and R. Sha, "Performance analysis of centralized and distributed systems for urban traffic control," Transportation Research Record: Journal of the Transportation Research Board, vol. 2557, pp. 66–76, 1 2016.

[35] J. Light, "Distributed graphical topic-oriented document search system," SPIE Proceedings, vol. 3017, pp. 129–135, 4 1997.

[36] H. Liu, Q. Du, Y. Song, and W. Guo, "Optimal control of distributed generation system: The state of the art," Applied Mechanics and Materials, vol. 433-435, pp. 1061–1064, 10 2013.

[37] H. Yang, B. Jiang, and V. Cocquempot, Switched Nonlinear Systems with Distributed Parameters, pp. 143–182. Springer International Publishing, 6 2014.

[38] H. Özbay, S. Gumussoy, K. Kashima, and Y. Yamamoto, Frequency Domain Techniques for ∞ Control of Distributed Parameter Systems. Society for Industrial and Applied Mathematics, 10 2018.

[39] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in

Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI, pp. 2701–2706, 2007.

[40] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Sensys - cartel: a distributed mobile sensor computing system," in Proceedings of the 4th international conference on Embedded networked sensor systems, pp. 125–138, ACM, 10 2006.

[41] J. Zhu, H. Deng, X. Li, Y. Yuan, and F.-Y. Wang, "Blockchain-based consensus study on distributed control systems*," in 2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI), pp. 164–167, IEEE, 7 2021.

[42] Y. Hong, Y. Ping, and J. P. Zhou, "A distributed monitoring system for spinning-machine's spindle," in SPIE Proceedings, vol. 6041, pp. 604119–, SPIE, 12 2005.

[43] J. S. Park, P. Chandramohan, A. T. Suresh, J. V. Giordano, and K. Kwiat, "Component survivability at runtime for mission-critical distributed systems," The Journal of Supercomputing, vol. 66, pp. 1390–1417, 9 2012.

[44] T. Loukopoulos, P. Lampsas, and I. Ahmad, "Ics - continuous replica placement schemes in distributed systems," in Proceedings of the 19th annual international conference on Supercomputing, pp. 284–292, ACM, 6 2005.

[45] M. Turoff, S. R. Hiltz, A. N. F. Bahgat, and A. R. Rana, "Distributed group support systems," MIS Quarterly, vol. 17, pp. 399–417, 12 1993.

[46] F. Boldrin, C. Taddia, and G. Mazzini, "Web distributed computing systems implementation and modeling," International Journal of Adaptive, Resilient and Autonomic Systems, vol. 1, pp. 75–91, 1 2010.

[47] J. Li, Y. Zhang, S. Lu, H. S. Gunawi, X. Gu, F. Huang, and D. Li, "Performance bug analysis and detection for distributed storage and computing systems," ACM Transactions on Storage, vol. 19, pp. 1–33, 6 2023.

[48] J. Zhou and Z. Feng, "Transient response of distributed parameter systems," in Volume 1D: 16th Biennial Conference on Mechanical Vibration and Noise, American Society of Mechanical Engineers, 9 1997.

[49] Q. Yu, Z. Jiang, Y. Liu, S. Shi, J. Wang, Y. Zhang, L. Li, X. Han, X. Bai, X. Xue, F. Qi, and J. Wang, "Research of integrated system of distributed multi-energy station," in 2021 11th International Conference on Power, Energy and Electrical Engineering (CPEEE), pp. 261–265, IEEE, 2 2021.

[50] Y. L. Gorrec, H. Ramirez, Y. Wu, N. Liu, and A. Macchelli, Energy Shaping Control of 1D Distributed Parameter Systems, pp. 3–26. Springer International Publishing, 4 2022.

[51] N. Soundarajan, FAABS - Refining Interactions in a Distributed System. Germany: Springer Berlin Heidelberg, 10 2001.

[52] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson, "Verdi: a framework for implementing and formally verifying distributed systems," ACM SIGPLAN Notices, vol. 50, pp. 357–368, 6 2015.

