Non-Stationary Environments: A Benchmark for Stochastic Optimization

Ivan Akinfiev*
Elizaveta Tarasova*
Pavel Averin*
Oleg Granichin*
ivan.akinfiev@spbu.ru
elizaveta.tarasova@spsu.ru,el.u.tarasova@gmail.com
st107211@student.spbu.ru"
o.granichin@spsu.ru
Saint Petersburg State University
Saint-Petersburg, Russia

Abstract

Stochastic optimization in non-stationary environments is a key component of modern AI and data-driven systems, including online learning, adaptive model tuning, reinforcement learning and bandit optimization, as well as real-time decision-making under changing conditions. In such settings, algorithms operate under uncertainty: observations are noisy (including heavy-tailed, quantized, and correlated noise), and the objective may change over time. The goal is therefore not only asymptotic convergence, but robust tracking of a drifting global optimum. We present a benchmark designed for reproducible evaluation of optimization methods under non-stationarity. The environment is parameterized by drift models of the optimum (e.g., linear drift, random walk, cyclic drift, abrupt shocks, adaptive drift), controllable loss-landscape geometry (including tunable smoothness and conditioning), and families of noise regimes. The benchmark follows a "clean environment" principle: the only interaction point is an oracle that returns noisy observations under a specified access configuration (gradient access; value-only access; and joint access to function values and gradients). This design enables fair comparison across a broad spectrum of methods, including zero-order approaches (e.g., FDSA, SPSA-based consensus, accelerated SPSA, Kiefer–Wolfowitz, etc.) and first-order schemes (e.g., SGD, AMSGrad, subgradient methods, etc.). Performance is evaluated using consistent error metrics (including geometry-aware variants), cumulative regret, and dynamic measures of shock response (e.g., recovery time), while query efficiency is explicitly accounted for in value-only settings. The experimental protocol supports reliable reporting via 95

*Both authors contributed equally to this research.

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Keywords

Stochastic optimization, Non-stationary environments, Benchmark, Reproducible evaluation, Drift, Heavy-tailed noise, Zeroth-order methods, First-order methods, Tracking

ACM Reference Format:
Ivan Akinfiev, Elizaveta Tarasova, Pavel Averin, and Oleg Granichin. 2018. Non-Stationary Environments: A Benchmark for Stochastic Optimization. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In many applied stochastic optimization problems, the objective is *non-stationary*: the location of the minimizer changes over time due to data drift, evolving external conditions, time-varying workloads, or unstable measurement channels. In such regimes, the relevant goal is not "static convergence" but *online tracking of a drifting optimum* under limited feedback and noise. Consequently, the primary performance target is tracking quality over time rather than asymptotic behavior. The appropriate theoretical language is provided by online optimization and regret-type criteria. In particular, *dynamic regret* compares the cumulative loss of an algorithm to the best admissible comparator trajectory that is allowed to vary over time, thereby formalizing tracking as a standalone objective [8, 9, 12, 13, 15, 34–36].

Despite the mature theory, a practical gap remains between formal non-stationary models and the engineering need to *reproducibly* compare algorithms under controlled dynamic regimes. Existing optimization suites are largely designed for stationary settings and do not provide a standardized tracking protocol for drifting optima. Moreover, support for realistic noise models—in particular, *correlated* and *heavy-tailed* noise—is typically absent or only partially available. This is summarized in Table 1: COCO, HPOBench, and LibOPT are effectively focused on stationary evaluation, while Nevergrad offers only limited coverage of non-ideal noise and does not define a unified tracking protocol [1–4]. As a result, algorithm comparisons often depend on implicit assumptions about the environment, ambiguous accounting of query costs (especially in zeroth-order regimes), and mismatched notions of available feedback.

Table 1: Coverage of major optimization suites and the proposed benchmark WIND

Suite	Regime	Drift	Correlated noise	Heavy-tailed noise
COCO [1]	Stationary	No	No	No
Nevergrad [4]	Stationary	No	Limited	Limited
HPOBench [2]	Stationary	No	No	No
LibOPT [3]	Stationary	No	No	No
NAME (ours)	Dynamic	Yes	Yes	Yes

In this work, we introduce the benchmark WIND, designed for reproducible evaluation of algorithms in non-stationary stochastic optimization. The benchmark targets tracking of a drifting global minimizer in a dynamic environment and specifies a family of instances parameterized by three orthogonal axes of difficulty. The first axis captures *drift of the optimum*: both smooth regimes (e.g., gradual drift) and shock-type changes (e.g., jumps) are included, enabling separation of robustness and recovery speed after regime shifts. The second axis controls *landscape geometry*: function families with tunable smoothness, conditioning, and constraints, which exposes sensitivity to ill-conditioned and high-dimensional regimes. The third axis specifies *observation noise*, including correlated and heavy-tailed distributions, as well as quantization and multiplicative distortions.

The architecture of WIND (Worst-case INspired Drift benchmark) follows the principle of a "clean environment": the benchmark provides a generative environment and an oracle, while the optimizer is external. This makes it possible to compare broad algorithm classes within a single interaction contract, ranging from first-order gradient methods to zeroth-order heuristics and evolutionary strategies. We therefore adopt a unified oracle interface with FO (First-Order), ZO (Zero-Order), Hybrid (FO and ZO), Offline modes. Zeroth-order support is essential because in many applications only function values are available, and performance is determined not only by tracking accuracy but also by the cost of oracle queries [18, 23].

A central component of WIND is a standardized *interaction protocol* ensuring validity and interpretability of comparisons. The protocol fixes the notion of a time step and the update order of the environment state, unambiguously defines what constitutes an "oracle call" and how query indices are counted, and specifies budget accounting across feedback modes. Importantly, it enforces an *information barrier* that prevents access to the true optimum location and other privileged state variables. The protocol is formalized by a set of operational principles, including temporal consistency (a fixed environment state within a step), correctness of regret computation, tail-based evaluation rules (choice of T_{tail}), metric direction semantics (min/max), reproducibility requirements (seeds and offline traces), statistical stability (confidence intervals), and fair query accounting (critical for ZO settings). An interaction architecture diagram and the full set of protocol rules are provided in Section 2.

Contributions.

(1) We propose a unified oracle benchmark supporting FO, ZO, Hybrid, Offline feedback under a single interaction contract.
(2) We construct parametric suites along drift, geometry, and noise axes, explicitly including correlated and heavy-tailed regimes.
(3) We provide protocol guarantees for valid comparison, including temporal consistency, an information barrier, and fair query-budget accounting.
(4) We define tracking metrics and reporting (including dynamic regret, tail-based errors, shock-response measures, and query efficiency) with fixed aggregation rules and statistical summaries.
(5) We release reference baselines and a reproducible runner, reducing the cost of algorithm development and validation by enabling rapid stress-testing prior to labor-intensive theoretical analysis.

The remainder of the paper formalizes the benchmark and interaction protocol (Section 2), specifies drift/landscape/noise families and metrics (Section 3), and reports baseline implementations and experimental results on standardized suites (Sections 4, 5).

2 Benchmark Definition and Interaction Protocol

This section formalizes the benchmark WIND and the interaction protocol between an evaluated optimizer and a dynamic environment. The goal is to make algorithm comparisons valid by construction: identical information across methods, comparable oracle-access cost, no leakage of latent state, reproducibility, and well-defined tracking metrics.

2.1 Interaction Architecture

The benchmark WIND follows the "clean environment" principle: the optimizer never interacts directly with the components defining the latent dynamics of the task. The only interface exposed to the optimizer is an *oracle* that returns observations in a fixed format.

Figure 1 illustrates the interaction architecture:

- Environment maintains the latent state at time step t, including the (hidden) optimum location θ_t.
- Drift specifies the evolution law for θ_t.
- Landscape specifies the objective family and its geometry.
- Noise specifies the observation noise model.
- Oracle implements the feedback contract (FO/ZO/Hybrid/Offline) and performs query-cost accounting.
- Metrics have privileged access to θ_t *only* for evaluation and never expose it to the optimizer.

We consider an arbitrary *evaluated optimizer* (candidate method) $\mathcal{A}$, which generates oracle queries over time. Depending on the feedback mode, $\mathcal{A}$ may issue a single query per time step (typical in FO) or multiple queries within a step (typical in ZO, where gradient surrogates require repeated function evaluations).

