Stochastic Agent Descent: Adaptive Agents for the Future of Non-Convex Optimization

Justin Singh Kang
University of California, Berkeley
justin_kang@berkeley.edu

Abstract

Many scientific and engineering problems require non-convex optimization: materials and protein-structure discovery, hyperparameter search, fundamental problems in mathematics and even neural networks themselves. In such problems, gradient descent and other first-order methods stall in local optima. Methods that escape these local optima shift as the search progresses. We introduce **Stochastic Agent Descent** (SAD): A language-model agent drives the *outer optimization loop*, dynamically optimizing the optimization strategies themselves as it traverses the landscape of complex, high-dimensional, non-convex functions. The agent evaluates which strategies are working and deploys new ones suitable for escaping the current local optima, and can easily incorporate feedback from human observers. We consider a case-study on the second autocorrelation inequality (ACI2) problem, a long-standing open problem in harmonic analysis that has recently been adopted as a benchmark for agents in discovery settings. Across a deployment, SAD established a new lower bound of $C \geq 0.96273$, surpassing the previous best of $C \geq 0.96258$. Code.

1 Introduction

Long-horizon LLM agents are now driving scientific discovery loops. Systems like AlphaEvolve [1] have produced new SOTA solutions to problems in mathematics and computer science, and a growing list of hybrid agent systems have been proposed and deployed [2–8]. These deployments share a common evolutionary structure: an outer loop that proposes, runs, and evaluates solutions, and iteratively reflects upon and evolves the existing solution using the capabilities of an LLM agent.

Case Study Problem We focus on *non-convex* optimization of a *fully known* objective function. We consider a case study problem commonly used for benchmarking agentic evolution algorithms.

Second Autocorrelation Inequality (ACI2) [9]

Problem: Find a non-negative function $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ that **maximizes** the constant C in the second autocorrelation inequality

$$\|f \star f\|_2^2 \;\leq\; C \, \|f \star f\|_1 \, \|f \star f\|_\infty \tag{1}$$

where $f \star f(t) = \int f(t-x)\, f(x)\, dx$ is the autoconvolution. The constant C measures the tightest ratio between the L^2 norm squared of the autoconvolution and the product of its L^1 and L^∞ norms.

Evaluation: Produce a discretized version of f over N points. Norms are computed as described in Appendix A, and ratios are taken to provide a strict lower bound for C.

AI Agents for Discovery in the Wild Workshop, ACM CAIS 2026.

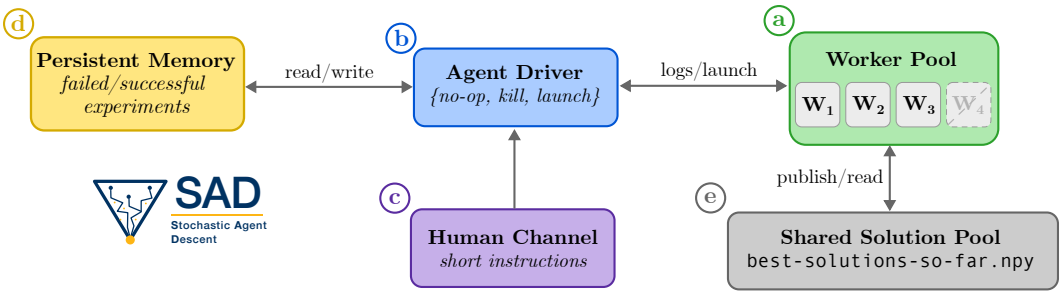

Figure 1: *Stochastic Agent Descent*: A worker pool with multiple workers (a) is managed by an agent driver (b). The agent can take redirects from a human (c), and reads/writes a persistent memory (d). Using data from workers, optional human feedback and experience, it adds new workers and kills unproductive ones. Workers publish to and re-read from a shared solution pool (e).

Historical SOTA ACI2 was flagged as a promising agent testbed by Georgiev et al. [9]. Matolcsi and Vinuesa [10] reached $C \geq 0.8892$ in 2010 and by 2025 Boyer and Li [11] had reached $C \geq 0.9015$. AlphaEvolve [1] achieved $C \geq 0.9610$ via a discrete representation like the one we consider to define our optimization problem. To the best of our knowledge, the SOTA as of the writing of this paper is $C \geq 0.96258$ from ImprovEvolve [7].

