DUOEVO: RECURSIVE SELF-IMPROVEMENT VIA HETEROGENEOUS CHALLENGER–SOLVER CO-EVOLUTION FOR INDUSTRIAL FAULT DIAGNOSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Industrial automation requires reliable fault diagnosis under changing sensing conditions. Yet developing robust diagnostic models still relies on expert choices of training disturbances and network architectures. Recursive self-improvement (RSI) offers a route to automating model development, but most recent approaches leave the underlying neural architecture fixed. We introduce DUOEVO, a heterogeneous Challenger–Solver co-evolution framework that integrates challenge discovery with joint adaptation of model parameters and source-level structure. A Large Language Model(LLM)-based agent proposes executable noise programs and architecture revisions. Guided by the deployed Solver's weaknesses, the Challenger evolves a training curriculum; the Solver updates its parameters and evaluates structural revisions under matched curricula and budgets. Validation gates control deployment, and a patience rule retires unproductive structural search while challenge evolution and parameter learning continue. Each deployed Solver then defines the next round's challenge fitness, closing the recursive loop. On three bearing-diagnosis datasets, DUOEVO achieves the highest mean balanced accuracy among the compared methods on held-out synthetic corruptions under equal deployment-training budgets, with search costs reported separately. Fixed-reference evaluation shows bidirectional progress across all nine trajectories. Controlled analyses attribute most gains to Challenger evolution and a smaller, less certain contribution to structural adaptation. These results support a supervised self-improvement process that automates challenge discovery and model revision within a single feedback loop.

1 INTRODUCTION

Industrial automation relies on fault diagnosis to monitor equipment health and support maintenance decisions (Lei et al., 2020; Fink et al., 2020). In vibration-based diagnosis, measurement noise can obscure fault signatures and make reliable classification difficult (Randall & Antoni, 2011). Deep learning can extract diagnostic features from raw signals, but robust performance still depends on training disturbances and network design (Zhang et al., 2017; Zhao et al., 2020; Liao et al., 2025). Practitioners must therefore select training disturbances, design suitable networks, and assess these choices through repeated experiments (Li et al., 2022). This reliance on expert-led development is becoming a bottleneck to deploying reliable diagnostic models at scale (Martínez-Arellano & Ratchev, 2025), which raises a central question: *can a learning system use experimental feedback to make and revise these decisions with less expert intervention?*

Recursive self-improvement (RSI) offers a framework for such feedback-driven development, in which the learner's current capabilities inform its next adaptation step. Recent approaches evolve training challenges, such as tasks and examples that target the learner's current weaknesses (Huang et al., 2026; Zhao et al., 2025; Yang et al., 2026; Heng et al., 2026). Other systems couple parameter learning with changes to agent runtime components (Fan & Huang, 2026; Xue et al., 2026). Most nevertheless retain a fixed neural architecture. In fault diagnosis, robustness failures may reflect limitations in feature extraction as well as insufficient coverage of disturbances during training. Parameter adaptation operates within the existing architecture, leaving the benefit of structural revision unexplored. Extending RSI to network development therefore requires both discovering informa-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

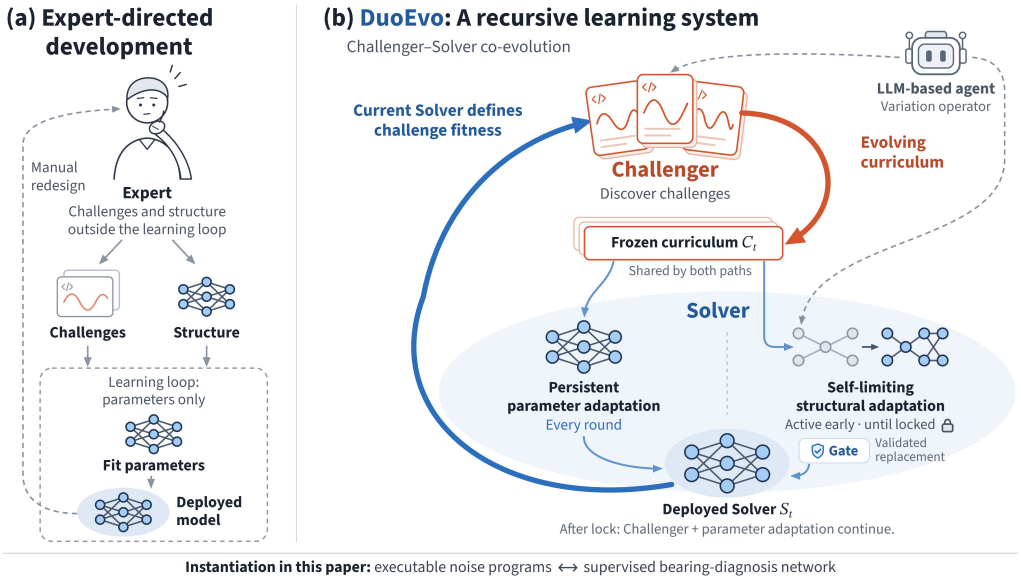


Figure 1: **Motivation for recursive Challenger–Solver co-evolution.** (a) Expert-directed development chooses challenges and model structure outside a parameter-only learning loop. (b) DUOEVO uses an LLM-based agent to revise executable challenges and Solver source code: the deployed Solver scores Challenger programs, whose selected curriculum drives persistent parameter adaptation and self-limiting, validation-gated structural adaptation. The updated deployed Solver defines the next round's challenge fitness; after structural search locks, Challenger evolution and parameter adaptation continue.

tive challenges and testing architectural responses. These decisions are coupled: challenges must track the model's changing weaknesses, while structural candidates must be compared with continued training of the incumbent. Otherwise, changes in training data or optimization budgets can be mistaken for structural gains. The central problem is to coordinate challenge discovery, parameter learning, and architectural revision through controlled experimentation.

We introduce DUOEVO, a heterogeneous Challenger–Solver co-evolution framework for joint challenge and model adaptation within a recursive feedback loop (Figure 1(b)). An large language model (LLM)-based agent provides a shared proposal mechanism for the Challenger and Solver, using observed outcomes to guide subsequent adaptation. The Challenger evolves additive-noise programs against the frozen deployed Solver, scoring their effect on classification margin at a fixed signal-to-noise ratio. Selected programs form a curriculum targeting the Solver's current weaknesses. The Solver continues parameter training and evaluates agent-proposed network revisions on this curriculum. Its deployed successor defines challenge fitness for the next round, closing the loop. Architecture search is thus embedded in an iterative experimental process whose training challenges also respond to model updates. The coupling is heterogeneous: adversarial program search and supervised network learning use distinct update mechanisms, with each directing the other's adaptation.

Structural adaptation is conditional within this loop. Candidate architectures and the incumbent are compared under matched curricula and training budgets, and validation gates govern deployment. When no candidate qualifies, the system retains the architecture and uses its updated parameters. A patience rule terminates structural search after repeated non-promotions, subject to a preset search cap. Challenge evolution and parameter learning continue after search stops. This policy bounds structural exploration while allowing parameter learning to remain active, so joint adaptation does not require architectural changes at every round.

We evaluate DUOEVO on three bearing fault diagnosis benchmarks, using one adaptation policy without per-round human intervention. It achieves the highest mean balanced accuracy among the compared methods on synthetic noise generators excluded from training and model selection.

Deployment-training budgets are equalized; search costs are reported separately. To distinguish reciprocal improvement from opponent drift (Cliff & Miller, 1995), we evaluate Solver generations on common challenges and Challenger generations against a common Solver panel. Both directions improve across all nine trajectories. Ablations and replay under identical curricula attribute most gains to Challenger evolution and a smaller positive mean contribution to the structural path, with greater uncertainty for the latter. These analyses assess both the complete development process and the contributions of its adaptation mechanisms.

Our contributions are:

- **Agent-guided heterogeneous co-evolution.** We introduce DUOEVO, where an LLM-based agent proposes heuristic revisions to executable noise programs and diagnostic networks. Evolving challenges guide parameter and architectural adaptation, while each updated Solver directs subsequent challenge discovery.
- **Controlled structural adaptation.** We develop a validation-gated update policy that compares architectural candidates with the incumbent under matched curricula and training budgets. A stopping rule bounds structural search while challenge evolution and parameter learning continue.
- **Evidence of reciprocal progress.** Across three fault-diagnosis benchmarks, DUOEVO achieves the highest mean balanced accuracy among the compared methods on held-out synthetic corruptions under matched deployment-training budgets. Fixed-reference evaluation shows reciprocal progress across all nine trajectories, while ablations and curriculum replay separate challenge-evolution and structural-path effects.

2 RELATED WORK

Robust fault diagnosis and its automation. Noise-robust diagnosis uses architectural priors including wide-kernel convolution, shrinkage, time–frequency modeling, quadratic neurons, and attention (Zhang et al., 2017; Zhao et al., 2020; Jia et al., 2023; Chen et al., 2024; Liao et al., 2023; Hou et al., 2023; Wang et al., 2023). ClassBD, our anchor, jointly learns blind-deconvolution filters and the classifier (Liao et al., 2025). Neural architecture search automates gearbox and bearing model design (Li et al., 2020; Wang et al., 2020a), and AutoML extends to predictive-maintenance pipelines (Tornede et al., 2021). These approaches automate model or pipeline design without recursively evolving corruptions according to the deployed diagnostic model. DUOEVO makes the training challenge distribution itself adaptive to the current Solver.

Adaptive challenges and co-evolution. Curriculum learning adapts example difficulty (Bengio et al., 2009); adversarial training constructs model-dependent perturbations within a prescribed threat set (Madry et al., 2018; Wang et al., 2020b; Yin & Ruan, 2024); PAIRED and POET adapt tasks or environments (Dennis et al., 2020; Wang et al., 2019). Data–model co-evolution predates modern RSI: Mayer (1998) co-evolved architectures and training sets cooperatively, and Shapiro (1998) studied its generalization. Recent RSI systems couple LLM learners with generated challenges or data (Huang et al., 2026; Zhao et al., 2025; Yang et al., 2026), executable transformations (Heng et al., 2026), or evolving runtime harnesses (Fan & Huang, 2026; Xue et al., 2026). In contrast to cooperative fitness or two trainable generative policies, DUOEVO gives executable corruption programs adversarial fitness from a label-supervised Solver whose parameters and deployed structure can change. Moving opponents make contemporaneous performance an unreliable measure of absolute progress (Cliff & Miller, 1995; Watson & Pollack, 2001); archives and fixed references help separate progress from opponent drift (Rosin & Belew, 1997; de Jong & Pollack, 2004; Cartlidge & Bullock, 2004).

Architecture and program search. NAS, neuroevolution, and population-based training adapt architectures, parameters, or hyperparameters (Elsken et al., 2019; Stanley & Miikkulainen, 2002; Real et al., 2019; Jaderberg et al., 2017). DAAS jointly searches architectures and augmentation policies over predefined operations (Wang et al., 2021). Quality-diversity methods retain diverse high-performing candidates (Mouret & Clune, 2015; Pugh et al., 2016), while LLMatic, FunSearch, and AlphaEvolve search executable source code with LLM guidance (Nasir et al., 2024; Romera-Paredes et al., 2024; Novikov et al., 2025). DUOEVO combines program-space variation with recursive Solver conditioning: a promoted Solver changes not only its response to current challenges but which executable challenges are selected next.