2.2 Benchmark Overview

Episodes and configuration. A single run of WIND is an *episode* of length T, specified by the configuration

$$\mathcal{E} = (\text{mode, drift, landscape, noise, } d, T, \text{ seed}), \quad (1)$$

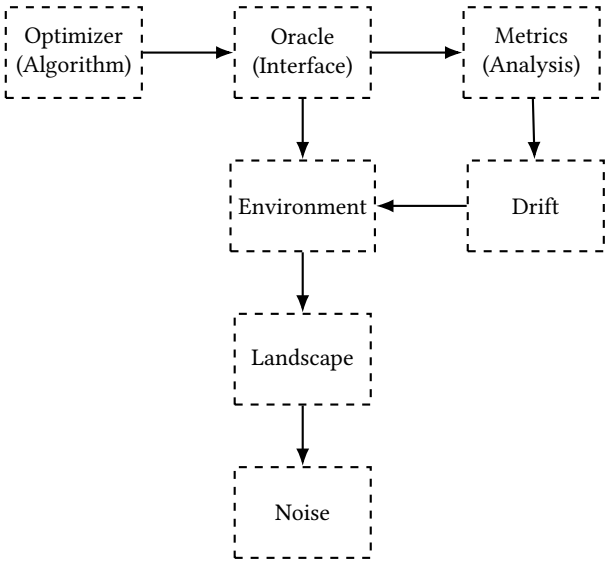

Figure 1: Interaction architecture of WIND

where mode $\in$ {FO, ZO, Hybrid, Offline} denotes the feedback mode, drift is the drift model for the latent optimum, landscape specifies the objective family and geometry parameters, noise specifies the observation noise model, d is the decision-space dimension, and seed controls reproducibility (including the option to use separate seeds for drift and noise).

Dynamic objective. At each time step $t = 1, \ldots, T$, the environment defines a loss function

$$L_t(x) \equiv L(x; \theta_t, \text{landscape}), \tag{2}$$

where $\theta_t \in \mathbb{R}^d$ is a latent optimum location. The optimum evolves according to the drift model:

$$\theta_{t+1} \sim \text{Drift}(\theta_t; \text{drift}, \text{seed}). \tag{3}$$

The optimizer $\mathcal{A}$ selects decisions and receives oracle feedback corrupted by noise according to noise.

Trajectories available for analysis. During an episode, WIND produces the trajectories of queries and oracle responses used for evaluation and reporting. In particular, we record:

- the sequence of optimizer queries (either $\{x_t\}$ or $\{x_{t,k}\}$ if multiple oracle calls are issued within step t);
- the corresponding oracle responses (values and/or gradients) with explicit linkage to time step and oracle-call counter;
- the episode configuration $\mathcal{E}$;
- per-step metric values and aggregate summaries (including tail-aggregated metrics);
- seeds and identifiers of randomness sources;
- offline traces (in the Offline mode) to enforce identical comparison conditions.

2.3 Oracle Interface and Feedback Modes

Oracle contract and query accounting. The oracle exposes a single query operation that maps a point $x \in \mathbb{R}^d$ to an observation whose content depends on the feedback mode. To account for information-access cost, we introduce a *global oracle-call counter* $q \in \mathbb{N}$, incremented at every oracle call:

$$q \leftarrow q + 1. \tag{4}$$

An episode is associated with a query budget Q, enforced as $q \le Q$. This is essential in ZO settings, where a single "logical" optimization step may require multiple oracle calls.

To describe within-step querying (common in ZO), we index queries as $x_{t,k}$ where $k = 1, \ldots, K_t$ enumerates oracle calls within time step t. Each oracle call $\textsc{Oracle}(x_{t,k})$ increments q and consumes budget.

Feedback modes (FO / ZO / Hybrid / Offline). The benchmark supports four feedback modes:

- **First-Order (FO).** The oracle returns a noisy gradient only:
$$g_t(x) = \nabla L_t(x) + \xi_{t,q}^{(g)}, \tag{5}$$
and does not provide the function value $L_t(x)$ to the optimizer.
- **Zero-Order (ZO).** The oracle returns a noisy function value only:
$$y_t(x) = L_t(x) + \xi_{t,q}^{(y)}, \tag{6}$$
and does not provide gradients. Any gradient surrogate (finite differences, SPSA-type estimators, etc.) must be constructed by the optimizer via additional oracle calls, hence comparison must explicitly account for q.
- **Hybrid.** The oracle returns both value and gradient:
$$y_t(x) = L_t(x) + \xi_{t,q}^{(y)}, \qquad g_t(x) = \nabla L_t(x) + \xi_{t,q}^{(g)}, \tag{7}$$
with (by default) independent noise channels
$$\xi_{t,q}^{(y)} \perp \xi_{t,q}^{(g)}, \tag{8}$$
unless a correlated design is explicitly specified by noise.
- **Offline.** To ensure strictly comparable conditions, the benchmark may provide offline traces that fix the latent trajectory (e.g., $\{\theta_t\}_{t=1}^T$) and/or the noise realization, so that different optimizers are evaluated under identical dynamics.

Correct oracle usage and query-cost comparability. In WIND, the unit cost is an *oracle call*. Any call increments q and consumes budget. This removes ambiguity in ZO regimes, where methods may use multiple function evaluations per optimization update. When needed, we report query-normalized performance measures. For instance, given a tracking error metric Err, a simple query-normalized indicator can be written as

$$\text{QE} = \frac{\text{Err}}{Q}, \tag{9}$$

so that accuracy and information-access cost are jointly reflected.

2.4 Protocol Design and Validity Guarantees

Comparisons in non-stationary environments are sensitive to operational details: the definition of a time step, the moment of latent-state updates, query accounting, and the information available to the optimizer. The WIND protocol therefore fixes a set of invariants that exclude the most common sources of invalid evaluation.

Time-step semantics and temporal consistency. Each time step t corresponds to a fixed latent environment state, including θ_t, and a fixed loss function $L_t(\cdot)$. All oracle calls issued within the same time step must be evaluated against the *same* θ_t. Concretely, if the optimizer issues a sequence $\{x_{t,k}\}_{k=1}^{K_t}$ at time t, then the oracle responses must satisfy

$$y_t(x_{t,k}) = L_t(x_{t,k}) + \xi_{t,q}^{(y)}, \qquad g_t(x_{t,k}) = \nabla L_t(x_{t,k}) + \xi_{t,q}^{(g)}, \quad (10)$$

and the transition $\theta_t \to \theta_{t+1}$ may occur only after the completion of step t. This prevents mixing different latent states within a single step and ensures that per-step evaluation is well-defined.

Information barrier. The optimizer $\mathcal{A}$ has no access to θ_t or any privileged variables that would allow reconstruction of the latent dynamics. The only information available to $\mathcal{A}$ is the oracle feedback defined by the chosen mode. Privileged access to θ_t is permitted only for metric computation and is never used in the decision rule that generates queries.

Consistency of dynamic-regret evaluation. Tracking performance is naturally expressed via regret-type criteria. In particular, we define dynamic regret as

$$R_T \;=\; \sum_{t=1}^{T} \Big(L_t(x_t) - L_t(\theta_t) \Big), \quad (11)$$

where x_t is the representative decision of the optimizer at time t (as specified by the evaluation protocol when multiple within-step queries exist). A critical requirement is that both terms $L_t(x_t)$ and $L_t(\theta_t)$ are evaluated under the *same* latent state θ_t; otherwise, regret can be artificially distorted.

Tail-based evaluation. To reduce sensitivity to transient effects, we also evaluate selected metrics on the tail segment of the trajectory. Let

$$T_{\text{tail}} = \lfloor \gamma T \rfloor, \qquad \gamma \in (0, 1), \quad (12)$$

and let M_t denote a per-step metric. The tail-aggregated metric is defined as

$$M_{\text{tail}} \;=\; \frac{1}{T_{\text{tail}}} \sum_{t=T-T_{\text{tail}}+1}^{T} M_t. \quad (13)$$

This rule is fixed by the benchmark specification to prevent post-hoc selection of evaluation windows.

Metric direction semantics. For each reported metric, the benchmark specifies whether it is to be minimized or maximized. This removes ambiguity when constructing leaderboards and summary tables combining heterogeneous metrics (tracking errors, regret, query efficiency, shock-response indicators, etc.).

