Supplementary Information for "Scientific discovery as meta-optimization: a combinatorial optimization case study"

Yuan-Hang Zhang,[1,*] Chesson Sipling,[1,†] and Massimiliano Di Ventra[1,‡]
[1]*Department of Physics, University of California San Diego, La Jolla, CA 92093*

I. BEST SOLVER ANALYSIS

Design 340, the best solver found, reduces the scaling exponent from $N^{2.51}$ (baseline) to $N^{1.33}$ and delivers a $\sim 67\times$ speedup at number of variables $N = 1810$ for a clause-to-variable ratio $\alpha_r = 4.3$. Below we trace the solver's genealogy, compare its modifications to the baseline DMM equations (Eqs. (4)-(6) of the main text), and examine why those modifications improve scaling.

A. Architectural modifications

Design 340 leaves the baseline variable dynamics (Eq. (4)) and long-term memory dynamics (Eq. (6)) intact. All innovation is concentrated in the short-term memory equation (Eq. (5)), which acquires an additive release term:

$$\dot{x}_{s,m} = \beta\big[\underbrace{(x_{s,m}+\epsilon)(c_m-\gamma)}_{\text{baseline}} + \underbrace{\mathcal{R}_m}_{\text{release}}\big], \tag{S1}$$

where the release term $\mathcal{R}_m$ is a product of five gating functions, each targeting a specific condition that must hold before the system intervenes:

$$\mathcal{R}_m = \kappa \cdot \text{gate}_{x_l} \cdot \text{weak\_band} \cdot \text{push\_up} \cdot \text{gate}_{\text{tail}} \cdot \text{amp\_norm}. \tag{S2}$$

The multiplicative structure means $\mathcal{R}_m$ is nonzero only when all five gates open at once. Each component is described below and visualized in Fig. 1.

1. Long-term memory gate (Fig. 1(a)). A sigmoid in log-space that switches on only for clauses carrying a large long-term memory:

$$\text{gate}_{x_l} = \sigma\left(\frac{\ln x_{l,m} - \ln x_{l,\text{thr}}}{\omega_l}\right), \tag{S3}$$

with $x_{l,\text{thr}} = 1000.4$ and $\omega_l = 2.03$. The release mechanism therefore acts only on persistently violated clauses.

2. Weak-satisfaction band (Fig. 1(b)). Two opposing sigmoids select clauses in a narrow satisfaction window:

$$\text{weak\_band} = \sigma\left(\frac{c_m - \eta}{\omega_b}\right) \cdot \sigma\left(\frac{\gamma - c_m}{\omega_b}\right), \tag{S4}$$

where $\eta = 0.336$, $\gamma = 0.282$, and $\omega_b = 0.063$. Notably, the optimized values satisfy $\eta > \gamma$, yielding a very narrow, low-amplitude response (peak ≈ 0.16). The band targets clauses hovering near the satisfaction boundary, which are partially satisfied but at risk of flipping. The inverted ordering $\eta > \gamma$ emerged from hyperparameter optimization and makes the release mechanism highly selective, delivering only a small nudge to clauses in this narrow region.

3. Bounded upward push (Fig. 1(c)). A ReLU gate that drives $x_{s,m}$ toward a target value:

$$\text{push\_up} = \frac{\max(x_s^* - x_{s,m},\ 0)}{x_s^*}, \tag{S5}$$

with $x_s^* = 0.091$. This is perhaps the most surprising parameter choice: the target sits near the lower bound of x_s, so the push shuts off almost as soon as x_s rises above ~ 0.09. Together with the narrow weak-band, it makes the release mechanism extremely conservative.

4. Tail-safety gate (Fig. 1(d)). A sigmoid that damps release when $x_{s,m}$ approaches its upper bound:

$$\text{gate}_{\text{tail}} = \sigma\left(\frac{x_{s,\text{tail}} - x_{s,m} + \mu_{\text{tail}} \cdot \text{gate}_{x_l}}{\omega_{\text{tail}}}\right), \tag{S6}$$

with $x_{s,\text{tail}} = 0.531$, $\mu_{\text{tail}} = 0.424$, and $\omega_{\text{tail}} = 0.107$. The x_l-dependent shift ($\mu_{\text{tail}} \cdot \text{gate}_{x_l}$) widens the safe operating range for high-penalty clauses, giving them more headroom before the safety cutoff kicks in. With the previous push_up term concentrating on small x_s, this gate becomes almost redundant. This is an artifact arising from hyperparameter optimization and might warrant simplification.

5. Amplitude normalization (Fig. 1(e)). A power-law decay with a clause-state-driven floor:

$$\text{amp}_{\text{pow}} = \left(\frac{x_{l,\text{norm}}}{x_{l,\text{norm}} + x_{l,m}}\right)^p, \tag{S7}$$

$$\text{floor\_gate} = 1 - (1 - \text{weak\_band})(1 - \text{push\_up}), \tag{S8}$$

$$\text{amp\_norm} = (1-f)\,\text{amp}_{\text{pow}} + f, \quad f = a_{\text{floor}} \cdot \text{floor\_gate}, \tag{S9}$$

where $x_{l,\text{norm}} = 10{,}087$, $p = 1.75$, and $a_{\text{floor}} = 0.0093$. The power-law decay (Eq. (S7)) regulates the release term as $x_{l,m}$ increases. The floor gate (Eq. (S8)) is a soft logical OR of the weak-band and push-up signals: it guarantees a minimum amplitude precisely when a clause needs intervention. Still, hyperparameter optimization yields a near-zero a_{floor}, suggesting that the floor gate mechanism could be simplified.

* yuz092@ucsd.edu
† csipling@ucsd.edu
‡ diventra@physics.ucsd.edu

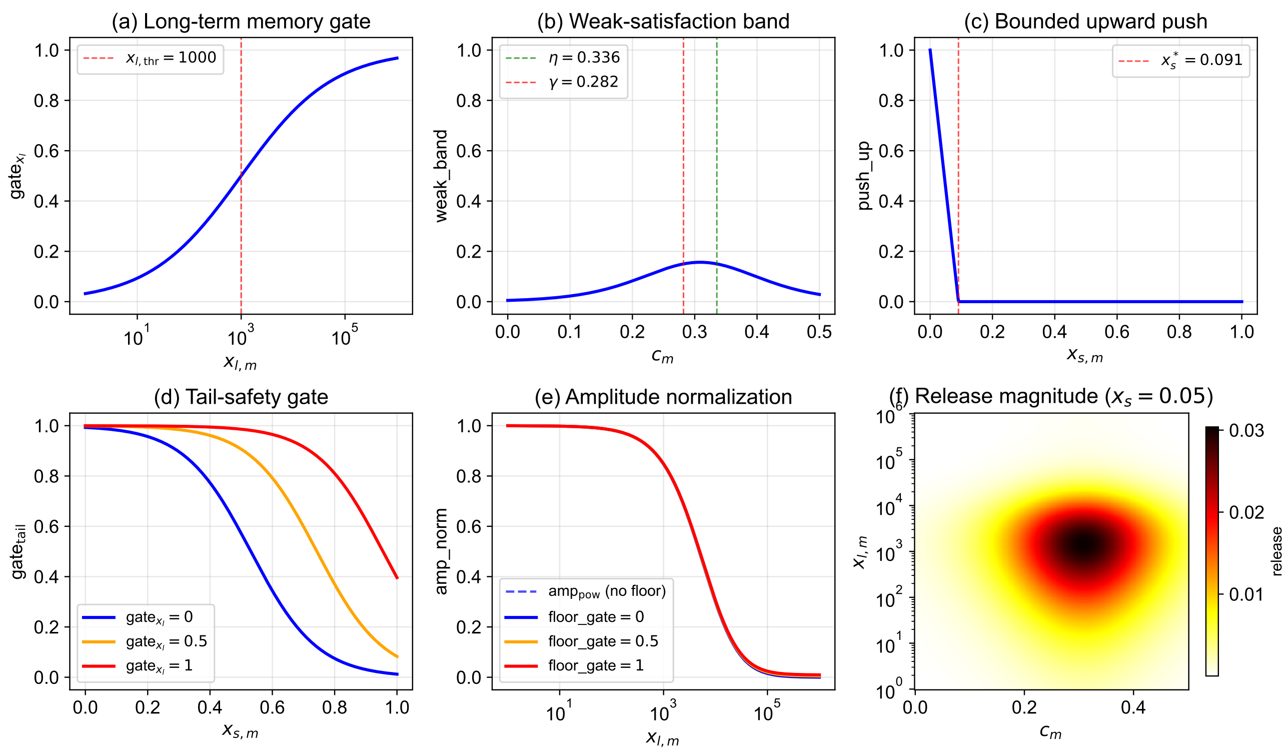

FIG. 1. **Gating components of the release term in design 340.** **(a)** Long-term memory gate gate_{x_l}: activates only when $x_{l,m}$ exceeds the threshold $x_{l,\mathrm{thr}} = 1000$ (dashed red line). **(b)** Weak-satisfaction band: peaks when c_m falls between η and γ; note that the optimized values $\eta = 0.336 > \gamma = 0.282$ produce a narrow, low-amplitude response (peak ≈ 0.16). **(c)** Bounded upward push: linearly decreasing in $x_{s,m}$, reaching zero at $x_s^* = 0.091$. **(d)** Tail-safety gate at three levels of gate_{x_l}, showing how the x_l-dependent shift extends the safe operating range. **(e)** Amplitude normalization with power-law decay and clause-state-driven floor at three levels of floor_gate. **(f)** Combined release magnitude in the $(c_m, x_{l,m})$ plane at $x_{s,m} = 0.05$, showing the narrow region where all gates are simultaneously open.

