Computational Engineering and Technology Innovations

www.ceti.reapress.com

Comput. Eng. Technol. Innov. Vol. 3, No. 3 (2026) 193–207.

Paper Type: Original Article

Adaptive Accuracy and Cost Control under Distribution Shift in Model Cascades

Renas Rajab Asaad[1,*], Viktor P. Sokolov[2]

[1] Department of Information Technology, Technical College of Informatics–Akre, Akre University for Applied Sciences, Duhok, Iraq; renas_rekany@yahoo.com; renas.beda89@gmail.com.

[2] Department of Computer Science, Kazan Federal University, Kazan, Russia.

Citation:

Received: 16 June 2025 *Revised: 10 August 2025* *Accepted: 04 January 2026*	Rajab Asaad, R., & Sokolov, V. (2026). Adaptive accuracy and cost control under distribution shift in model cascades. *Computational Engineering and Technology Innovations,* 3(3), 193-207.

Abstract

Streaming prediction under distribution shift requires decisions about inference cost and model refresh. We examine ABACUS, a confidence-gated cascade with drift-triggered refitting and feedback on average proxy cost, and outline a separate conditional design for admission under per-item deadlines. The supplied study reports ten seeded simulations of 10,000 items each on rotating-hyperplane and Streaming Ensemble Algorithm (SEA) generators with two scheduled abrupt changes, quality perturbations, and a 200-item label delay. At nominal budget B=3, reported mean accuracies are 0.9148 and 0.9397 at assigned mean costs 2.99 and 2.94, compared with 0.6775 and 0.8987 at costs 3.00 and 2.85 for a learned static cascade. Reported areas under accuracy-versus-nominal-budget curves are 0.853 and 0.916 versus 0.651 and 0.883. The ablations attribute the observed gain at this operating point to drift-triggered refitting; budget feedback mainly tracks the target mean cost. These are author-reported simulation results: the named experiment script, shared generator, seed-level results, and raw JavaScript Object Notation (JSON) file were not furnished with the materials reviewed. The assigned cost units are not measured wall-clock latency. The experiment did not exercise worst-case stage admission, preemption, pool-weight updates, projected-gradient threshold control, or an input-channel detector. It therefore establishes neither a hard per-item deadline nor the comparative cost of enforcing one. Independent reproduction and a matched, refitted single-model baseline are required before a general performance claim.

Keywords: Distribution shift, Model cascade, Drift-triggered refitting, Average compute cost, Conditional deadline admission, Synthetic stream.

1 | Introduction

Streaming systems may need prompt predictions while the input-label relationship evolves. In such settings, a fixed model can lose accuracy after a shift and a more expensive model need not remain more accurate. The

Corresponding Author: renas_rekany@yahoo.com

https://doi.org/10.48314/ceti.v3i3.92

Licensee System Analytics. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0).

relevant engineering questions are how frequently to refresh a model, how to route items among stages, what resources each path consumes, and whether the deployed system has a genuine response deadline. The present experiment addresses accuracy and an assigned average-cost proxy; a hard arrival-to-response deadline is a proposed systems constraint that it does not test.

Continuous queries [1], distributed dataflow systems [2–4], and load shedding [5] provide ways to process streams and manage latency. Their systems goals differ from this study's predictive accuracy under scheduled synthetic drift. Tail latency remains a separate deployment measure [6]; the supplied experiment does not instrument an end-to-end serving stack.

Cascades and early exits offer one route to conditional computation, from the Viola–Jones detector [7] to neural designs [8], [9]. A confidence gate can avoid further work when an item is easy for an earlier stage, but a distribution shift may change both calibration and which stage is useful. Here model refresh, rather than routing alone, is the principal observed mechanism. The value of the cascade over a refitted single model remains to be tested.

We study a specific combination of cascade routing, drift-triggered refitting, and feedback on average proxy cost. A distinct conditional design admits paths against certified stage and system overhead bounds. The experiment below implements only a subset of that design. Mean assigned cost, tail latency, and a hard deadline are different endpoints.

The paper formulates incremental path cost and a conditional deadline-admission rule that includes queueing and non-model overhead. It then describes a broader controller while identifying the narrower mechanisms actually simulated. The synthetic campaign compares five policies and reports budget sweeps, ablations, and paired tests. These tabulated results are retained as author-reported evidence; scripts and per-seed outputs remain necessary for independent reproduction.

Section 2 reviews relevant systems, drift, and cascade research. Section 3 states the questions and the conditional operating model. Section 4 distinguishes the broader architecture from the implementation reported in Section 5. Sections 6 and 7 discuss limitations and next validation steps.

2 | Related Work

2.1 | Streaming Systems and Latency Management

The database community established the foundations of computing over unbounded inputs: continuous queries over data streams [1] and the systems that realised them, from early monitoring engines to modern distributed runtimes. MillWheel provided fault-tolerant, low-latency per-key streaming [2], and the dataflow model gave a practical account of balancing correctness, latency, and cost with event time and watermarks [3]. Flink demonstrated that one engine can serve both stream and batch workloads [4]. Under overload, load shedding and approximation are the classical responses, with principled treatments of where to shed for aggregation workloads [5] and of bounded-error, bounded-time query answering [10]. The tail-latency literature adds a systems-design perspective: percentiles, not means, determine user-visible performance, and taming them requires bounding every stage of the critical path [6]. Our framework consumes these lessons directly, since hard deadlines must be enforced by construction, per stage, rather than hoped for on average.

2.2 | Learning under Concept Drift and Dataset Shift

The learning side of the problem is studied under two overlapping banners. Concept-drift research treats streams whose joint distribution of inputs and labels changes over time, with comprehensive surveys of adaptation strategies [11] and of the learning algorithms themselves [12]. Dataset-shift research emphasises the taxonomy of shift mechanisms and their consequences for prediction, generalising the classical covariate-shift setting [13], [14] and transfer learning surveys the reuse of past models under distributional mismatch [15]. Algorithmically, the stream-learning community contributed very fast decision trees [16] and their time-changing variants [17], streaming ensembles [18], online bagging and boosting [19], dynamically weighted

ensembles for drift tracking [20], and incremental algorithms that learn under non-stationarity without full retraining [21]. Change detection itself is a distinct craft, with classical sequential inspection [22] and stream-specific detectors such as Drift Detection Method (DDM) [23], Early Drift Detection Method (EDDM) [24], and Adaptive Windowing (ADWIN) [25], and platforms such as Massive Online Analysis (MOA) standardising evaluation [26].