Reproducibility and statistical reporting. Reproducibility is controlled at the episode level by seed in (1), with explicit identification of randomness sources and optional offline traces. For statistically meaningful comparisons, results are aggregated over multiple independent runs and reported with confidence intervals (e.g., 95%). The benchmark fixes aggregation rules and minimal requirements on estimate stability, preventing conclusions based on single trajectories or insufficient replication.

Fair resource accounting. Finally, the protocol enforces fair accounting through the oracle-call budget Q. This is essential for comparability across feedback modes, particularly in ZO settings where performance is strongly shaped by the number of oracle calls required to obtain informative gradient surrogates.

3 Benchmark Specification

Table 1 highlights a structural gap in widely used optimization suites: most platforms are designed around *stationary* objectives and do not provide a unified specification that simultaneously supports (i) drifting optima, (ii) correlated noise, and (iii) heavy-tailed noise under a single oracle contract. In WIND, we therefore separate the benchmark specification into orthogonal components—Drift, Landscape/Geometry, and Noise—and couple them with a fixed set of metrics and reporting rules consistent with the interaction protocol in Section 2.

3.1 Drift Models (Non-Stationary Dynamics)

Non-stationarity is introduced through a latent, time-varying optimum $\theta_t \in \mathbb{R}^d$. Over an episode of length T, the environment generates a trajectory $\{\theta_t\}_{t=1}^{T}$, and the transition $\theta_t \mapsto \theta_{t+1}$ is determined by a drift family with explicit parameters. We include the following drift families.

Stationary (control regime):

$$\theta_{t+1} = \theta_t. \quad (14)$$

Linear drift (smooth trend):

$$\theta_{t+1} = \theta_t + v\,u, \qquad \|u\|_2 = 1, \qquad v \in [0, 0.5]. \quad (15)$$

This regime models persistent gradual changes.

Random walk (smooth but unpredictable drift):

$$\theta_{t+1} = \theta_t + \sigma_d\,\varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, I), \qquad \sigma_d \in [0.01, 0.2]. \quad (16)$$

Cyclic drift (structured non-stationarity):

$$\theta_t = \theta_0 + A \sin\Big(\frac{2\pi t}{P}\Big), \qquad A \in [0.1, 2.0], \qquad P \in [50, 500]. \quad (17)$$

This regime captures periodic effects and seasonal patterns.

Jump process (shocks):

$$\theta_{t+1} = \theta_t + \Delta\,\mathbb{I}[t \bmod P = 0], \qquad \|\Delta\| \in [0.5, 5.0], \qquad P \in [100, 1000]. \quad (18)$$

Adaptive drift (response to the optimizer):

$$\theta_{t+1} = \theta_t - \alpha\,\text{sign}(x_t - \theta_t), \qquad \alpha \in [0, 1]. \quad (19)$$

Here the drift depends on the optimizer trajectory and represents an adverse dynamic regime.

Sparse drift (partial-coordinate changes). Let $S_t \subset \{1, \ldots, d\}$ be an active coordinate set with $|S_t| = k$, $k \in [1, d]$. Then

$$\theta_{t+1}[i] = \begin{cases} \theta_t[i] + \sigma_d\,\xi_{t,i}, & i \in S_t,\ \xi_{t,i} \sim \mathcal{N}(0, 1), \\ \theta_t[i], & i \notin S_t. \end{cases} \quad (20)$$

Reproducibility. The benchmark configuration allows separate randomness sources for drift and noise (i.e., separate seeds and generators). This enables controlled experiments where the same latent trajectory $\{\theta_t\}$ is paired with different noise realizations, and vice versa.

3.2 Landscape Families and Geometry Control

The LANDSCAPE component specifies the family of loss functions $L_t(\cdot)$ and the associated geometry/conditioning parameters. In the main implementation of WIND, the available landscape classes are: `Quadratic`, p-`Norm`, `Rosenbrock`, `MultiExtremal`, and `Robust`. Riemannian/Stiefel landscapes are listed as potential extensions in the specification but are not included in the current codebase.

A central design choice is that, for the base families, the ground-truth optimum is consistent with the latent state:

$$x_t^\star \in \arg\min_x L_t(x), \qquad \text{and} \qquad x_t^\star = \theta_t. \tag{21}$$

This alignment is essential for dynamic regret and distance-to-optimum metrics, while preserving a strict information barrier (the optimizer does not observe θ_t).

p-Norm family (Hölder-controlled geometry). Let $p = \rho + 1$ with $\rho \in [0.05, 1]$. The loss is defined as

$$L_t(x) = \frac{1}{p}\left\|M_\kappa(x - \theta_t)\right\|_p^p, \qquad p = \rho + 1, \tag{22}$$

where M_κ is a linear operator controlling anisotropy and conditioning, and $\kappa \geq 1$ is the conditioning parameter. In particular, M_κ can be implemented as a rotation followed by a diagonal scaling that realizes a prescribed ratio of principal axes.

Quadratic family (conditioning control):

$$L_t(x) = \frac{1}{2}(x-\theta_t)^\top A(x-\theta_t), \qquad A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d), \qquad \kappa = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{23}$$

This family provides a transparent knob for valley geometry through κ.

Rosenbrock-type family (non-convex valley). We include Rosenbrock-type landscapes that generate narrow curved valleys and violate global strong convexity. Non-stationarity is induced by coupling the landscape to θ_t, so that the minimizer moves in time while the optimizer must track it under noise.

MultiExtremal family (multimodality). This family introduces multiple local basins of attraction and is used to probe sensitivity of methods (especially ZO heuristics and population-based optimizers) to multimodality under drift.

Robust family (robust variants). Robust landscapes are used to disentangle geometric difficulty from observation irregularities (including heavy tails and outliers), and to test stability of tracking under adverse feedback.

3.3 Noise Models

The NOISE component specifies the stochastic perturbations applied to oracle feedback in FO/ZO/Hybrid modes. We include Gaussian (control), heavy-tailed, correlated, quantized, multiplicative, and sparse regimes. Below, ξ denotes a generic noise term.

Additive Gaussian noise: $\xi \sim \mathcal{N}(0, \sigma^2), \qquad \sigma \in [0.01, 1.0]$.

Heavy-tailed noise (Pareto-type). A standard construction uses $u \sim \mathcal{U}(0,1)$ and a random sign $s \in \{\pm 1\}$:

$$\xi = s\Big((1-u)^{-1/\alpha} - 1\Big), \qquad \alpha \in [1.5, 3.0]. \tag{24}$$

This regime models outliers and violates sub-Gaussian assumptions.

Multiplicative noise:

$$\xi = L_t(x)\,\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma_m^2), \qquad \sigma_m \in [0.05, 0.5]. \tag{25}$$

Quantization noise:

$$\xi = \Delta \cdot \mathrm{round}\big(L_t(x)/\Delta\big) - L_t(x), \qquad \Delta \in [2^{-8}, 2^{-2}]. \tag{26}$$

Correlated noise (AR(1)):

$$\xi_t = \phi\,\xi_{t-1} + \sqrt{1-\phi^2}\,\eta_t, \qquad \eta_t \sim \mathcal{N}(0, \sigma^2), \qquad \phi \in [0, 0.9]. \tag{27}$$

Sparse noise (Bernoulli-gated):

$$\xi = \begin{cases} \mathcal{N}(0, \sigma^2), & \text{with probability } p, \\ 0, & \text{otherwise,} \end{cases} \qquad p \in [0.1, 1.0]. \tag{28}$$

Noise injection in FO/ZO/Hybrid oracle feedback. Under FO, the oracle returns noisy gradients,

$$g_t(x) = \nabla L_t(x) + \xi_t^{(g)}, \tag{29}$$

while values are not accessible. Under ZO, the oracle returns noisy values,

$$y_t(x) = L_t(x) + \xi_t^{(y)}, \tag{30}$$

while gradients are not accessible. Under Hybrid, both components are available and the dependence structure between $\xi_t^{(g)}$ and $\xi_t^{(y)}$ is dictated by the selected noise scenario.

Implication for ZO finite-difference estimators. In ZO mode, value noise induces additional instability in gradient surrogates. For example, the central-difference estimator along direction u with step h reads

$$\widehat{g}_t(x;u) = \frac{\big(L_t(x+hu) + \xi(x+hu)\big) - \big(L_t(x-hu) + \xi(x-hu)\big)}{2h}\,u, \tag{31}$$

so the effective noise is amplified by $1/h$, which becomes critical under heavy tails and temporal correlation.