Table I gives the full hyperparameter comparison between the baseline and design 340.

Design choices and potential simplifications. The five-gate structure was found without any explicit simplicity constraint: agents were judged purely on solver performance, with no penalty for parameter count or code length. The evaluation schedule's wall-time timeout acts as an implicit regularizer: complex designs that run slower are less likely to pass higher fidelity levels. But this pressure is indirect and does not penalize architectural complexity sufficiently. A systematic ablation study would sort out which components are essential and which are evolutionary artifacts.

An explicit simplicity term, like a parameter-count or description-length penalty, could bias the search toward more parsimonious solutions and reduce the hyperparameter optimization burden. Calibrating such a penalty without unintentionally suppressing useful mechanisms would require further fine-tuning.

B. Genealogy

Design 340 is the product of 32 generations of LLM-guided search spanning 154 planner rounds. Fig. 2 shows the primary lineage, traced by following the highest-weight parent reference at each step. Table II lists every design in this lineage with its key innovation and maximum solved problem size. A consistent pattern runs through the entire ancestry: every architectural change modifies only the short-term memory dynamics $\dot{x}_{s,m}$; the variable update (Eq. (4)) and long-term memory update (Eq. (6)) are never touched.

The evolution falls into four phases, each contributing distinct pieces of the final solver.

Phase 1: Core release structure (designs 1-12)

The first phase laid down the release term's basic architecture. Design 5 introduced the central idea: an additive "bump" in the x_s dynamics, gated by long-term memory: $\mathcal{R}_m = \kappa \cdot \mathrm{gate}_{x_l}(x_{l,m}) \cdot c_m \cdot \max(\gamma - c_m, 0)$. It fires for persistently violated clauses ($x_{l,m} \gg x_{l,\mathrm{thr}}$) whose satisfaction monitor falls below γ, cutting the baseline's median

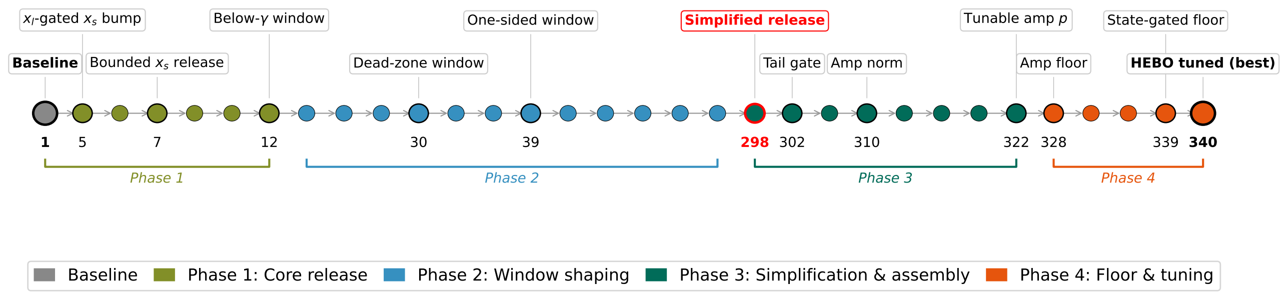

FIG. 2. **Primary lineage of design 340.** Each node is a milestone design, colored by innovation phase. Design IDs appear below each node; several intermediate designs are omitted for clarity. The full lineage spans 32 designs across 154 planner rounds.

TABLE I. **Hyperparameter comparison.** The baseline uses 7 parameters; design 340 uses 20, with the 13 additional parameters controlling the release mechanism. Values shown are the HEBO-optimized defaults for design 340.

Parameter	Role	Baseline	Design 340
Shared parameters (backbone)			
α_l	x_l growth rate	5.0	11.24
β	x_s update rate	20.0	42.07
γ	x_s threshold	0.25	0.282
δ	x_l threshold	0.05	0.080
ϵ	x_s regularization	10^{-3}	8.5×10^{-4}
ζ	Rigidity coupling	10^{-3}	0.021
Δt_0	Integration time step	1.0	2.99
Release mechanism (new)			
κ	Release strength	—	1.015
$x_{l,\text{thr}}$	Memory gate threshold	—	1000.4
ω_l	Memory gate sharpness	—	2.025
η	Band lower edge	—	0.336
ω_b	Band sharpness	—	0.063
x_s^*	Push-up target	—	0.091
$x_{s,\text{tail}}$	Tail-safety threshold	—	0.531
ω_{tail}	Tail-safety sharpness	—	0.107
μ_{tail}	Tail x_l-shift	—	0.424
$x_{l,\text{norm}}$	Amp decay scale	—	10,087
p	Amp decay power	—	1.753
a_{floor}	Amp floor level	—	0.0093

step count at $N = 320$ from 37,729 to 9,241. Design 6 then added a $(1-x_s)$ damping factor so the bump would not saturate x_s.

Design 7 was the first real breakthrough. It replaced the ad-hoc $c_m \cdot \max(\gamma - c_m, 0)\cdot(1-x_s)$ product with three cleanly separated components—weak_band, push_up, and x_s^*—establishing the multiplicative gating structure that survives into the final solver:

$$\mathcal{R}_m = \kappa \cdot \text{gate}_{x_l} \cdot \text{weak\_band}(c_m)\cdot \text{push\_up}(x_{s,m}). \quad \text{(S10)}$$

$N_{\max}$ jumped from 320 to 453 after this change. Designs 10-12 refined the release window: design 10 added a "near-γ" gate to concentrate release on clauses closest to the satisfaction threshold, and design 12 restricted action to $c_m < \gamma$ (below-γ windowed release), pushing $N_{\max}$ to 905. HEBO tuning of design 10 alone cut median steps at $N = 453$ from 45,234 to 12,409—a 3.6× speedup from hyperparameters.

Phase 2: Window shaping and amplitude refinement (designs 16-285)

With the basic release structure working up to $N \le 905$, the search spent designs 16-285 (12 generations, planner rounds 8-133) exploring progressively more elaborate amplitude shaping functions. Design 16 added a floor to the amplitude window. Design 20 brought in power-law shaping (t^p). Design 30 introduced a dead-zone threshold. Design 34 swapped the amplitude for a smoothstep function ($6t^5 - 15t^4 + 10t^3$). Design 36 applied a rational warp ($s/(s+(1-s)^2)$). These refinements pushed $N_{\max}$ to 1,280 (design 30), though step counts at that size remained high (130,867–230,042).

Two other directions were explored in parallel: x_l-dependent amplitude boosting (design 43, replacing gate_{x_l} with $\text{gate}_{x_l}(1+g\cdot\text{gate}_{x_l})$) and x_l-adaptive sharpness control (designs 222–285, making the window sharpness depend on clause memory state). These additions pushed the parameter count upward (from 13 to 24) without matching gains in scaling. Design 285, the most complex solver in this phase, ran 183 lines of code with 24 hyperparameters yet could only solve up to $N = 320$.

Phase 3: Simplification and component assembly (designs 298-322)

Design 298 marks a turning point. The LLM discarded the entire accumulated complexity of Phase 2 (smoothstep, rational warp, dead zones, x_l-adaptive sharpness, threshold shifts) and reverted to the clean three-gate structure of design 7: $\mathcal{R}_m = \kappa \cdot \text{gate}_{x_l} \cdot \text{weak\_band} \cdot \text{push\_up}$. Code shrank from 183 lines and 24 hyperparameters to 118 lines and 13 hyperparameters. Reach dropped ($N_{\max}$ fell from 1,280 to 640), but the simplified

TABLE II. **Complete primary lineage of design 340.** Each row shows a design along the highest-weight ancestry path, its key innovation, the release term structure at that point, maximum solved problem size $N_{\max}$ (with median steps at $N_{\max}$), and consensus score across all objectives. HEBO-tuned designs share identical code with their parent; only hyperparameters change. $N_{\max}$ is determined by the largest N at which unsolved fraction < 0.5.