Concept-drift studies often emphasize predictive performance and resource use under changing streams [11], [12], while anytime algorithms explicitly trade quality against available computation [27], [28]. These strands motivate joint measurement of predictive quality, adaptation effort, and latency. They do not by themselves validate a hard deadline for the controller studied here; that would require an implemented serving path and an arrival-to-response timing certificate.

2.3 | Cascades, Early Exit, and Budget-Aware Inference

Viola and Jones [7], BranchyNet [8], and multi-scale dense networks [9] show different forms of cascade or early-exit inference; Clipper [29] concerns serving. Expert-weighting [30], [31], bandit feedback [32], and online optimization [33] provide candidate update tools, each under its own assumptions. The proposed architecture combines several ideas, but the supplied experiment implements only the limited refit and feedback mechanisms specified in Section 5. It supplies no test of a general pool-weight update or a hard deadline.

2.4 | Synthesis and Positioning

Existing streaming systems, drift-aware learners, and cascades each address parts of the problem [1–13]. ABACUS is a proposed composition of path selection, model refresh, and resource accounting. The present synthetic experiment isolates the observed value of refitting under drift and mean proxy-cost control. The broader promise of hard deadline compliance remains a design proposition with explicit prerequisites, rather than an empirical result of this campaign.

3 | Problem Formulation and Research Questions

3.1 | Research Questions and Hypotheses

Three questions separate what the supplied study measures from what it leaves open. RQ1 concerns average proxy-cost and accuracy in the tested cascades; RQ2 asks which signals drive adaptation; RQ3 concerns the response to larger nominal budgets and the untested hard-deadline variant. The hypotheses below are study expectations; no dated preregistration record was provided.

RQ1 asks how drift-triggered refitting in a cascade compares with a learned static cascade at a similar nominal mean-cost target on the specified streams. RQ2 asks what delayed-label accuracy and exit-entropy signals contribute to alarms, model refits, and prediction quality, and what a label-only comparison would add. RQ3 asks how this configured controller responds to larger nominal cost targets and what a future deadline-enforcing implementation might sacrifice compared with mean-latency control.

H1 expects the refitting cascade to improve accuracy on these synthetic streams at a comparable mean proxy cost to a learned static cascade; a refitted single-model arm would be needed to isolate routing. H2 expects exit entropy to add useful alarms beyond delayed labels, but there is no label-only detector arm. H3 asks whether worst-case admission trades accuracy for a hard deadline, but that comparison is not implemented.

The questions and hypotheses define the intended comparisons. Because neither a preregistration record nor the named experiment artifacts were available in the reviewed materials, the numerical results below are described as author-reported. The system model also distinguishes abstract cost units from elapsed time.

3.2 | System Model and Notation

Items arrive as $x_1, x_2, \ldots$ with arrival-to-response deadline D. A cascade has L levels m_1 through m_L. Let δc_l be the incremental cost of traversing level l, so that total cost for an exit at z is $C(z) = \sum_{l \le z} \delta c_l$. This definition is necessary to reconcile the experiment's endpoint costs 1, 3, and 10 with a cascade that can stop after each level: the corresponding incremental costs would be 1, 2, and 7 if work is shared exactly as assumed. The implementation and feature-reuse account have not been supplied. Let τ_l bound a level's elapsed service time, and let q_{max} and τ_{over} bound upstream waiting and routing/output overhead. Admission can enforce D only if these quantities are genuinely bounded and a feasible initial/fallback path exists. Confidence gates $\theta_l(t)$, optional pool weights, and delayed labels control later choices. *Table 1* defines the core notation.

Table 1. Principal notation for the ABACUS framework.

Symbol	Meaning	Unit
x_t	Item at index t	-
D	Conditional arrival-to-response deadline	ms
L	Number of cascade stages	-
m_l	Model at stage l	-
δc_l	Incremental cost of level l; total $C(Z) = \sum_{L \le z} \delta C_L$	cost units
τ_l	Certified upper bound on service time at level l, if available	ms
$\theta_l(t)$	Exit threshold at stage l and time t	-
z_t	Exit stage selected for item t	-
Δ_{lab}	Label delay	ms
P_{det}	Set of change-point detectors	-
λ	Cost weight in the objective	-
$A(t)$	Running accuracy estimate	-
$C(t)$	Running cost per item	-

3.3 | Formal Problem Statement

The objective prices accuracy and incremental computation on a stream. With z_t the exit level, δc_l the incremental cost, and λ a cost weight, an illustrative epoch objective is

$$\underset{\text{admissible } (\theta, w)}{\text{maximize}} \; E\left[1\{\text{correct}_t\} - \lambda \sum_{l \le z_t} \delta c_l\right]. \quad \textbf{(1)}$$

Eq. (1) describes a quality-versus-cost target, not a proven online optimum. A hard arrival-to-response promise additionally requires a certified bound on queueing, dispatch, stage execution, and output for every admitted path. In particular,

$$q_{max} + \tau_{over} + \sum_{l \in path(z)} \tau_l \le D \quad \text{for every admitted exit path } z. \quad \textbf{(2)}$$

The bound in *Eq. (2)* is conditional: Statistical tail percentiles are not worst-case certificates; shared-system queues may be unbounded; and an item that cannot complete even the first level needs a specified, bounded fallback. The simulation in Section 5 uses none of these timing bounds and constrains mean assigned cost instead. Updating thresholds must preserve an actually feasible path set if this conditional deadline design is ever deployed.