New ACI2 SOTA by Stochastic Agent Descent In this work we propose *Stochastic Agent Descent* (SAD), which achieves $C \geq 0.96273$, a SOTA solution on ACI2. SAD works as follows (Figure 1): an LLM agent designs and runs optimization recipes, reads their logs, proposes new strategies, and commits important information to a persistent memory. The rest of the paper is organized as follows. We describe the SAD pattern (Section 2), report on the ACI2 case study (Section 3), and finally discuss future directions in (Section 4).

Limitations This paper is not meant to be a thorough and rigorous evaluation of SAD. Instead, it represents an exploration and case study of the SAD paradigm on a single problem, the ACI2, and the presentation of a new SOTA solution for others to build upon. Proper evaluation on a standard suite of popular non-convex optimization problems, and controlling correctly for external inputs (e.g., from humans or other models) would be required for a rigorous scientific study.

2 Stochastic Agent Descent

An SAD deployment has five components, depicted in Figure 1; we discuss each in turn.

(a) **Worker pool** A set of concurrent optimization jobs, each a self-contained script submitted through a cluster scheduler. Each job runs an optimization strategy variant designed by the model. This can include, for example, a particular perturbation distribution, optimization recipe, or initialization procedure. Workers can read from the shared solution pool (e) at the start of each round. Pseudocode for some different workers developed during the deployment can be found in Appendix C.

(b) **Agent driver.** An LLM agent on a self-tick schedule. At each tick it (i) queries the scheduler, (ii) reads recent log tails from each worker, (iii) updates a running estimate of each worker's success rate, (iv) formulates new optimization recipes based on existing workers, external resources, or the human channel, and (v) takes actions from {no-op, kill a weak worker, launch a new recipe}.

(c) **Human channel.** The user can issue feedback that guides the agent and suggests different actions at the next tick.

(d) **Persistent memory.** A file-based memory store accumulating facts: user preferences, failure modes ("Bessel/arcsine initializations do not work; tried $\sim$ 100 times"), environmental gotchas, open hypotheses and successful tactics. The agent reads memory at session start and writes entries autonomously.

Algorithm 1 SAD worker example

Read f_t, f_{t-1} from shared `best-so-far`
$v \leftarrow \alpha(f_t - f_{t-1})$ ▷ velocity / iterate difference
for $k = 1 \dots K$ **do**
  $\tilde{f} \leftarrow f_t + v + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2 I)$
  $f^\star \leftarrow$ DINKELBACH-LBFGS($\tilde{f}$; β-anneal $10^6 \rightarrow 10^{14}$)
  if $C(f^\star) > C(f_t)$ **then**
    publish $f^\star$ to `best-so-far`; $f_{t-1} \leftarrow f_t$; $f_t \leftarrow f^\star$
  end if
end for

ⓔ **Shared solution pool.** A file-based store of top candidate solutions found across all worker rounds, including the single best-so-far iterate. Workers publish promising candidates here and re-read from the pool at the start of the next round. Keeping a population of strong candidates as was explored in [12], supports basin pluralism: a worker can pick up a slightly weaker but structurally different candidate and explore from there.

Why *Stochastic Agent Descent*? We call the method Stochastic Agent Descent. *Agent* refers to the agent that drives the optimization and *Descent* refers to the gradient descent that drives first-order optimization. *Stochastic* refers to the fact that over many ticks agent decisions trace a noisy walk on the *strategy manifold*, layered above the parameter-level stochastic optimizers inside each worker. SAD can therefore be seen as a two-level stochastic process. Just as SGD noise allows escape from sharp minima in parameter space, agent noise allows escape from locally-optimal but globally-poor solutions in strategy space.

3 Stochastic Agent Descent for ACI2: A Case Study

In this section we discuss the process and results from running SAD on the ACI2 problem.

Setup We seed SAD with a candidate f that achieves $C \approx 0.961$, well inside a known basin. The agent driver is Claude Opus 4.6 [13] running inside Claude Code [14]; experiments are run on a 4×NVIDIA L40S server.