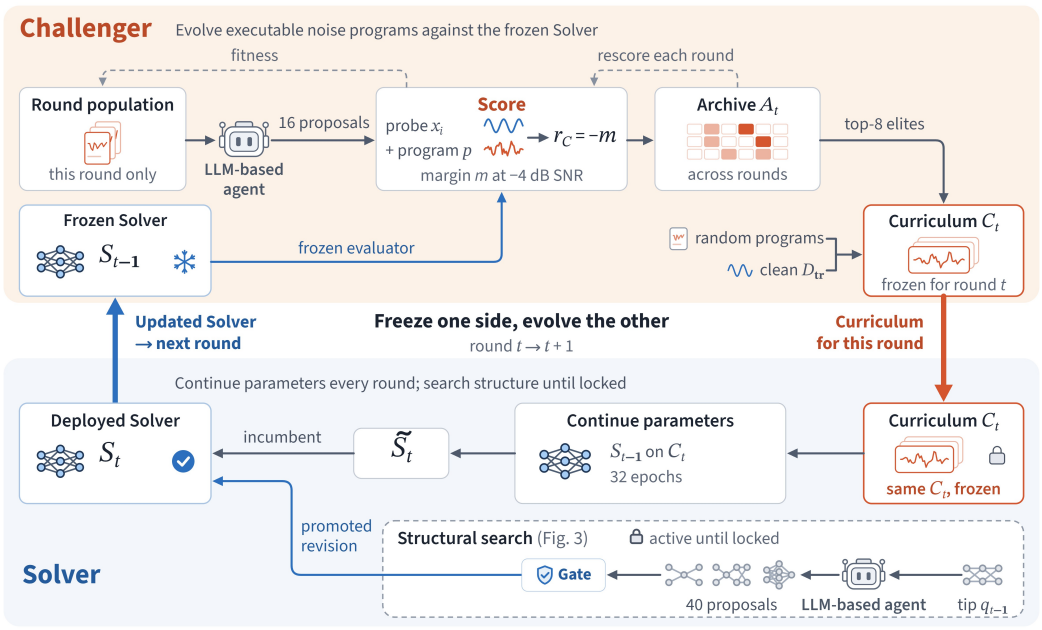


Figure 2: The recursive DUOEVO loop. The Challenger evolves executable noise programs against the frozen deployed Solver and materializes the selected programs into curriculum C_t. The Solver continues parameter adaptation on the same frozen C_t and conditionally revises its architecture (Figure 3). The deployed S_t then defines the next round's challenge fitness, while the LLM-based agent supplies executable code proposals for both noise-program and architecture search.

3 DUOEVO: RECURSIVE CHALLENGER–SOLVER CO-EVOLUTION

Overview: freeze one side, adapt the other. DUOEVO couples two heterogeneous forms of adaptation. The Challenger changes what the learner sees: its state is a program archive $\mathcal{A}_t$ (Figure 2), the executable noise programs retained across rounds, starting from $\mathcal{A}_0 = \varnothing$. The deployed Solver $S_t = (a_t, w_t)$, with architecture source a_t and parameters w_t, changes what the learner is. We use *co-evolution* in the reciprocal sense: the two sides share no optimizer, but each side's update depends on the current state of the other. A fixed LLM proposes code for both sides and is never updated.

We initialize S_0 by training anchor architecture a_0 for 8 epochs on a curriculum C_0 built from randomly sampled noise programs. Each subsequent round has two controlled stages. First, S_{t-1} is frozen while the Challenger searches for programs that expose its current weaknesses, updates $\mathcal{A}_t$, and builds a new curriculum C_t. Second, C_t is frozen while the Solver adapts to it: its weights always, its architecture only while structural search remains active. The deployed S_t then becomes the fitness reference for the next Challenger round:

$$(\mathcal{A}_{t-1}, S_{t-1}) \xrightarrow{\text{Challenger}} (\mathcal{A}_t, C_t) \xrightarrow{\text{Solver adaptation}} S_t \to \mathcal{A}_{t+1}.$$

The alternating freeze gives both sides a fixed reference: programs compete against one deployed Solver, and Solver alternatives train on identical data (Figure 2; Algorithm 1). Structural search is a conditional Solver-side mechanism, not a third co-evolving component.

3.1 CHALLENGER: EVOLVING THE TRAINING DISTRIBUTION

Executable programs as reusable challenges. The Challenger represents a corruption as an executable noise program rather than a stored waveform or a per-example perturbation. Each valid `generate_noise(rng, fs, n)` program must be finite and seed-reproducible and cannot access the clean input signal, so the Challenger searches for reusable corruption mechanisms.

Round-wise search and persistent archive. Each round runs a fresh 16-proposal OpenEvolve search (Sharma, 2025) from the same seed program. Within the round, scored proposals return to the

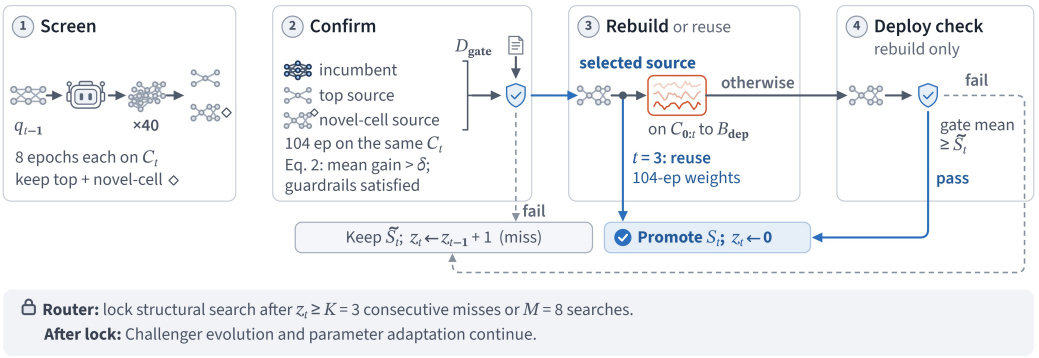


Figure 3: **Conditional structural update** (search box in Figure 2): screening, confirmation against the incumbent on D_{gate} (Eq. 2), and rebuild or reuse at the deployment budget.

round's population, and the fixed LLM rewrites a parent drawn from it, conditioned on the parent's score against the frozen S_{t-1} and on the round's top-scoring and diverse programs, so each rewrite is steered by what currently lowers S_{t-1}'s margin; this population does not persist across rounds. Cross-round memory is carried instead by the quality-diversity archive $\mathcal{A}_t$, which keeps one elite per waveform-behavior cell (405 cells; Appendix C) and absorbs every valid program of the round. It acts through rescoring and curriculum materialization, not through prompts, so what changes across rounds is not the proposer but the Solver that decides which programs remain difficult.

Solver-conditioned fitness. Programs are scored on class-balanced development probes (x_i, y_i) fixed once per run (sizes in Appendix C). Let $\widetilde{x}_i(p)$ denote the probe after adding program p's power-normalized waveform at the fixed attack SNR of -4 dB; normalization prevents a program from gaining fitness through amplitude alone. A program is fitter the more it lowers the Solver's margin between the true class and its strongest alternative:

$$r_C(p; S_{t-1}) = -m(p; S_{t-1}), \qquad m(p; S) = \frac{1}{n}\sum_{i=1}^{n}\Big[P_S(y_i \mid \widetilde{x}_i(p)) - \max_{k \neq y_i} P_S(k \mid \widetilde{x}_i(p))\Big]. \quad (1)$$

Only the deployed Solver defines this score; unpromoted architecture candidates never influence Challenger selection. At the start of every round, before new proposals enter, the archive is rescored against the frozen S_{t-1}, so old elites and new programs compete under the same Solver and each cell keeps its highest-fitness program. A program that exposed a weakness of an earlier Solver can therefore lose its rank once the Solver has adapted, shifting selection toward weaknesses that remain; this rescoring is the feedback from the model side to the challenge side.

Curriculum materialization. After search, up to eight highest-fitness archive elites are combined with fresh random programs to generate noisy examples amounting to 35% of the clean training set D_{tr}, with equal target shares from elites and fresh programs and random backfill when needed. The materialized C_t is frozen and every Solver update in round t uses its arrays, so challenges evolve *between* rounds while Solver alternatives are compared under matched data *within* one.

3.2 Solver: Continual Parameter and Self-Limiting Structural Adaptation

Continual parameter adaptation. Every round, the deployed Solver first continues training on the frozen C_t for 32 epochs without changing its architecture, yielding the weight-updated incumbent $\widetilde{S}_t = (a_{t-1}, \widetilde{w}_t)$. Including the 8-epoch bootstrap, the deployment-training budget after round t is $B_{\text{dep}} = 8 + 32t$ epochs. If no structural revision is attempted or accepted, $S_t = \widetilde{S}_t$.

Why structural adaptation is conditional. Whether the anchor architecture suffices is unknown in advance, so revision is allowed early, when new challenges may expose representational limits that weight updates alone do not resolve efficiently. Structural search costs far more than weight continuation and its marginal benefit can diminish as the architecture adapts. Consecutive rounds

without a deployable revision are therefore an empirical signal of *structural saturation* under the current search space, proposer, curriculum, and gate; the Solver then relies on its parameter path.

Matched structural comparison. While structural search is active (Section 3.3; Figure 3), the fixed LLM proposes source-level architecture variants from the current search lineage. After short screening, the incumbent and two candidate sources are retrained from scratch on the same frozen C_t, with the same seed and epoch budget (Appendix C), so a candidate cannot appear superior merely because it saw different data or received more optimization.

Validation-gated deployment. A proposal never replaces the deployed Solver automatically. Finalists are evaluated on a frozen validation fixture D_{gate} disjoint from D_{tr}. The best non-incumbent finalist is selected only if it raises the mean accuracy over five noise families by more than a margin δ over the retrained incumbent while keeping clean accuracy, the mean of the two weakest families, and every individual family within fixed regression limits (Eq. 2); all thresholds are calibrated before the main runs and then frozen (Appendix C). The selected source is then brought to the current budget B_{dep}: at $t = 3$, where $B_{\mathrm{dep}} = 104$, it reuses its confirmation weights; otherwise it is rebuilt over the saved curricula $C_{0:t}$ and deployed only if the rebuilt checkpoint matches or exceeds $\widetilde{S}_t$ on the gate (Appendix C.2). A rejected revision leaves $S_t = \widetilde{S}_t$. In either case, the deployed S_t immediately becomes the fitness reference for the next Challenger round.

3.3 Self-limiting structural adaptation

Structural search is intentionally temporary. The router uses patience $K = 3$ and a cap of $M = 8$ searches. A promotion resets the miss counter z_t; a round without a deployable revision, including a refused deployment, increments it. Reaching $z_t = K$, our operational signal of structural saturation, or exhausting the cap permanently locks the architecture by assigning the identity architecture action to all later rounds. If the architecture deployed at lock time differs from the anchor, a one-time switch audit retrains the anchor under that architecture's confirmation curriculum, seed, and budget; the evolved source is retained only if it remains non-inferior in mean and within the guardrails.

After lock, Challenger evolution and parameter adaptation continue through all remaining rounds. The gate, the identity action, and the router make deployed structural changes conditionally non-degrading on the gate, keep the fixed-architecture policy inside the policy class, and bound total structural cost (Propositions 1–3; proofs in Appendix B).

4 Experiments

4.1 Experimental protocol

Datasets and splits. We use the Paderborn University (PU) (Lessmeier et al., 2016), Case Western Reserve University (CWRU) (Smith & Randall, 2015), and Jiangnan University (JNU) (Li et al., 2013) bearing datasets, with 3, 10, and 4 classes. HN evaluation evaluation measures corruption generalization on independent test repetitions or time segments from the training operating conditions. JNU is anti-alias resampled from 50 to 12 kHz so absolute-frequency noise programs keep their meaning. The 2048-sample evaluation windows share no raw samples with training (Appendix C).

Perturbations and evaluation. Each round augments 35% of the clean set, split between at most eight archive elites and fresh random programs, at $\{0, -3, -6\}$ dB. Checkpoint selection reads a held-out validation split carrying the same curriculum noise as training, following the published ClassBD recipe. The 256-example gate uses independent noise at $\{-1.5, -4.5, -7.5\}$ dB from five in-language families, i.e., additive families expressible in the Challenger's program interface. The report-only HN battery crosses four held-out generator implementations (chirp, multiplicative speckle, heavy-tailed burst, and their mixture) at $\{0, -3, -6\}$ dB and reports macro balanced accuracy over the 12 cells; it selects neither checkpoints nor rounds, and its generators are excluded from training and selection (Appendix C). This tests transfer to held-out implementations of synthetic corruption; their temporal or spectral properties can overlap with training primitives.