4 | Proposed Methodology

4.1 | Architecture Overview

The broader ABACUS design has an item loop, a window-level adaptation loop, and a slower model-maintenance loop. The item loop would route an item through admitted stages and log its exit. Adaptation would update thresholds and optional pool weights from arrived labels and proxy statistics, while maintenance could retrain models. The experiment in Section 5 implements a narrower version: confidence gates, an entropy-plus-delayed-accuracy alarm, recent-window refits, threshold retuning, and proportional feedback on average assigned cost. It does not implement the certified item path, general detector battery, or pool-weight and projected-gradient updates.

Placing updates outside the immediate inference loop can reduce response-path work, but their compute, synchronization, and contention must still be measured. In the reported simulation, roughly ten refits per 10,000-item run are counted as events; their Central Processing Unit (CPU) time and effect on an actual serving deadline are not included in the assigned per-decision cost.

4.2 | Hard-Deadline Cascade Control

A deadline-aware implementation would reserve bounded time for queueing, routing, output, and a fallback before admitting a model stage. It would run level l only if the remaining certified budget covers τ_l and any required return path. If even the initial level cannot finish in time, the service must reject, abstain, or return a predetermined fallback within a stated bound. Conditional feasibility requires isolation or another defensible queue bound, measured and conservatively certified execution times, and bounded preemption or non-preemptive stages. Thresholds alone cannot enforce a hard deadline, and a measured high percentile is not a deterministic upper bound. None of these mechanisms was executed in the reported experiment.

If a stage can be safely interrupted, an anytime implementation may return the deepest completed prediction [27]. If it cannot, admission must reserve its full certified time before starting it. Certification and re-certification require a declared platform, load envelope, measurement and analysis method, as well as a documented fallback when a bound fails. The numerical results below do not include per-item arrival-to-response latencies or violations.

4.3 | Drift Detection and Signal Design

The broader design could monitor delayed-label error, confidence and exit-rate distributions, and input features, using detectors such as Page-style sequential tests [22] or ADWIN [25]. The reported experiment is narrower: an exponentially weighted mean of exit entropy and the accuracy of arrived 200-item label blocks are compared with prefix-calibrated limits, and an alarm initiates a refit. No input-feature detector, Page-Hinkley test, ADWIN, or label-only detector comparison is reported. False alarms require an explicit event definition and matching tolerance, especially on a continuously rotating hyperplane.

Changes in confidence, error, or exit share can prompt investigation but do not uniquely identify covariate, concept, or recurring shift. For example, miscalibration or a noisy-label burst can change accuracy without a new concept. A diagnostic controller would compare feature and conditional-label evidence and test its classifications against known generator states. The supplied two-stream study does not perform that diagnosis; it uses a common refit action after an alarm.

4.4 | Online Adaptation Rule

In the broader design, thresholds and optional pool weights could be updated separately. A proposed threshold step would use a defined surrogate loss and project onto an actually feasible path set. The source gives no estimator, smooth surrogate, step-size schedule, or proof for that step, and it was not implemented in the reported campaign. A candidate exponential-weights update for a K-model pool is

$$w_k(t) = \frac{w_k(t-1)\exp(-\eta \hat{\ell}_{k,t})}{\sum_j w_j(t-1)\exp(-\eta \hat{\ell}_{j,t})}. \quad (3)$$

Here $\hat{\ell}_{k,t}$ denotes an estimated recent loss and η an update rate. Standard full-information multiplicative-weights analysis [30], [31] assumes suitable bounded observed losses; delayed, selectively observed labels and confidence surrogates do not inherit that guarantee automatically. The sampling or importance-weighting scheme, loss range, and proof would have to be stated before claiming regret bounds. No pool-weight update, threshold gradient, or controller-cost charge was measured in Section 5; the tested controller uses refits and proportional mean-cost feedback instead.

4.5 | The ABACUS Controller

Algorithm 1 is a design sketch for the broader deadline-aware system. The numerical campaign exercises a subset described separately in Section 5; the algorithm is not pseudocode for the reported simulation. The sketch assumes valid bounds for queueing, overhead, stages, and a feasible bounded fallback.

Algorithm 1. ABACUS bounded-latency cascade control.

Input: Cascade $m_1, \dots, m_L$ with costs c_l and worst-case times τ_l, deadline D, cost weight λ, window W, detectors P_{det}
1: Calibrate confidences; set thresholds $\theta_l \leftarrow \theta_0$ and pool weights $w \leftarrow$ uniform; certify the admissible set given D
2: For each item x_t do
3: Remaining bound $b \leftarrow D - q_{max} - \tau_{over}$; use bounded fallback if b cannot admit level 1
4: For $l = 1, \dots, L$, admit l only if $b \geq \tau_l$ and a completed prediction can be returned in time
5: $(pred_l, conf_l) \leftarrow m_l(x_t)$; $b \leftarrow b - \tau_l$
6: If $conf_l \geq \theta_l$ then exit with $pred_l$
7: If no confidence exit, return the deepest completed prediction or bounded fallback
8: Every window W do
9: Update accuracy A, cost C, exit-rate and confidence statistics
10: In a fully implemented version, run specified detectors on arrived labels and proxy/input channels
11: Investigate shift type; do not infer it uniquely from confidence or error alone
12: If full-information or valid estimated losses exist, update optional pool weights by *Eq. (3)*
13: If a surrogate and feasible projection are defined, update thresholds
14: Monitor timing bounds; suspend the deadline claim and switch to fallback if certification fails
15: Rate-limit all changes; log configuration and detector states
16: If the accuracy target is unreachable within the cost ceiling then flag maintenance
17: Maintenance loop: retrain or replace pool models on recent windows
Output: predictions or declared fallback within D only under certified queue, stage, and overhead bounds.

4.6 | Complexity, Assumptions, and Failure Conditions

The proposed item-loop bookkeeping is $O(L)$ in the number of traversed levels, in addition to model execution. Detector updates depend on their window sizes and feature dimensions; retraining can dominate the adaptation workload. Neither worst-case runtime nor maintenance compute was measured in the supplied experiment. Consequently, an $O(L + K)$ bookkeeping expression cannot establish a deployed timing guarantee or that adaptation is free under the reported proxy cost.