3.4 Metrics and Reporting

The metrics in WIND are designed for *tracking* a drifting optimum rather than for static convergence. Metrics are computed along the trajectory $\{x_t\}$ and aggregated using fixed reporting rules.

Step-level decision x_t (including ZO methods). For each time step t, the benchmark defines a single decision x_t. If a method performs multiple internal oracle calls within the same step (common in ZO methods such as SPSA or CMA-ES), metrics use the *final* iterate produced at that step:

$$x_t \;\equiv\; x_{t,K_t}. \tag{32}$$

Tracking error (geometry-consistent). A primary accuracy measure is the distance to the latent optimum. For the p-norm geometry,

$$\mathrm{Err}_{p,\mathrm{tail}} = \frac{1}{T_{\mathrm{tail}}}\sum_{t=T-T_{\mathrm{tail}}+1}^{T} \|x_t - \theta_t\|_p, \qquad T_{\mathrm{tail}} = \lfloor \gamma T \rfloor, \qquad \gamma = 0.2. \tag{33}$$

For quadratic geometry, we also report the A-metric:

$$\mathrm{Err}_{A,\mathrm{tail}} = \frac{1}{T_{\mathrm{tail}}}\sum_{t=T-T_{\mathrm{tail}}+1}^{T} \sqrt{(x_t - \theta_t)^\top A (x_t - \theta_t)}. \tag{34}$$

Dynamic regret. Dynamic regret measures cumulative loss relative to the instantaneous optimum trajectory:

$$R_T = \sum_{t=1}^{T}\Big(L_t(x_t) - L_t(\theta_t)\Big). \tag{35}$$

The interaction protocol guarantees that both terms are evaluated under the same latent state θ_t and that θ_t is never exposed to the optimizer.

Shock response and time-to-recover (TTR). For jump-type drift, we quantify recovery speed after a shock at time t_0. Let $\varepsilon > 0$ be a fixed benchmark constant. We define

$$\mathrm{TTR}(\varepsilon) = \min\Big\{\tau \geq 0 : \ \|x_{t_0+\tau} - \theta_{t_0+\tau}\| \leq \varepsilon\Big\}, \tag{36}$$

and aggregate $\mathrm{TTR}(\varepsilon)$ over all shock times within an episode (e.g., mean and quantiles). The choice of norm is specified by the metric configuration (defaulting to the geometry-consistent norm for the given landscape family).

Query efficiency. Since oracle access is the fundamental resource in WIND, we also report query-normalized performance. Let Q be the number of oracle calls consumed in an episode. Representative indicators include

$$\mathrm{QE}_{\mathrm{err}} = \frac{\mathrm{Err}_{\mathrm{tail}}}{Q}, \qquad \text{and} \qquad \mathrm{QE}_{\mathrm{reg}} = \frac{R_T}{Q}, \tag{37}$$

depending on whether the primary objective is tail tracking error or cumulative dynamic regret.

Standard reporting and aggregation. Results are reported over multiple independent runs (distinct seeds) with fixed aggregation rules and confidence intervals (e.g., 95%). In addition to overall summaries, WIND mandates stratified reporting along the main axes of difficulty: drift family, noise family, landscape/geometry, and feedback mode. Tail evaluation (with $\gamma = 0.2$), the direction of each metric (min/max), and the oracle-call accounting are fixed by specification and do not depend on the evaluated method.

4 Experiments

We begin with infrastructure validation before algorithm evaluation. A suite of 16 stress tests verifies scientific invariants required for valid dynamic regret analysis. These tests enforce non-negotiable properties: all landscapes satisfy $L(\theta_t, \theta_t) = 0$ exactly; the environment locks its state during oracle queries so every evaluation within a step observes identical θ_t; observations never leak θ_t to optimizers; Lyapunov metrics use the correct $\ell_{\rho+1}$ geometry for ρ-Hölder smooth gradients; and offline oracles replay identical trajectories across runs. Passing these tests guarantees that regret values are mathematically meaningful, and comparisons reflect algorithmic behavior rather than implementation artifacts.

After validation we executed the main comparison across 23 optimization algorithms. The experiment covers three Hölder smoothness regimes ($\rho = 1.0, 0.5, 0.2$) and six drift magnitudes ($A \in \{0.001, 0.01, 0.1, 0.3, 0.6, 1.0\}$) in dimension five. Each algorithm runs for 500 steps across five random seeds. Critically, we did not tune hyperparameters per instance or environment. First-order methods use standard learning rates from original publications (SGD at 0.1, Adam at 0.001). All zeroth-order methods—including SPSA, FDSA, Kiefer–Wolfowitz, and their accelerated variants—use a fixed learning rate of 0.005. This reflects realistic deployment where practitioners rarely perform extensive per-instance tuning. The benchmark therefore functions as a stress test of out-of-the-box robustness rather than an optimized leaderboard.

4.1 Stress Tests and Ablations

The stress test suite validates five categories of scientific integrity.

Landscape invariants. Every loss function $L(x; \theta_t)$ must satisfy $L(\theta_t; \theta_t) = 0$ and $\nabla L(\theta_t; \theta_t) = 0$ exactly. We verify this across all six landscape families (quadratic, p-norm, Rosenbrock, multi-extremal, robust) in multiple dimensions. Violation invalidates dynamic regret and breaks first-order convergence.

Temporal consistency (Protocol A). The environment locks its latent state θ_t during an optimizer's step. All oracle queries within the same step see identical θ_t, and the environment advances only after the optimizer signals completion. The test confirms that calling `env.step()` while an oracle is active raises a runtime error—preventing lookahead.

Information barrier. Observations never expose θ_t directly. We check that attributes like `theta`, `optimal_point`, or `_theta` are absent from the observation object. In blind-value mode, pure gradient oracles return `value=None`, ensuring fair comparison between FO and ZO access.

Metric correctness. Lyapunov metrics must use the theoretically correct norm: for a ρ-Hölder smooth landscape, distance is measured in $\ell_{\rho+1}$ and raised to the power $\rho + 1$. We also confirm that normalized Lyapunov values $V_n / A^{\rho+1}$ denormalize to identical raw values across drift magnitudes—essential for cross-regime comparison.

Reproducibility and export fidelity. We run identical experiments with fixed seeds and verify bit-for-bit trajectory matching. JSON exports contain optimizer signatures, environment configs, ground-truth $\{\theta_t\}$ trajectories, and full metric histories. Any failure halts the benchmark; passing ensures subsequent comparisons are scientifically valid.

4.2 Baselines and Reference Implementations

In the experiments, we benchmark two algorithmic classes that match the oracle feedback regimes implemented in : *first-order* (FO) and *zero-order* (ZO). This separation is essential: FO methods assume access to a (noisy) gradient oracle, whereas ZO methods rely only on function values and/or zeroth-order gradient surrogates. Consequently, they require explicit query-cost accounting and a protocol-level alignment of what constitutes an oracle call (see Section 2 for the oracle contract and cost rules). Full algorithmic specifications, update equations, hyperparameter conventions, and budget normalization details are deferred to Appendix A.

First-order baselines. Our FO set includes Nesterov's accelerated gradient method [24], the classical Robbins–Monro stochastic approximation scheme [30], Polyak's adaptive step-size rule [27], and Polyak's heavy-ball momentum method [26]. We further include proximal/regularized variants based on forward–backward splitting (proximal SGD) [10] and regularized dual averaging [33], as well as communication-efficient robust updates via signSGD [6]. As widely used adaptive optimizers, we report Adam [17], AMSGrad [29], and AdamW [19]. For stochastic mirror descent, we follow the robust stochastic approximation framework in [22], and for iterate averaging (Polyak–Ruppert averaging) we use [28].

Zero-order baselines. Our ZO set includes CMA-ES [14], classical finite-difference stochastic approximation (Kiefer–Wolfowitz /

FDSA) [16], and random search as a standard baseline [5]. We also include SPSA as a canonical simultaneous-perturbation ZO gradient estimator [31] together with an accelerated tracking-oriented variant [11]. For one-dimensional optimization, we use quadratic (parabolic) interpolation in the classical formulation of [7]. For Bayesian optimization, we include GP-UCB [32]. Finally, in selected configurations we report a distributed subgradient method as a reference distributed baseline [21].