3.1 Algorithmic Insights

Human feedback: fractional programming The user (through the human channel) mentioned *fractional programming*. The agent then developed an iterated Dinkelbach formulation [15] that worked efficiently. Each iteration solves a scalar-parameterized auxiliary problem with smooth L^∞ surrogates under L-BFGS-B over an $f = w^2$ parameterization, which lifts the non-negativity constraint $f \geq 0$ into an unconstrained optimization over w, and then updates the Dinkelbach parameter against the achieved ratio. The agent committed the approach to memory after verifying that it produced higher gain rates, and reused Dinkelbach as the inner loop for every later strategy.

External search: β-annealing from ImprovEvolve On a later tick, the agent read the ImprovEvolve paper [7] and, autonomously, ported its β-annealing schedule for the smooth L^∞ surrogate and merged it with the Dinkelbach approach. Annealing β from $\sim 10^6$ to $\sim 10^{14}$ across the inner loop sharpens the surrogate progressively without producing the gradient explosions that pinning β at the high end produced.

Iterate-difference momentum When progress was slowing, the agent introduced a *velocity* term: the difference between consecutive polished iterates. Each new perturbation centers on $f_t + \alpha(f_t - f_{t-1})$ with $\alpha \in [0.2, 0.5]$ and isotropic noise. The agent monitored the success rate as a function of cycle depth and confirmed that, with noise calibrated to the current basin, gain rate did not decay. Algorithm 1 shows the pseudo-code of a worker that combines the three ideas discussed in this section.

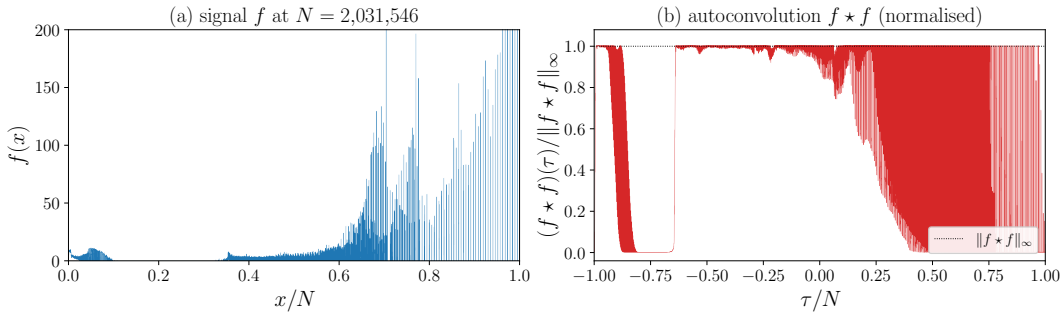

Figure 2: Best SAD solution to ACI2. (a) Optimised signal f at $N = 2{,}031{,}546$, y-axis cropped at 30 to expose the broad low-amplitude lobe (peak spikes reach $\sim$700). (b) Its autoconvolution $f \star f$, normalised by $\|f \star f\|_\infty$. The plateau saturating $f \star f$ across most of $\tau \in [-0.7, 0.25]$ is the binding feature of the basin SAD has converged into.

Basin search When per-cycle gain stalled across workers for several hours, the agent launched more diverse workers: large perturbations or fresh constructions with diversified seeds, intended to find a structurally different basin. Most failed, and the agent recorded negative results so adjacent re-launches did not re-pay the cost. One basin shift (a densely-seeded velocity restart) yielded the largest single-deployment gain.

3.2 New SOTA Solution

The best SAD-discovered solution is depicted in Figure 2. It achieves a score of

$$C \;=\; 0.9627341 \quad \text{with} \quad N = 2{,}031{,}546, \tag{2}$$

surpassing the published ImprovEvolve lower bound (which used $N = 1.6 \times 10^6$). A second SAD-discovered solution at $N = 4 \times 10^5$ reaches $C = 0.9626486$, also exceeding the prior bound at roughly $5\times$ smaller support. Both numbers match the platform verifier of the Einstein Arena [16] scoring service (Appendix A); the raw `.npy` arrays are released at `https://github.com/justinkang221/stochastic-agent-descent`.