A deployed version would require calibrated or monitored confidence estimates, certified stage and system timing bounds, representative arrived labels for the error channel, and a stated loss-observation model before any online regret result. Gross miscalibration may misroute items; tail events may escape measurement; and proxy confidence may conceal a real loss of accuracy.

Several failure modes can be monitored, but not every undetected shift or rare tail event is observable before its consequences. A deployed controller must define how it abstains or downgrades when timing certification or calibration fails; the simulation does not test these responses.

Slow drift may remain below detector limits until error accumulates. Simultaneous resource pressure and predictive shift would compete for the same budget, and a refitted model may invalidate confidence calibration and thresholds. A deployment would need explicit downgrade, abstention, and coordinated model-and-threshold update behavior; none of these failure responses is demonstrated in the simulation.

5 | Experimental Results and Analysis

5.1 | Setup: Streams, Cascade, Policies, and Executed Scale

The source reports a campaign dated 2026-10-11 using two synthetic stream generators: a ten-feature rotating hyperplane with continuous change and abrupt jumps at $t = 2{,}500$ and 7,500, and a three-feature Streaming Ensemble Algorithm (SEA) variant whose concept threshold changes at the same indices [17], [18]. Each stream has 10,000 items per seed for ten seeds (110–119). A shared overlay adds six 100–300-item Missing Completely At Random (MCAR) missingness bursts at 10–30% entry missingness, five label-flip bursts at 5–15%, and 2% exact duplicates; the bursts are said to begin at least 400 items from scheduled abrupt changes. Missing values are filled from first-1,000-item feature means, SEA features are scaled by 0.1, and accuracy is scored against pre-noise labels. These generator specifications are author-reported because stream_gen.py and the seed-level traces were not furnished. The exact SEA threshold order, class balance, drift schedule, burst indices, and imputation/training separation require the generator and manifest.

The source reports three candidate exit levels with assigned total feature-acquisition-plus-inference proxy costs: L0 is logistic regression on the first 20% of features (cost 1), L1 logistic regression on all features (cost 3), and L2 a 100-tree random forest [34] (cost 10). If these are cumulative exit costs for one shared cascade, the incremental path costs in *Eq. (1)* are 1, 2, and 7; whether feature extraction and inference are actually shared must be checked against exp_15.py. The levels are initially fitted on the first 1,000 pre-shift items, with labels assumed available after a warm start. The comparators are always-L0, always-L2, a fixed 0.8 confidence-threshold cascade, and a learned static cascade whose thresholds are grid-searched on the prefix for each mean-cost target B. The simulated ABACUS variant adds an alarm from exit-entropy EWMA (decay 0.9) and arrived 200-item delayed-label block accuracy, with prefix-calibrated limits (three and 2.5 nominal standard deviations, respectively). Upon alarm it reportedly refits all levels on the most recent 500 arrived labelled instances, retunes thresholds in that same window, and enforces a 500-item cooldown. Proportional feedback rescales both thresholds by $\mathrm{clip}\left(\sqrt{\frac{B}{\mathrm{EWMA(cost)}}}, 0.5, 2.0\right)$. These choices could overfit a short, reused window; an independent post-refit evaluation and a refitted L1-only comparator would help isolate the cascade's value. Every mechanism is stated to consume only labels that have arrived after a one-block (200-item) delay; the timing needs code-level verification.

Nominal targets $B \in \{1, 1.5, 2, 3, 4, 5, 6, 8, 10\}$ are swept for the learned static and adaptive cascades; the fixed policies have a single realized-cost point. The main operating target is $B = 3$. Reported measures include accuracy against pre-noise ground truth, mean assigned exit cost, a trapezoidal area under accuracy versus nominal B over $B = 1 \ldots 10$ (AULC), post-change recovery by rolling accuracy, exit shares, and alarm/refit counts. AULC uses nominal targets rather than realized costs: integrating the rounded adaptive accuracies in the budget table gives approximately 0.8530 and 0.9163, as stated. The source says Python/scikit-learn [35] experiments ran on two CPU cores in 383 seconds, but the scripts and run logs were absent from the reviewed materials. The study does not measure wall-clock per-item latency or retraining cost, enforce a worst-case deadline, compare a refitted single L1 model, or implement the broader controller in Section 4. Thus H3 is untested, and H2 lacks a label-only comparator.

5.2 | Main Comparison (RQ1)

Table 2 reproduces the author-reported operating-point means and standard deviations over ten seeds, and *Fig. 1* shows the associated accuracy-versus-realized-mean-cost plot. On the rotating hyperplane, ABACUS is reported at 0.915 ± 0.010 accuracy and cost 2.99 versus 0.678 ± 0.056 at cost 3.00 for the learned static

cascade, 0.663 ± 0.063 at cost 4.94 for the fixed-threshold cascade, 0.668 ± 0.061 at cost 10 for always-L2, and 0.512 ± 0.042 at cost 1 for always-L0. For SEA the corresponding reported values are 0.940 ± 0.007 at cost 2.94, 0.899 ± 0.004 at 2.85, 0.894 ± 0.005 at 5.58, 0.888 ± 0.004 at 10, and 0.747 ± 0.006 at 1. At the nominal target $B = 3$, ABACUS has a large reported accuracy advantage over the learned static cascade at similar mean assigned cost; it does not Pareto-dominate always-L0, whose cost is lower, or strictly dominate the SEA learned static cascade in both axes, whose realized cost is 2.85. The source reports hyperplane exit shares of about 98% at L1 and 0.4% at L2, and SEA shares of 28%, 65%, and 7% at L0/L1/L2. Those shares are broadly compatible with mean endpoint costs near 3, but seed-level traces and precise accounting are needed to check them. The static forest was never refitted, so the contrast with ABACUS also compares model-refresh policies.