4.3 Experimental Protocol

Each time step t corresponds to a fixed latent optimum θ_t. The environment locks its state before oracle queries and advances only after the optimizer completes its step. Optimizers receive only noisy function values, gradients, or both—but never θ_t directly. Tracking quality is measured by the Lyapunov function

$$V_n(x) = \|x - \theta_n\|_{\rho+1}^{\rho+1}$$

and its drift-normalized variant $V_n/A^{\rho+1}$. Dynamic regret computes $\sum_{t=1}^{T}(L_t(x_t) - L_t(\theta_t))$ with both terms evaluated under identical θ_t. Every run uses explicit seeding for drift, noise, and perturbations. Results export as self-contained JSON files enabling third-party verification. The workload comprises 2,340 total runs (23 algorithms $\times$ 3 ρ values $\times$ 6 A values $\times$ 5 seeds), with tail-aggregated metrics over the final 20% of steps.

4.4 First Results and Implications

This experiment is not a tuned leaderboard—it is a stress test of out-of-the-box algorithmic behavior under realistic non-stationary conditions. All 23 methods run with fixed, theory-motivated hyperparameters: first-order optimizers use standard learning rates (e.g., SGD at 0.1, Adam at 0.001), while all zeroth-order methods—including SPSA variants—use a uniform learning rate of 0.005. No per-instance or per-drift tuning is performed. This mirrors real-world deployment, where practitioners rarely have the luxury of extensive hyperparameter search across unknown drift regimes.

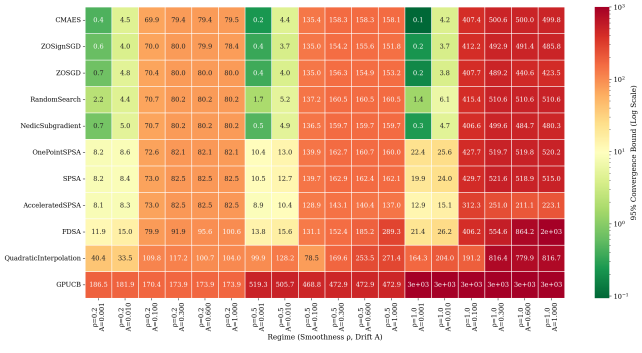

Figure 2: Convergence bounds (95% CI) for zero-order methods. Green cells indicate stable/low error (< 1).

The results in Figure 2 and Figure 3 reveal which algorithms remain functional when the optimum moves unpredictably. Only three zeroth-order methods—`AcceleratedSPSA`, `OnePointSPSA`, and `ZOSGD`—maintain bounded Lyapunov values across all smoothness levels ($\rho = 0.2$ to 1.0) and drift magnitudes ($A = 0.001$ to 1.0). In contrast, classical finite-difference schemes (`FDSA`, and population-based methods (`CMAES`, `GPUCB`) diverge rapidly as drift increases, often exceeding error bounds by orders of magnitude.

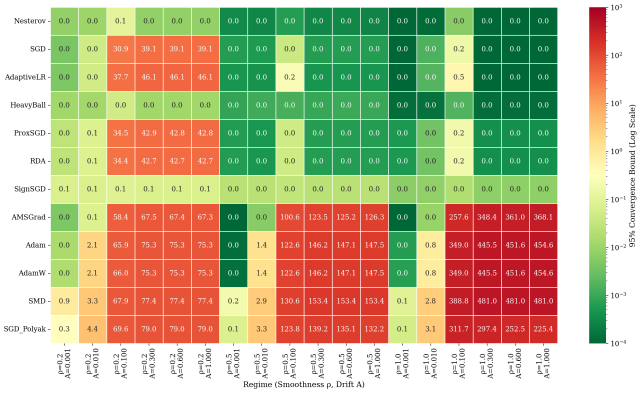

Figure 3: Convergence bounds (95% CI) for first-order methods. Green cells indicate stable/low error (< 1).

Among first-order methods, momentum-based schemes (`HeavyBall`, `Nesterov`) show robust tracking for $\rho \geq 0.5$, but fail sharply in non-smooth regimes ($\rho = 0.2$), confirming that convergence in stationary settings does not imply stabilization under drift.

Full drift sensitivity profiles, scaling law validations, and additional heatmaps supporting these conclusions are provided in Appendix B. Together, these findings validate the benchmark as a diagnostic tool: it exposes which algorithms are genuinely ready for deployment in drifting environments and which collapse without manual intervention. The experiment measures practical robustness—not peak performance under ideal tuning. The original code and the results may be found at our public repository [20].

5 Discussion

Benchmark Insights and Failure Modes. The WIND benchmark reveals a critical gap: many algorithms that converge in stationary settings fail to stabilize under even mild drift. This is not an implementation flaw but a consequence of violated assumptions—e.g., finite-difference methods assume near-static function values, while adaptive first-order methods accumulate stale gradient history. The benchmark systematically exposes these failure modes, showing that robustness does not correlate with popularity or static performance.

Reproducibility and Accessibility. All experiments are fully reproducible. Runs include deterministic seeding, version-locked dependencies, and self-contained JSON and CSV exports with optimizer configurations. environment parameters, ground-truth trajectories $\{\theta_t\}$, and full metric histories. The codebase is open-source and extensible under the MIT license.

Ethics and Responsible Use. This work uses synthetic environments and involves no human subjects or sensitive data. However, we caution against deploying any optimizer—even those that pass

our tests—in high-stakes domains without domain-specific validation. The benchmark measures tracking stability under controlled drift, not safety, fairness, or societal impact. Results should be seen as a necessary but insufficient condition for real-world use.

Limitations and Future Extensions. The current benchmark assumes isotropic, known-Hölder drift. Real-world problems often involve high-dimensional, structured, or adversarial dynamics. We plan to extend WIND to multi-agent settings with communication constraints, heterogeneous agent roles (FO/ZO/observer), and hybrid drift models. An MIT-licensed interactive visualizer already supports live replay, side-by-side algorithm comparison, and real-time metric overlays—enabling diagnostic inspection beyond aggregated scores.

6 Conclusion

In this paper, we introduced WIND, a benchmark for reproducible evaluation of stochastic optimization methods in non-stationary environments, where the minimizer θ_t drifts over time and observations may be corrupted by irregular noise, including heavy-tailed, quantized, and temporally correlated regimes. The central design principle of WIND is a *clean environment* contract: the optimizer interacts with the system only through an oracle with a formally specified interface (FO/ZO/Hybrid/Offline), while the latent state and the ground-truth optimum θ_t remain hidden and are accessed exclusively by the evaluation layer. This architecture fixes the semantics of a time step, prevents information leakage across modules, and yields fair comparisons by construction.

We specified WIND through three orthogonal axes of difficulty. First, drift models cover both smooth evolution and abrupt shocks via parametrized dynamics of θ_t. Second, landscape families provide geometry control (e.g., Hölder regularity, conditioning, multimodality, and constraints) under an explicit definition of the ground-truth minimizer. Third, noise models include correlated and heavy-tailed perturbations, with a precise account of how noise enters FO/ZO/Hybrid feedback and how it is amplified under zeroth-order finite-difference queries. For evaluation, WIND adopts tracking-oriented metrics, including geometry-consistent tracking error (with tail variants), dynamic regret, shock-response characteristics (e.g., TTR(ε)), and query efficiency with explicit query-cost accounting in ZO regimes.

Our experimental results demonstrate that WIND serves as an effective stress-test suite out of the box: under a unified reporting protocol and broad grids over smoothness and drift magnitude, it clearly separates stable from unstable algorithmic behaviors. In particular, strong performance in stationary optimization does not necessarily translate to robustness under drift: several classical schemes degrade rapidly as non-stationarity increases, whereas a smaller subset maintains controlled tracking error over a wide range of regimes. Overall, WIND closes a practical gap between the theory of non-stationary optimization and the engineering need for standardized, reproducible comparisons under realistic dynamics and noise, providing a unified oracle interface, principled query accounting, and a consistent reporting format.

Acknowledgments

This work was supported by the Ministry of Economic Development of the Russian Federation (Agreement No. 139-15-2025-007, dated April 16, 2025; ID: 000000C313925P3O0002).