ID	Short name	Innovation	$N_{\max}$	Steps@$N_{\max}$	$\mathcal{R}_m$ structure
Phase 1: Core release structure (designs 1–12)					
1	Baseline	—	320	37,729	—
5	x_l-gated x_s bump	gate$_{x_l}$, κ	320	9,241	$\kappa \cdot \text{gate}_{x_l} \cdot c \cdot \max(\gamma - c, 0)$
6	x_s hysteresis bump	$(1 - x_s)$ damping	320	9,452	$\kappa \cdot \text{gate}_{x_l} \cdot c \cdot \max(\gamma - c, 0) \cdot (1 - x_s)$
7	Bounded x_s release	weak_band, push_up, x_s^*	**453**	66,681	$\kappa \cdot \text{gate}_{x_l} \cdot \text{wb} \cdot \text{pu}$
10	Near-γ release gate	near_gamma gate	453	45,234	+ near$_\gamma$ factor
11	HEBO-tuned	(hyperparameters)	453	12,409	(same)
12	Below-γ windowed	below-γ filter	**905**	229,470	+ below$_\gamma$ window
Phase 2: Window shaping & amplitude refinement (designs 16–285)					
16	Floored window	win_floor	905	203,389	+ amp with floor
20	Power-shaped window	win_pow	905	191,288	+ t^p shaping
21	HEBO-tuned	(hyperparameters)	640	32,071	(same)
30	Dead-zone window	win_tau dead zone	**1,280**	220,176	+ τ dead zone
34	Smootherstep amp	smootherstep function	1,280	230,042	+ $s(t)$ smoothing
36	Warp amp	rational warp	1,280	225,071	+ $s/(s + (1 - s)^2)$ warp
39	One-sided window	simplified to one-sided	1,280	130,867	one-sided sigmoid window
43	x_l-gain amp	x_l-dependent boost	1,280	191,946	+ $(1 + g \cdot \text{gate}_{x_l})$
222	Safe x_l-sharp cap	sharp_min_ratio, sharp_gain	905	35,104	+ x_l-adaptive sharpness
228	γ-threshold shift	thr_shift	905	62,151	+ x_l-dependent γ shift
280	Shift sharpening	shift_pow	1,280	143,552	+ power-law shift
285	Dead-zone shift	shift_tau	320	3,476	+ τ dead zone on shift
Phase 3: Simplification & component assembly (designs 298–322)					
298	Weak-band release	**radical simplification**	640	64,272	$\kappa \cdot \text{gate}_{x_l} \cdot \text{wb} \cdot \text{pu}$
302	Tail gate	gate_tail, $x_{s,\text{tail}}$	640	51,263	+ gate$_{\text{tail}}$
305	x_l-shifted tail	tail_mu	320	6,001	+ $\mu \cdot \text{gate}_{x_l}$ in tail
310	Amplitude norm	amp_norm	640	45,978	+ amp_norm
316	Sign flip in tail	$-\mu \to +\mu$	640	38,994	sign correction
320	Over-damped	amp_norm^2	640	58,343	squared amp_norm
321	HEBO-tuned	(hyperparameters)	**1,280**	118,834	(same)
322	Tunable amp power	amp_p	905	28,192	amp_norm $= (\cdot)^p$
Phase 4: Floor refinement & final tuning (designs 328–340)					
328	x_l-gated floor	amp_floor $\cdot$ gate$^2_{x_l}$	905	41,271	+ $f \cdot \text{gate}^2_{x_l}$ floor
332	Linear floor	amp_floor $\cdot$ gate$_{x_l}$	905	31,154	+ $f \cdot \text{gate}_{x_l}$ floor
336	Ungated floor	amp_floor (constant)	320	2,916	+ f constant floor
339	State-gated floor	**floor_gate** (soft-OR)	**1,810**	691,058	+ $f \cdot \text{OR}(\text{wb}, \text{pu})$
340	HEBO-tuned	(hyperparameters)	**1,810**	**95,503**	(same)

base gave the remaining innovations a clean foundation.

From there, the system assembled the final release term's remaining components in quick succession:

- Design 302: Added gate$_{\text{tail}}$, a sigmoid suppressing release at high $x_{s,m}$ to prevent saturation.
- Design 305: Added $\mu_{\text{tail}} \cdot \text{gate}_{x_l}$ to the tail gate, letting the safety threshold shift with x_l.
- Design 310: Added amp_norm $= x_{l,\text{norm}}/(x_{l,\text{norm}} + x_{l,m})$, a monotone decay preventing release amplitude from growing with x_l.
- Design 316: Flipped the tail-gate shift sign from $-\mu$ to $+\mu$, extending (not reducing) the safe range for high-x_l clauses.
- Design 320: Squared amp_norm for stronger damping at large x_l.
- Design 322: Generalized the square to a tunable power p, replacing amp_norm^2 with amp_base^p and making p a hyperparameter.

By design 321 (HEBO tuning of design 320), $N_{\max}$ had recovered to 1,280 at 118,834 median steps. The release term now contained all five gating components of the final solver.

Phase 4: Floor refinement and final tuning (designs 328-340)

One problem remained: amp_norm decays to zero as $x_{l,m} \to \infty$, which can starve clauses that genuinely need sustained help. Design 328 introduced an amplitude floor, amp_norm $= (1 - f) \cdot$ amp_pow $+ f$, with $f = a_{\text{floor}} \cdot \text{gate}_{x_l}^2$. But because the release term already multiplies by gate_{x_l}, this created a double-gating pathology: the effective floor scaled as $\text{gate}_{x_l}^3$.

Three floor variants followed in rapid succession:

- Design 332: Reduced to $f = a_{\text{floor}} \cdot \text{gate}_{x_l}$ (linear; effective $\propto \text{gate}_{x_l}^2$).
- Design 336: Dropped x_l dependence entirely: $f = a_{\text{floor}}$ (constant floor; effective $\propto \text{gate}_{x_l}$).
- Design 339: Conditioned the floor on clause state instead: $f = a_{\text{floor}} \cdot [1 - (1 - \text{weak\_band})(1 - \text{push\_up})]$. This soft-OR activates the floor when the clause sits in the weak-satisfaction band *or* when $x_{s,m}$ is below target, and shuts off otherwise.

Design 339 reached $N_{\max} = 1{,}810$ (691,058 steps), the first solver in the lineage to clear the largest benchmark instances. Design 340 (HEBO tuning of 339, no code changes) then brought the median steps at $N = 1{,}810$ down from 691,058 to 95,503, a 7.2× speedup from hyperparameters alone, confirming the importance of both architecture and tuning.

Summary: Building up the release term

Table III records which design first introduced each component of the final release term. The core three-gate structure ($\text{gate}_{x_l} \cdot$ weak_band $\cdot$ push_up) was in place by design 7; assembling the full five-gate structure with a clause-state-driven floor took 32 generations and a critical simplification event at design 298.

TABLE III. **Release term component origins.** Each row shows the first design in the lineage that introduced a component of the final release term (Eq. (S2)).

Component	Symbol	Design	Short name
Coupling strength	κ	5	x_l-gated x_s bump
Long-term memory gate	gate_{x_l}	5	x_l-gated x_s bump
Weak-satisfaction band	weak_band	7	Bounded x_s release
Bounded upward push	push_up	7	Bounded x_s release
Tail-safety gate	$\text{gate}_{\text{tail}}$	302	x_s-cap tail gate
x_l-shifted tail	μ_{tail}	305	x_l-shifted tail gate
Amplitude norm	amp_norm	310	Tail-safe release
Tunable power	p	322	Tunable amp_p
Amplitude floor	a_{floor}	328	x_l-gated amp floor
Clause-state floor	floor_gate	339	State-gated floor

C. Analysis

Why the release mechanism improves scaling. The baseline x_s dynamics (Eq. (5)) amounts to a simple proportional controller: $x_{s,m}$ grows when $c_m > \gamma$ (clause violated) and shrinks when $c_m < \gamma$ (clause satisfied). Every clause gets the same treatment: one that has been stuck for thousands of steps receives the same dynamical response as one that was violated only briefly. At large problem sizes, where clauses compete for variable updates through the shared variable dynamics (Eq. (4)), this indiscriminate response leads to wasted computational effort.

The release term (Eq. (S2)) provides a targeted intervention that fires only when five conditions hold simultaneously:

1. The clause has been persistently violated ($x_{l,m} \gg x_{l,\text{thr}}$, via gate_{x_l}).
2. The clause sits in a critical satisfaction state ($c_m \approx 0.3$, via weak_band).
3. The short-term memory is below target ($x_{s,m} < x_s^*$, via push_up).
4. The short-term memory is not saturated ($x_{s,m}$ below safety threshold, via $\text{gate}_{\text{tail}}$).
5. The release amplitude is properly regulated (via amp_norm).

The conjunction channels effort toward persistently stuck clauses that are close to flipping and need a gentle push, rather than clauses that are far from satisfaction or would resolve on their own through the baseline dynamics.

Conservative parameter regime. The HEBO-optimized [1] hyperparameters significantly restrain the release mechanism. Three noteworthy choices:

- *High memory threshold* ($x_{l,\text{thr}} = 1000$). The long-term memory gate demands $x_{l,m} > 1000$ before it opens. Since $x_{l,m}$ starts at 1 and grows at rate $\alpha(c_m - \delta)$, a clause must be persistently violated before the release mechanism engages at all.
- *Inverted weak-band* ($\eta = 0.336 > \gamma = 0.282$). The band edges are flipped relative to their original design intent (η was meant to be less than γ), producing a narrow, low-amplitude response peak of ≈ 0.16. The weak-band filter ends up highly selective.
- *Very low push-up target* ($x_s^* = 0.091$). The push deactivates once $x_{s,m} > 0.091$: a small initial nudge, not sustained forcing.

Together these produce a release mechanism that fires rarely (high $x_{l,\text{thr}}$), responds weakly (inverted band, low x_s^*), and acts briefly (push-up saturates quickly). The combined amplitude is typically 1–3% of the baseline x_s dynamics. This small intervention suffices to break deadlocks at large N without disrupting the baseline dynamics that handle the majority of clauses well.