Table 2. Author-reported comparison at nominal mean-cost target B = 3.

Policy (Operating Budget B=3)	Hyperplane: Accuracy	Hyperplane: Cost	SEA: Accuracy	SEA: Cost
Always-L0 (LR, 20% features)	0.512±0.042	1.00±0.00	0.747±0.006	1.00±0.00
Always-L2 (RF, 100 trees)	0.668±0.061	10.00±0.00	0.888±0.004	10.00±0.00
Static threshold cascade ($\tau = 0.8$)	0.663±0.063	4.94±0.39	0.894±0.005	5.58±0.29
Learned static cascade (grid search)	0.678±0.056	3.00±0.03	0.899±0.004	2.85±0.08
ABACUS (adaptive + drift recalibration)	0.915±0.010	2.99±0.01	0.940±0.007	2.94±0.02

mean ± standard deviation over ten seeds; accuracy against pre-noise ground truth; assigned feature-and-inference proxy cost. Fixed policies have no budget parameter.

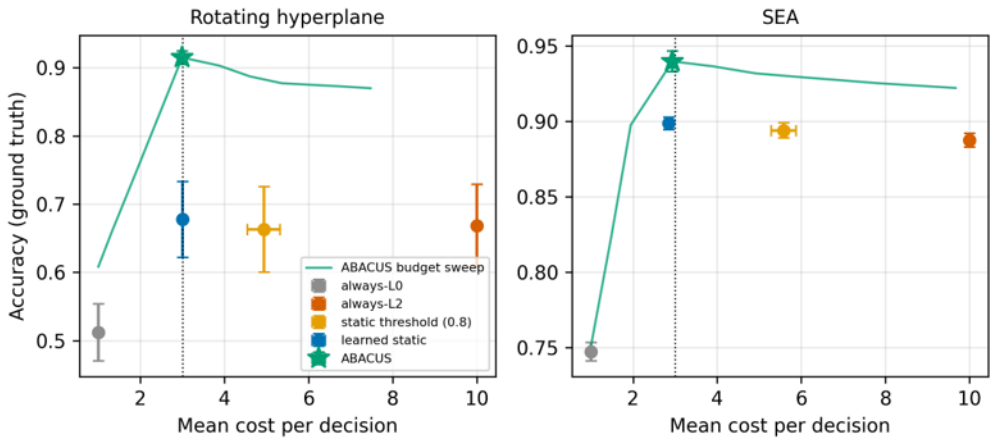


Fig. 1. Author-reported accuracy against realized mean assigned cost (ten-seed means and standard deviations): Rotating hyperplane (left) and SEA (right).

The green line connects ABACUS runs at increasing nominal targets B, and the dotted line marks $B = 3$; it is the response of this configured policy, not an optimal accuracy-cost frontier. The adaptive operating point improves accuracy at approximately the learned static cascade's cost and at a lower cost than the fixed 0.8 cascade or always-L2; always-L0 has a lower cost.

Fig. 2 shows author-reported 200-item rolling accuracy near the first scheduled change. For the rotating hyperplane, the source gives ABACUS mean recovery of 1,310 items (per-seed range 1,100–1,800), while its plotted static policies do not regain the stated pre-change threshold within the remaining horizon. On SEA the reported mean is 2,330 (range 700–6,100), while learned static and always-L2 reportedly cross the recovery threshold later, around 6,770 and 6,970 items. The recovery definition is three sustained windows above pre-

change accuracy minus 0.02, but exact alignment of overlapping windows and censoring must be checked in the raw traces. *Fig. 2* is a mean curve and cannot by itself verify per-seed recovery times.

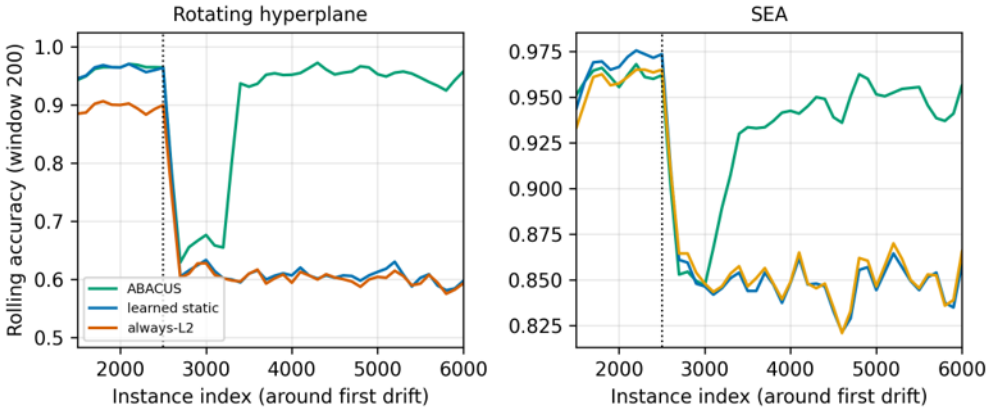


Fig. 2. Author-reported rolling accuracy (window 200, step 100; means over ten seeds) around the first scheduled abrupt change at $\mathbf{t = 2,500}$: Rotating hyperplane (left) and SEA (right).

The plot illustrates recovery of the refitted policy. The static policies shown do not recover in the hyperplane horizon; some static SEA policies reportedly cross the recovery threshold later in the full 10,000-item stream.

5.3 | Ablations of Model Refresh and Budget Feedback

Table 3 reports the author-reported $B = 3$ ablations. Removing alarm-triggered model refits while keeping mean-cost feedback reduces hyperplane accuracy from about 0.915 to 0.677 and SEA accuracy from 0.940 to 0.900, close to the learned static cascade. Removing feedback but retaining refits leaves accuracy nearly unchanged (about 0.915 and 0.939) while SEA mean assigned cost shifts from 2.94 to 2.87. At this operating point and among these ablations, refitting accounts for the reported accuracy gain; this does not show that cascade routing is intrinsically beneficial, since a matched refitted single-model arm is missing. The source reports 9.5 hyperplane and 9.9 SEA alarms/refits per 10,000-item run, while only two abrupt change points were scheduled. Additional alarms cannot simply be labelled false positives without an alarm-to-event window and a decision about the continuously rotating hyperplane and injected quality bursts. Refitting cost and contention are absent from the mean assigned-cost column; they are material to any deployment claim.