References

[1] [n. d.]. COCO: COmparing Continuous Optimizers. https://coco-platform.org/getting-started/. Accessed 2026-02-08.
[2] [n. d.]. HPOBench: A Collection of Reproducible Hyperparameter Optimization Benchmarks. https://github.com/automl/HPOBench. Accessed 2026-02-08.
[3] [n. d.]. LibOPT: An Open-Source Platform for Fast Prototyping Soft Optimization Techniques. https://github.com/jppbsi/LibOPT. Accessed 2026-02-08.
[4] [n. d.]. Nevergrad: A Gradient-Free Optimization Platform. https://github.com/facebookresearch/nevergrad. Accessed 2026-02-08.
[5] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305. http://jmlr.org/papers/v13/bergstra12a.html
[6] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. 2018. signSGD: Compressed Optimisation for Non-Convex Problems. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 560–569. https://proceedings.mlr.press/v80/bernstein18a.html
[7] Richard P. Brent. 1973. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, Englewood Cliffs, New Jersey. Reprinted by Dover Publications, 2002.
[8] Tsung-Jui Chang and Shahin Shahrampour. 2021. On Online Optimization: Dynamic Regret Analysis of Strongly Convex and Smooth Problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 6966–6973. doi:10.1609/aaai.v35i8.16858
[9] Joshua Cutler, Dmitriy Drusvyatskiy, and Zaid Harchaoui. 2021. Stochastic Optimization under Time Drift: Iterate Averaging, Step-Decay Schedules, and High Probability Guarantees. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 11859–11869. https://proceedings.neurips.cc/paper_files/paper/2021/file/62e7f2e090fe150ef8deb4466fdc81b3-Paper.pdf
[10] John Duchi and Yoram Singer. 2009. Efficient Online and Batch Learning Using Forward Backward Splitting. *Journal of Machine Learning Research* 10 (2009), 2899–2934. http://www.jmlr.org/papers/v10/duchi09a.html
[11] Victoria Erofeeva, O. Granichin, Munira Tursunova, Anna Sergeenko, and Yuming Jiang. 2022. Accelerated Simultaneous Perturbation Stochastic Approximation for Tracking Under Unknown-but-Bounded Disturbances. In *Proceedings of the American Control Conference (ACC)*. 1582–1587. doi:10.23919/ACC53348.2022.9867491
[12] Eric Hall and Rebecca Willett. 2013. Dynamical Models and Tracking Regret in Online Convex Programming. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 579–587. https://proceedings.mlr.press/v28/hall13.html
[13] Eric C. Hall and Rebecca M. Willett. 2016. Online Optimization in Dynamic Environments. arXiv:1307.5944 [stat.ML] https://arxiv.org/abs/1307.5944
[14] Nikolaus Hansen. 2023. The CMA Evolution Strategy: A Tutorial. arXiv:1604.00772 [cs.LG] https://arxiv.org/abs/1604.00772
[15] Andrew Jacobsen and Francesco Orabona. 2024. An Equivalence Between Static and Dynamic Regret Minimization. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., 98835–98870. doi:10.52202/079017-3137
[16] J. Kiefer and J. Wolfowitz. 1952. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics* 23, 3 (1952), 462–466. doi:10.1214/aoms/1177729392
[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1412.6980
[18] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Guoyin Zhang, Alfred O. III Hero, and Pramod K. Varshney. 2020. A Primer on Zeroth-Order Optimization in Signal Processing and Machine Learning: Principles, Recent Advances, and Applications. *IEEE Signal Processing Magazine* 37, 5 (2020), 43–54. doi:10.1109/MSP.2020.3003837
[19] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1711.05101
[20] muffin003. 2026. WIND: ST-HRD Benchmark for Stochastic Tracking under Hölder Regularity and Drift. https://github.com/muffin003/WIND. Accessed: 2026-02-09.
[21] Angelia Nedić and Asuman Ozdaglar. 2009. Distributed Subgradient Methods for Multi-Agent Optimization. *IEEE Trans. Automat. Control* 54, 1 (2009), 48–61. doi:10.1109/TAC.2008.2009515
[22] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. 2009. Robust Stochastic Approximation Approach to Stochastic Programming. *SIAM Journal on Optimization* 19, 4 (2009), 1574–1609. doi:10.1137/070704277

[23] Yurii Nesterov and Vladimir Spokoiny. 2017. Random Gradient-Free Minimization of Convex Functions. *Foundations of Computational Mathematics* 17 (2017), 527–566. doi:10.1007/s10208-015-9296-2
[24] Yu. E. Nesterov. 1983. A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$. *Doklady Akademii Nauk SSSR* 269, 3 (1983), 543–547. http://mi.mathnet.ru/dan46009 In Russian.
[25] T. Papastergiou and V. Megalooikonomou. 2017. A Distributed Proximal Gradient Descent Method for Tensor Completion. In *2017 IEEE International Conference on Big Data (Big Data)*. 2056–2065. doi:10.1109/BigData.2017.8258152
[26] B. T. Polyak. 1964. Some Methods of Speeding Up the Convergence of Iteration Methods. *U. S. S. R. Comput. Math. and Math. Phys.* 4, 5 (1964), 1–17.
[27] B. T. Polyak. 1969. Minimization of Unsmooth Functionals. *U. S. S. R. Comput. Math. and Math. Phys.* 9, 3 (1969), 14–29. doi:10.1016/0041-5553(69)90061-5
[28] B. T. Polyak and A. B. Juditsky. 1992. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization* 30, 4 (1992), 838–855. doi:10.1137/0330046
[29] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1711.05105
[30] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400–407. doi:10.1214/aoms/1177729586
[31] J. C. Spall. 1992. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *IEEE Trans. Automat. Control* 37, 3 (1992), 332–341. doi:10.1109/9.119632
[32] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. 2012. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Transactions on Information Theory* 58, 5 (2012), 3250–3265. doi:10.1109/TIT.2011.2182033
[33] Lin Xiao. 2010. Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization. *Journal of Machine Learning Research* 11, 88 (2010), 2543–2596. http://jmlr.org/papers/v11/xiao10a.html
[34] Lijun Zhang, Shiyin Lu, and Zhi-Hua Zhou. 2018. Adaptive Online Learning in Dynamic Environments. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 1330–1340.
[35] Peng Zhao, Yu-Jie Zhang, Lijun Zhang, and Zhi-Hua Zhou. 2020. Dynamic Regret of Convex and Smooth Functions. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 12510–12520.
[36] Peng Zhao, Yu-Jie Zhang, Lijun Zhang, and Zhi-Hua Zhou. 2024. Adaptivity and Non-Stationarity: Problem-Dependent Dynamic Regret for Online Convex Optimization. *Journal of Machine Learning Research* 25, 98 (2024), 1–52. https://jmlr.org/papers/v25/23-0118.html

A Baseline Algorithms and Implementation Conventions

This appendix specifies the baseline optimizers used in WIND and fixes the implementation conventions needed for reproducibility and fair accounting across feedback regimes. Throughout, the benchmark operates over a time-indexed objective sequence $\{f_t\}_{t=1}^{T}$ with a decision variable $x_t \in \mathbb{R}^d$. The optimizer interacts with the benchmark through the oracle contract defined in Section 2. In FO mode, the oracle provides (possibly noisy) first-order feedback; in ZO mode, the oracle provides function values and the optimizer may construct gradient surrogates via finite differences or random perturbations.

Time indexing and reporting. At each outer time step t, an optimizer produces a single reported decision x_t (the *step decision*). Algorithms that perform internal iterations within a single step (e.g., evolution strategies or multi-evaluation ZO estimators) are treated as producing a final iterate x_t after spending a step budget; the benchmark metrics are computed on this reported x_t. The latent optimum θ_t is *never* exposed to the optimizer and is used only by the evaluation layer.

Query-cost accounting. We count *oracle calls* according to the oracle contract. In FO mode, one gradient query at a point x (and, if applicable, the accompanying function value) constitutes a single call. In ZO mode, each function evaluation $f_t(x)$ counts as one call. Therefore, ZO gradient surrogates constructed from m function evaluations incur cost m per surrogate. The benchmark normalizes comparisons by reporting performance as a function of the total number of oracle calls (and/or a fixed per-step query budget), as specified in Section 2.

Notation. We use g_t for a (noisy) gradient returned by the FO oracle at time t; $\widehat{g}_t$ denotes a gradient surrogate in ZO mode. The operator $\mathrm{prox}_{\eta R}$ denotes the proximal mapping of a regularizer R with stepsize $\eta > 0$:

$$\mathrm{prox}_{\eta R}(v) \;=\; \arg\min_{x} \left\{ R(x) + \frac{1}{2\eta}\|x - v\|_2^2 \right\}. \tag{38}$$

For elementwise operations we use $\odot$, and for elementwise square root we use $\sqrt{\cdot}$.