[53] P. Chitnelawong, A. A. Klishin, N. Mackay, D. J. Singer, and G. van Anders, "No free lunch for avoiding clustering vulnerabilities in distributed systems.," Scientific reports, vol. 14, pp. 12789–, 6 2024.

[54] C. Ramachandran, S. Misra, and M. Obaidat, "On evaluating some agent-based intrusion detection schemes in mobile ad-hoc networks," Proceedings of the SPECTS 2007, pp. 594–601, 2007.

[55] G. Chen and F. L. Lewis, "Distributed tracking control for networked mechanical systems," Asian Journal of Control, vol. 14, pp. 1459–1469, 5 2012.

[56] J. R. Wright, G. Gavilan, Y. Zhang, and K. Redinbo, "Emerging technologies for developing distributed database systems," in Building Partnerships, pp. 1–9, American Society of Civil Engineers, 9 2000.

[57] I. Blank, Z. Balewski, K. Mahowald, and E. Fedorenko, "Syntactic processing is distributed across the language system.," NeuroImage, vol. 127, pp. 307–323, 12 2015.

[58] J. Wang, S. Han, K.-Y. Lam, and A. K. Mok, "Maintaining data temporal consistency in distributed real-time systems," Real-Time Systems, vol. 48, pp. 387–429, 4 2012.

[59] K. P. Birman and T. A. Joseph, "Exploiting virtual synchrony in distributed systems," 2 1987.

[60] V. Subramonian and C. Gill, "Fine-grained composition of distributed sensor system infrastructure," in IEEE MTT-S International Microwave Symposium Digest, 2005., vol. 1, pp. 377–380, IEEE, 6 2005.

[61] W. He, F. Qian, G. Chen, and Q.-L. Han, Distributed Impulsive Control of Leader-Following Multi-agent Systems, pp. 1–54. Springer Nature Singapore, 4 2019.

[62] R. N. Schauer and A. Joshi, "Icac - a probabilistic approach to distributed system management," in Proceedings of the 7th international conference on Autonomic computing, pp. 77–78, ACM, 6 2010.

[63] L. Zhao, D. Wang, B. Huang, and L. Xie, "Distributed filtering-based autonomous navigation system of uav," Unmanned Systems, vol. 3, pp. 17–34, 1 2015.

[64] D. Dering and C. L. Swartz, "Integration of scheduling and control for plants controlled by distributed mpc systems," Industrial & Engineering Chemistry Research, vol. 63, pp. 12016–12034, 6 2024.

[65] R. C. Coghill, "The distributed nociceptive system: a novel framework for understanding pain.," Scandinavian journal of pain, vol. 22, pp. 679–680, 9 2022.

[66] W. A. Adamson and O. Kornievskaia, "Infrasec - a practical distributed authorization system for gara," Lecture Notes in Computer Science, pp. 314–324, 9 2002.

[67] A. Prokscha, F. Sheikh, N. Zarifeh, I. B. Mabrouk, and T. Kaiser, "Distributed antenna system for improving thz coverage inside rooms," in 2021 IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (APWC), pp. 112–112, IEEE, 8 2021.

[68] H. Baruh and L. Meirovitch, "On the placement of actuators in the control of distributed-parameter systems," in Dynamics Specialists Conference, American Institute of Aeronautics and Astronautics, 4 1981.

[69] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in IMMERSCOM, p. 18, 2009.

[70] J. Liu, O. Arden, M. D. George, and A. C. Myers, "Fabric: Building open distributed systems securely by construction," Journal of Computer Security, vol. 25, pp. 367–426, 7 2017.

[71] G. Tackett, T. McKelvy, and W. Greenleaf, "Missile system modeling in a distributed virtual environment," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, vol. 1, pp. 71–82, 4 2004.

[72] V. Natarajan and J. Bentsman, "Approximate local output regulation for nonlinear distributed parameter systems," Mathematics of Control, Signals, and Systems, vol. 28, pp. 24–, 7 2016.