Table 1: **Main comparison by noise severity.** Held-out-noise (HN) balanced accuracy (%), mean ± sample SD over three seeds at 648 deployment epochs. Each SNR averages four held-out generators; Mean averages all 12 cells. Bold marks the best mean per dataset and column.

Data	Configuration	0 dB	−3 dB	−6 dB	Mean
PU	TFN-STTF (2024)	66.63±2.60	58.91±2.24	50.75±2.84	58.76±2.44
	QCNN (2023)	86.82±1.25	78.30±0.85	64.61±1.20	76.57±0.35
	DRSN-CW (2020)	77.52±0.84	70.44±0.60	65.30±0.52	71.09±0.59
	WDCNN (2017)	94.31±0.17	88.74±0.17	81.32±0.57	88.13±0.28
	GTFENet (2023)	98.31±0.33	94.55±0.65	87.95±1.22	93.61±0.45
	ClassBD-BDWDCNN (2025)	98.84±0.06	95.60±0.19	88.68±0.93	94.37±0.25
	ClassBD w/ MART+LORE (2024)	**99.47±0.14**	96.07±1.20	83.64±2.31	93.06±1.21
	DuoEvo	99.36±0.20	**97.48±0.41**	**93.16±0.78**	**96.67±0.46**
CWRU	TFN-STTF (2024)	37.20±4.89	30.14±2.97	23.16±1.73	30.16±2.68
	QCNN (2023)	91.92±2.10	81.77±3.84	58.34±3.23	77.34±2.80
	DRSN-CW (2020)	90.84±1.47	77.79±3.65	60.22±2.52	76.28±2.54
	WDCNN (2017)	98.33±0.34	92.85±0.14	79.30±1.90	90.16±0.58
	GTFENet (2023)	99.30±0.34	94.70±1.56	80.70±5.03	91.56±2.27
	ClassBD-BDWDCNN (2025)	**99.80±0.10**	97.57±0.19	83.60±2.86	93.66±0.93
	ClassBD w/ MART+LORE (2024)	99.71±0.09	98.83±0.51	91.02±2.97	96.52±1.19
	DuoEvo	99.74±0.08	**98.95±0.38**	**94.93±0.78**	**97.87±0.40**
JNU	TFN-STTF (2024)	73.67±1.83	62.30±1.35	54.06±1.68	63.35±1.60
	QCNN (2023)	63.44±8.87	50.51±8.39	37.71±5.51	50.55±7.58
	DRSN-CW (2020)	83.50±2.17	73.06±3.55	59.64±4.81	72.07±3.40
	WDCNN (2017)	96.46±1.21	89.23±2.97	76.08±3.05	87.26±2.41
	GTFENet (2023)	98.66±0.13	93.56±0.61	76.69±1.52	89.64±0.42
	ClassBD-BDWDCNN (2025)	98.39±0.52	94.49±0.88	85.86±1.18	92.91±0.66
	ClassBD w/ MART+LORE (2024)	99.23±0.14	96.17±0.89	85.80±1.22	93.73±0.70
	DuoEvo	**99.31±0.11**	**96.98±0.98**	**88.43±5.22**	**94.91±2.10**

Policies, budgets, and statistics. The common anchor is ClassBD-BDWDCNN (Liao et al., 2025). DuoEvo uses $K = 3$ and an eight-search cap; Section 4.4 defines the fixed-architecture, random-noise, and donor-replay controls. Trajectories receive an 8-epoch bootstrap plus 32 epochs per round, reaching 648 epochs at $T = 20$, and are read at the final round; seeds are 11, 23, and 37. Automated search, including screening, confirmation, rebuilds, and LLM calls, is accounted for separately from deployment training (Appendix F). Total training charges are 2,544–4,744 official-equivalent epochs (OEE), including the 648 deployment epochs; this is epoch accounting, not equal total compute. For each dataset, seed, and configuration, all reported HN cells use the same endpoint checkpoint.

4.2 Endpoint generalization under held-out corruptions

We compare with ClassBD, DRSN-CW, GTFENet, QCNN, WDCNN, and TFN-STTF (Liao et al., 2025; Zhao et al., 2020; Jia et al., 2023; Liao et al., 2023; Zhang et al., 2017; Chen et al., 2024). All run for 648 deployment-training epochs under the shared optimizer and checkpoint-selection protocol; ClassBD-anchored controls additionally share the same backbone and loss recipe (Appendix C).

ClassBD trained with MART+LORE provides an adversarial-training control on the same anchor and static curriculum; its attack budget and KL weight are selected per dataset using the preregistered validation protocol in Appendix C.5 (Wang et al., 2020b; Yin & Ruan, 2024).

Endpoint results. DuoEvo has the highest aggregate HN mean on all three datasets, exceeding static ClassBD by 2.30, 4.21, and 1.99 points on PU, CWRU, and JNU. Gains are largest at −6 dB (+4.48, +11.33, and +2.56 points), where static methods degrade most. Against the validated MART+LORE control, DuoEvo also has higher means on all three datasets (+3.61, +1.35, and +1.17 points). Appendix E.2 reports paired uncertainty; we interpret these comparisons as mean differences rather than universal statistical superiority.

Performance across the deployment budget. The advantage is not tied to the 648-epoch readout: across the 20 round-end budgets (Figure 4), DuoEvo exceeds the 648-epoch static model at 19, 20, and 15 on PU, CWRU, and JNU, from 72 epochs onward on PU and at every budget on CWRU. Its

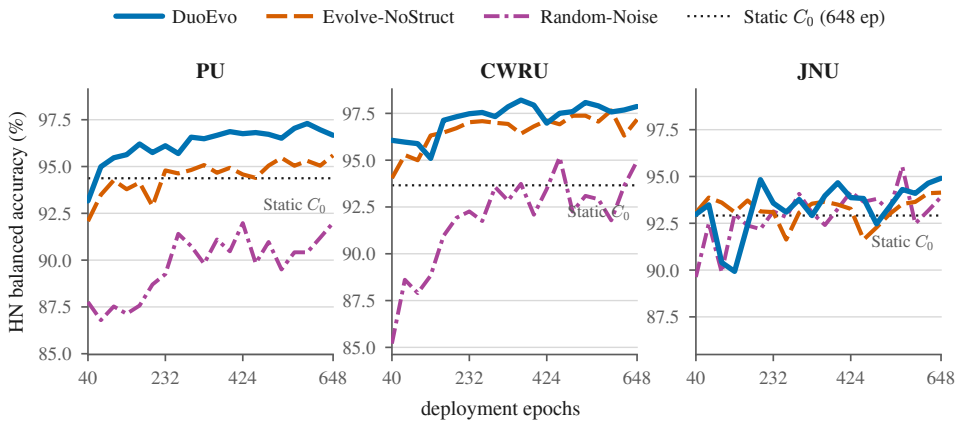


Figure 4: **Performance across the deployment budget.** Three-seed mean HN accuracy after the shared bootstrap; configurations as in Section 4.4. Dotted lines: the 648-epoch static baseline.

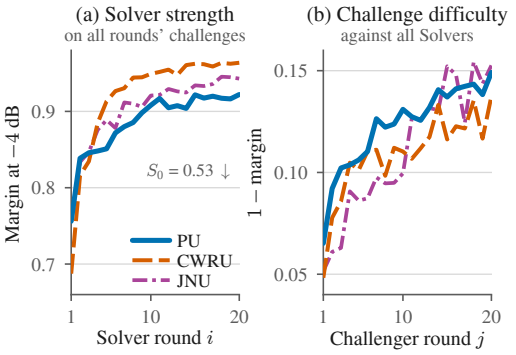


Figure 5: **Fixed-reference cross-generation progress.** Three-seed mean margin at -4 dB. (a) Later Solvers score higher on the same Challenger elites. (b) Later Challenger elites lower the margin of the same Solver panel.



Figure 6: **Replay-controlled structural-path effect.** Final HN difference (pp), DUOEVO minus REPLAY-ANCHOR, under byte-identical donor curricula. Open markers denote the two runs without structural promotion.

mean over the last five budgets exceeds that over the first five by 1.81, 1.79, and 2.43 points, so the gain accumulates over rounds rather than appearing at one checkpoint.

4.3 CROSS-GENERATION EVIDENCE OF RECURSIVE CO-EVOLUTION

Because both sides change, an online score against the current opponent is a moving target (Cliff & Miller, 1995); we therefore use fixed-reference cross-generation evaluation. Every deployed Solver S_i is scored on the elites of every Challenger generation j, yielding a 21×20 margin table per run (Appendix D.1). Row means compare Solver generations on one common challenge set, while column means compare Challenger generations against one common Solver panel.

The fixed-reference results show both directions of progress. Mean Solver margin on the common challenge set rises from 0.53 at S_0 to 0.94 at S_{20}, while Challenger elites from rounds 16–20 induce lower margins on the common Solver panel than elites from rounds 1–5 (0.86 vs. 0.92; Figure 5). Both trends hold in all nine trajectories: each side improves against fixed versions of the other, the reciprocal progress that recursive co-evolution requires. S_{20} also remains within 0.04 of the best margin reached on each round's elites, indicating limited forgetting on earlier evolved challenges.

Table 2: **Data- and model-level ablations.** HN balanced accuracy (%), averaged over all 12 generator–SNR cells; mean ± sample SD over three seeds at 648 deployment epochs. Bold marks the highest mean per dataset. Table 7 gives the full SNR breakdown.

Configuration	PU	CWRU	JNU
Static ClassBD	94.37 ± 0.25	93.66 ± 0.93	92.91 ± 0.66
RANDOM-NOISE	92.00 ± 0.63	94.96 ± 1.20	93.91 ± 0.20
EVOLVE-NOSTRUCT	95.60 ± 0.82	97.17 ± 0.78	94.14 ± 1.11
DUOEVO	$\mathbf{96.67 \pm 0.46}$	$\mathbf{97.87 \pm 0.40}$	$\mathbf{94.91 \pm 2.10}$

4.4 ATTRIBUTING ENDPOINT GAINS TO CHALLENGER AND SOLVER PATHS

Table 2 compares four ClassBD-anchored configurations. Static ClassBD fixes C_0; RANDOM-NOISE keeps the round-wise augmentation schedule but replaces the Challenger with uniformly sampled primitive-library programs, without an archive or Solver-conditioned scoring; EVOLVE-NOSTRUCT restores the evolving Challenger with a fixed architecture; and DUOEVO adds conditional structural revision of the Solver. All four share the training recipe of Section 4.2.

Challenger contribution. With architecture fixed, EVOLVE-NOSTRUCT exceeds static ClassBD by 1.22, 3.52, and 1.22 points. Replacing the Challenger with matched-volume random programs lowers the mean by 3.60, 2.22, and 0.23 points. This control measures the Challenger branch as a whole: LLM program variation, archive retention, and Solver-conditioned selection (Appendix C.3).

Structural-path contribution. DUOEVO exceeds EVOLVE-NOSTRUCT by 1.07, 0.70, and 0.77 points, but this online difference also includes post-promotion curriculum changes. We therefore use REPLAY-ANCHOR, which fixes the anchor architecture while replaying the exact materialized curricula of its matched DUOEVO donor. Under fixed curricula, the donor-conditioned effect averages $+0.87$, $+0.72$, and $+1.50$ pp on PU, CWRU, and JNU (Figure 6). The two no-promotion runs reproduce exact zeros; paired 95% intervals exclude zero on PU but not on CWRU or JNU. An anchor-retraining control finds a positive architecture-path component in all five runs that retain an evolved architecture (Appendix E.1). Structural search locks by epoch 232 in all runs; subsequent progress comes from evolving curricula and weight adaptation (family-level effects in Appendix E.2).

5 CONCLUSION

DUOEVO brings the choice of training disturbances and network revisions into a recursive loop between the Challenger and Solver, extending self-improvement from the training distribution to the learner's source-level structure. On PU, CWRU, and JNU, the final Solvers achieve the highest mean accuracy on held-out corruptions among the compared methods under equal deployment budgets. Evaluation against fixed references shows reciprocal progress across all nine trajectories. Controlled analyses indicate that most of the gain comes from Challenger evolution, while the structural path provides a smaller and less certain contribution. Overall, these results support bounded recursive improvement of a supervised learner through Solver-conditioned challenge evolution and gated architectural revision.