Backbone parameter shifts. The optimized backbone parameters also change a lot from the baseline: α

more than doubles ($5 \to 11.2$), β doubles ($20 \to 42$), and the time step Δt_0 triples ($1 \to 3.0$). These shifts make the baseline dynamics faster and more responsive overall; the release mechanism then provides a focused correction for the few clauses that get stuck.

Scaling comparison. Fig. 3 compares scaling in detail. The speedup grows with problem size: no speedup at small N ($\sim$1× at $N = 10$), modest at intermediate N ($\sim$4× at $N = 320$), and accelerating at large N (67× at $N = 1810$). The pattern matches the picture above: at small N most clauses resolve through the baseline dynamics alone, so the release mechanism adds little. At large N the growing web of clause interactions creates more persistent deadlocks, and the targeted release becomes increasingly valuable.

II. OBJECTIVE EVOLUTION ANALYSIS

Over the course of the search, the objective agent produced 42 objective functions across eight independent workspaces that were later merged. These objectives pass through four phases, tracking how the system's notion of a good solver changes as understanding deepens. Table VII lists all 42 objectives with their origin and final consensus weights. We trace this evolution below, document three reward hacking episodes that were detected and mitigated, and analyze the shifting correlation structure of the objective portfolio.

A. Objective phases

Phase 1a: Power-law extrapolation (objectives 0–5). The first six objectives came from workspace 22. They shared a common strategy: fit power-law ($\log s \sim a + b \log N$) and exponential ($\log s \sim a + bN$) models to the observed median steps s vs. problem size N, then extrapolate to a fixed target $N = 2000$. Objective 0, the initial human-designed baseline, used an R^2-weighted softmax blend of both models. Successive objectives show incremental refinements: reliability penalties based on `unsolved_fraction` (objective 1), coverage penalties when the largest tested N fell far short of the target (objective 2), AIC-weighted model averaging (objective 3), and smooth failure-margin penalties (objectives 4-5).

All six shared a basic limitation: they worked entirely from small-N data ($N \leq 640$ under low-fidelity evaluation) and tried to extrapolate to $N = 2000$. There is also a misunderstanding on the `unsolved_fraction` variable: a problem is considered solved and terminates when `unsolved_fraction` falls below 0.5 (effectively a binary success/fail signal), but the objectives treat it as a continuous variable and reward designs with smaller `unsolved_fraction`. The result was a tight cluster of correlated objectives: the mean within-phase pairwise Kendall's τ was 0.588, with no negative pairs.

Phase 1b: Censored-aware scaling (objectives 6-15). In parallel, workspace 24 produced ten objectives through a similar iterative refinement loop. The key advance was a restart-steps model: instead of raw median step s, these objectives computed an effective cost $s_{\text{eff}} = s + (\texttt{unsolved\_fraction}/(1 - \texttt{unsolved\_fraction})) \cdot \texttt{max\_steps}$, accounting for the restarts needed when some runs fail. They also adopted AIC-weighted model averaging between polynomial and exponential scaling models and used Theil-Sen (median) regression for outlier robustness.

Nevertheless, Phase 1b objectives still use the extrapolation-to-N=2000 paradigm and still treated `unsolved_fraction` as a continuous variable below the success threshold 0.5. Without the meta-agent (not yet introduced), objective evolution in this workspace amounted to incremental polishing of one approach, producing a tight cluster with mean within-workspace $\tau = 0.536$.

Phase 2: Reach-dominant metrics (objectives 16-31). The later exploratory workspaces exhibit a paradigm shift. Instead of extrapolating to a fixed target N, these objectives maximized the largest problem size the solver could actually clear—a metric we call "reach." Objective 20 introduced the structure that every subsequent objective adopted: a reach term (e.g., $-\log_2 N_{\max}$) dominates by orders of magnitude, with tail-scaling and budget-headroom terms serving only as tie-breakers among designs that reach the same N.

Hard gates at schedule milestones also appeared in this phase: objective 20 penalized solvers failing before $N = 640$ and, when medium/high-fidelity data were available, before $N = 1280$. Later objectives (25-28) added conservative worst-window tail fits and budget-cliff hazard terms for sharp jumps in `median_step/max_steps` at the frontier.

The meta-agent entered in workspace 36, and its effect on diversity is visible in the numbers: workspace 36 yielded the most diverse objective set (within-workspace mean $\tau = 0.359$, versus 0.587 and 0.536 in the pre-meta-agent workspaces). Phase 2 also produced the most problematic objective in the entire search (objective 17; see Sec. II B).

Phase 3: Schedule-faithful objectives (objectives 32–41). The last ten objectives were generated in the merged workspace, with all earlier objectives and designs available as context. They learned from the lessons in Phases 1 and 2, and produced a mature design pattern resting on three principles:

1. *Strict binary pass/fail.* Success means `unsolved_fraction` < 0.5; a value of 0.49 is never penalized.

2. *Schedule-faithful headroom.* Budget headroom is computed from the *schedule budget* $B(N)$ (a deterministic function of N and the fidelity cap), not from the run's `max_steps`. This blocks designs from

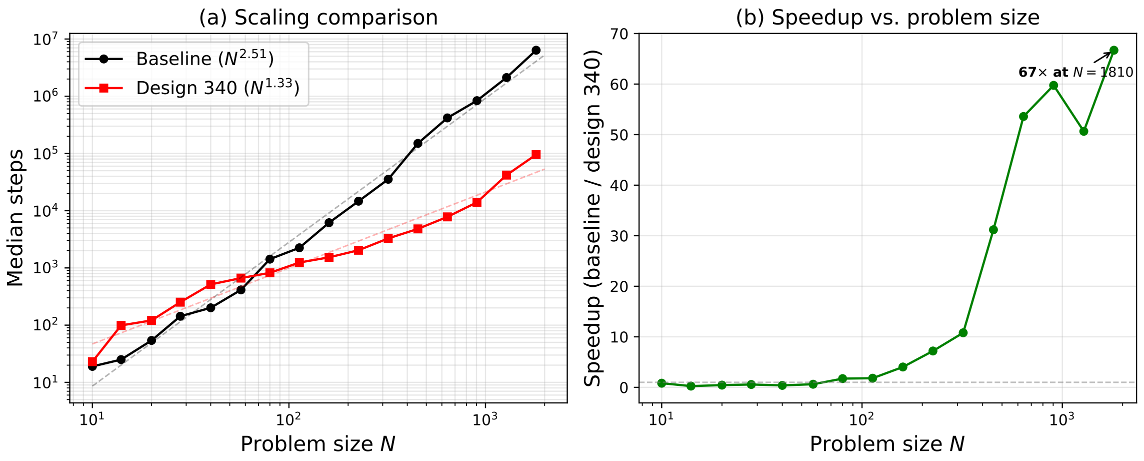

FIG. 3. **Scaling comparison.** **(a)** Median solution steps vs. problem size N for the baseline (black) and design 340 (red); power-law fits are dashed. **(b)** Speedup ratio (baseline steps / design 340 steps) vs. N, growing monotonically to 67× at $N = 1810$. The horizontal dashed line corresponds to a speedup of 1× (no change).

inflating headroom by running with inflated budgets.

3. *Smooth-max bottleneck detection.* Rather than sampling headroom at a single point, the objective takes a smooth-max (log-sum-exp) of $\log(\texttt{median\_step}/B(N))$ over the last 3 cleared levels plus conservative worst-window predictions at $N = 1810$ and $N = 2560$. The worst bottleneck is identified without being dominated by a single noisy data point.

Further components included multi-step budget-cliff hazard penalties for acceleration in $\log(\texttt{median\_step}/B)$ across the $905 \to 1280 \to 1810 \to 2560$ transitions (gated so they fire only once the solver reaches those regimes), and optional repeat-robustness terms penalizing mixed pass/fail outcomes across repeated runs at the same N.

Phase 3 objectives agreed most closely with the consensus ranking (mean $\tau = 0.624$), and their per-objective winners had a median consensus rank of 2 (compared with 95 for Phase 1b).

B. Reward hacking episodes

Three reward hacking patterns surfaced during the search. Each was caught through correlation analysis and corrected by the meta-agent's weight adjustments.

Episode 1: Small-N over-optimization (objectives 0–15). Phase 1 objectives rewarded designs whose scaling curves looked favorable on small-N data, regardless of whether the extrapolation held up. The meta-agent's first assessment (round 1) diagnosed the issue: "designs reach $N = 640$ under the adaptive schedule, but this is largely driven by hovering just below the `unsolved_fraction`< 0.5 gate while `median_step` explodes at higher N." By round 6 of the merged workspace, the meta-agent had zeroed out all 16 Phase 1 objectives (weights set to 0.0), concentrating the consensus on Phase 2 and 3 metrics. Even after suppression, these objectives still contributed passively to the consensus through agreement-based weighting: objectives 0, 2, 4, 7, 12, and 15 retained Kendall's $\tau > 0.78$ with the final consensus ranking. Their winners were often poor, but their overall ranking structure was partially aligned.