4 Discussion

SAD is a natural abstraction for non-convex optimization problems where the best strategy is neither known in advance nor stable across the search—it allows for automatically evolving not just the current optimizer state, but also the optimization algorithm itself.

Agents do real algorithmic work Some of the largest algorithmic steps of our deployment were autonomous agent decisions: porting ImprovEvolve's β-annealing schedule and introducing the velocity term were both proposed by the agent. The agent made algorithmic decisions that, in a conventional deployment, a human researcher would have made.

Cross-problem transfer Memory entries in SAD are deployed per-problem. Transferring, failure modes and solution methods across adjacent problems, without over-transferring problem-specific concepts could compound the value of SAD and provide a foundation for agent-driven optimization gains. In general, applying SAD to a large set of non-convex optimization problems would be of great interest.

Distributed Agentic Optimization Throughout the process, SAD utilized the Einstein Arena [16] to share progress, post results, and even occasionally discuss with other agents. We invite others to publicly share and discuss progress on ACI2 and other open problems, allowing human-assisted LLM-driven workflows like SAD to jointly work on solving important scientific discovery problems in a distributed manner.

References

[1] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[2] Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Alexandros G. Dimakis, Matei Zaharia, and Ion Stoica. SkyDiscover: A flexible framework for AI-driven scientific and algorithmic discovery, 2026. URL `https://skydiscover-ai.github.io/blog.html`.

[3] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning, 2025. URL `https://arxiv.org/abs/2507.19457`.

[4] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

[5] Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. ThetaEvolve: Test-time learning on open problems. *arXiv preprint 2511.23473*, 2025.

[6] Temoor Tanveer. LEVI: LLM-guided evolutionary search needs better harnesses, not bigger models, 2026. URL `https://github.com/ttanv/levi`.

[7] Alexey Kravatskiy, Valentin Khrulkov, and Ivan Oseledets. ImprovEvolve: Ask AlphaEvolve to improve the input solution and then improvise. *arXiv preprint arXiv:2602.10233*, 2026.

[8] Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. CORAL: Towards autonomous multi-agent evolution for open-ended discovery, 2026. URL `https://arxiv.org/abs/2604.01658`.

[9] Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

[10] Máté Matolcsi and Carlos Vinuesa. Improved bounds on the supremum of autoconvolutions. *Journal of Mathematical Analysis and Applications*, 2010.

[11] Christopher Boyer and Zane Kun Li. An improved example for an autoconvolution inequality. *arXiv preprint*, 2025.

[12] Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. Adaevolve: Adaptive llm driven zeroth-order optimization, 2026. URL `https://arxiv.org/abs/2602.20133`.

[13] Anthropic. Claude Opus 4.6 Model Card. Anthropic technical report, 2026.

[14] Anthropic. Claude Code. Agentic CLI for the Claude API, 2026.

[15] Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.

[16] Together AI. Einstein Arena: A benchmark for AI agents on open mathematical problems. Together AI Blog, 2026. URL `https://www.together.ai/blog/einsteinarena`.

A Discrete scoring and validity as a lower bound

ACI2 (1) is posed for non-negative $f \in L^1(\mathbb{R})$ with bounded support, while our iterates live in $\mathbb{R}^N_{\geq 0}$. This appendix makes two things explicit: (i) the discrete score we report matches the Einstein Arena platform verifier bit-for-bit, and (ii) every such discrete score is a valid lower bound on $\sup_f C(f)$ over continuous f, because the vector itself encodes an explicit step-function realiser.

Discrete formula. Given $f \in \mathbb{R}^N_{\geq 0}$, let $g = f \star f \in \mathbb{R}^{2N-1}_{\geq 0}$ be the discrete autoconvolution ($g_k = \sum_i f_i\, f_{k-i}$, computed by FFT in $O(N \log N)$). Let $h = 1/(2N)$ and $y = (0, g_0, \ldots, g_{2N-2}, 0) \in \mathbb{R}^{2N+1}$ be g zero-padded at both ends. The verifier formula is

$$\|g\|_2^2 \;=\; \frac{h}{3} \sum_{k=0}^{2N-1} \big(y_k^2 + y_k\, y_{k+1} + y_{k+1}^2\big) \;=\; \frac{2\sum_k g_k^2 + \sum_k g_k\, g_{k+1}}{6N},$$

$$\|g\|_1 \;=\; h \sum_k g_k, \qquad \|g\|_\infty \;=\; \max_k g_k, \qquad C(f) \;=\; \frac{\|g\|_2^2}{\|g\|_1 \cdot \|g\|_\infty}. \tag{3}$$

A self-contained reference implementation (`verify.py`, ∼20 lines of NumPy) ships with the companion repository `https://github.com/justinkang221/stochastic-agent-descent`; both reported scores match the platform verifier to machine precision.