A.1 First-Order (FO) Baselines

SGD / Robbins–Monro. Given a stochastic gradient g_t at x_t, SGD updates

$$x_{t+1} \;=\; x_t - \eta_t\, g_t, \tag{39}$$

where $\{\eta_t\}$ is a stepsize schedule (constant, polynomial decay, or tuned per suite). We cite Robbins–Monro [30] as the classical stochastic approximation origin.

Nesterov accelerated gradient. We use the standard accelerated scheme (in its gradient form), with a momentum-like auxiliary sequence:

$$\begin{aligned} y_t &= x_t + \beta_t(x_t - x_{t-1}), \\ x_{t+1} &= y_t - \eta_t\, g_t(y_t), \end{aligned} \tag{40}$$

where $g_t(y_t)$ is the FO oracle gradient at y_t and $\beta_t \in [0, 1)$ is the acceleration parameter. We cite Nesterov [24].

*Polyak adaptive step-size (*Polyak step*).* When an estimate of the optimal value $f_t^\star$ is available (or a proxy is provided by the benchmark configuration), we use the Polyak step-size rule:

$$\eta_t \;=\; \frac{f_t(x_t) - f_t^\star}{\|g_t\|_2^2 + \varepsilon}, \qquad x_{t+1} \;=\; x_t - \eta_t\, g_t, \tag{41}$$

with a small $\varepsilon > 0$ for numerical stability. We cite Polyak [27]. In settings where $f_t^\star$ is unknown, we use a configured proxy (e.g., best-so-far estimate) consistently across methods.

HeavyBall / Polyak momentum. We use the classical momentum update

$$v_{t+1} \;=\; \mu v_t + g_t, \qquad x_{t+1} \;=\; x_t - \eta_t\, v_{t+1}, \tag{42}$$

with momentum coefficient $\mu \in [0, 1)$. We cite Polyak [26].

Proximal SGD (forward–backward splitting). For composite objectives with a (possibly non-smooth) regularizer R, we apply a gradient step on the smooth part followed by a proximal step:

$$x_{t+\frac{1}{2}} \;=\; x_t - \eta_t\, g_t, \qquad x_{t+1} \;=\; \mathrm{prox}_{\eta_t R}\Big(x_{t+\frac{1}{2}}\Big). \tag{43}$$

We cite Duchi and Singer [10].

Regularized Dual Averaging (RDA).. RDA maintains an accumulated gradient $\bar{g}_t = \sum_{s=1}^{t} g_s$ and computes

$$x_{t+1} \;=\; \arg\min_{x} \left\{ \langle \bar{g}_t, x \rangle + \frac{1}{\eta_t}\psi(x) + \lambda R(x) \right\}, \tag{44}$$

where ψ is a strongly convex prox function and R is a regularizer (e.g., ℓ_1). We cite Xiao [33].

Stochastic Mirror Descent (SMD).. With a distance-generating function ψ and Bregman divergence $D_\psi(x, y) = \psi(x) - \psi(y) - \langle \nabla\psi(y), x - y \rangle$, mirror descent updates

$$x_{t+1} \;=\; \arg\min_{x \in \mathcal{X}} \left\{ \eta_t \langle g_t, x \rangle + D_\psi(x, x_t) \right\}. \tag{45}$$

We follow the robust stochastic approximation perspective in Nemirovski *et al.* [22].

signSGD.. Given a stochastic gradient g_t, signSGD uses the elementwise sign:

$$x_{t+1} \;=\; x_t - \eta_t \,\mathrm{sign}(g_t), \tag{46}$$

where $\mathrm{sign}(\cdot)$ is applied componentwise. We cite Bernstein *et al.* [6].

Adam. Adam maintains first and second moment estimates of gradients:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t, \\ \widehat{m}_t &= \frac{m_t}{1-\beta_1^t}, \qquad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}, \\ x_{t+1} &= x_t - \eta_t \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}. \end{aligned} \tag{47}$$

We cite Kingma and Ba [17].

AMSGrad. AMSGrad modifies Adam by enforcing a non-decreasing second-moment accumulator:

$$\tilde{v}_t \;=\; \max(\tilde{v}_{t-1}, v_t), \qquad x_{t+1} \;=\; x_t - \eta_t \frac{\widehat{m}_t}{\sqrt{\tilde{v}_t} + \varepsilon}. \tag{48}$$

We cite Reddi *et al.* [29].

AdamW.. AdamW decouples weight decay from the adaptive gradient step:

$$x_{t+1} \;=\; x_t - \eta_t \left(\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon} + \lambda x_t \right), \tag{49}$$

where λ is the weight decay coefficient. We cite Loshchilov and Hutter [19].

Polyak–Ruppert iterate averaging. Given a base sequence $\{x_t\}$, the averaged iterate is

$$\bar{x}_t \;=\; \frac{1}{t} \sum_{s=1}^{t} x_s, \tag{50}$$

and the benchmark reports either x_t or $\bar{x}_t$ depending on the configuration (fixed consistently across methods within a suite). We cite Polyak and Juditsky [28].

Distributed subgradient (reference distributed baseline). For a networked setting with agents $i \in \{1, \ldots, n\}$, local variables $x_{i,t}$, and mixing weights w_{ij}, the distributed subgradient step takes the form

$$x_{i,t+1} \;=\; \sum_{j=1}^{n} w_{ij} x_{j,t} \;-\; \eta_t \, g_{i,t}, \tag{51}$$

where $g_{i,t}$ is a (sub)gradient of the local objective at agent i. We cite Nedić and Ozdaglar [21]. In WIND, this baseline is included where the benchmark configuration explicitly enables distributed execution.

A.2 Zero-Order (ZO) Baselines

In ZO mode, the oracle provides function evaluations $f_t(x)$, and the optimizer may form $\widehat{g}_t$ using finite differences or randomized perturbations. Unless stated otherwise, each evaluation $f_t(\cdot)$ counts as one oracle call.

Finite-difference stochastic approximation (FDSA / Kiefer–Wolfowitz). A coordinatewise finite-difference estimator at x_t uses perturbations $\pm c_t e_i$:

$$\widehat{g}_{t,i} \;=\; \frac{f_t(x_t + c_t e_i) - f_t(x_t - c_t e_i)}{2c_t}, \qquad i = 1, \ldots, d, \tag{52}$$

and then performs $x_{t+1} = x_t - \eta_t \widehat{g}_t$. This costs $2d$ function evaluations per surrogate. We cite Kiefer and Wolfowitz [16]. (When the benchmark config uses a one-sided variant, the estimator is defined analogously with cost $d + 1$.)

Randomized gradient-free SGD (ZOSGD).. We use a randomized two-point estimator based on random directions u_t sampled uniformly from the unit sphere:

$$\widehat{g}_t \;=\; \frac{d}{2\delta_t} \Big(f_t(x_t + \delta_t u_t) - f_t(x_t - \delta_t u_t) \Big) u_t, \qquad x_{t+1} = x_t - \eta_t \widehat{g}_t, \tag{53}$$

with query cost 2 per step. We cite Nesterov and Spokoiny [23] for randomized gradient-free minimization.

Random search. Random search samples candidates $x^{(k)}$ from a proposal distribution (uniform or Gaussian around the incumbent) and selects the best observed value:

$$x_{t+1} \;=\; \arg \min_{x \in \{x_t, x^{(1)}, \ldots, x^{(K)}\}} f_t(x), \tag{54}$$

with cost K evaluations per step. We cite Bergstra and Bengio [5] as a canonical random-search baseline.

SPSA (simultaneous perturbation). SPSA uses a random Rademacher perturbation vector $\Delta_t \in \{\pm 1\}^d$ and constructs

$$\widehat{g}_t \;=\; \frac{f_t(x_t + c_t \Delta_t) - f_t(x_t - c_t \Delta_t)}{2c_t} \, \Delta_t^{-1}, \qquad x_{t+1} = x_t - \eta_t \widehat{g}_t, \tag{55}$$

where Δ_t^{-1} denotes elementwise inverse (equivalently, $\Delta_t^{-1} = \Delta_t$ for ± 1 entries). This costs 2 evaluations per step. We cite Spall [31].

Accelerated SPSA (tracking under bounded disturbances). We use the accelerated SPSA variant provided in the benchmark implementation, which retains the SPSA gradient surrogate while modifying the update to improve tracking under unknown-but-bounded disturbances (including acceleration/weighting mechanisms specified

in the implementation). We cite Erofeeva *et al.* [11]. The per-step query cost remains 2 function evaluations.