Episode 2: Gate-passing artifacts (objective 17). Objective 17 was the worst reward hacking case in the search. It used a soft threshold at $u_{\text{eff}} \leq 0.40$ (with uncertainty-aware unsolved fraction $u_{\text{eff}} = u + \text{SE}_{\text{binom}}$) and computed a composite score from effective solved size N_{eff}, a time-to-solution (TTS) surrogate, and a scaling exponent penalty. In practice, the formulation rewarded designs achieving moderate solve rates ($u \approx 0.35$–0.40) across many small N values, even if those designs failed catastrophically at larger N.

The consequences were dramatic: objective 17 had a Kendall's τ of -0.472 with the consensus ranking, anti-correlated with the majority view of quality. Its top 10 designs all sat between consensus ranks 345 and 414 (bottom 5%). Conversely, it placed the consensus winner (design 340) at rank 390 out of 414. The meta-agent detected the anti-correlation and set objective 17 to weight 0.0 in every round after its creation.

Episode 3: Echo chamber in pre-meta-agent workspaces. Without the meta-agent's diversity-promoting guidance, objective evolution in workspaces 22 and 24 amounted to incremental refinement of the same approach. Workspace 22's objectives shared a mean pairwise τ of 0.587; workspace 24's shared 0.536.

Introducing the meta-agent in workspace 36 visibly increased diversity. Its strategic guidance explicitly steered later objectives toward different facets of solver quality (headroom vs. hazard vs. robustness), yielding within-workspace $\tau = 0.359$—the lowest among all multiobjective workspaces. In the merged workspace, the meta-agent diversified Phase 3 objectives further by specifying distinct tie-breaking mechanisms and introducing optional components.

Detection and mitigation. The meta-agent's weight trajectory (Table IV) shows weight steadily migrating from early to late objectives. By round 6, all Phase 1 objectives (0-19) were zeroed out. In the final round only 13 of 42 objectives retained nonzero weights, with 8 receiving weights ≥ 1.0. The heaviest-weighted objectives (39, 40, 37, 35, 34) were all Phase 3 schedule-faithful metrics. Importantly, suppression was not merely age-based: objectives 20-22 and 24 (Phase 2) kept nonzero weights in the final round because of their high correlation with the consensus, while the more recent objective 32 (Phase 3, $\tau = -0.162$) was suppressed.

TABLE IV. **Meta-agent weight trajectory.** Aggregate weight per objective phase across meta-agent rounds, showing the progressive shift from Phase 1 extrapolation objectives to Phase 3 schedule-faithful metrics.

Round	Phase	Ph. 1a	Ph. 1b	Ph. 2	Ph. 3
1	stuck	1.3	3.1	15.2	0.0
3	breakthrough	0.4	0.5	17.1	1.0
6	stuck	0.0	0.0	16.7	7.7
8	refining	0.0	0.0	7.7	8.7
11	refining	0.0	0.0	1.3	16.8

Limitation of agreement-based weighting

The meta-agent's progressive suppression of Phase 1 objectives (Table IV) raises a natural question: was the intervention necessary, or would the built-in age decay have done the same job without external override?

Agreement-based weighting gives each objective a weight proportional to its median pairwise Kendall's τ with all others. This works well when misaligned objectives are isolated outliers—their low correlation with the majority suppresses their influence. But the mechanism has a structural weakness: when a block of correlated objectives forms the majority, each member's median τ is inflated by agreement with other block members, regardless of whether what they collectively measure aligns with the true research goal. Agreement is computed within the current portfolio, so the mechanism cannot distinguish a genuinely informative majority from an echo chamber of redundant proxies.

That is precisely what happened in the merged workspace. At the time of merging, Phase 1a and 1b objectives (IDs 0-15) made up 16 of 32 total objectives. They formed a tight cluster (mean pairwise $\tau = 0.561$; Table V; zero negatively correlated pairs) that collectively rewarded small-N performance, even when those designs scaled poorly past $N = 1000$. Because block members agreed with each other, their per-objective median τ values stayed high (mean 0.49, vs. 0.40 for Phase 2), and the consensus mechanism gave them a combined weight share of 55% despite age decay.

Age decay alone could not fix this. At $\lambda = 0.9$, an objective still retains $0.9^{10} \approx 0.35$ of its original contribution after 10 rounds. With 16 objectives in the cluster, collective influence decays slowly. Furthermore, a newly generated objective that happens to correlate with the Phase 1 cluster would receive high agreement-based weight regardless of recency, perpetuating the misalignment.

The meta-agent supplies an orthogonal correction. Instead of measuring statistical agreement among objectives, it assesses whether each objective's evaluation criterion still fits the current research strategy. Here, the meta-agent recognized that Phase 1 objectives were optimizing for small-N scaling when the search needed large-N reach and tail efficiency. Its weight multipliers broke the cluster's grip: by round 6 all 16 Phase 1 objectives were set to $m_i = 0.0$, shifting the consensus entirely to Phase 2 and 3 metrics. The intervention was gradual, so the consensus ranking shifted smoothly, avoiding destabilization of the planner and designer agents that depend on consistent rankings.

The multipliers compose with, rather than replace, the agreement-based weights: the final weight is $w'_i = w_i \cdot m_i / \sum_k w_k \cdot m_k$, where w_i is the consensus weight and m_i is the meta-agent multiplier. The meta-agent therefore cannot amplify an objective that the consensus mechanism has already flagged as an outlier. Multipliers only modulate the relative influence among objectives that already carry nonzero consensus weight, providing a targeted correction for cluster dominance without undermining the consensus mechanism's built-in outlier suppression.

C. Correlation structure

The pairwise Kendall's τ matrix (Fig. 2(a) of the main text) captures the full correlation structure of the 42-objective portfolio. Here we quantify how that structure evolved as objectives were added.

Table V tells a clear story: the mean pairwise τ fell from 0.588 (Phase 1a alone, 6 objectives) to 0.399 (all 42), while the fraction of negatively correlated pairs rose from 0% to 11.3%. This growing disagreement is not pathological. Instead, it reflects increasing diversity in how solvers are evaluated. Early objectives agreed because they were variations on a single extrapolation theme; later objectives introduced fundamentally different evaluation criteria (reach vs. headroom vs. hazard), which naturally reduced pairwise agreement while broadening the coverage of the portfolio.

The within-phase vs. cross-phase structure reveals an informative asymmetry. Phase 1a objectives have higher within-phase τ (0.588) than cross-phase τ (0.452), which is a classic echo-chamber signature. Phase 2 shows the

TABLE V. **Pairwise Kendall's τ snapshots.** As objectives accumulate across phases, mean pairwise correlation drops and the fraction of negative pairs grows, reflecting greater diversity.

Objective subset	Mean τ	Median τ	Negative pairs
Phase 1a only (0–5)	0.588	0.553	0.0%
Phases 1a+1b (0–15)	0.561	0.555	0.0%
Through Phase 2 (0–31)	0.391	0.432	9.5%
All objectives (0–41)	0.399	0.431	11.3%

reverse: within-phase τ (0.287) is *lower* than cross-phase τ (0.369), meaning these objectives were more diverse among themselves than in their relationship to other phases. That inversion traces directly to the meta-agent's diversity-promoting guidance in workspace 36.

Per-objective Kendall's τ with the consensus ranking (Table VI) confirms that no single objective captures the consensus perfectly. Even the best-aligned (objectives 0, 2, 21, 28, 39) reach only $\tau \approx 0.84$–0.85, so roughly 8% of design pairs are ranked differently. At the other end, objectives 17, 32, 23, 25, and 27 have $\tau < 0.16$—near-random or anti-correlated.

TABLE VI. **Per-phase agreement with the consensus ranking.** Mean and range of Kendall's τ between individual objectives and the consensus, by phase.

Phase	Mean τ	Min τ	Max τ
1a (Power-law)	0.665	0.459	0.849
1b (Censored)	0.603	0.343	0.842
2 (Reach-dominant)	0.489	−0.472	0.844
3 (Schedule-faithful)	0.624	−0.162	0.845

D. Why the consensus outperforms individual objectives

Three observations from the data above illustrate why the consensus mechanism matters.

Winner divergence. Only 7 of 42 objectives (17%) rank the same design as the consensus winner (design 340) at position #1. Just 24 of 42 (57%) place their #1 design in the consensus top-10. The mean consensus rank of per-objective winners is 84.3 (out of 414 designs), and for Phase 1b objectives the median consensus rank of their winners is 95.

Visibility of the best design. Design 340 falls outside the top-10 under 14 of 42 objectives (33%), outside the top-20 under 10 (24%), and outside the top-50 under 8 (19%). Objective 17 ranks it 390th out of 414—worse than 94% of all designs. Objective 3 ranks it 239th. No single objective consistently identifies design 340 as a top candidate. By weighting objectives on agreement and recency, the consensus aggregation ensures that design 340's strong performance across the *majority* of objectives lifts it to rank #1 despite its poor standing under a minority.

Phase-dependent selection quality. Phase 3 objectives' per-objective winners have a mean consensus score of 0.064 (near the best achievable score of 0.028), while Phase 1b winners have a mean consensus score of 0.344—over 5× worse. Later objectives, drawing on more experimental data and meta-agent guidance, exert substantially better selection pressure. The consensus mechanism picks this up automatically through the age decay factor ($\lambda = 0.9$), which down-weights older objectives even before the meta-agent explicitly suppresses them.