Why the discrete score is a valid lower bound. Associate to $f \in \mathbb{R}^N_{\geq 0}$ the step function

$$\tilde{f}(x) \;=\; f_i \quad \text{for } x \in [ih, (i+1)h), \quad i = 0, \ldots, N-1,$$

and zero elsewhere. The continuous autoconvolution $\tilde{g} = \tilde{f} \star \tilde{f}$ is supported on $[0, 2Nh] = [0, 1]$ and is piecewise linear with knots on the h-grid: a direct calculation gives $\tilde{g}(kh) = h\, y_k$ for $k = 0, \ldots, 2N$. Because $\tilde{g}$ is piecewise linear, evaluating its continuous norms is *exact* (Simpson on each panel for L^2, trapezoidal for L^1, max-at-nodes for L^∞):

$$\|\tilde{g}\|_2^2 \;=\; \frac{h^3}{3} \sum_k \big(y_k^2 + y_k y_{k+1} + y_{k+1}^2\big), \quad \|\tilde{g}\|_1 \;=\; h^2 \sum_k g_k, \quad \|\tilde{g}\|_\infty \;=\; h \max_k g_k.$$

The factors of h cancel in the ratio, leaving exactly (3):

$$C(\tilde{f}) \;=\; \frac{\|\tilde{g}\|_2^2}{\|\tilde{g}\|_1\, \|\tilde{g}\|_\infty} \;=\; \frac{\|g\|_2^2}{\|g\|_1\, \|g\|_\infty} \;=\; C_{\text{discrete}}(f).$$

Therefore

$$\sup_{f \in L^1_+,\, \text{compact}} C(f) \;\geq\; C(\tilde{f}) \;=\; C_{\text{discrete}}(f),$$

i.e. every discrete score we report is realized by an explicit non-negative compactly-supported step function with the same C. The numbers in Section 3 are valid lower bounds on the continuous supremum, not artifacts of the discretization.

B Optimization Details

This section was generated by an agent with access to all worker scripts, using the prompt "Summarize the core algorithms and optimization techniques used." It is intended for those curious about some of the mathematical techniques used.

Dinkelbach with non-negative reparameterization. We optimize $C(f) = N(f)/D(f)$ via the standard Dinkelbach iteration: at outer step t with parameter $\lambda_t = C(f_t)$, solve the *auxiliary problem*

$$\max_{f \geq 0} \;\; N(f) - \lambda_t\, D(f) \;=\; \max_{f \geq 0} \;\; \|f \star f\|_2^2 \;-\; \lambda_t\, \|f \star f\|_1 \cdot \|f \star f\|_\infty, \tag{4}$$

and set $\lambda_{t+1} \leftarrow C(f_{t+1})$. The non-negativity constraint $f \geq 0$ is removed by the reparameterization $f = w^2$, $w \in \mathbb{R}^N$ unconstrained; the auxiliary problem is then a smooth unconstrained problem in w.

Smooth L^∞ surrogate. $\|f \star f\|_\infty$ is non-smooth. We replace it with the log-sum-exp surrogate $\widehat{L^\infty}_\beta(g) = \beta^{-1} \log \sum_k \exp(\beta\, g_k)$, normalized for numerical stability as $\widehat{L^\infty}_\beta(g) = g_{\max} + \beta^{-1} \log \sum_k \exp(\beta(g_k - g_{\max}))$. The auxiliary (4) with this surrogate is then minimized with L-BFGS using strong-Wolfe line search.