CMA-ES.. CMA-ES maintains a Gaussian search distribution $\mathcal{N}(m_t, \sigma_t^2 C_t)$ and iteratively updates its mean and covariance using weighted elite samples. In each step, it samples λ candidates $x_t^{(k)} \sim \mathcal{N}(m_t, \sigma_t^2 C_t)$, evaluates $f_t(x_t^{(k)})$, and updates (m_t, C_t, σ_t) using the μ best samples. The per-step query cost equals the population size λ. We cite Hansen's tutorial [14], and we treat CMA-ES as producing the final reported iterate $x_t \equiv m_t$ after consuming its per-step evaluation budget.

Quadratic (parabolic) interpolation in 1D.. For one-dimensional objectives, quadratic interpolation constructs a parabola through three points $(x_a, f(x_a))$, $(x_b, f(x_b))$, $(x_c, f(x_c))$ and selects the minimizer of the interpolant to propose the next query. We follow the classical treatment in Brent [7]. In WIND, this baseline is used only for 1D suites, with query costs equal to the number of function evaluations required by the interpolation step.

GP-UCB (Bayesian optimization). GP-UCB models $f_t(\cdot)$ as a Gaussian process and selects the next query by maximizing an upper-confidence bound acquisition function:

$$x_{t+1} = \arg\max_{x \in X} \left\{ \mu_t(x) + \beta_t^{1/2} \sigma_t(x) \right\}, \tag{56}$$

where μ_t and σ_t are the GP posterior mean and standard deviation after t observations. Each iteration uses one function evaluation plus internal optimization of the acquisition (performed with a fixed internal budget, reported separately when applicable). We cite Srinivas *et al.* [32].

ZO-signSGD (benchmark implementation). In configurations labeled ZO-signSGD, we apply a sign-based update using a ZO gradient surrogate:

$$x_{t+1} = x_t - \eta_t \operatorname{sign}(\widehat{g}_t), \tag{57}$$

where $\widehat{g}_t$ is constructed from ZO feedback under the configured estimator (e.g., randomized two-point or SPSA-style). We cite Papastergiou and Megalooikonomou [25] as provided in the baseline list; operationally, the query cost is determined by the underlying ZO surrogate used to compute $\widehat{g}_t$.

Hyperparameter selection and defaults. For each suite and feedback regime, hyperparameters (e.g., η_t, (β_1, β_2), c_t or δ_t, population size λ) are selected via the benchmark's standardized calibration procedure and are held fixed across repetitions. The tuning grid/ranges and the resulting chosen values are reported in the experiment configuration files released with the WIND artifact, consistent with the reproducibility requirements in Section 2.

B Additional simulation Results

This appendix contains supplementary visualizations referenced in Section 4. Figure 4 shows the Dolan–Moré reliability profile across all 23 algorithms. It demonstrates that `AcceleratedSPSA` consistently solves the largest fraction of problem instances within a factor of two of the best performer, despite using no gradient information. Its curve dominates most first-order methods for performance ratios $\tau > 2$, indicating rare catastrophic failure.

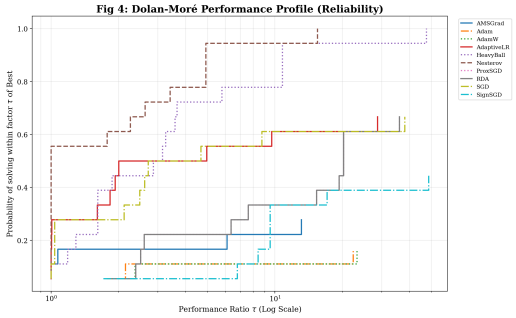

Figure 4: Dolan–Moré performance profile (reliability). Higher curves indicate more robust solvers.

Figures 5 and 6 validate the theoretical scaling law $E[V] \propto A^{\rho+1}$ for first-order and zero-order algorithms. The close alignment between measured slopes and theoretical predictions confirms the correctness of the normalized Lyapunov metric and justifies cross-regime comparisons.

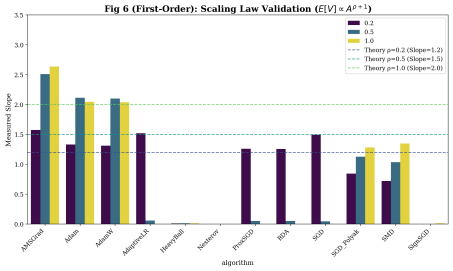

Figure 5: Scaling law validation for first-order methods: measured slope vs theory ($E[V] \propto A^{\rho+1}$).

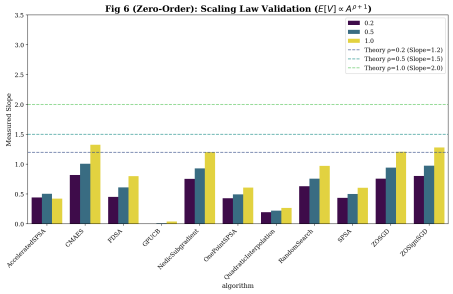

Figure 6: Scaling law validation for zero-order methods: measured slope vs theory ($E[V] \propto A^{\rho+1}$).

Figures 7 and 8 present drift sensitivity profiles for zero-order and first-order methods, respectively. A flat response indicates ideal invariance to increasing drift magnitude; deviations reveal degradation under non-stationarity.

Received 08 February 2026; revised 08 February 2026; accepted ? May 2026

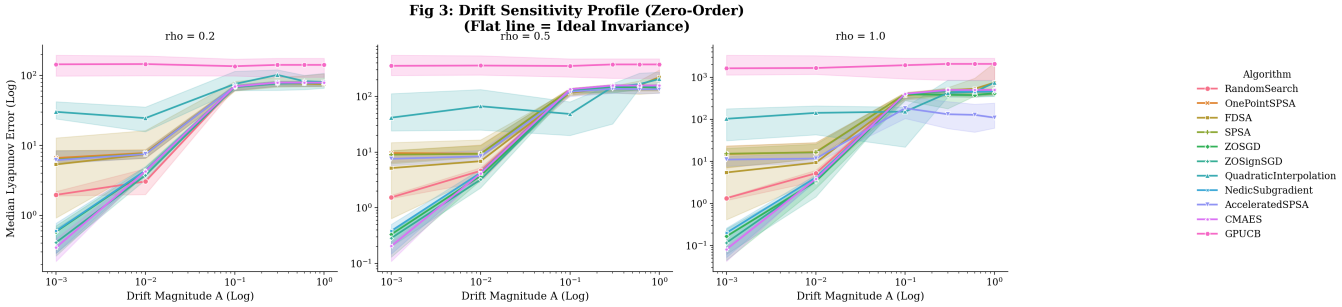

Figure 7: Drift sensitivity profile (zero-order). Flat line indicates ideal invariance.

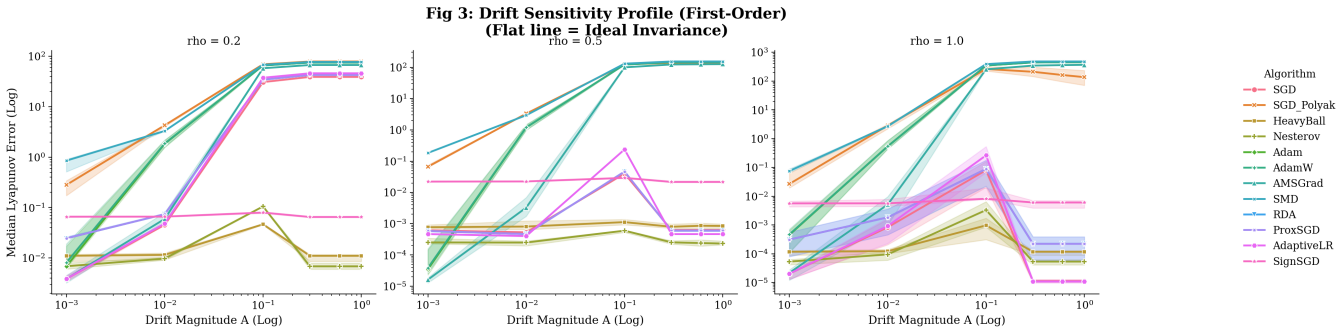

Figure 8: Drift sensitivity profile (first-order). Flat line indicates ideal invariance.