E. Complete objective list

Table VII lists all 42 objectives with their origin and final meta-agent weight multipliers.

III. METHODOLOGY EVOLUTION

The framework described in the main text went through several rounds of redesign across independent workspaces. This section records the key methodological changes and the empirical reasoning behind each.

A. Meta-agent introduction

Before workspace 36, the objective agent ran on its own: it proposed new objective functions and set their weights with no external oversight. The result was the *echo-chamber effect* documented in Sec. II B (Episode 3): the agent repeatedly reinforced a narrow set of evaluation criteria. Objectives produced this way exhibited high mutual correlation (mean pairwise Kendall $\tau = 0.587$–0.949 within echo-chamber clusters; see Table V) and consistently ranked the same small subset of designs at the top, narrowing the search.

Workspace 36 introduced a *meta-agent* as an oversight layer between the objective agent and the consensus mechanism. The meta-agent reviews newly proposed objectives, adjusts their consensus weights, and can suppress objectives that are redundant or misaligned with the broader research goals. In practice, it progressively reallocated weight from Phase 1 and Phase 2 objectives to the more informative Phase 3 (schedule-faithful) objectives as those were created (Table IV). This broke the echo-chamber pattern and restored the objective diversity that the consensus mechanism needs to work well (Sec. II D).

B. Evaluation schedule

An early design question was whether the evaluation schedule (problem sizes N, clause-to-variable ratios, and

TABLE VII. **Complete list of all 42 objectives.** Each objective is listed with its ID, source workspace, phase, description (abbreviated), and final meta-agent weight multiplier. Objectives with weight 0.0 were suppressed by the meta-agent; a dash (—) indicates the objective was created after the final meta-agent round.

ID	Source	Phase	Description	Wt.
Phase 1a: Power-law extrapolation				
0	merged	1a	R^2-weighted power-law / exponential blend, extrapolate to N=2000	0.0
1	ws-22	1a	Predicted $\log_{10}$ steps at N=2000 incl. scaling + reliability	0.0
2	ws-22	1a	$\log_{10}$ expected steps with reliability-adjusted cost + coverage penalty	0.0
3	ws-22	1a	AIC-mixed scaling to N=2000 with smooth fail/coverage	0.0
4	ws-22	1a	Smooth reliability and budget-cap penalties	0.0
5	ws-22	1a	Scaling + reliability-margin penalties	0.0
Phase 1b: Censored-aware scaling				
6	ws-24	1b	Success-rate + scaling extrapolation to N=2000	0.0
7	ws-24	1b	Censor-aware AIC model avg. with failure lower-bounds	0.0
8	ws-24	1b	Success + scaling + coverage extrapolation	0.0
9	ws-24	1b	AIC-avg $\log_{10}$ expected solve-steps incl. success + coverage	0.0
10	ws-24	1b	AIC-avg robust scaling + reliability	0.0
11	ws-24	1b	AIC-avg $\log_{10}$ restart-steps + reliability margin	0.0
12	ws-24	1b	AIC-avg restart-steps + slope and reliability	0.0
13	ws-24	1b	p-adjusted steps with smooth coverage and slope penalties	0.0
14	ws-24	1b	Restart-steps with reliability margin and censoring	0.0
15	ws-24	1b	AIC-weighted censored scaling fit of restart-steps	0.0
Phase 2: Reach-dominant				
16	ws-32	2	Extrapolated log-steps + near-threshold failure penalty + reach reward	0.0
17	ws-33	2	Composite N_{eff} at $u \le 0.40$ + TTS surrogate + scaling exponent	0.0
18	ws-34	2	Robust high-N solves ($u \le 0.3/0.2$) + brittleness penalty	0.0
19	ws-34	2	Soft-pass area + top-level reward + anti-truncation	0.0
20	ws-36	2	Reach-dominant: hard fail before 640/1280 + tail slope + slack	0.2
21	ws-36	2	Reach + budget-margin/slack + conservative next-N clearance	0.3
22	ws-36	2	Successful-prefix clearance N^* + near-budget tail penalty	0.4
23	ws-36	2	Reach + conservative worst-of-multi-fit clearance at 1280/2560	0.0
24	ws-36	2	Reach + worst-window predicted earliest schedule failure	0.3
25	ws-36	2	Robust scheduled N^* with ratio-space clearance ≤ 0.60	0.0
26	ws-36	2	Gated tail-only headroom-to-1280 + slope/curvature penalties	0.0
27	ws-36	2	Tail-local polynomial scaling + budget-clearance robustness	0.0
28	ws-36	2	Two-jump headroom at $640 \to 905 \to 1280$ + exponent penalty	0.0
29	ws-37	2	Reach + low scaling exponent + censored extrapolation to N=2000	0.0
30	ws-37	2	Reach + scaling + headroom + frontier robustness	0.0
31	ws-37	2	Schedule-faithful lexicographic: reach, then steps, then slope	0.1
Phase 3: Schedule-faithful				
32	merged	3	Tail headroom: conservative worst-window next-N overshoot	0.0
33	merged	3	Lexicographic reach + strong tail headroom tie-break	1.2
34	merged	3	Reach + tail headroom + worst-window 1810/2560 overshoot	2.0
35	merged	3	Reach + schedule-budget headroom + budget-cliff hazard	2.2
36	merged	3	Reach + smooth-max headroom + hazard + repeat-stability	1.2
37	merged	3	Reach + smooth-max headroom + focus $1280 \to 1810 \to 2560$ + hazard	2.4
38	merged	3	Reach + smooth-max headroom + tail exponent + post-1280 cliff	1.6
39	merged	3	Reach + smooth-max log(headroom) + multi-jump hazard + censor	3.0
40	merged	3	Reach + smooth-max log(headroom) + cliff hazard + robustness	3.2
41	merged	3	Three-variant objective (balanced / hazard / robustness)	—

computational budgets used to benchmark each solver) should co-evolve with the solver designs. Initial experiments let the objective agent propose new schedules alongside new evaluation criteria.

Three problems emerged:

1. **Inconsistent evaluation.** When the schedule shifts between iterations, scores from different iterations are not directly comparable. A design that appears to improve may simply have been tested on an easier schedule, making consensus rankings unreliable.

2. **Schedule echo chamber.** The LLM agent, aware of the current top designs and existing schedules, tended to propose schedules favoring those same

designs, which is self-reinforcing bias analogous to the objective echo chamber (Sec. II B).

3. **No quality criterion.** Unlike solver designs, which can be ranked by objective performance, there is no obvious ground truth for schedule quality. The system had no reliable signal for judging whether a new schedule was more informative than its predecessor.

We resolved these issues by fixing the evaluation schedule and pairing it with rule-based multi-fidelity promotion. All designs face the same base schedule, so rankings are consistent and comparable. Computational resources are allocated by deterministic rules: the top 50% of designs by consensus advance to medium fidelity (larger N, more instances), the top 10% to high fidelity. This preserves the efficiency benefits of adaptive evaluation without the instabilities of co-evolving schedules.

Whether flexible schedule generation might help in domains where the evaluation landscape is less well-characterized remains an open question. Possible criteria for schedule quality include: (i) discriminative power (does the schedule separate good designs from bad?), (ii) predictive validity (do rankings on the schedule predict rankings under held-out test conditions?), and (iii) cost-efficiency (does the schedule achieve comparable discrimination at lower cost?). We leave these for future work.

C. Workspace merging

The search initially ran across multiple independent workspaces, each operating the full framework autonomously with its own design pool, objective history, and MCGS graph. Running in parallel increases diversity: independent workspaces explore different regions of the design space without interfering, reducing the risk that an early success monopolizes the search.

a. Selection and import. From each source workspace, the top 5 designs by consensus ranking were selected together with their *complete* parent-child genealogies. Preserving the genealogy matters: it gives the planner in the merged workspace the full evolutionary context behind each imported design, not just the final product. Roughly 200 designs were imported in total.

b. Merged workspace dynamics. Once merged, the planner can see designs from all source workspaces at once. The MCGS graph is rebuilt from the combined pool, and UCB scoring treats every design uniformly regardless of origin. This opens the door to *cross-workspace recombination*: the planner can pick parents from different source workspaces, combining innovations that evolved independently. The sub-tree structures in Fig. 3(a) of the main text are a direct imprint of this merging: each sub-tree traces back to a particular source workspace, with inter-tree edges marking cross-workspace references.

About 200 additional designs were generated in the merged workspace. Design 340, the best solver found, emerged from this phase, incorporating architectural elements that trace back to multiple source workspaces (see Sec. I B for the full lineage). Cross-workspace recombination appears to have been productive: innovations that evolved in isolation were combined in ways unlikely to have been found within any single workspace.

c. Practical considerations. Merging introduces a cold-start problem: imported designs have not been evaluated under the merged workspace's objective history, and MCGS visit counts reset to zero. The UCB parameterization partially addresses this by maintaining exploration pressure even with a large initial pool, and the consensus mechanism helps by aggregating objectives from all phases, including those created in the source workspaces.

IV. BASE SOLVER FRAMEWORK

This section describes the baseline DMM solver provided to the LLM agents and the modular code framework that supports automated solver generation.