Table 3. Author-reported $B = 3$ ablations (mean ± standard deviation over ten seeds where given).

Variant ($B = 3$)	Hyperplane: Acc	Hyperplane: Cost	Hyperplane: Alarms	SEA: Acc	SEA: Cost	SEA: Alarms
ABACUS (full)	0.915±0.010	2.99	9.5	0.940±0.007	2.94	9.9
ABACUS, no drift recalibration	0.677±0.056	3.00	0.0	0.900±0.004	2.94	0.0
ABACUS, no budget controller	0.915±0.010	2.99	9.5	0.939±0.007	2.87	9.8

Alarm counts are per 10,000-item run; each alarm is said to trigger a model refit whose resource cost is not included in the assigned per-decision proxy.

5.4 | Observed Budget Response

Table 4 shows the reported response of this controller to increasing nominal mean-cost targets. Trapezoidal integration across nominal B values from 1 to 10 yields approximately 0.8530 (hyperplane) and 0.9163 (SEA) from the rounded adaptive table entries; the source gives 0.651 ± 0.053 and 0.883 ± 0.004 for the learned

static policy, which cannot be recalculated from its absent sweep table. The adaptive policy's observed accuracy peaks at $B = 3$ (0.9148 and 0.9397) and then decreases as the controller routes more items toward a less effective forest. This is a non-monotone response of the specified heuristic, not a non-monotone optimal accuracy-budget frontier: a policy allowed $B = 10$ can always choose the $B = 3$ configuration when the constraint is an upper bound. At $B = 1$, adaptive refits still occur, so comparison with never-refit always-L0 mixes routing and refresh. No per-item deadline is imposed, and the proposed hard-versus-soft latency question H3 remains unanswered.

Table 4. Author-reported ABACUS response by nominal mean-cost target B: realized assigned mean cost and accuracy over ten seeds.

Budget B	Hyperplane: Realized Cost	Hyperplane: Accuracy	SEA: Realized Cost	SEA: Accuracy
1.0	1.00	0.6082	1.00	0.7513
1.5	1.31	0.6580	1.48	0.8228
2.0	1.89	0.7461	1.94	0.8975
3.0	2.99	0.9148	2.94	0.9397
4.0	3.87	0.9031	3.91	0.9364
5.0	4.60	0.8872	4.91	0.9318
6.0	5.37	0.8771	5.90	0.9295
8.0	6.70	0.8728	7.86	0.9252
10.0	7.48	0.8699	9.67	0.9221

AULC integrates accuracy over nominal B\in \left[1,10\right], not over realized cost. Learned static AULC is stated as 0.651 ± 0.053 and 0.883 ± 0.004; its budget-by-budget values and all seed-level data were not furnished.

5.5 | Statistical Analysis

The source reports paired Wilcoxon signed-rank statistics [36] for seed-matched differences and Holm [37] adjustment across nine contrasts (*Table 5*). Conditional on all ten nonzero differences favoring ABACUS without ties, the two-sided exact minimum p is $\frac{2}{2^{10}} = 0.001953$; for twenty it is $\frac{2}{2^{20}} = 1.907 \times 10^{-6}$. The $W = 0$ rows are arithmetically compatible with that pattern, and the adaptive AULC means can be checked from the rounded budget table. However, for ten nonzero untied pairs, $W = 6$ has two-sided exact $p = 0.02734$ and $W = 1$ has $p = 0.003906$, whereas the last two rows report 0.161 and 0.0108. Zeros, tied magnitudes, a smaller effective n, or an approximate calculation could explain a different procedure; the table should not call all p values exact until raw differences and software options are provided. Holm decisions for those rows are provisional. Actual paired ranks, static AULC, and recovery distributions cannot be audited without seed-level output. The twenty-block pooled result combines two generator types and is exploratory; per-generator estimates are more interpretable. Demšar [38] concerns multiple datasets, not independent validation of inference from ten seeds on one generated stream.

Table 5. Author-reported paired Wilcoxon statistics (W, reported p, and Holm-adjusted p across nine contrasts; ten seeds per generator and twenty blocks in the exploratory pooled contrast).

Contrast (Paired Wilcoxon Signed-Rank)	W	p	Holm p	Outcome
AULC, pooled ($n = 20$ blocks)	0.0	1.91×10^{-6}	1.72×10^{-5}	ABACUS better (0.8846 vs 0.7672)
AULC, hyperplane ($n = 10$)	0.0	1.95×10^{-3}	1.56×10^{-2}	ABACUS better (0.8530 vs 0.6513)
AULC, SEA ($n = 10$)	0.0	1.95×10^{-3}	1.56×10^{-2}	ABACUS better (0.9163 vs 0.8831)
Accuracy at $B = 3$, hyperplane ($n = 10$)	0.0	1.95×10^{-3}	1.56×10^{-2}	ABACUS better (0.9148 vs 0.6775)
Accuracy at $B = 3$, SEA ($n = 10$)	0.0	1.95×10^{-3}	1.56×10^{-2}	ABACUS better (0.9397 vs 0.8987)
ABACUS vs no-drift ablation, hyperplane	0.0	1.95×10^{-3}	1.56×10^{-2}	ABACUS better (10/10)
ABACUS vs no-drift ablation, SEA	0.0	1.95×10^{-3}	1.56×10^{-2}	ABACUS better (10/10)
ABACUS vs no-budget ablation, hyperplane	6.0	1.61×10^{-1}	1.61×10^{-1}	not significant (Holm)
ABACUS vs no-budget ablation, SEA	1.0	1.08×10^{-2}	2.16×10^{-2}	ABACUS better (8/10)

W, reported p, and Holm-adjusted p across nine contrasts; ten seeds per generator and twenty blocks in the exploratory pooled contrast.