Limitations. Evaluation is limited to synthetic corruptions and same-condition splits; transfer to measured noise, cross-condition shifts, and different LLM proposers remains untested. The Challenger search space is restricted to additive, signal-independent programs. With three seeds, structural-path effects remain uncertain on CWRU and JNU, and cross-generation analysis does not isolate online recursion from an offline-evolved frozen curriculum. Structural search also adds 1,896 to 4,096 OEE per run. Finally, CWRU and JNU share source-record normalization statistics across temporal splits.

AI USE STATEMENT

Generative AI tools assisted with brainstorming and refining research hypotheses and conceptual framing, experimental design, method implementation, result interpretation, manuscript organization, English drafting and revision, LaTeX preparation, figure refinement, code review, experimental diagnosis, and supplementary analysis code. Generative AI was not used to generate or alter the benchmark datasets, determine reported numerical results, or independently validate mathematical claims or proofs. Separately, the method itself uses a fixed LLM to propose executable noise and architecture programs (Section 3); every accepted program is saved with its hash (Appendix C). The authors reviewed all AI-assisted work. The authors checked AI-assisted analysis code and result interpretations against the saved run artifacts. The control and replay summaries reported here can be regenerated from the saved run artifacts with the evidence exporter described in Appendix G. The authors are responsible for the scientific claims, citations, code, and results.

REPRODUCIBILITY STATEMENT

Appendix C specifies the executed data protocol, including splits, boundary purging, normalization, and loader identifiers (Table 3); the frozen configuration (Table 4); the calibration of the gate thresholds; the executable interfaces and search spaces for noise and Solver programs; and the per-dataset deployment settings. Algorithm 1 states the deployed policy, and Appendix B gives complete proofs. Appendix C.5 documents the adversarial-training baseline and its preregistered selection protocol, Appendix E the replay checks, and Appendix F the per-run compute. Because LLM sampling is not deterministically reproducible, every accepted noise and architecture source is saved with its SHA-256 hash, and the materialized curricula are stored so that training can be replayed byte for byte. We provide the source code, experiment configurations, and evidence exporter as supplementary material; the datasets, checkpoints, and materialized curricula are not included.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 41–48, 2009. doi: 10.1145/1553374.1553380.

John Cartlidge and Seth Bullock. Combating coevolutionary disengagement by reducing parasite virulence. *Evolutionary Computation*, 12(2):193–222, 2004. doi: 10.1162/106365604773955148.

Qian Chen, Xingjian Dong, Guowei Tu, Dong Wang, Changming Cheng, Baoxuan Zhao, and Zhike Peng. TFN: An interpretable neural network with time-frequency transform embedded for intelligent fault diagnosis. *Mechanical Systems and Signal Processing*, 207:110952, 2024. doi: 10.1016/j.ymssp.2023.110952.

Dave Cliff and Geoffrey F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Advances in Artificial Life (ECAL 1995)*, volume 929 of *Lecture Notes in Computer Science*, pp. 200–218. Springer, 1995. doi: 10.1007/3-540-59496-5_300.

Edwin D. de Jong and Jordan B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004. doi: 10.1162/106365604773955139.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13049–13061, 2020.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. URL https://jmlr.org/papers/v20/18-598.html.

Tianyu Fan and Chao Huang. HELIX: Model-harness co-evolution for recursive self-improvement. *arXiv preprint arXiv:2608.13951*, 2026. doi: 10.48550/arXiv.2608.13951. URL https://arxiv.org/abs/2608.13951.

Olga Fink, Qin Wang, Markus Svensén, Pierre Dersin, Wan-Jui Lee, and Melanie Ducoffe. Potential, challenges and future directions for deep learning in prognostics and health management applications. *Engineering Applications of Artificial Intelligence*, 92:103678, 2020. doi: 10.1016/j.engappai.2020.103678.

Yongrui Heng, Chaoya Jiang, Han Yang, Shikun Zhang, and Wei Ye. EVE: Verifiable self-evolution of MLLMs via executable visual transformations. *arXiv preprint arXiv:2604.18320*, 2026. doi: 10.48550/arXiv.2604.18320. URL https://arxiv.org/abs/2604.18320.

Yandong Hou, Jinjin Wang, Zhengquan Chen, Jiulong Ma, and Tianzhi Li. Diagnosisformer: An efficient rolling bearing fault diagnosis method based on improved Transformer. *Engineering Applications of Artificial Intelligence*, 124:106507, 2023. doi: 10.1016/j.engappai.2023.106507.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=oSGOE5z2iU.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017. doi: 10.48550/arXiv.1711.09846.

Linshan Jia, Tommy W. S. Chow, and Yixuan Yuan. GTFE-Net: A gramian time frequency enhancement CNN for bearing fault diagnosis. *Engineering Applications of Artificial Intelligence*, 119:105794, 2023. doi: 10.1016/j.engappai.2022.105794.

Yaguo Lei, Bin Yang, Xinwei Jiang, Feng Jia, Naipeng Li, and Asoke K. Nandi. Applications of machine learning to machine fault diagnosis: A review and roadmap. *Mechanical Systems and Signal Processing*, 138:106587, 2020. doi: 10.1016/j.ymssp.2019.106587.

Christian Lessmeier, James Kuria Kimotho, Detmar Zimmer, and Walter Sextro. Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: A benchmark data set for data-driven classification. In *PHM Society European Conference*, volume 3, 2016. doi: 10.36001/phme.2016.v3i1.1577.

Ke Li, Xueliang Ping, Huaqing Wang, Peng Chen, and Yi Cao. Sequential fuzzy diagnosis method for motor roller bearing in variable operating conditions based on vibration analysis. *Sensors*, 13 (6):8013–8041, 2013. doi: 10.3390/s130608013.

Xudong Li, Yang Hu, Jianhua Zheng, and Mingtao Li. Neural architecture search for fault diagnosis, 2020. URL https://arxiv.org/abs/2002.07997.

Xudong Li, Jianhua Zheng, Mingtao Li, Wenzhen Ma, and Yang Hu. One-shot neural architecture search for fault diagnosis using vibration signals. *Expert Systems with Applications*, pp. 116027, 2022. doi: 10.1016/j.eswa.2021.116027.

Jing-Xiao Liao, Hang-Cheng Dong, Zhi-Qi Sun, Jinwei Sun, Shiping Zhang, and Feng-Lei Fan. Attention-embedded quadratic network (qttention) for effective and interpretable bearing fault diagnosis. *IEEE Transactions on Instrumentation and Measurement*, 72:1–13, 2023. doi: 10.1109/TIM.2023.3259031.

Jing-Xiao Liao, Chao He, Jipu Li, Jinwei Sun, Shiping Zhang, and Xiaoge Zhang. Classifier-guided neural blind deconvolution: A physics-informed denoising module for bearing fault diagnosis under noisy conditions. *Mechanical Systems and Signal Processing*, 222:111750, 2025. doi: 10.1016/j.ymssp.2024.111750.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. doi: 10.48550/arXiv.1706.06083.

Giovanna Martínez-Arellano and Svetan Ratchev. Towards frugal industrial AI: A framework for the development of scalable and robust machine learning models in the shop floor. *The International Journal of Advanced Manufacturing Technology*, 138:169–191, 2025. doi: 10.1007/s00170-024-14508-5.

Helmut A. Mayer. Symbiotic coevolution of artificial neural networks and training data sets. In *Parallel Problem Solving from Nature – PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pp. 511–520. Springer, 1998. doi: 10.1007/BFb0056893. URL https://link.springer.com/chapter/10.1007/BFb0056893.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. doi: 10.48550/arXiv.1504.04909.

Muhammad U. Nasir, Sam Earle, Christopher Cleghorn, Steven James, and Julian Togelius. LLMatic: Neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2024. doi: 10.1145/3638529.3654017. URL https://arxiv.org/abs/2306.01102.

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. doi: 10.48550/arXiv.2506.13131.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016. doi: 10.3389/frobt.2016.00040.

Robert B. Randall and Jérôme Antoni. Rolling element bearing diagnostics—a tutorial. *Mechanical Systems and Signal Processing*, 25(2):485–520, 2011. doi: 10.1016/j.ymssp.2010.07.017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019. doi: 10.1609/aaai.v33i01.33014780.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997. doi: 10.1162/evco.1997.5.1.1.

J. L. Shapiro. Does data-model co-evolution improve generalization performance of evolving learners? In *Parallel Problem Solving from Nature – PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pp. 540–549. Springer, 1998. doi: 10.1007/BFb0056896. URL https://link.springer.com/chapter/10.1007/BFb0056896.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent. GitHub repository, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Wade A. Smith and Robert B. Randall. Rolling element bearing diagnostics using the case western reserve university data: A benchmark study. *Mechanical Systems and Signal Processing*, 64–65:100–131, 2015. doi: 10.1016/j.ymssp.2015.04.021.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811.

Tanja Tornede, Alexander Tornede, Marcel Dominik Wever, and Eyke Hüllermeier. Coevolution of remaining useful lifetime estimation pipelines for automated predictive maintenance. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2021. doi: 10.1145/3449639.3459395.

Jie Wang, Haidong Shao, Shen Yan, and Bin Liu. C-ECAFormer: A new lightweight fault diagnosis framework towards heavy noise and small samples. *Engineering Applications of Artificial Intelligence*, 126:107031, 2023. doi: 10.1016/j.engappai.2023.107031.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. POET: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 142–151, 2019. doi: 10.1145/3321707.3321799.

Ruixin Wang, Hongkai Jiang, Xingqiu Li, and Shaowei Liu. A reinforcement neural architecture search method for rolling bearing fault diagnosis. *Measurement*, 154:107417, 2020a. doi: 10.1016/j.measurement.2019.107417.

Xiaoxing Wang, Xiangxiang Chu, Junchi Yan, and Xiaokang Yang. DAAS: Differentiable architecture and augmentation policy search. *arXiv preprint arXiv:2109.15273*, 2021. doi: 10.48550/arXiv.2109.15273.

Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. Improving adversarial robustness requires revisiting misclassified examples. In *International Conference on Learning Representations*, 2020b.

Richard A. Watson and Jordan B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 702–709, 2001.

Shuhan Xue, Jianyuan Zhong, Ziyuan Nan, Wenbin Li, Zhaochen Yu, Jinchao Ding, Qiang Gao, Pengyu Zhan, Yuntong Zhang, Tian Cheng, Zhenfei Yin, Yingcheng Wu, and Ling Yang. Sciencebuddy: Recursive-in-recursive self-improvement for interactive scientific agents, 2026. URL https://arxiv.org/abs/2609.17523.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. CoEvolve: Training LLM agents via agent-data mutual evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23015–23036. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1055. URL https://aclanthology.org/2026.acl-long.1055/.

Xiangyu Yin and Wenjie Ruan. Boosting adversarial training via fisher-rao norm-based regularization, 2024. URL https://arxiv.org/abs/2403.17520.

Wei Zhang, Gaoliang Peng, Chuanhao Li, Yuanhang Chen, and Zhujun Zhang. A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals. *Sensors*, 17(2):425, 2017. doi: 10.3390/s17020425.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025. doi: 10.48550/arXiv.2505.03335.

Minghang Zhao, Shisheng Zhong, Xuyun Fu, Baoping Tang, and Michael Pecht. Deep residual shrinkage networks for fault diagnosis. *IEEE Transactions on Industrial Informatics*, 16(7):4681–4690, 2020. doi: 10.1109/TII.2019.2943898.

A THE DEPLOYED POLICY

Algorithm 1 states the deployed policy; deployment settings are given in Appendix C.2. Stage names refer to Figures 2 and 3, and the gate condition is Eq. 2 below. EVOLVE-NOSTRUCT is the same procedure with step 4 taken unconditionally, and REPLAY-ANCHOR is EVOLVE-NOSTRUCT with step 2 replaced by a byte-exact copy of the donor's curriculum.

Gate rule. Confirmation evaluates each finalist on the frozen fixture D_{gate}: $G(a) = G(a, w_a^{104}; D_{\text{gate}})$, where w_a^{104} denotes its confirmation weights. Let G_M be mean accuracy over five noise families, G_{clean} clean accuracy, G_W mean accuracy over the two weakest families, and G_f accuracy on family f. Let $i = a_{t-1}$ be the retrained incumbent and y the non-incumbent finalist with highest G_M. Source selection requires a mean gain under three guardrails:

$$\begin{aligned} &G_M(y) > G_M(i) + \delta, && G_{\text{clean}}(y) \geq G_{\text{clean}}(i) - \epsilon_c, \\ &G_W(y) \geq G_W(i) - \epsilon_w, && \min_f \{G_f(y) - G_f(i)\} \geq -\epsilon_f. \end{aligned} \tag{2}$$

Threshold values and their calibration are given in Appendix C.

Algorithm 1: DUOEVO with the patience router
Input: anchor a_0, rounds T, patience K, search cap M, budgets, gate G
Initialize: bootstrap $S_0 = (a_0, w_0)$; $q_0 \leftarrow a_0$; $z_0 \leftarrow 0$; $n_A \leftarrow 0$; LOCKED←`false`

1. For $t = 1, \ldots, T$:
2. *Challenger:* freeze S_{t-1}; rescore the Challenger archive; evolve programs and freeze C_t.
3. *Weight move:* continue S_{t-1} on C_t for 32 epochs to obtain $\widetilde{S}_t$.
4. If LOCKED: set $u_t^A = \text{id}$, $S_t \leftarrow \widetilde{S}_t$, $z_t \leftarrow z_{t-1}$, continue.
5. *Structural move:* search 40 sources from tip q_{t-1} with 8-epoch screening; gate the retrained incumbent, top-screened source, and a different-cell source, if available, at 104 epochs.
6. *Deployment:* if Eq. 2 selects a source, reuse confirmation when $B_{\text{dep}} = 104$; otherwise rebuild at matched B_{dep} over $C_{0:t}$ and require the rebuilt gate mean to match or exceed that of $\widetilde{S}_t$; on deployment $q_t \leftarrow a_t$, $z_t \leftarrow 0$. Otherwise keep $\widetilde{S}_t$, update or roll back the search tip, $z_t \leftarrow z_{t-1} + 1$.
7. Set $n_A \leftarrow n_A + 1$. If $z_t \geq K$ or $n_A \geq M$: run the switch audit and set LOCKED←`true` for round $t + 1$.

Output: S_T; read report-only $F_{\text{HN}}(S_T)$ after the loop.

Search tip. The search tip and the deployed Solver have distinct roles. After a failed promotion, the next search may start from the confirmed source with the highest gate mean, even when that source did not pass all deployment guardrails. Two rounds without a promotion reset the tip to the deployed source. This permits local exploration while preserving the incumbent's accumulated weights; the Challenger continues to face the deployed Solver throughout. Each new round rebuilds the screening lineage and reevaluates sources on its newly frozen curriculum.

B PROOFS AND BOUNDARY CONDITIONS

The main text uses three policy properties to summarize the structural safeguards. We state them here before the proofs. They are operational properties of the implemented policy, not guarantees that the LLM search finds an optimal architecture or that gate improvements transfer to held-out corruptions. Let Π_{DuoEvo} allow all architecture actions including identity, $\Pi_{\text{fixA}} = \{\pi \in \Pi_{\text{DuoEvo}} : u_t^A = \text{id}\ \forall t\}$ fix the architecture, and $J_B(\pi) = \mathbb{E}_\pi[F_{\text{HN}}(S_T)]$ be expected endpoint HN accuracy under cap B.

Proposition 1 (Within-round non-degradation)**.** *For a fixed gate, let $V_t^- = G_M(\widetilde{S}_t)$ and $V_t^+ = G_M(S_t)$. If deployment retains $\widetilde{S}_t$ unless a source passes Eq. 2 and its deployed checkpoint scores at least V_t^-, then $V_t^+ \geq V_t^-$ in every round.*

The guarantee concerns the gate score, not HN accuracy, and it holds where the checkpoint condition is checked: rebuilds check it, while round-3 reuse and the switch audit do not (Appendix C.2).

Proposition 2 (Same-budget policy containment)**.** *Let Π^B denote policies within cap B. If Π_{DuoEvo} contains the state, history, and actions of Π_{fixA}, with a zero-cost identity action, then $\Pi_{\text{fixA}}^B \subseteq \Pi_{\text{DuoEvo}}^B$ and $\sup_{\pi \in \Pi_{\text{DuoEvo}}^B} J_B(\pi) \geq \sup_{\pi \in \Pi_{\text{fixA}}^B} J_B(\pi)$.*

Opening structural search thus cannot lower the best achievable endpoint, although a finite LLM search can find less in the larger space (Appendix B).

Proposition 3 (Finite structural cost)**.** *With patience $K \geq 1$, cap $M < \infty$, and horizon T, the router (Section 3.3) performs N_A structural searches and P_A promotions satisfying*

$$N_A \leq \min\{T, M, K(P_A + 1)\}. \tag{3}$$

It locks after K consecutive misses or cap exhaustion and takes identity thereafter. If each search costs at most c_A and the one-time switch audit costs c_{audit}, the total structural cost is at most $c_A \min\{T, M, K(P_A + 1)\} + c_{\text{audit}}$.

The bound does not depend on how many rounds follow the lock. In our runs the router locked after three to six searches, below the cap of eight (Table 9).

B.1 Proof of Proposition 1

Proof. Fix round t. If no candidate is deployed, the deployment rule sets $S_t = \widetilde{S}_t$, so $V_t^+ = V_t^-$. If a candidate is deployed, the rule requires the deployed checkpoint to score at least V_t^- on the gate, so $V_t^+ \geq V_t^-$. Thus $V_t^+ \geq V_t^-$ in either case. □

Cross-round extension. Adding the continuation premise $V_t^- \geq V_{t-1}^+$ gives $V_t^+ \geq V_t^- \geq V_{t-1}^+$, and induction makes $\{V_t^+\}$ non-decreasing. If each of k promotions also satisfies $V_t^+ \geq V_t^- + \delta_{\min}$, then $V_t^+ \geq V_0^+ + k\delta_{\min}$. Since $V_t^+ \leq 1$, this gives $k \leq \lfloor (1 - V_0^+)/\delta_{\min} \rfloor$.

Where the premise holds. The implementation checks the within-round inequality when auditing a rebuilt checkpoint. At $t = 3$, it reuses the 104-epoch confirmation checkpoint, so budget equality alone does not establish non-degradation against the continued incumbent. The source-based switch audit also does not reverify its final rebuilt checkpoint. The proposition therefore remains conditional on its checkpoint-score premises. Another 32 epochs on a new curriculum need not improve the fixed gate, and a confirmation score need not transfer to a rebuilt checkpoint; neither implication is assumed.

B.2 Proof of Proposition 2

Proof. Fix any $\pi \in \Pi_{\text{fixA}}$ that satisfies budget cap B. By assumption, Π_{DuoEvo} exposes the state variables and history read by π and contains every action that π may emit. Construct $\iota(\pi) \in \Pi_{\text{DuoEvo}}$ as follows: on every history shared with π, copy its curriculum, weight, and deployment actions and set the architecture action to id. Couple the environment, trainer, and random streams of the two executions. Because identity has zero charge in the inactive architecture-budget coordinate, $\iota(\pi)$ satisfies the same cap B. Induction on t now gives identical curricula, deployed states, and histories: equality holds at initialization, and copying all effective actions preserves it for the next round. Hence $J_B(\iota(\pi)) = J_B(\pi)$. This constructs a budget-preserving injection from Π_{fixA} into Π_{DuoEvo}, and taking suprema yields the result. □

The zero-cost premise is coordinate-specific. In the implementation, an identity architecture action consumes no candidate, confirmation, rebuild, or Solver-proposer budget, so containment is exact for deployment epochs and for the active-action ledger. It does not imply that an executed search policy has the same total design cost as EVOLVE-NOSTRUCT.

Finite-budget search dilution. Containment is an oracle policy-class property, not a guarantee for a particular searcher. Let the restricted class contain only an optimal action a^*, and let the full class contain a^* plus 999 ineffective actions. A one-proposal uniform search succeeds with probability one in the restricted class and 10^{-3} in the full class. The oracle inequality remains true, while the expected output of the finite searcher is lower in the larger space.

B.3 Proof of Proposition 3

Proof. The counter n_A increases by one only after a structural-search round and is never reset. The router locks when $n_A = M$, so no execution can contain more than M searches. Independently, the miss counter increases after each non-promotion and resets after a promotion. Once it reaches K,

the next-search state is set to the absorbing lock value. The routing function maps that value to id for every later round. Hence K consecutive misses trigger immediate permanent lock, and the cap remains valid even when promotions repeatedly reset the miss counter. Partition the N_A searches into P_A blocks ending in a promotion and one final block with no promotion. Each promotion block contains at most $K-1$ misses followed by one promotion, and the final block contains at most K misses. Hence $N_A \le K(P_A+1)$. There is at most one search per round, so combining this bound with $N_A \le T$ and $N_A \le M$ gives Eq. 3. Summing at most c_A over these searches and adding the audit, which runs at most once, gives the stated cost bound. □

B.4 Asymptotic coverage

Permanent lock ends structural search, so the result below does not apply to DuoEvo as run. It applies to variants of the same policy class that keep searching with non-vanishing probability, such as an unbounded backoff policy.

Proposition 4 (Persistent-coverage convergence)**.** *Let $\mathcal{X}_R$ be the finite set of sources reachable under a stationary gate G, let $\mathcal{X}^* \subseteq \mathcal{X}_R$ be its maximizers, and let τ^* be the first round in which a member of $\mathcal{X}^*$ is confirmed. Suppose that for every history $\mathcal{H}_{t-1}$ with $\tau^* > t-1$, the conditional probability of confirming a member of $\mathcal{X}^*$ in round t is at least $\epsilon_t \in [0,1]$, that $\sum_t \epsilon_t = \infty$, and that retention is elitist. Then* $\Pr(\tau^* > T) \le \exp\big(-\sum_{t=1}^{T} \epsilon_t\big) \to 0$.

Proof. Let $E_t = \{\tau^* > t\}$ be the event that no maximizer has been confirmed through round t. Before a hit, the conditional miss probability is at most $1-\epsilon_t$. Iterating conditional expectations gives

$$\Pr(E_T) \le \prod_{t=1}^{T}(1-\epsilon_t) \le \exp\left(-\sum_{t=1}^{T}\epsilon_t\right), \tag{4}$$

where the last step uses $1-x \le e^{-x}$. The divergent coverage sum sends this bound to zero. The optimum is therefore hit almost surely, and elitist retention keeps it after the first hit. □

Positive probability in each round is not enough. With $\epsilon_t = 2^{-t}$, the sum is finite and the infinite miss product is strictly positive. Under permanent lock, $\epsilon_t = 0$ after lock, which is why the result does not cover DuoEvo as run.

C Reproducibility Protocol

C.1 Data construction and executed splits

Table 3 reports values reconstructed from the saved run configuration and the resulting artifacts. Because CWRU and JNU are windowed with stride below the window length, consecutive windows share raw samples, and a split index alone does not separate the segments. The executed runs therefore purge $\lceil L/\text{stride} \rceil - 1$ windows after every split boundary (one window on CWRU, seven on JNU), so no evaluation window shares a sample with a training window. PU needs no purge: it partitions whole measurement repetitions.

Table 3: Executed data protocol. Counts are clean windows before curriculum augmentation.

Data	Classes	Rate	Window	Stride	Train	Dev	Test	Reserved cross-condition
PU	3	12 kHz	2048	1024	39,017	4,882	4,877	`N09_M07_F10`
CWRU	10	12 kHz	2048	1024	3,436	1,112	1,132	motor load 3
JNU	4	12 kHz[†]	2048	256	3,342	1,056	1,062	1000 rpm

[†]JNU is recorded at 50 kHz and resampled with polyphase anti-alias filtering to 12 kHz. PU uses repetitions 1–16/17–18/19–20 for train/dev/test. CWRU and JNU use 60/20/20 time-ordered partitions of each non-held-out record, with boundary windows purged as described above. HN accuracy is evaluated on the Test column, not on the reserved cross-condition data. The ten-class CWRU loader excludes the eight 0.028-inch fault files because the frozen taxonomy is Normal plus $\{\mathrm{IR}, \mathrm{B}, \mathrm{OR}\} \times \{0.007, 0.014, 0.021\}$ inch.

All signals are normalized per source record before windowing. The PU split hash is `9774ff5bcfe43b0a`; the CWRU and JNU loader identifiers are `cwru_2048_1024_load3_10c` and `jnu_2048_256_speed1000_4c`. Dataset roots are environment-specific and are not part of these identifiers. Record-wise normalization uses the full source record, including its held-out time segments, before splitting; it does not use labels but differs from training-only normalization.

C.2 Training, search, and gate configuration

Table 4: Frozen DuoEvo configuration.

Quantity	Value
Seeds	11, 23, 37
Rounds T	20
Bootstrap / continuation	8 / 32 epochs
Candidate screening	40 proposals $\times$ 8 epochs
Full-fidelity shortlist	3 sources $\times$ 104 epochs
Challenger proposals per round	16
Augmented fraction	0.35
Random/archive target split	0.5 / 0.5
Archive materialization cap	8 elites
Training SNRs	$0, -3, -6$ dB
Gate SNRs	$-1.5, -4.5, -7.5$ dB
Challenger attack SNR	-4 dB
Challenger reward probes (PU / CWRU / JNU)	600 / 330 / 340
Screen / gate probe size	128 / 256
Router patience / search cap	3 misses / 8 searches
Search-tip rollback patience	2 rounds
Parameter / epoch-time cap	8M / 90 s

All runs use SGD with learning rate 0.01, momentum 0.9, batch size 64, no gradient clipping, and cosine annealing to 10^{-8}. The loss is class-weighted cross-entropy for every architecture; ClassBD, and evolved sources that expose them, add the published ClassBD uncertainty-weighted auxiliary terms. Each adaptive training call starts a fresh cosine over its own duration: 8 epochs at bootstrap, 32 for a continuation, or 104 for confirmation. Static baselines use one 648-epoch cosine on C_0. On all three datasets, promotion and anchor-restoration rebuilds use the accumulated curriculum $C_{0:t}$, restarting the cosine for each 8- or 32-epoch leg; round-3 promotions reuse their 104-epoch confirmation checkpoint. Each static baseline is trained from scratch with model-independent epoch permutations; the adaptive trainer uses seeded minibatch shuffles. Checkpoint selection uses the noisy validation rows materialized with each arm's curriculum. cuDNN benchmark mode is disabled and deterministic mode is enabled. Seed-derived streams control initialization, curriculum generation, gate noise, and HN noise. LLM sampling is not assumed to be deterministically reproducible; every accepted executable source and its hash are persisted. Post-search SHA-256 checks confirm that every candidate in a round trained on the same curriculum arrays.

Promotion uses Eq. 2 with $(\delta, \epsilon_c, \epsilon_w, \epsilon_f) = (0.0283, 0.01, 0.0845, 0.03)$. Two independent five-run studies trained the anchor for 104 epochs on two realized curricula. Pooling their within-fixture, across-training variances with eight degrees of freedom gave standard deviations $\sigma_M = 0.0142$ for the gate mean and $\sigma_W = 0.0282$ for the two-worst mean; $\delta = 2\sigma_M$ and $\epsilon_w = 3\sigma_W$, while ϵ_c and ϵ_f are prior guardrails. These are single-training standard deviations, not paired-difference errors or significance thresholds. All four were frozen before the main trajectories. Candidate ties are resolved by higher gate mean, higher two-worst-family mean, fewer trainable parameters, then lexicographically smaller source SHA-256. Screening ties never authorize deployment. The switch audit uses the same mean, clean, worst, and family guardrails with a zero anchor margin.

Executed deployment settings. All source-selection guardrails are applied against the 104-epoch incumbent source. Outside round 3, selected sources are rebuilt over $C_{0:t}$ and the rebuilt gate mean is checked against the continued incumbent; round 3 reuses the confirmation weights. PU, CWRU, and JNU use the same deployment settings.

Executable interfaces and search space. Noise sources return a finite array of shape $(n,)$ from `generate_noise(rng, fs, n)` using NumPy and `noise_lib`. The library includes Gaussian, colored, ARMA, filtered, sinusoidal, line-frequency, impulsive, resonant, harmonic, cyclostationary, and burst primitives; NumPy permits additional compositions. A source is checked for seed reproducibility and an 8-second evaluation-time limit. Signal-dependent multiplicative noise is outside this additive interface. Each Challenger round runs OpenEvolve 0.2.26 (Sharma, 2025) for 16 full-rewrite proposals on one island with population 16, always starting from the same seed program, a mixture of colored background noise, sparse amplitude bursts, and a resonant impulse train. A prompt carries the parent's source and metrics against the frozen Solver, the round's three top-scoring programs, two further diverse programs, and inspiration programs sampled from the same population; the persistent archive reaches training through curriculum materialization rather than through prompts. The evolved programs move well beyond the seed. Across the nine final archives (35–44 elites per run, 362 in total), the median elite combines four primitives, 12 of the 14 primitives occur, the most frequent one (cyclostationary modulation, in 74% of elites) is absent from the seed, and only 4 elites are near-verbatim copies of it. Solver sources expose `build_solver(n_classes, length)`, returning a PyTorch module mapping $(B, 1, L)$ signals to (B, C) class probabilities. The prompt permits changes to quadratic time filters, full-capacity spectral filters, and convolutional classifiers using PyTorch and `solver_lib`, under the parameter and time caps in Table 4. Screening uses noisy-validation balanced accuracy; archive cells are $(\mathrm{round}(\log_{10}\text{parameters}, 1), \text{convolution count})$. The configured proposer is `glm-5.2`, temperature 0.7, with maximum output lengths of 8,192 tokens (Challenger) and 16,384 tokens (Solver). The saved experiment configurations point to shared LLM configuration files rather than recording an immutable provider model revision.

Example evolved source. The PU seed-11 endpoint retains a full $L \times L$ learned spectral filter and combines quadratic time-domain filters of widths 63, 31, and 15 with learned scale weights, channel attention, spectral-band attention, and residual convolutional blocks. It has 5,131,695 parameters. Endpoint counts for PU seeds 23/37 are 5,618,523/4,581,038; CWRU seeds 11/23/37 have 4,269,275/4,269,275/4,388,574; JNU has 4,268,669/4,343,302/4,268,669. CWRU seeds 11/23 and JNU seeds 11/37 end at the anchor, including two switch-audit rollbacks; a final anchor does not imply an all-anchor trajectory.

C.3 What the Random-Noise arm controls

Random-Noise removes the Challenger branch and replaces it with uniform draws from the noise-primitive library: eight distinct programs per round, no archive, no scoring against the current Solver, at the same total augmented-row count. Eight is the evolved arm's own archive-materialization cap, so the two arms match the elite cap and perturbed-data volume, not necessarily total distinct program count; a fresh program per augmented example would instead multiply program coverage and confound it with Solver-conditioned selection.

The evolved curriculum targets a half-random, half-archive mixture, with at most eight reused archive elites and random backfill when the archive is underfilled. Its composition and reuse pattern therefore still differ from a uniform eight-program pool. The comparison consequently identifies the contribution of the Challenger branch as a whole (LLM authorship, archive retention, and Solver-conditioned selection), rather than any one component. Byte-exact replay addresses a different question by fixing all materialized training arrays while varying the architecture path.

On PU, Random-Noise trails static ClassBD in all three seeds. The deficit is concentrated in swept chirp: its paired family–SNR difference is -7.49 pp at -3 dB and -18.88 pp at -6 dB, averaged over seeds; differences in the other ten cells are at most 2.75 pp in magnitude. This localizes the endpoint gap but does not establish whether program-pool coverage caused it.

C.4 Evaluation fixtures

The internal gate averages five diagnostic families. Its two-worst statistic averages the two lowest family means rather than taking the raw minimum. The HN battery uses four generator implementations withheld from training and model selection: linear swept chirp; correlated multiplicative speckle; Student-t bursts with drifting inter-arrival time; and their mixture. Each is evaluated at 0,

-3, and -6 dB on the test split. The 12 balanced accuracies are macro-averaged. HN evaluation is a separate command, and an assertion rejects any HN field in an online round log.

C.5 ADVERSARIAL-TRAINING BASELINE

The ClassBD w/ MART+LORE row of Table 1 reimplements MART (Wang et al., 2020b) with full MSE LORE (Yin & Ruan, 2024) from the loss equations in the official LOAT release (`github.com/TrustAI/LOAT`, commit `6ff9873`, `mart/mart.py`). MART is the adversarial cross-entropy plus a complement-label term plus β times a confidence-weighted KL between the clean and adversarial predictive distributions; LORE adds a signed dispersion term over correctly and incorrectly classified base inputs and an adaptive clean/adversarial pairing term, active in an early and a late phase with MART alone in between.

Four adaptations carry the objective from images to 1-D signals. *(i)* The threat model is a per-example L_2 ball of radius $\|x\|_2\, 10^{-q/20}$ rather than an L_∞ box, since a vibration window has no $[0, 1]$ range to clip to; PGD takes ten normalized-gradient ascent steps of length $2\,\mathrm{radius}/\mathrm{steps}$ from a random direction at a uniformly drawn radius, and a zero signal receives zero perturbation. *(ii)* The reference phase breakpoints $E_1 = 1$ and $E_2 = 100$ of a 110-epoch schedule are scaled to the 648-epoch budget as $E_1 = \max(1, \mathrm{round}(E/110))$ and $E_2 = \lceil 100E/110 \rceil$, placing LORE in epochs 1–6 and 590–648 at the same proportions. *(iii)* Following this paper's shared recipe, train-derived inverse-frequency class weights enter the cross-entropy only; the complement-label term, the KL, and both LORE terms keep their unweighted definitions, and the ClassBD auxiliary terms stay off. *(iv)* The clean distribution enters the KL through `log_softmax` instead of as an `F.kl_div` target. The two express the same quantity and agree to float32 rounding on real batches, but ClassBD's quadratic front end spreads clean logits far enough for float32 `softmax` to underflow to exact zeros, after which `kl_div` differentiates $\log(\mathrm{target})$ at zero and the softmax backward turns $0 \cdot (-\infty)$ into `NaN` in every parameter gradient while the forward loss still reads finite. Image ResNets do not reach that regime.

The baseline uses the backbone rows' optimizer recipe: SGD at 0.01 with momentum 0.9, cosine annealing to 10^{-8} over 648 epochs, batch size 64, no gradient clipping, model-independent epoch permutations, and the same noisy-validation checkpoint criterion. Validation selects the objective's q and β per dataset, with $(\theta, \gamma) = (0.1, 0.05)$ left at the released full-LORE values throughout.

Selecting q and β. The released launch configuration, $q = 20$ dB with $\beta = 6$, is calibrated for an image ResNet, and Table 6 shows that one nominal radius does not transfer as one threat: the same q costs a clean anchor 61.1, 97.3, and 78.9 percent of its balanced accuracy on CWRU, JNU, and PU. Both are therefore chosen per dataset on validation, under a protocol registered before any run of the sweep and archived with the code.

Admissibility. Ranking configurations by validation accuracy alone would select the weakest attack, because a smaller ball makes the objective approach plain cross-entropy: the procedure would tune the defense away rather than improve it. A configuration is therefore admissible only if its attack costs a clean anchor at least as much, relatively, as $q = 20$ dB costs one on CWRU, that is, a relative drop of at least 0.6114 read off Table 6. On PU the two weakest rungs score higher on validation than the selected one and are excluded.

Ranking. Among admissible configurations, the one with the highest balanced accuracy on the seed-11 noisy-validation split is selected; ties within 0.010 are broken by the mean over the last ten epochs, and further ties by proximity to the released values. The swept grid is $q \in \{20, 26, 32, 38\}$ dB at $\beta = 6$, then $\beta \in \{1, 3, 6\}$ at the chosen q. Seeds 23 and 37 never enter any selection decision, and no HN command is issued until the configuration is frozen; the unsealing timeline is archived alongside the runs.

Table 5 lists the selected configurations. CWRU keeps the published q: every weaker rung falls under the floor, so only β moves. JNU keeps the published β. The three datasets do not agree on either value, which is the reason for selecting per dataset rather than transferring one setting. PU is screened at 108 epochs because its 52,673-row curriculum makes a full run an order of magnitude more expensive; the screen is validated by reproducing the released configuration's known failure there before its ranking is used. All nine confirmation runs complete 648 epochs and reproduce their recorded best validation score on checkpoint reload before any HN command is issued.

Table 5: Selected configuration and its validation evidence. Relative drop is the admissibility statistic of Table 6; the floor is 0.6114.

Dataset	q (dB)	β	Relative drop	Validation BA	Last-10 mean
PU	26	1	0.662	0.9519	0.9470
CWRU	20	1	0.611	0.9828	0.9768
JNU	26	6	0.627	0.9804	0.9784

Table 6: **The same nominal attack budget is three different threats.** Balanced accuracy (%) of a clean ClassBD anchor on its noisy validation rows under a ten-step L_2 PGD attack, with the relative drop from the unattacked value in parentheses. Seed 11; the anchor is trained for 104 epochs with the shared recipe on the same static C_0. The allowed perturbation is $\|x\|_2\, 10^{-q/20}$, that is, 19.95 to 1.26 percent of the input norm from left to right.

Dataset	Clean	$q = 14$ dB	20 dB	26 dB	32 dB	38 dB
CWRU	97.24	6.45 (93.4)	37.79 (61.1)	75.80 (22.1)	90.98 (6.4)	94.47 (2.8)
JNU	94.58	0.59 (99.4)	2.54 (97.3)	35.25 (62.7)	73.27 (22.5)	87.45 (7.5)
PU	94.64	16.35 (82.7)	19.97 (78.9)	31.99 (66.2)	58.14 (38.6)	80.36 (15.1)

Two further readings of Table 6 matter. Attack severity alone does not determine what the objective delivers: JNU is the dataset the attack hurts most and is not the dataset with the lowest HN mean in Table 1. And the realized perturbation is a second per-dataset quantity: PGD reaches 99–100 percent of the allowed radius on CWRU and JNU at every rung, and 87–91 percent on PU, so the nominal q is a lower bound on the applied signal-to-noise ratio rather than its value. Balanced accuracy is monotone in q on all three datasets.

D Additional Results

Table 7 retains the per-SNR ablation results summarized in Table 2.

Table 7: **Full SNR breakdown of the ablations in Table 2.** HN balanced accuracy (%), mean $\pm$ sample SD over three seeds at 648 deployment epochs; columns as in Table 1. Bold marks the highest mean.

Data	Configuration	0 dB	-3 dB	-6 dB	Mean
PU	Static ClassBD	98.84±0.06	95.60±0.19	88.68±0.93	94.37±0.25
	Random-Noise	99.00±0.14	94.01±0.80	82.98±1.27	92.00±0.63
	Evolve-NoStruct	99.34±0.18	96.87±0.78	90.59±1.52	95.60±0.82
	DuoEvo	**99.36±0.20**	**97.48±0.41**	**93.16±0.78**	**96.67±0.46**
CWRU	Static ClassBD	**99.80±0.10**	97.57±0.19	83.60±2.86	93.66±0.93
	Random-Noise	**99.80±0.16**	98.26±0.69	86.81±2.91	94.96±1.20
	Evolve-NoStruct	99.62±0.29	98.45±0.56	93.45±1.85	97.17±0.78
	DuoEvo	99.74±0.08	**98.95±0.38**	**94.93±0.78**	**97.87±0.40**
JNU	Static ClassBD	98.39±0.52	94.49±0.88	85.86±1.18	92.91±0.66
	Random-Noise	98.58±0.51	96.09±0.48	87.05±0.51	93.91±0.20
	Evolve-NoStruct	98.87±0.31	96.14±0.13	87.40±3.20	94.14±1.11
	DuoEvo	**99.31±0.11**	**96.98±0.98**	**88.43±5.22**	**94.91±2.10**

Table 8 reports both evaluation readouts for the published backbones. Figure 7 and Table 9 record routing decisions, Table 10 gives family-level online effects, and Table 11 lists per-seed endpoints.

These online effects combine architecture and curriculum changes; Figure 8 shows their family-by-SNR breakdown. The largest gains occur in swept chirp on PU (+5.34) and heavy burst on JNU (+4.90). The replay-conditioned family comparison below evaluates the structural path under fixed donor curricula.

The primary endpoint is round 20. Per-round logs retain gate measurements, promotion decisions, source identities, and router state; HN exports retain all 12 family–SNR cells.

Table 8: **Published backbones at 648 epochs, both readouts.** Balanced accuracy (%), mean over three seeds on static C_0, using the same runs as Table 1. Gate is the in-language fixture; HN is the report-only held-out-noise battery. Bold marks the highest mean per column.

	PU		CWRU		JNU	
Backbone	Gate	HN	Gate	HN	Gate	HN
ClassBD-BDWDCNN	90.11	**94.37**	**87.71**	**93.66**	**94.93**	**92.91**
GTFENet	**93.36**	93.61	76.58	91.56	93.17	89.64
WDCNN	74.61	88.13	74.07	90.16	89.96	87.26
QCNN	71.60	76.57	66.60	77.34	60.34	50.55
DRSN-CW	79.59	71.09	74.34	76.28	79.21	72.07
TFN-STTF	63.12	58.76	36.66	30.16	59.95	63.35

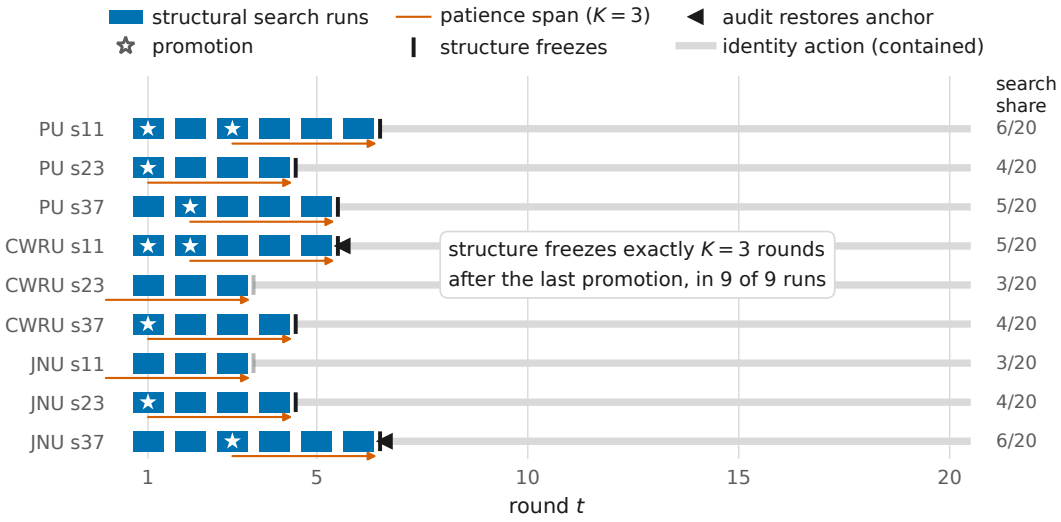


Figure 7: **Per-run routing behavior.** Each row is one DUOEVO trajectory. In each run, the structural branch runs three to six searches and closes after K consecutive misses; the nine runs yield nine promotions in total. After the freeze mark, every remaining round takes the identity action. The lock-time switch audit retains the evolved source five times, restores the anchor twice, and is not required twice (gray freeze mark, where the champion was already the anchor).

Table 9: Per-seed routing and switch-audit record. "First contained" is the first post-lock identity-action round.

Data	Seed	Horizon	Searches	Promotions	First contained	Switch audit	Frozen source
PU	11	20	6	2	7	keep evolved	evolved
PU	23	20	4	1	5	keep evolved	evolved
PU	37	20	5	1	6	keep evolved	evolved
CWRU	11	20	5	2	6	restore anchor	anchor
CWRU	23	20	3	0	4	not required	anchor
CWRU	37	20	4	1	5	keep evolved	evolved
JNU	11	20	3	0	4	not required	anchor
JNU	23	20	4	1	5	keep evolved	evolved
JNU	37	20	6	1	7	restore anchor	anchor

Table 10: Family-level online policy effects. Cells are mean paired HN changes (percentage points), DUOEVO minus EVOLVE-NOSTRUCT, across three SNRs and three seeds.

HN family	PU	CWRU	JNU
Swept chirp	+5.34	+0.40	−1.14
Mult. speckle	−1.37	+1.59	−0.78
Heavy burst	+0.24	+0.25	+4.90
OOD mixture	+0.07	+0.54	+0.10

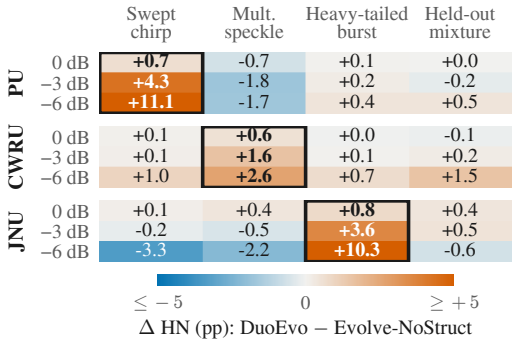


Figure 8: **Online policy differences.** Three-seed mean HN change (pp), DUOEVO minus EVOLVE-NOSTRUCT, at round 20. Boxes mark the weakest family under EVOLVE-NOSTRUCT; color saturates at ±5 pp.

Table 11: Per-seed endpoint balanced accuracy (%). NoStruct and Replay denote EVOLVE-NOSTRUCT and REPLAY-ANCHOR.

Data	Seed	DUOEVO gate	NoStruct gate	Replay gate	DUOEVO HN	NoStruct HN	Replay HN	Worst HN
PU	11	96.53	93.23	94.19	96.58	94.74	95.44	85.01
PU	23	95.24	94.15	92.28	97.17	96.37	96.42	87.88
PU	37	95.50	94.33	91.88	96.26	95.68	95.55	85.07
CWRU	11	91.57	92.30	92.60	98.21	96.59	97.09	86.86
CWRU	23	91.51	90.60	91.51	97.42	98.06	97.42	81.93
CWRU	37	91.87	90.75	89.93	97.98	96.87	96.94	87.78
JNU	11	97.79	97.50	97.79	92.48	93.98	92.48	76.26
JNU	23	97.69	98.06	97.75	96.16	95.32	94.46	80.31
JNU	37	97.92	98.23	97.67	96.08	93.11	93.29	88.93

D.1 CROSS-GENERATION EVALUATION

For each DUOEVO run we load the 21 deployed Solvers $S_0, \ldots, S_{20}$ from their round checkpoints and the eight elites that each Challenger round materialized into its curriculum. Every elite is regenerated with the Challenger evaluator's own noise stream, injected at -4 dB into the fixed class-balanced development probes, and scored with the margin of Eq. 1, so every Solver sees identical noisy inputs; M_{ij} is the mean margin of S_i over the round-j elites. Recomputing $M_{j-1,j}$ reproduces the reward logged during the run to within 2×10^{-4} in all 180 run–round pairs. Against its own opponent, the Challenger's elites score $M_{j-1,j} = 0.65$ at $j = 1$ and 0.91 at $j = 20$ (mean over runs), illustrating why the moving-opponent diagonal is not itself a progress measure. Row means of M therefore measure each Solver on one fixed challenge set, and column means measure each Challenger round on one fixed Solver panel (Figure 9, Table 12). As an external panel, the same elites are also scored by the 21 deployed Solvers of the RANDOM-NOISE arm with the same dataset and seed; these Solvers train on random primitive programs and never serve as this Challenger's opponent. The evaluation is inference only.

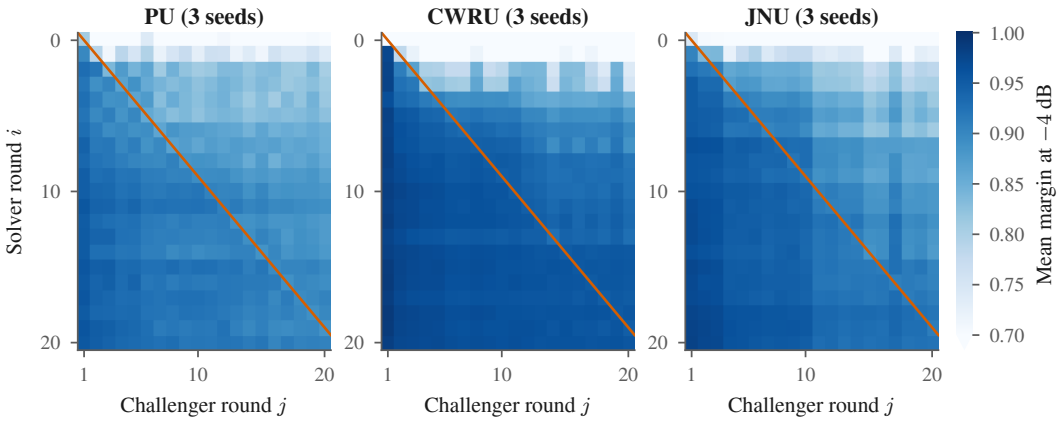


Figure 9: **Cross-generation margin tables.** Mean margin at -4 dB of Solver S_i (rows) on the round-j elites (columns), averaged over three seeds. The orange line marks $i = j - 1$, the opponent each round's elites were selected against; cells below it belong to Solvers that have trained on those elites. Values below 0.7, mostly from S_0, share the lightest color.

Table 12: Cross-generation summaries, three-seed means. Fixed challenges: mean of M_{ij} over all 20 Challenger rounds, at $i = 0$ and $i = 20$. Fixed panel: mean over all 21 DUOEVO Solvers, averaged over rounds 1–5 and 16–20; lower is harder. Trained/untrained: mean M_{ij} for $i \geq j$ and $i < j$. Forgetting: largest gap, over rounds and seeds, between S_{20} and the best Solver on a round's elites. External: the fixed-panel mean over the 21 RANDOM-NOISE Solvers, with the number of seeds in which rounds 16–20 are harder.

Data	Fixed challenges	Fixed panel	Trained/untrained	Forgetting	External panel
PU	$0.69 \to 0.92$	$0.906 \to 0.857$	0.919 / 0.835	0.040	$0.682 \to 0.655$ (2/3)
CWRU	$0.29 \to 0.96$	$0.916 \to 0.873$	0.961 / 0.821	0.011	$0.618 \to 0.538$ (2/3)
JNU	$0.61 \to 0.94$	$0.930 \to 0.855$	0.943 / 0.836	0.023	$0.880 \to 0.772$ (3/3)

E REPLAY VERIFICATION

The curriculum-controlled, donor-conditioned effect (CDE) is

$$\mathrm{CDE}_s = F_{\mathrm{HN}}\big(S_{T,s}^{\text{DUOEVO}} \mid C_{0:T}^{\text{DUOEVO}}\big) - F_{\mathrm{HN}}\big(S_{T,s}^{\text{REPLAY-ANCHOR}} \mid C_{0:T}^{\text{DUOEVO}}\big). \tag{5}$$

It measures the final HN difference along the structural path under the donor's realized curriculum.

At round t, REPLAY-ANCHOR performs four steps before training:

1. locate the matched donor's materialized `data.npz` and metadata;
2. copy both files into the replay round without invoking a Challenger;
3. recompute SHA-256 and compare it with the digest recorded by the donor at materialization;
4. load the exact train, label, development, selection, and OOD-validation arrays from the copy.

A missing file or digest mismatch aborts the run. Across the completed matrix, all 180 donor–copy pairs match, 60 per dataset. Each replay row records `replay=true`, donor path, digest, row counts, realized random/archive share, and the donor's materialization accounting. Replay uses zero Challenger and Solver LLM calls.

The zero-promotion invariant follows by induction. The bootstrap states agree. If the states agree through round $t-1$, both arms load the same C_t, use the same continuation source, weights, seed, and epoch count, and choose identity as the architecture action. Their round-t states therefore agree. In the two completed no-promotion trajectories, this construction yields an endpoint CDE of exactly 0.00 percentage points.

E.1 Anchor retraining at matched structural events

To separate a weight reset from an architecture change, Anchor-Retrain keeps the ClassBD anchor source and the donor's saved $C_{0:20}$, but retrains the anchor from scratch at every round in which the matched DuoEvo donor promotes a source or performs a switch audit. Each reset reaches the donor's $8 + 32t$ epoch budget before later rounds resume the ordinary 32-epoch continuation. Each reset replays the donor's own rebuild schedule and cosine restarts. The same event seeds are used, including the 104-epoch confirmation seed when budgets coincide. Each control uses 824–992 anchor-training epochs, with no LLM call or structure search. Every saved curriculum digest was checked; the initial anchor weights match Replay-Anchor exactly, and the two restored-anchor donor checkpoints match Anchor-Retrain weight for weight.

At the round-20 HN endpoint, the retraining component is Anchor-Retrain minus Replay-Anchor, and the remaining architecture-path component is DuoEvo minus Anchor-Retrain. These add exactly to CDE for each seed (Table 13). The latter component includes interactions with subsequent training under the fixed donor curriculum; it is not an architecture-only main effect. No new training is needed for the two zero-promotion runs, whose components are exactly zero.

Table 13: Round-20 HN decomposition in percentage points. All seven promoted runs are reported; zero-promotion rows follow from the exact replay invariant. Unrounded components sum exactly to CDE; displayed values can differ by 0.01 pp due to rounding.

Data	Seed	Retraining	Architecture path	CDE
PU	11	-0.85	$+1.99$	$+1.14$
	23	-2.02	$+2.77$	$+0.75$
	37	$+0.51$	$+0.20$	$+0.71$
CWRU	11	$+1.12$	0.00	$+1.12$
	23	0.00	0.00	0.00
	37	$+0.74$	$+0.30$	$+1.03$
JNU	11	0.00	0.00	0.00
	23	-2.65	$+4.35$	$+1.70$
	37	$+2.79$	0.00	$+2.79$

Across three seeds, the mean retraining/architecture-path components are $-0.79/+1.65$ pp on PU, $+0.62/+0.10$ pp on CWRU, and $+0.05/+1.45$ pp on JNU. Their paired 95% t intervals are respectively $[-3.93, 2.36]/[-1.61, 4.92]$ pp, $[-0.80, 2.04]/[-0.33, 0.53]$ pp, and $[-6.72, 6.81]/[-4.79, 7.69]$ pp. The architecture-path component is positive in all five runs that keep an evolved architecture.

E.2 PAIRED UNCERTAINTY AND FAMILY EFFECTS

Intervals use $\bar{d} \pm t_{0.975,2} s_d/\sqrt{3}$ on independent seed-level paired differences, not on the 12 correlated family–SNR cells. Aggregate effects (mean $\pm$ SD) are 0.87 ± 0.24, 0.72 ± 0.62, and 1.50 ± 1.41 pp for PU, CWRU, and JNU; their 95% intervals are $[0.27, 1.46]$, $[-0.83, 2.27]$, and $[-2.00, 4.99]$ pp. These small-sample descriptive intervals do not correct for selecting promoted structures.

Table 14 applies the same seed-level paired interval to the main endpoint comparisons. The MART+LORE rows use the validation-selected attack setting for each dataset.

Table 14: Paired endpoint differences, DUOEVO minus comparator, in HN percentage points. Mean and 95% t interval over seeds 11, 23, and 37; each pair uses the same dataset and seed.

Data	Comparator	Mean	95% interval
PU	Static ClassBD	$+2.30$	$[+1.18, +3.41]$
	MART+LORE	$+3.61$	$[-0.42, +7.64]$
	Evolve-NoStruct	$+1.07$	$[-0.61, +2.75]$
CWRU	Static ClassBD	$+4.21$	$[+0.92, +7.50]$
	MART+LORE	$+1.35$	$[-2.11, +4.82]$
	Evolve-NoStruct	$+0.70$	$[-2.23, +3.62]$
JNU	Static ClassBD	$+1.99$	$[-2.44, +6.43]$
	MART+LORE	$+1.17$	$[-2.73, +5.08]$
	Evolve-NoStruct	$+0.77$	$[-4.79, +6.33]$

Table 15: Replay-controlled family effects: DUOEVO minus REPLAY-ANCHOR (pp), averaging the three SNRs within each seed, then reporting mean $\pm$ sample SD. Saved family cells are rounded to four decimals before aggregation.

Held-out generator	PU	CWRU	JNU
Swept chirp	3.40 ± 1.33	0.33 ± 0.39	0.38 ± 0.88
Mult. speckle	-0.81 ± 0.70	1.62 ± 1.47	0.76 ± 1.24
Heavy burst	0.21 ± 0.12	0.45 ± 0.63	3.83 ± 3.90
OOD mixture	0.66 ± 0.18	0.48 ± 0.65	1.01 ± 0.88

Table 15 reports the replay-controlled family effects. Structural-path effects vary across generators. The largest replay-conditioned family gain matches the largest online gain in each dataset; the PU speckle effect is negative.

F COMPUTE ACCOUNTING

One OEE is one epoch of the official ClassBD training recipe on the current curriculum. It is an accounting unit, not a hardware-normalized FLOP. For a search round with n_p screened proposals, n_c confirmed sources, and B_r rebuild/audit epochs, the structural charge is

$$B_{\text{struct}} = 8n_p + 104n_c + B_r. \tag{6}$$

The unavoidable delivered-trajectory charge is 8 bootstrap epochs plus 32 epochs per round. Table 16 lists these charges per run. LLM calls and wall time remain separate because API latency and hardware throughput are not commensurate with training epochs.

Wall time is the sum of recorded round durations and is machine- and load-dependent. Structural OEE includes candidate screening, shortlist confirmation, promotion rebuilds when needed, and the switch audit. EVOLVE-NOSTRUCT and REPLAY-ANCHOR have zero structural OEE. Their deployment OEE is 648 on all three datasets; EVOLVE-NOSTRUCT still incurs 16 Challenger calls per round, whereas replay incurs none. Total training charges range from 2,544 to 4,744 OEE, excluding API latency; equal deployment epochs therefore do not imply equal total compute.

Table 16: Per-run design and deployment cost. OEE is an epoch-count accounting unit; wall time is recorded elapsed round time.

Data	Seed	Searches	Proposals	Deploy OEE	Structural OEE	Wall h	LLM calls	Lock round
PU	11	6	240	648	3,936	34.23	560	7
PU	23	4	160	648	2,672	24.83	480	5
PU	37	5	200	648	3,336	36.34	520	6
CWRU	11	5	200	648	3,544	6.11	520	6
CWRU	23	3	120	648	1,896	3.38	440	4
CWRU	37	4	160	648	2,672	3.87	480	5
JNU	11	3	120	648	1,896	3.54	440	4
JNU	23	4	160	648	2,672	9.85	480	5
JNU	37	6	240	648	4,096	7.08	560	7

G Research Artifacts

The local experiment artifacts contain saved configurations, accepted source files, materialized curricula, round logs, and report-only HN exports. The supplementary evidence exporter `tools/export_review_evidence.py` reconstructs the control and replay summaries, checks complete endpoint batteries and budgets, and optionally verifies curriculum bytes with `--verify-bytes`. Its machine-readable output is `evidence/review_evidence.json`, containing results for all completed controls.

No human participants, personal data, or animal subjects are involved. The experiments use public machine-condition datasets.