A. Research goal and problem context

The high-level research goal given to the meta-agent is:

> *Develop an algorithm that efficiently solves hard, large-scale 3-SAT problems. Prioritize robust scaling, with focus on large-N behaviors. Aim for polynomial step vs. N scaling, ideally sub-quadratic. Common pitfall: polynomial at small N, exponentially diverging at large N.*

a. 3-SAT and the DMM relaxation. A 3-SAT instance has N Boolean variables $V_i \in \{0,1\}$ and M clauses $C_m = L_i \vee L_j \vee L_k$, where each literal L_i is either V_i or its negation. A digital MemComputing machine (DMM) [2, 3] relaxes each V_i to a continuous variable $v_i \in [-1,1]$ and introduces auxiliary *memory* variables that encode information about the dynamics' history. The sign of v_i at the fixed point gives the Boolean assignment.

b. Baseline DMM equations. The baseline dynamics (Eqs. (4)-(6) of the main text) evolve three sets of variables. The state variables v_n follow

$$\dot{v}_n = \sum_{m=1}^{M} x_{l,m}\, x_{s,m}\, G_{n,m} + (1 + \zeta\, x_{l,m})(1 - x_{s,m})\, R_{n,m}, \tag{S11}$$

where $G_{n,m}$ is a gradient term nudging variables toward clause satisfaction and $R_{n,m}$ is a rigidity term preventing satisfied literals from flipping:

$$G_{n,m} = \tfrac{1}{2}\, q_{n,m} \min\bigl[(1 - q_{j,m} v_j),\, (1 - q_{k,m} v_k)\bigr], \tag{S12}$$

$$R_{n,m} = \begin{cases} \tfrac{1}{2}\, q_{n,m}(1 - q_{n,m} v_n), & \text{if } c_m = \tfrac{1}{2}(1 - q_{n,m} v_n), \\ 0, & \text{otherwise,} \end{cases} \tag{S13}$$

with $q_{n,m} \in \{+1, -1\}$ the literal polarity and the clause cost

$$c_m = \tfrac{1}{2} \min\bigl[(1 - q_{i,m} v_i),\, (1 - q_{j,m} v_j),\, (1 - q_{k,m} v_k)\bigr]. \tag{S14}$$

Two auxiliary memory variables per clause track satisfaction history:

$$\dot{x}_{s,m} = \beta\,(x_{s,m} + \epsilon)\,(c_m - \gamma), \tag{S15}$$

$$\dot{x}_{l,m} = \alpha\,(c_m - \delta). \tag{S16}$$

The short-term switch $x_{s,m} \in [0, 1]$ toggles between "push" ($x_s \approx 1$) and "hold" ($x_s \approx 0$) modes; the long-term weight $x_{l,m} \in [1, 10^6]$ grows monotonically for persistently violated clauses. Default hyperparameters are $\alpha = 5$, $\beta = 20$, $\gamma = 0.25$, $\delta = 0.05$, $\epsilon = 10^{-3}$, and $\zeta = 10^{-3}$.

B. Modularized solver framework

The solver codebase [4] separates problem-specific dynamics from generic solving infrastructure. A single Python file fully defines each solver experiment through three components:

1. `VARIABLES_SPEC`: a dictionary mapping variable names to their initialization, shape, and bounds. For the baseline:

   - `v`: shape (B, N), bounds $[-1, 1]$, randomly initialized;
   - `xl`: shape (B, M), bounds $[1, 10^6]$, initialized to 1;
   - `xs`: shape (B, M), bounds $[0, 1]$, initialized to 0;

   where B is the batch size. The LLM can add, remove, or modify variables to introduce new memory mechanisms.

2. `HYPER_SPACE`: a dictionary defining the hyperparameter search space. Each entry specifies a type (uniform, log-uniform, integer, or categorical), default value, and bounds. The baseline defines seven parameters ($\alpha, \beta, \gamma, \delta, \epsilon, \zeta$, and the integration step size). A Bayesian optimizer (HEBO [1]) tunes these parameters for each solver design.

3. `_grad_single(vars, idx, sgn, hp)`: the core dynamics function computing per-instance gradients for all state variables. Inputs are the current variable values (`vars`), clause structure (`idx`, `sgn`), and hyperparameters (`hp`). It returns a gradient dictionary with the same keys as `vars`. This function encodes the dynamical equations (Eqs. (S11)-(S16)) and is the primary target of LLM-driven design.

a. Framework integration. The solver framework (`sat_solver.py`) dynamically imports these three components at runtime. Launching an experiment with solver ID k loads `solvers/solver_`k`.py`. The `SATSolver` class reads `VARIABLES_SPEC` to allocate and initialize state tensors, then vectorizes `_grad_single` over the batch dimension via `torch.vmap` [5]. The solving loop is standard: zero gradients, compute dynamics through the vmapped function, take an optimizer step, clamp variables to their specified bounds, check for satisfying assignments.

b. LLM-generated solvers. The designer agent produces new solver files containing modified versions of these three components. Because the interface is fixed—any file exporting `VARIABLES_SPEC`, `HYPER_SPACE`, and `_grad_single` with the correct signatures is automatically integrated—the LLM can freely redesign the dynamics, introduce new auxiliary variables, or restructure the hyperparameter space without touching framework code. This modularity is what makes the automated search described in the main text possible: each MCGS iteration generates a new solver file, evaluates it through the framework, and records the results.

V. LLM AGENT PROMPTS

This section provides the complete prompt templates used by each LLM agent in the system, organized by agent role.

A. Planner Agent

The Planner Agent analyzes research progress and assigns strategic directions to Designer Agents. It operates in two stages: first selecting promising experiments to review from MCGS rankings, then synthesizing insights into distinct, non-overlapping research directions.

a. System prompt. Sets the agent's role and scope.

```
You are the Planner Agent in an LLM-driven autonomous
research system. Guide the research direction and provide
strategic recommendations for Designer Agents.
```

b. Stage 1: Direction selection. The planner receives an overview of all experiments ranked by UCB score from MCGS and selects which experiments to examine in detail.

```
## Research Context
{main_research_context}

## Task
Select {DESIGNER_AGENT_COUNT} promising research
directions and identify experiments to review in detail.

## Database Overview
- Total experiments: {total_experiments}
- Best performance: {best_performance}
- Recent experiments: {recent_summary}

## Experiment Summaries
Up to {MAX_EXPERIMENT_SUMMARY} experiments ranked by
upper confidence bound (UCB) from Monte Carlo graph
search (MCGS):
{experiment_summaries}

## Guidelines
- Request details for up to {MAX_EXPERIMENT_DETAIL}
  experiments
- Prioritize high-UCB experiments and complementary ideas
- Avoid redundant or near-duplicate directions
- Hyperparameter tuning is automatic via HEBO optimizer -
  don't assign this to Designer Agents
- Your goal: MINIMIZE the objective function

## Output Format
‘‘‘json
{
  "lookup_experiment_ids": [/* int IDs to review */],
  "context_rationale": "Concise reasoning behind chosen
    directions and their relevance"
}
‘‘‘
```

c. Stage 2: Designer assignment. After reviewing the requested experiment details, the planner synthesizes findings into non-overlapping research directions with reference experiments for each Designer Agent.

```
## Experiment Details
{design_details}

## Task
Summarize progress and assign {DESIGNER_AGENT_COUNT}
designer agents distinct research directions with up to
{MAX_DESIGNER_REFERENCE} reference experiments each.

## Output Format
‘‘‘json
{
  "current_phase": "early_exploration|systematic_search
    |exploitation|stagnation|breakthrough_needed",
  "key_insights": ["Top takeaways explaining what works"],
  "success_patterns": [
    "Shared traits of strong experiments"],
  "failure_patterns": [
    "Shared traits of weak experiments"],
  "research_directions": [
    "Direction for d1", "Direction for d2"],
  "strategy_rationales": [
    "Rationale for d1", "Rationale for d2"],
  "focus_areas": ["themes for d1", "themes for d2"],
  "avoid_areas": ["pitfalls for d1", "pitfalls for d2"],
  "reference_design_ids": [
    [int IDs for d1], [int IDs for d2]]
}
‘‘‘

## Guidelines
- All arrays must have length {DESIGNER_AGENT_COUNT}
  and align by index
- Directions must be non-overlapping, complementary,
  and distinct
```

B. Designer Agent

The Designer Agent creates new experiment designs based on the Planner's strategic directions and reference experiments. It operates through a multi-turn conversation: first proposing a design, then receiving experiment results, and finally analyzing outcomes. If execution errors occur, an error-recovery prompt is used.