The $W = 0$ p values are consistent with a no-tie exact calculation; the $W = 6$ and $W = 1$ rows require seed-level differences, effective sample size, zero/tie handling, and software options before the test method or adjusted values can be verified.

5.6 | Findings and Scope

The supplied results support a narrower reading than a hard-latency performance claim. At $B = 3$ on both specified synthetic streams, the author-reported adaptive policy substantially exceeds the learned static cascade at approximately the same assigned mean cost; the no-refit ablation identifies recent-window model refresh as the principal mechanism in this comparison. Feedback mainly tracks the mean proxy target. Increasing the target above $B = 3$ hurts this particular policy because it apparently escalates to a less useful forest, but the feasible optimum under a larger upper bound cannot get worse by retaining the earlier configuration. The approximate ten-per-run alarm count flags material maintenance work, not a measured false-alarm rate or low operational overhead.

The result depends on the specified 200-item label delay, a 500-label refit window, and abrupt synthetic changes. Label delay, prevalence, gradual or recurring shift, and adversarial confidence shaping were not varied. The designated pre-noise truth is available in a synthetic generator but may be unobservable in deployment. An independent evaluation should include a refitted standalone L1 logistic model, a matched refresh schedule without drift detection, a label-only detector arm, and an alarm/event matching rule. The source's confidence-entropy and delayed-accuracy detector was the only detector exercised; input-channel signals, general pool updates, stage certification, and preemption were not. Assigned inference costs exclude refit time and serving contention, so neither total compute savings nor end-to-end latency savings can be inferred from the present table.

Evidence status: The manuscript contains a coherent set of author-reported synthetic results, figures, and summary statistics. The named exp_15.py, stream_gen.py, exp_15_results. JavaScript Object Notation (JSON), and seed-level traces were not present in the reviewed source or accessible project folder. Publication-ready reproduction and stronger claims are contingent on supplying these exact artifacts and regenerating all tables, plots, tests, and cost accounts.

6 | Discussion: Limitations and Counterarguments

6.1 | Limitations

The architectural deadline claim depends on actual, certified bounds for queueing, routing, execution, and fallback on a declared serving platform. The reported study has no elapsed-time trace, so it cannot estimate

tail latency or guarantee a deadline. Proxy confidence can be misleading under shift, and alarms from quality perturbations can cause needless refits. The only evaluated streams are synthetic tabular tasks; large vision or language models introduce batching, loading, and preemption issues not tested here. The three-loop controller's joint stability also remains unanalysed.

Assigned feature-and-inference costs are ordinal proxy units, not CPU milliseconds, FLOPs, joules, or prices. The source's total path-cost notation was inconsistent with the endpoint values 1, 3, and 10; *Eq. (1)*-now uses incremental costs, subject to code-level confirmation of shared work. Model-refresh cost was counted in alarms but excluded from the cost objective, making cost-effectiveness claims conditional on a more complete account.

6.2 | Alternative Viewpoints and Responses

A simple refitted model could explain much of the improvement. The supplied no-refit ablation shows that refresh matters, but it does not distinguish a cascade from a periodically refitted L1 logistic model at the same average cost. A matched refitted single-level baseline and an identical model-update schedule would separate routing value from refresh value. Any such comparison must charge refitting and feature-acquisition effort consistently.

Mean-cost control also differs from a per-item response bound. Tail latency and queueing matter to time-sensitive services [6], but the present experiment contains no wall-clock latency observations and does not compare worst-case admission against expected-latency scheduling. This is the still-open H3 question. Similarly, its alarm counts cannot be called false-alarm rates without a defined event-matching rule across abrupt changes, continuous rotation, and injected data-quality bursts.

7 | Conclusion and Future Directions

We formulated a conditional deadline-aware cascade and examined an implemented subset that adapts to two synthetic shifts under an assigned average-cost target. The author-reported ten-seed campaign gives accuracy of about 0.915 and 0.940 at proxy mean cost near 3, compared with 0.678 and 0.899 for a learned static cascade at similar targets. Reported ablations show that drift-triggered refits, rather than the mean-cost feedback, account for most of this operating-point improvement. The table's non-monotone budget response concerns the configured controller, not an optimal frontier. These findings are provisional pending the named code and raw results. They do not verify a hard deadline, deployed latency, or an independent cascade benefit over a matched refitted single model.

The immediate validation step is to supply exp_15.py, stream_gen.py, exp_15_results.JSON, seeded streams, and figure generation scripts, with precise timing of label availability, refits, and cost accounting. The next experiment should add the matched refitted L1 and label-only detector arms, price maintenance, and stress label sparsity and gradual or recurring shifts. Only after implementing a serving stack with bounded queueing, stage and overhead times, safe fallback, and a declared load envelope can the hard-deadline admission rule and H3 be evaluated against a mean-latency alternative.

Data Availability

The source states that synthetic streams were generated by seeded code. No external dataset is named. The listed generator, seed list, and raw result JSON were not furnished in the reviewed materials. Authors should provide the stream-generation version, parameters, per-seed outputs, and timing of data-quality events to support reproduction.

Code Availability

The source names exp_15.py, stream_gen.py, exp_15_results.JSON, exp_15_f1.png, and exp_15_f2.png as a submission package. Only the manuscript with two embedded figures was available in the reviewed folder; the scripts, raw JSON, and standalone image files should be supplied with the final submission.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflicts of Interest

The authors declare no conflicts of interest.

Author Contributions

R. R. A.: Conceptualization, methodology, software, and writing original draft.

V. P. S.: Formal analysis, validation, supervision, and writing review and editing.

All authors read and approved the final manuscript.

Generative AI disclosure

AI-assisted drafting tools were used in preparing the manuscript text under the direction and scientific responsibility of the authors; all methodological choices, claims, experimental results, and final content were determined and verified by the authors.