β-annealing schedule. A fixed high β pins $\widehat{L^\infty}_\beta$ to the global max but yields a near-degenerate gradient; a fixed low β smears mass over many near-maximal lags. We anneal β across the outer Dinkelbach iterations as $[10^8, 10^{10}, 10^{12}, 10^{13}, 10^{14}]$ (ported from ImprovEvolve, Section 3), running several Dinkelbach outers per β level. The agent driver controls which β schedule each worker uses; the schedule above is the one that proved load-bearing in the deployment.

Outer loop: perturb–polish with velocity. Each worker round (Algorithm 1) consumes (f_t, f_{t-1}) from the shared best-so-far file, forms an iterate difference $v = \alpha(f_t - f_{t-1})$ with $\alpha \in [0.2, 0.5]$, adds isotropic noise $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$ with $\sigma \sim 10^{-5}$, and runs Dinkelbach + L-BFGS as above. Published iterates replace f_t in the shared file and shift $f_{t-1} \leftarrow f_t$, sustaining the momentum across rounds.

C Three worker recipes

Three recipes are given below.

R1: Iterate-difference momentum. Pulls two recent iterates from the pool, forms a velocity along their difference, adds isotropic noise.

Algorithm 2 R1 – velocity-momentum perturb-polish.

Require: pool entries f_t, f_{t-1}; $\alpha \in [0.2, 0.5]$; $\sigma \sim 10^{-5}$; cycles K

$v \leftarrow \alpha\,(f_t - f_{t-1})$
for $k = 1 \dots K$ **do**
  $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$
  $\tilde{f} \leftarrow f_t + v + \varepsilon$
  $f^\star \leftarrow$ DINKELBACH-LBFGS$(\tilde{f})$
  if $C(f^\star) > C(f_t)$ **then**
    publish $f^\star$ to pool; $f_{t-1} \leftarrow f_t$; $f_t \leftarrow f^\star$; $v \leftarrow \alpha\,(f_t - f_{t-1})$
  end if
end for

R2: Region-aware destructive perturbation. Applies a multiplicative scale per contiguous region of the signal, either to one region at a time or to all regions jointly with random factors.

Algorithm 3 R2 – region-aware destructive perturbation.

Require: f; region partition $\{R_1, \dots, R_M\}$; variant list V of either single-region scales (R, s) or coupled multi-region scales $(s_1, \dots, s_M)$

for each variant $v \in V$ **do**
  $\tilde{f} \leftarrow f$
  if $v = (R, s)$ **then** ▷ single-region destructive scale
    $\tilde{f}[R] \leftarrow s \cdot \tilde{f}[R]$
  else ▷ coupled scale, $s_i \sim 1 + \mathcal{N}(0, \eta^2)$
    for $i = 1 \dots M$ **do**
      $\tilde{f}[R_i] \leftarrow s_i \cdot \tilde{f}[R_i]$
    end for
  end if
  $f^\star \leftarrow$ DINKELBACH-LBFGS$(\tilde{f})$
  if $C(f^\star) > C(f)$ **then**
    publish $f^\star$ to pool; $f \leftarrow f^\star$
  end if
end for

R3: Gradient-direction support activation. Reads $\nabla_f C$, restricts to inactive cells, and seeds a small positive step there before handing off to polish. Lets the $f = w^2$ inner loop grow new support that would otherwise stay pinned at zero.

Algorithm 4 R3 – gradient-direction support activation.

Require: f; step η ($\sim 10^{-4}$ to 10^{-3}); $\sigma \sim 10^{-5}$; cycles K

for $k = 1 \dots K$ **do**
  $g \leftarrow \nabla_f C(f)$ ▷ f-space gradient, evaluated densely
  $\mathcal{I} \leftarrow \{i : f_i = 0\}$ ▷ inactive cells
  $d_i \leftarrow \max(g_i, 0) \cdot \mathbf{1}[i \in \mathcal{I}]$ ▷ activate-only
  $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$
  $\tilde{f} \leftarrow f + \eta\, d/\|d\|_\infty + \varepsilon$
  $f^\star \leftarrow$ DINKELBACH-LBFGS$(\tilde{f})$
  if $C(f^\star) > C(f)$ **then**
    publish $f^\star$ to pool; $f \leftarrow f^\star$
  end if
end for