a. System prompt. Sets the agent's role and scope.

```
You are the Designer Agent in an LLM-driven autonomous
research system. You receive strategic recommendations
from the Planner Agent and create targeted experiments.
```

b. Design creation. The designer receives the research context, planner guidance, baseline components, and reference experiments, then proposes a new design with one principled modification.

```
## Research Context
{main_research_context}

## Planner Context
{planner_context}

## Architecture
- ‘domain_knowledge/{framework_module}.py‘ - Main
  framework
- ‘domain_knowledge/{baseline_filename}‘ - Baseline
  components
- ‘solvers/solver_N.py‘ - Your experiment components
- Objective: Consensus of all generated objectives
  (to MINIMIZE)
- ‘schedules/schedule_{current_schedule_id}.py‘ - Current
  experiment schedule

{framework_module}.py dynamically imports your
{num_components} components from solver_N.py.

## Baseline Components
‘‘‘python
{base_solver_code}
‘‘‘

## Reference Experiments
{reference_experiments}

## Current Objective
{objective_description}

## Current Schedule
{schedule_description}

## Task
Design a new experiment by modifying the {num_components}
core components:
1. Make ONE small, principled modification to baseline
2. Build on proven ideas from reference experiments
```

~~~
3. Follow the Planner's direction and rationale
4. Explain how changes should improve the objective

## Required Components
{component_descriptions}

Available imports: 'math', 'numpy', 'scipy', 'torch',
and standard libraries

## Output Format
Return a JSON object with:
```json
{
  "explanation": "Rationale for modification, referencing
    evidence and strategy",
  "solver_code": "Complete Python code with imports and
    components: {component_names}"
}
~~~
```

c. Experiment results. After the design is executed, the experiment results are appended to the conversation as context. No LLM response is requested at this stage.

```
## Experiment Results

You have completed experiments at **{current_fidelity}**
fidelity.

### Results
```json
{experiment_results}
```

Objective value (lower is better): {objective_value}
```

d. Result analysis. The designer analyzes outcomes and extracts actionable insights. Reference weights are computed for MCGS graph updates, reflecting how much each parent design influenced the current result.

```
## Task
Analyze results and extract actionable insights. Focus on
why the outcome occurred and what to do next.
Evaluate how much each reference design influenced this
result for Monte Carlo Graph Search (MCGS) updates.

## Success Levels
- **excellent**: Major breakthrough or validated
  improvement
- **good**: Noticeable improvement with well-understood
  cause
- **moderate**: Partial progress or useful insight
  despite limited gains
- **poor**: No improvement or regression, but still
  informative

## Reference Weights
- Include only referenced design IDs
- Each weight (0-1) represents influence on current
  design
- Weights must sum to 1.0
- Higher weights for ideas/parameters that most strongly
  shaped results

## Output Format
Return a JSON object with:
```json
{
  "short_name": "Concise descriptive title (<= 40 chars)",
  "key_insight": "Most important takeaway (1 line)",
  "success_level": "poor|moderate|good|excellent",
  "detailed_analysis": "Comprehensive explanation of
    mechanisms and outcomes",
  "comparison_to_references": "How results compare with
    referenced designs",
  "recommended_next_steps": "Concrete suggestions for
    future designs",
  "reference_weights": [
    {"design_id": int, "weight": float}
  ]
}
```
```

e. Error recovery. When a design produces runtime errors, this prompt is appended to the existing conversation so the designer can see the full context of the failed attempt.

```
## Error
```
{error_message}
{full_traceback}
```

## Task
Fix the error in your implementation components and
return all {num_components} corrected components:
{component_names}.

## Output Format
Return a JSON object with:
```json
{
  "error_summary": "What went wrong and how you fixed it",
  "solver_code": "Complete Python code with imports and
    corrected components"
}
```
```

C. Objective Agent

The Objective Agent generates proxy objective functions that guide the search. The evaluation schedule remains fixed at the baseline; only the objective function is generated. The Meta-Agent provides strategic guidance on what properties the next objective should emphasize.

a. System prompt. Sets the agent's role and scope.

```
## Role
You are the **Objective Agent**. You design proxy
objective functions that guide the discovery of better
algorithms.
```

b. Objective generation. The Objective Agent receives the current research state, Meta-Agent guidance, baseline code, existing objectives, and recent results, then proposes a new proxy objective function.

```
## Research Context
{main_research_context}

## Meta-Agent Guidance
{meta_agent_directions}
```
```

~~~
Consider this guidance when designing your objective
function. The meta-agent has analyzed research progress
and identified areas that need attention.

## Task
Generate a proxy objective function that better estimates
the research goal. The experiment schedule will use the
baseline schedule (shown below for reference).

## Discovery Philosophy
The baseline objective is deliberately simple. You have
freedom to design objectives that:
- Target any experiment or combination of all experiments
- Use any combination of available metrics
- Apply any scaling model or none at all
- Incorporate uncertainty, robustness, or other advanced
  concepts

Learn from existing results and think about what truly
matters for identifying the best algorithms.

## Baseline Code
**Objective:**
```python
{baseline_objective_code}
```

Schedules (for reference - will be used unchanged):
```python
{baseline_schedule_code}
```

Existing Objectives
{existing_objective_summary}

Recent Experiment Results
{recent_experiments_objectives_summary}

Required Function

Objective function:
```python
def objective(experiment_results):
    """Estimate the research goal from experiment results.

    Args:
        experiment_results: List of dicts with experiment
            details
    Returns:
        Float to be MINIMIZED
    """
    return objective_value
```

Experiment Interface
```python
{experiment_code}
```

Guidelines
- `experiment_results` is a list of dicts containing all
  experiment kwargs and outputs
- Objective should be smooth and friendly to Bayesian
  optimization (avoid large penalties for failures)
- Objective should adapt to different schedules and remain
  backward compatible when possible
- Available imports: `math`, `numpy`, `scipy`, `torch`,
  and standard libraries
Output Format

Return a JSON object with:
```json
{
  "objective_description": "What this objective measures
    (one line, comprehensive but extremely concise)",
  "objective_code": "Complete Python code with imports and
    objective() function"
}
```
~~~

c. Error recovery. Appended to the conversation when the generated objective function produces a runtime error.

~~~
## Error
```
{error_message}
{full_traceback}
```

## Task
Fix the error in your objective code.

## Output Format

Return a JSON object with:
```json
{
  "error_summary": "What went wrong and how you fixed it",
  "objective_code": "Corrected Python code with imports
   and objective() function"
}
```
~~~

D. Meta-Agent

The Meta-Agent oversees the entire research process. It analyzes objective function performance using Kendall tau correlations, adjusts objective weights in the consensus mechanism, and provides strategic guidance for the Objective Agent's next generation.

a. System prompt. Sets the agent's role and scope.

```
You are the **Meta-Agent**. You oversee the entire
research process, analyze what's working and what's not,
guide the objective agent, and adjust objective weights
to improve research progress.
```

b. Research analysis. The Meta-Agent receives a comprehensive view of all objective functions, their correlation structure, weighting state, and recent progress, then produces an assessment with updated weights and directions for the Objective Agent.

```
## Research Context
{main_research_context}

## High-Level Research Goal
{high_level_research_goal}

## Task
Analyze research progress, evaluate objective functions,
and provide strategic guidance.
```

~~~
Your responsibilities:
1. Assess research progress
2. **Maintain and update an evolving consensus objective
   function**
   - Objective functions are periodically generated by
     Objective Agent
   - Planner/Designer Agents/hyperparameter optimizer
     minimize the consensus objective.
   - You can adjust objective weights: amplify useful
     ones, suppress harmful ones.
3. Guide the Objective Agent in generating new objectives.

## Experiment Schedule
(for reference - will be used unchanged):
```python
{baseline_schedule_code}
~~~

## Current Objective Functions

{objective_summary_with_code}

## Objective Performance Analysis

Kendall Tau Correlation Matrix (measures agreement
between objectives):
{objective_correlation_matrix}

## Objective Weighting Mechanism

```python
weight = default_weight * weight_multiplier
# You assign weight_multiplier, default 1.0
default_weight = agreement * age_decay
agreement = max(median_tau, 0.0)
age_decay = 0.9 ** rounds_since_creation
```

Objective Agreement Summary:
{objective_agreement_summary}

Recent Progress:
{recent_progress_summary}

Top Designs:
{top_designs_summary}

## Previous Meta-Agent Guidance

{previous_meta_guidance}

## Guidelines

For Objective Evaluation:
- Objectives with low Kendall tau (disagreeing with others)
  may be misleading
- Objectives that rank failed designs highly are
  problematic
- Objectives that don't differentiate between designs are
  not useful
- Consider whether the objective aligns with the
  high-level research goal

## Output Format

Return a JSON object:
```json
{
  "research_assessment": "Overall assessment of research
    progress (2-4 sentences)",
  "research_phase": "exploring|converging|stuck
    |breakthrough_needed|refining",
  "objective_analysis": [
    {
      "objective_id": 0,
      "assessment": "How this objective is performing",
      "weight_multiplier": 1.0
    }
  ],
  "objective_directions": "Strategic guidance for next
    objective generation (be specific)"
}
```
```

[1] A. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R.-R. Griffiths, A. Maravel, J. Hao, J. Wang, J. Peters, and H. Bou Ammar, Hebo: Pushing the limits of sample-efficient hyperparameter optimisation, Journal of Artificial Intelligence Research **74** (2022).

[2] M. Di Ventra, *MemComputing: Fundamentals and Applications* (Oxford University Press, 2022).

[3] S. R. B. Bearden, Y. R. Pei, and M. Di Ventra, Efficient solution of Boolean satisfiability problems with digital memcomputing, Scientific Reports **10**, 19741 (2020).

[4] Y.-H. Zhang, Github repository: Scientific reserch as meta-optimization, Available at: https://github.com/yuanhangzhang98/LLM_meta_optimization (2026).

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in *Advances in Neural Information Processing Systems*, Vol. 32 (Curran Associates, Inc., 2019).
```