References

[1] Babu, S., & Widom, J. (2001). Continuous queries over data streams. *ACM Sigmod Record*, *30*(3), 109–120. https://doi.org/10.1145/603867.603884

[2] Akidau, T., Balikov, A., Bekiroglu, K., Chernyak, S., Haberman, J., Lax, R., Mills, D., & Paul Nordstrom, S. W. (2013). MillWheel: Fault-tolerant stream processing at internet scale. *In procceding 6th biennial conf* (pp. 1033–1044). Innovative Data Systems Research (CIDR). https://www.vldb.org/pvldb/vol6/p1033-akidau.pdf

[3] Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández Moctezuma, R. J., & Lax, R. (2015). The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, *8*(12), 1792–1803. https://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf %28Google

[4] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache flink™: Stream and batch processing in a single engine. *IEEE.*, *38*(4), 28–38. https://research.tudelft.nl/en/publications/apache-flink-stream-and-batch-processing-in-a-single-engine/

[5] Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., & Stonebraker, M. (2003). Load shedding in a data stream manager. *Proceedings 2003 VLDB conference* (pp. 309–320). Elsevier. https://doi.org/10.1016/B978-012722442-8/50035-5

[6] Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM*, *56*(2), 74–80. http://doi.acm.org/10.1145/2408776.2408794

[7] Viola, P., & Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, *57*(2), 137–154. https://doi.org/10.1023/B:VISI.0000013087.49260.fb%0A%0A

[8] Teerapittayanon, S., McDanel, B., & Kung, H. T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd international conference on pattern recognition (ICPR)* (pp. 2464–2469). IEEE. https://doi.org/10.48550/arXiv.1709.01686

[9] Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., & Weinberger, K. Q. (2018). *Multi-scale dense networks for resource efficient image classification*, *1*, 1–14. https://doi.org/10.48550/arXiv.1703.09844

[10] Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., & Stoica, I. (2013). BlinkDB: Queries with bounded errors and bounded response times on very large data. *Proceedings of the 8th ACM european conference on computer systems* (pp. 29–42). ACM. https://doi.org/10.1145/2465351.2465355

[11] Gama, J., Žliobaite, I. ., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, *46*(4), 1–37. https://doi.org/10.1145/2523813

[12] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, *31*(12), 2346–2363. https://doi.org/10.1109/TKDE.2018.2876857

[13] Quiñonero Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (2008). *Dataset shift in machine learning*. Mit Press. https://www.researchgate.net/publication/234817678

[14] Sugiyama, M., & Kawanabe, M. (2012). *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press. https://doi.org/10.7551/mitpress/9780262017091.001.0001

[15] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[16] Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. *Proceedings of the sixth ACM sigkdd international conference on knowledge discovery and data mining* (pp. 71–80). ACM. https://doi.org/10.1145/347090.347107

[17] Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. *Proceedings of the seventh ACM sigkdd international conference on knowledge discovery and data mining* (pp. 97–106). ACM. https://doi.org/10.1145/502512.502529

[18] Street, W. N., & Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. *Proceedings of the seventh ACM sigkdd international conference on knowledge discovery and data mining* (pp. 377–382). ACM. https://doi.org/10.1145/502512.502568

[19] Oza, N. C., & Russell, S. J. (2001). Online bagging and boosting. *International workshop on artificial intelligence and statistics* (pp. 229–236). Proceedings of Machine Learning Research (PMLR). https://proceedings.mlr.press/r3/oza01a.html

[20] Kotler, J., & Maloof, M. (2003). Dynamic weighted majority: A new ensemble method for tracking concept drift. *IEEE international conference on data mining* (pp. 123–130). IEEE Computer Society, Washington, DC, USA. https://dl.acm.org/doi/pdf/10.5555/1314498.1390333

[21] Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, *22*(10), 1517–1531. https://doi.org/10.1109/TNN.2011.2160459

[22] Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*(1/2), 100–115. https://doi.org/10.2307/2333009

[23] Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. *Brazilian Symposium on Artificial Intelligence* (pp. 286–295). Springer. https://doi.org/10.1007/978-3-540-28645-5_29%0A%0A

[24] Baena Garcia, M., del Campo Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales Bueno, R. (2006). Early drift detection method. *Fourth international workshop on knowledge discovery from data streams* (Vol. 6, pp. 77–86). ACM. https://www.researchgate.net/publication/245999704

[25] Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM international conference on data mining* (pp. 443–448). SIAM. https://doi.org/10.1137/1.9781611972771.42

[26] Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., & Seidl, T. (2010). Moa: Massive online analysis, a framework for stream classification and clustering. *Proceedings of the first workshop on applications of pattern analysis* (pp. 44–50). PMLR. https://www.researchgate.net/publication/220320178

[27] Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, *17*(3), 73. https://doi.org/10.1609/aimag.v17i3.1232

[28] Hellerstein, J. M., Haas, P. J., & Wang, H. J. (1997). Online aggregation. *Proceedings of the 1997 ACM sigmod international conference on management of data* (pp. 171–182). ACM. https://doi.org/10.1145/253260.253291

[29] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2017). Clipper: A low-latency online prediction serving system. *14th usenix symposium on networked systems design and implementation (NSDI 17)* (pp. 613–627). USENIX Association. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[30] Littlestone, N., & Warmuth, M. K. (1994). The weighted majority algorithm. *Information and Computation*, *108*(2), 212–261. https://doi.org/10.1006/inco.1994.1009

[31] Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge University Press. https://www.researchgate.net/profile/Gabor-Lugosi/publication/220690817

[32] Bubeck, S., & Cesa Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® In Machine Learning*, *5*(1), 1–122. https://doi.org/10.1561/2200000024

[33] Shalev Shwartz, S. (2025). Online learning and online convex optimization. *Foundations and Trends®In Machine Learning*, *4*(2), 107–194. https://doi.org/10.1561/2200000018

[34] Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324%0A%0A

[35] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. https://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

[36] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 80–83. https://doi.org/10.2307/3001968

[37] Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 65–70. https://doi.org/10.2307/4615733

[38] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*(Jan), 1–30. https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf