OPTEVOLVE: HARDWARE-AWARE DISCOVERY OF CERTIFIED OPTIMIZATION ALGORITHMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Canonical optimization algorithms each excel in distinct regimes, yet identifying the optimal combination of formulation, algorithm, and hardware strategy for a target instance remains non-obvious. While recent algorithmic discovery frameworks utilize LLM-guided evolutionary search, naive source-code mutation fails to provide structural guarantees of mathematical convergence. We introduce **OptEvolve**, an automated framework for optimization modeling and verified solver synthesis. OptEvolve synthesizes the optimal solver for a target problem instance by co-optimizing mathematical formulations, algorithmic schemes, and hardware-level operator implementations. Solvers are evolved as typed genomes in a monotone-operator calculus, using a symbolic gate to discard invalid candidates prior to compilation while admitting custom operator kernels under strict behavioral contracts. An empirical cost model guides the search across hyperparameters and stopping criteria, while optimizing GPU iteration fusion tiles on target devices. OptEvolve matches or beats domain-specialist solvers on structured problems and outperforms general-purpose baselines where no specialist exists. Furthermore, its targeted CUDA fusion tiles outperform XLA by up to $2/7\times$, and the framework generalizes effectively across unseen, held-out QP, LP, and SOCP problems.

1 INTRODUCTION

Selecting and tuning the most suitable algorithm or solver for a class of optimization problems is an extensive and non-obvious task. Classical optimization algorithms require trial and error across numerous mathematical formulations, yet black box deep learning methods lack interpretability and guaranteed verification. The challenge in selecting the best optimization method for a target task on specific hardware is therefore daunting, especially in large scale applications where iteration is costly. However reliance on neural networks alone reduce trust and are heavily training data dependent, which is challenging in applications such as the medical field. Automating solver design is therefore a complex yet valuable task, and despite the many progressive advancements which span PDHG Chambolle & Pock (2011) to Enhanced PDLP Lu & Zhang (2025), a comprehensive automated optimization algorithm discovery framework remains difficult.

In this work, we present an evolutionary automated framework for optimization algorithm discovery with strictly convergence by design guarantees. Our key insight is that although evolutionary deep learning frameworks have proven sufficient in large scale combinatorial discovery, they struggle in guaranteed mathematical verification and correctness. Yet many optimization techniques which seem to target different problem families at first glance, can be viewed as sharing a common set of low level fundamental operations. For example, a splitting method can handle parts of an objective function through simpler updates, such as a gradient step and projection onto a constraint set. Their composition is an operator mapping the solver's current state to its next state. Choosing that composition changes both convergence conditions and work per iteration. Since this class of optimization problems has wide application through numerous fields, specialized methods have combined primal dual splitting with preconditioning, adaptive steps, and restarts to target specific problem instances. In contrast, RLQP Ichnowski et al. (2021) learns quadratic program (QP) penalty updates via reinforcement learning, and HPR-LP combines another splitting scheme with adaptive policies (Applegate et al., 2021; Ichnowski et al., 2021; Chen et al., 2024). Recent LLM-driven discovery, notably FunSearch and AlphaEvolve, propose further designs for automating discovery

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.

(Romera-Paredes et al., 2024; Novikov et al., 2025). Yet algorithmic convergence and verification is not guaranteed.

We introduce **OptEvolve**, the convergent by design framework.[1] Given an input problem family structure, training dataset instances, a target accuracy, and available hardware devices, OptEvolve outputs solver algorithms and implementations that are targeted for user specific problem classes. A typed genome specifies a splitting scheme, parameters, solution-preserving formulation, projection implementation, and CPU or GPU backend with suitable kernel optimization. Before timing a candidate, a symbolic gate checks sufficient convergence conditions for five exact base schemes and admissible relaxations. Synthesized projections and kernels undergo separate finite numerical tests. Our theoretical analysis covers the stated exact splitting-iteration grammar, and our empirical results demonstrate remarkable generalization across unexpected problem instances, for given input problem classes.

Contributions.

- **Convergent by Design.** Every admitted and discovered scheme in the exact splitting-iteration grammar must converge: the gate checks sufficient construction conditions *before* execution (Theorem 1). Numerical implementations are tested separately.
- **Joint solver and hardware search.** The evolutionary framework evolves schemes and parameters with solution-preserving formulations, contract-tested projection implementations, and measured per-device costs. Tested synthesized projections meet the target on $9/9$ unseen instances versus $5/9$ for an unchecked faster implementation (Section 5.2). The best fused GPU tile changes across devices (Section 5.4).
- **End-to-end gains and their boundary.** On budget-constrained least squares at $n = 48{,}000$, with no packaged specialist for this family, a searched formulation and projection give a $25.22\times$ faster cold solve at 10^{-4} than the fastest accepted engine at shipped defaults (Appendix G.7). A separate $n = 12{,}000$ ablation shows that formulation accounts for most of this family's gain, as one example across a broad family of test problems.
- **OptEvolve-Bench protocol.** We release the OptEvolve-Bench protocol and measured field. Each result faces the same original-problem acceptance check rather than a solver's own stopping rule, comparisons include recognized structure-specific specialists, and the finest accepted tolerance is reported beside time. The release pairs six application classes at packaged-solver defaults with 806 acceptance records over 14 engines on Netlib, Maros–Mészáros, and generated LP, QP, and conic instances. The accuracy check changes the wall-clock verdict on three of the six application classes (Section 5.3).

2 Related Work

Mathematical checking. AutoOPT is the closest optimization-discovery comparison. Numerical BnB–PEP design precedes LLM derivation of analytic methods and proofs, Lean verification, and human interpretation (Kim et al., 2026). Its formal proof checks are stronger than ours, while its staged proof workflow and fixed-step smooth-minimization objective differ from a pre-execution gate inside a population loop priced by time to accepted answers on target hardware. Proof-assistant approaches such as CvxLean likewise address a stronger verification goal (Bentkamp et al., 2023). OptEvolve checks sufficient mathematical side conditions for exact splitting iterations, it does not machine-check generated floating-point code. Hawkeye uses architecture-specific tests for generated GPU kernels (Tschand et al., 2026); our measured backend choices also affect solver formulation and routing.

Solver design and selection. Generic engines in the comparison field include Clarabel, PIQP, OSQP, SCS, ECOS, and HiGHS (Goulart & Chen, 2024; Schwan et al., 2023; Stellato et al., 2020; O'Donoghue et al., 2016; Domahidi et al., 2013; Huangfu & Hall, 2018; HiGHS, 2026). PDLP and HPR-LP combine splitting with preconditioning, adaptive parameters, or restarts (Applegate et al., 2021; Chen et al., 2024); RLQP learns penalty updates for a QP solver (Ichnowski et al.,

[1] code, genomes and reproduction scripts: `https://anonymous.4open.science/r/OptEvolve-8104/README.md`

2021). Their adaptive policies are not certified by our gate. Solver selection itself has precedents: Hydra configures complementary portfolio members, and formulation selection has been learned for mixed-integer quadratic programming (Xu et al., 2010; Bonami et al., 2022). OptEvolve differs by expanding a portfolio across scheme parameters, solution-preserving formulations, tested projection implementations, and hardware choices, then pricing evolved candidates against existing engines. Discussion on related work continues in Appendix B.

3 METHODOLOGY

Figure 1 shows how candidate generation, mathematical and numerical checks, measured costs, and routing interact. The checks have different scopes: a scheme certificate concerns an exact iteration; a projection contract screens generated code; the final check accepts a point on the original problem.

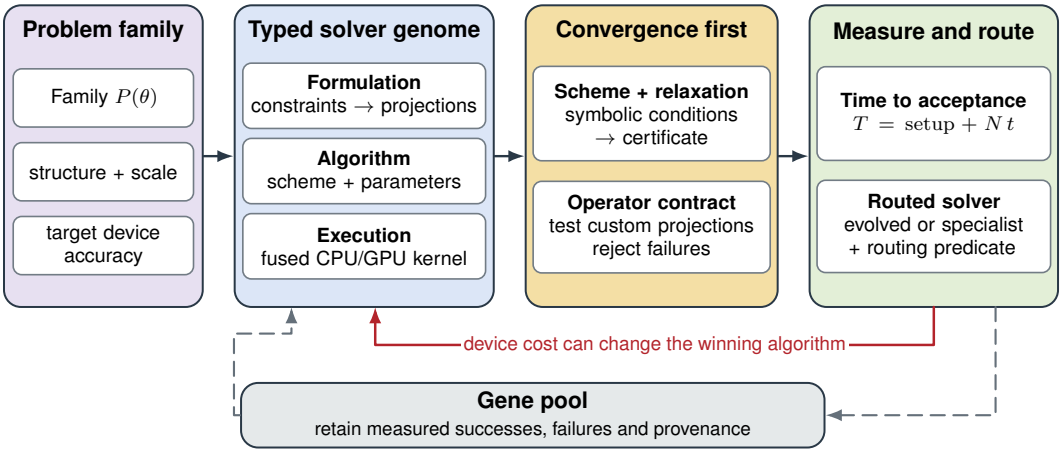

Figure 1: OptEvolve searches typed solver designs for a problem family. The scheme gate checks convergence conditions before evaluation; generated projections pass separate contract tests. Measured device costs and outcomes feed back into search, while accepted solves determine the route.

Canonical problem class. A family supplies matrix-vector and adjoint operators, block metadata, training instances, a target instance and accuracy, and available devices. OptEvolve retains reusable candidates and costs; on the target it returns a route and a point that passes the original-problem check (Section 5.1), or reports no answer within budget. Its composite convex model is

$$\min_x f(x) + g(x) + h(Lx), \tag{1}$$

where f has a Lipschitz gradient, g, h have computable proximal steps, and L is a bounded linear operator with computable adjoint L^* and a known norm bound $\ell \geq \|L\|$. A proximal step minimizes $g(u) + \|u - v\|^2/(2\gamma)$; for a constraint indicator it projects onto the feasible set. An iteration operator T maps the current (possibly primal–dual) state to the next. Splitting constructs T from updates for the separate terms (Ryu & Yin, 2022).

Solvers as typed genomes. A genome specifies a base scheme (forward–backward, Douglas–Rachford, Davis–Yin, primal–dual hybrid gradient, or Condat–Vũ), stepsizes, and optional relaxation (Chambolle & Pock, 2011; Condat, 2013; Vũ, 2013). Associated genes select a solution-preserving formulation, projection implementation, and backend. Canonical serialization deduplicates proposals; typed mutations cannot silently change the mathematical meaning of a field. The certified grammar is defined in Section 4. The gate certifies base schemes and relaxation, not arbitrary blends, adaptive policies, or free-form switching. Conditional blend and safeguarded-step rules appear in Appendix A; they are not used in the reported experiments.

Before executing a scheme, the gate checks its stepsize, operator-norm, and relaxation inequalities (Table 3) and records the resulting exact-iteration construction certificate. Invalid proposals return the

violated condition to the proposer rather than being silently clamped. The gate assumes an admissible formulation and exact proximal maps; it is not a proof of floating-point code.

Compilation and certificates. An admitted exact scheme is compiled as a JAX iteration. The construction certificate establishes convergence under Section 4's assumptions. Runtime residuals and, where available, duality gaps monitor the solve; the reported point must independently pass the original-problem acceptance check. Synthesized numerical projections are run first inside a contract harness, not admitted by the symbolic scheme gate. Appendix F details these distinct checks.

Hardware-aware and formulation genes. The backend genes choose precision, batching, and per-device execution, including a temporally fused GPU tile or an unfused loop. Candidate kernels are checked against reference updates; the exact-operator certificate does not cover their floating-point behavior. Calibration prices each tile on its target device rather than transferring a winner across GPUs (Appendix F). Formulations can equilibrate rows or move recognized constraints from $h(Lx)$ into the projection in g, changing iteration count and cost without changing the underlying solution. Relocation requires a valid recovery map, a new bound for the transformed operator, and an exact mathematical projection; only then can the scheme certificate transfer (Proposition 2, Appendix A).

Operator synthesis under contract. When projection time limits a promising route, an LLM proposes a faster implementation for a specified set and device. Before a proposal is used in a solve, a contract harness checks it against a reference projection on tested inputs, including degeneracies. Passing finite tests does not prove equality to the exact projection for all inputs. Admitted implementations are timed on each device (Appendix F).

The gene pool and the cost model. An append-only gene pool stores accepted designs, measurements, and rejection feedback. The router estimates $T = \text{setup} + N(\text{formulation})\, t(\text{operator}, \text{device})$, using measured iteration counts and calibrated per-device costs, not an operator-norm proxy. It considers evolved candidates and recognized generic or structure-specific engines; unmeasured choices are explored within budget. Every returned answer passes the same original-problem check, regardless of route.

3.1 Evolutionary loop

The LLM proposer receives archived results, a mutation menu, and the genome schema, and a random proposer controls for typed-space search at the same evaluation budget. Candidates are admitted before timing, scored by training time to an accepted answer, and selected on held-out instances. Algorithm 1 ties the search and target routing together; Appendix F gives proposal and fitness details.

Algorithm 1 OptEvolve: search, certify, measure, and route.

Require: Problem family, training and target instances, accuracy, devices
Ensure: Reusable gene pool; accepted point and route, or no answer
while evaluation budget remains **do**
  Propose a typed scheme, formulation, operator, or backend edit.
  Check the exact scheme and formulation; contract-test generated code.
  if rejected **then**
    Record the reason for the next proposal.
  else
    Compile, time accepted training solves, and update measured costs.
  end if
end while
Include recognized specialists; explore unmeasured routes within budget.
return a route and an original-problem-accepted point, or no answer.

Implementation. JAX compiles the admitted iterations. Search uses Qwen2.5-Coder-1.5B-Instruct for genome proposals and Qwen3.8-27B for projection synthesis; timing is on an RTX 4090 workstation and an H100 node, with CPU specialists run on the corresponding host. Machine, cache, and timing details appear in Appendix F and Section 5.1.

3.2 AN EXACT-PATH LASSO SOLVER AS A SEARCH-SPACE EXAMPLE

Reachability is not rediscovery: the expert's decisions must be expressible as mutations, and the fitness must make their effects visible. An exact-path lasso solver illustrates this design. Each of the four decisions in Table 1 becomes a mutation of one layer.

Table 1: The four decisive steps of an expert derivation, each reachable as a single typed mutation.

Step	Layer patched	The mutation	Gate rule that keeps it honest	Why the fitness sees it
D3 carry the factorization	3 state	`factorization.update`: refactorize to insert_delete	drift check on the maintained factor	per-step cost, visible immediately
D4 free screening bound	4 restrict	`rule`: none to equiangular_bound	admitted only if provably safe, or paired with a full re-admission pass, checked statically	step-count identity proves exactness, so the win is pure
D5 hybridize	5 handoff	`handoff`: null to {to: working_set, predicate}	the existing switch_k rule generalized: exact prefix exempt, tail passes the full gate	neither engine dominates, so only a sweep shows it
D6 measure, do not bound	5 handoff	`predicate.source`: bound to counted	none needed; it is a source swap	only oracle regret on a sweep sees it

The D6 decision illustrates why fitness matters. The gene value $\texttt{source} \in \{\text{bound}, \text{fitted}, \text{probe}, \text{counted}\}$ is an enumeration, so swapping it is a legal single mutation. Under a wall-clock fitness, the negated shifted geometric mean of time-to-certificate over a training set, `bound` looks fine, because it is calibrated on the instance where the bound is tight. Under regret against a per-problem oracle across 54 shape-and-grid combinations at two thread counts, `bound` costs $14\times$ to $19\times$ and `counted` costs $1.10\times$ with a worst case of $1.64\times$. The same mutation gets opposite verdicts under the two fitness measures; the chosen fitness thus determines which predicate a search would favor.

The example shows how an expert's four-step derivation can be expressed as four single mutations; it does not show that a search run rediscovered them. An ablation of that discovery claim would withhold the four decisive gene-pool cards, compare LLM mutation without a gene pool, and run a random proposer over the same typed space. SimpleTES (Ye et al., 2026) is another algorithm-discovery system with a lasso-path solver and offers an external comparison target.

3.3 AUTOMATIC HARDWARE KERNEL FUSION

Fusion is layer 8, and layer 8 is not a post-processing step applied to the winning algorithm. It is one of three hardware genes, all set from the same quantity, the work per step of whatever inner loop layers 1 through 6 produced.

The fusion tile and the thread count are the same gene at two granularities. Both answer how wide the parallel region should be relative to the work the algorithm gives it, both are read off the device profile, and both have a cliff when set by an environment default instead of by the algorithm. That is what makes the RTX 4090 against H100 tile result and the 16 against 64 thread result one finding rather than two anecdotes.

Algorithm 1 closes the loop by feeding the measured $t(\text{operator}, \text{device})$ back into the layer-5 routing predicate. A better fused tile lowers t for an iterative candidate and moves the crossover at which it beats another algorithm. Hardware choice therefore changes the route, not just the cost of a fixed solver (Equation 8).

Table 2: The three hardware genes. Each has a GPU form and a CPU form, and each has a measured cliff, which is what distinguishes a gene from an environment default.

Gene	Question	GPU form	CPU form	The cliff that proves it is a gene
7 admission	cross the device at all?	streamed bytes against the bandwidth difference, decided before touching the runtime	stay	importing the runtime costs more than a small solve
8 width	how much work per launch or barrier?	fusion tile: T iterations per launch on a $(B+2T)^2$ tile with halo in shared memory	BLAS thread count from work per step	the wrong device's tile costs $1.06\times$ to $1.63\times$; 64 threads instead of 16 costs $103\times$
6 contract	what precision, and where	per-expression fastmath scope	same	fastmath on a Schur complement turned a 10^{-14} certificate into 10^{-2}

4 Convergence by construction

For problem (1), assume finite-dimensional spaces, convex f with L_f-Lipschitz gradient, closed proper convex g, h, a nonempty saddle set, exact required proximal maps, and a supplied $\ell \geq \|L\|$. For a positive-definite metric H, an operator is θ-averaged if $T = (1-\theta)I + \theta S$, with S nonexpansive in $\|\cdot\|_H$ and $0 < \theta < 1$ (Ryu & Yin, 2022). Let $\mathcal{X}^\star$ be the primal solution set and $\operatorname{dist}_H(z, C) = \min_{c \in C} \|z - c\|_H$.

Definition 1 (Construction certificate)**.** A certificate for the exact iteration T_G is (H_G, θ_G, π_G) with $H_G \succ 0$ and (C1) T_G θ_G-averaged in $\|\cdot\|_{H_G}$, (C2) $\operatorname{Fix} T_G \neq \emptyset$, and (C3) $\pi_G(\operatorname{Fix} T_G) \subseteq \mathcal{X}^\star$. The decoder π_G maps a fixed point to a solution.

The gate admits the grammar

$$G ::= \operatorname{base}_s(p) \mid \operatorname{relax}_\rho(G'), \qquad s \in \{\text{fbs}, \text{drs}, \text{dys}, \text{pdhg}, \text{cv}\}, \tag{2}$$

where p denotes stepsizes and $\rho > 0$ relaxes the base iteration. The fixed metric may differ between genomes but not across steps of one solve. Table 3 specifies the sufficient gate conditions; additional formulation, numerical, and device choices are not terms of this grammar.

Theorem 1 (Gate soundness)**.** For an admissible formulation under the assumptions above, every G generated by (2) and accepted by the gate carries a construction certificate for its *exact* iteration. For a base scheme, Table 3 gives H_G and $\theta_G = 1/(2 - c_G)$; for relaxation, the gate requires $\rho\theta_{G'} < 1$ and yields $(H_{G'}, \rho\theta_{G'}, \pi_{G'})$.

The proof is induction over the grammar, using the base-scheme and relaxation rules in Appendix A.

base scheme	metric H_G	gate condition	c_G
FBS	I	$0 < \alpha < 2/L_f$	$\alpha L_f/2$
DRS	I	$\alpha > 0$	0
Davis–Yin	I	$0 < \alpha < 2/L_f$	$\alpha L_f/2$
PDHG	$M_{\tau\sigma}$	f affine, $\tau\sigma\ell^2 < 1$	0
Condat–Vũ	$M_{\tau\sigma}$	$\tau L_f/2 + \tau\sigma\ell^2 < 1$	$\dfrac{\tau L_f}{2(1 - \tau\sigma\ell^2)}$

Table 3: Base-scheme conditions; $M_{\tau\sigma} = \left[\begin{smallmatrix} \tau^{-1}I & -L^\top \\ -L & \sigma^{-1}I \end{smallmatrix}\right]$, $\ell \geq \|L\|$. Positive stepsizes and each stated inequality ensure a valid metric and $c_G < 1$. The relaxation gate also checks $\rho\theta_G < 1$.

Corollary 1 (Convergence and residual rate)**.** For certified T_G, the exact iterates $z_{k+1} = T_G z_k$ converge from any initial state to a fixed point decoding to a solution. Their step residual is

nonincreasing and satisfies

$$\|z_{k+1} - z_k\|_{H_G}^2 \leq \frac{\theta_G}{(1-\theta_G)(k+1)} \operatorname{dist}_{H_G}^2(z_0, \operatorname{Fix} T_G). \tag{3}$$

This follows from averagedness and the fixed-point correspondence (Appendix A). The rate concerns the exact step residual, not an objective-gap guarantee for floating-point code.

Remark 1 (The search can spend certified relaxation slack). The base certificate permits $0 < \rho < 2 - c_G$. For Condat–Vũ the evolved TV parameters $(\tau, \sigma, \rho) = (0.05, 1.0, 1.9)$ satisfy this condition: a smaller primal step widens the admissible relaxation range. Appendix E compares this window with the classical condition.

A formulation must additionally preserve solutions and pass the gate with its transformed operator bound (Appendix A). Finite projection and kernel tests do not establish the required exact-atom assumption on all inputs; a returned numerical point instead faces the independent acceptance check of Section 5.1. Halpern and safeguarded-step guarantees, neither used in the reported results, are detailed in Appendix E.

5 Experiments

Section 5.1 and the six classes of Section 5.3 address (i), whether a comparison can be trusted, by grading every entrant with the same check and completing the field with the structure-specific specialist. Section 5.2 addresses (ii), by separating what the gate admits from what finite tests screen. Sections 5.3 through G.8 address (iii), by varying formulation, algorithm and device together rather than one at a time. Results on the generated families, including the routes the system chose and the three it misjudged, are in Appendix G.7. Detailed settings, per-device sweeps, and further losses are in Appendix G.

5.1 Evaluation protocol

Every route and competitor must satisfy the same check on the *original* problem: relative constraint violation and objective gap, normalized by $1 + |f^\star|$, are each at most the target tolerance; $f^\star$ is a high-accuracy reference computed outside timing. Where $f^\star$ is numerical rather than a proven bound, this is an independent empirical acceptance check, not a formal optimality certificate. The comparison field includes generic LP/QP/conic engines and recognized specialists such as coordinate descent for lasso, LIBLINEAR/LIBSVM for SVM, and network simplex for transport. Each engine runs at its shipped default settings, so the comparison concerns out-of-the-box performance at the requested accuracy. Head-to-head timings include detection, formulation, compilation, and solve in a fresh process with compilation cache disabled, except CUDA context creation; the comparison engines likewise use fresh processes. We report two quantities against this check. *Accepted time* is the wall clock to the first point that passes it, with a failure charged the full budget. *Reachability* is the finest tolerance a solver passes it at, over every setting that solver was offered, or none if there is no such tolerance. The two are independent: a solver can be fastest to a loose target and have no reachability at a tight one, which is what happens in three of the six classes of Section 5.3. Reporting time alone hides that, which is why both appear in every table. Held-out routing has a 20 s budget, with exploration charged on first visits. Machines and warm-versus-cold conventions are specified in Appendix G.

5.2 What evolution discovers

On anisotropic total-variation denoising of held-out retinal OCT patches, the certified evolved Condat–Vũ scheme takes 0.073 s geometric-mean time to the target duality gap, against 0.621 s for its certified textbook default. Random and LLM proposals tie on held-out time, so this is an internal scheme-tuning result, not evidence that the LLM beats random search or an external TV specialist. On box–hyperplane projection, a contract-tested synthesized implementation is $3.16\times$ faster per call than classical breakpoint search on the RTX 4090; the still-faster unchecked LLM implementation passes only $1/17$ contract clauses. Solvers using the tested implementation meet the target on $9/9$ unseen instances versus $5/9$ for the unchecked one. The contracts screen numerical code, not prove its exactness. Full ablations and per-call times are in Appendix G.

5.3 SIX PROBLEM CLASSES AGAINST STANDARD SOLVERS IN THEIR SPECIALIZED DOMAINS

The following six problem classes are derived from published benchmarks and clinical data. This distinguishes synthetic data results from real world application and efficacy of of our framework.

Table 4: Six application classes under the independent original-problem acceptance check of Section 5.1; baselines run at shipped defaults.

class	data	ours	baselines, shipped defaults	margin
Simplex-constrained least squares	AVIRIS Cuprite, $188 \times 47{,}750$, 498-signature library	10^{-4} in $\mathbf{53.3}$ s (H100), 165.3 s (4090)	SUnSAL 9,000 s measured, worst $1.19\text{e}{-3}$, **never**; per-column rates scaled to batch: Clarabel 30,379 s, ECOS 22,864 s, SCS 5,654 s never, OSQP 14,117 s never	$\mathbf{570}\times$ (scaled rate)
Cell-type deconvolution	LM22 (547×22) and TIL10, 10,000 samples	10^{-8} in $\mathbf{1.31}$ s for all 10,000, RMSE 0.0051	CVXPY-Clarabel 4.73 s for 300 samples, 0.0158 s per sample (scaled per-column rate); `scipy.nnls` violates sum-to-one by $2.2\text{e}{-2}$	$\mathbf{120}\times$ per sample
Isotropic TV denoising	DIV2K; clinical OCT; clinical CT	OCT 256^2: 10^{-4} in $\mathbf{0.16}$ s, 10^{-8} in 10.16 s	scikit-image 0.023 s but gap $1.7\text{e}{-2}$: reaches **no target at any size**	reachability
Isotropic TV deblurring	DIV2K, 128^2 to 1024^2	10^{-3} in $\mathbf{0.74}$ s (128^2) and $\mathbf{0.57}$ s (256^2)	no packaged solver exists; CVXPY-SCS 118.6 s and 1,063.9 s, landing at $8.4\text{e}{-3}$ and $1.6\text{e}{-3}$, so it reaches **neither** target	$> 160\times$, $> \mathbf{1{,}870}\times$
TV-Poisson deconvolution	Poisson counts, peak 100 photons	256^2: 10^{-8} in $\mathbf{2.87}$ s	no packaged solver; `richardson_lucy` solves a different objective	Not Applicable
Batched QP layers	OptNet form, n=64..256, B=1024..16,384	$\mathbf{0.0029}$ s warm (n=64); 0.0253 s (n=256)	`qpth` 0.188 s and 2.372 s, both at machine precision	$65\times$ to $\mathbf{94}\times$

Formulation decisions dominate kernel speed. On the unmixing and deconvolution classes the constraint set is a simplex, so relocating it into g leaves an f whose proximal map is *one* inverse shared by all $47{,}750$ pixels or $10{,}000$ samples; a first-order method on the same problem cannot use that factorization and needed more than 10^5 iterations where the relocated form needs thousands. On the batched QP layers the same move shares one factorization across the batch where an interior-point method rebuilds a KKT system per element, which is why the margin *grows* with dimension, $65\times$ at $n = 64$ to $94\times$ at $n = 256$. On the Poisson class the formulation is forced rather than chosen: the data term has no Lipschitz gradient, so the smooth-plus-proximable split is inapplicable and the term must go behind the linear map.

For example, the lasso regularization path is piecewise linear in λ, so an exact traversal returns every requested grid point inside a segment by interpolation rather than by solving. Four decisions carry it. The factorization of $X_A^\top X_A$ is maintained across the whole path by Cholesky update and downdate rather than refactorized, drifting 9.5×10^{-16} over 300 interleaved insert and delete operations. The entering-variable scan is screened by a bound that is free because it reuses $G_{AA}d = s$, the equation that defines the direction, and is exact: the traversal visits the identical sequence of breakpoints, asserted on step counts. The traversal hands its tail to a working-set engine at a predicate obtained by writing out both costs, where n and p cancel and the decision reduces to the active-set size against a multiple of the grid size. And that active-set size is obtained by counting variables as they enter rather than by bounding the support with $\min(n, p)$.

5.4 HELD-OUT GENERALIZATION AND SYSTEM ABLATIONS

Across 28 held-out QP, LP, and SOCP instances at 10^{-3}, the full system solves 27 within 20 s, versus 26 for a best-per-instance oracle restricted to the completed field of *existing competitor engines* (excluding evolved candidates), 25 for its strongest single engine, and 18 for an LLM-only evolved operator in a fixed pipeline. The LLM-only arm uses the same LLM and search engine, but lacks

the scheme gate, gene pool, and router; these components are not individually isolated by that comparison. The router without an agent session solves 26. On 25 instances both the full system and the existing-engine oracle solve, repeat visits have a $1.29\times$ geometric-mean speedup, while first visits have $0.035\times$ because they pay for exploration. Table 5 gives per-arm means over each arm's own solved instances; the $1.29\times$ and $0.035\times$ comparisons above use only jointly solved instances.

	solved (of 28)	geo. first visit (s)	geo. repeat (s)
full system (LLM agent + gene pool + router)	**27**	1.204	0.0374
router without the agent session	26	1.163	0.0284
LLM-only evolved operator, fixed pipeline	18	1.066	0.8625
best-per-instance existing-engine oracle	26	–	0.0441
best single engine (Clarabel)	25	–	0.0839

Table 5: Held-out solve rate at tolerance 10^{-3} with a 20 s budget, RTX 4090. The oracle chooses among completed existing engines, excluding evolved candidates. Geometric means are over each row's own solved instances.

Three GPU architectures. The temporally fused kernel was swept on three devices spanning consumer and datacenter parts. The number of admissible tiles tracks opt-in shared memory per block monotonically ($101{,}376$, $\approx 163{,}840$ and $232{,}448$ bytes admit 17, 23 and 24 of the 21-to-24 candidate tiles), which is the mechanism rather than a correlation. The *choice*, however, is not monotone: the A100 sits between the other two on both shared memory and float64 rate yet takes the largest gain at every size, and it disagrees with the H100 at three of four sizes as well as with the RTX 4090 at all four. A selector that read device capability as a scalar would get this wrong.

The RTX 4090 never selects $T = 8$ and both datacenter parts select it almost everywhere, because buying HBM traffic with $2.25\times$ to $4.00\times$ halo recomputation is affordable only where float64 is not rate limited. In this later four-size three-device sweep, the RTX 4090 at size 128 has a $2.01\times$ hand-written single-step gain over XLA and a further $1.619\times$ fusion gain over that CUDA step, giving about $3.25\times$ relative to XLA. This is a kernel-loop comparison, with more results provided in Appendix D.

6 Conclusion and Future Work

OptEvolve searches reusable convex-solver designs and routes instances by measured time to an independently accepted answer on target hardware. Its pre-execution gate checks sufficient convergence conditions for exact splitting iterations; finite tests screen generated numerical code; and an original-problem check accepts returned points. These checks have different scopes.

On budget-constrained least squares at $n = 48{,}000$, where no packaged specialist exists, formulation relocation and a synthesized projection produce a $25.22\times$ faster cold solve at 10^{-4} than the fastest accepted engine at shipped defaults. An ablation at $n = 12{,}000$ shows that formulation accounts for most of this family's gain. Across the held-out field, routing solves $27/28$ instances within budget versus $26/28$ for a best-per-instance oracle limited to existing competitor engines. In three of six application classes, a baseline returns a point that fails the common accuracy check; GPU tile choices also differ by device. These results support joint mathematical and hardware-aware solver search for reusable workloads, while specialists win in other regimes and first-visit exploration remains costly.

Future work. Reducing first-visit cost through reusable cost estimates and more selective candidate evaluation could make search practical for less frequently repeated workloads. A complementary direction is to develop error-aware certificates for synthesized projections and finite-precision kernels, narrowing the gap between convergence guarantees for exact iterations and the behavior of their implementations.

Reproducibility Statement

The anonymized code, reproduction scripts, solver genomes and certificates, configurations, seeds, and data manifests are available at https://anonymous.4open.science/r/OptEvolve-8104/README.md. Archived genomes allow rerunning the reported solvers without regenerating them; an independent LLM search may produce different proposals.

Ethics Statement

We followed the ICLR Code of Ethics. No new patient data or human-participant experiments were collected. Retinal OCT images come from a previously released dataset with randomized patient identifiers; we measure denoising computation, not diagnostic accuracy. A solver's convergence certificate does not establish clinical safety; consequential uses require independent application-specific validation.

AI Use Statement

Generative AI assisted with refining writing, feedback on methods and experiments, code editing, identifying gaps, keywords, and interpretation of the executable-loop experiments. It was not used to generate synthetic data or provide critical proof ingredients. Translation, data cleaning or reformatting, and qualitative or thematic data analysis were not applicable. We independently checked cited sources, reviewed and tested assisted code, and take responsibility for all content and artifacts.

References

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

David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient, 2021. URL https://arxiv.org/abs/2106.04756.

Jean-Bernard Baillon and Georges Haddad. Quelques propriétés des opérateurs angle-bornés et n-cycliquement monotones. *Israel Journal of Mathematics*, 26(2):137–150, 1977.

Heinz H. Bauschke and Patrick L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. CMS Books in Mathematics. Springer, 2nd edition, 2017.

Alexander Bentkamp, Ramon Fernández Mir, and Jeremy Avigad. Verified reductions for optimization. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 74–92. Springer, 2023. doi: 10.1007/978-3-031-30820-8_8. URL https://doi.org/10.1007/978-3-031-30820-8_8.

Quentin Bertrand, Quentin Klopfenstein, Pierre-Antoine Bannier, Gauthier Gidel, and Mathurin Massias. Beyond L1: Faster and better sparse models with skglm. In *Advances in Neural Information Processing Systems*, 2022.

Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX. *Operations Research*, 70(6):3303–3320, 2022.

Nicolas Bonneel, Michiel van de Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using Lagrangian mass transport. *ACM Transactions on Graphics*, 30(6):158:1–158:12, 2011.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

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

Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011. doi: 10.1007/s10851-010-0251-1.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011.

Kaihuang Chen, Defeng Sun, Yancheng Yuan, Guojun Zhang, and Xinyuan Zhao. HPR-LP: An implementation of an HPR method for solving linear programming, 2024. URL https://arxiv.org/abs/2408.12179. arXiv:2408.12179.

Patrick L. Combettes. Quasi-Fejérian analysis of some optimization algorithms. In *Inherently Parallel Algorithms in Feasibility and Optimization and Their Applications*, volume 8 of *Studies in Computational Mathematics*, pp. 115–152. Elsevier, 2001.

Patrick L. Combettes. Solving monotone inclusions via compositions of nonexpansive averaged operators. *Optimization*, 53(5–6):475–504, 2004.

Patrick L. Combettes and Bằng Công Vũ. Variable metric forward-backward splitting with applications to monotone inclusions in duality. *Optimization*, 63(9):1289–1318, 2014.

Patrick L. Combettes and Isao Yamada. Compositions and convex combinations of averaged nonexpansive operators. *Journal of Mathematical Analysis and Applications*, 425(1):55–70, 2015.

Laurent Condat. A primal-dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms. *Journal of Optimization Theory and Applications*, 158(2): 460–479, 2013. doi: 10.1007/s10957-012-0245-9.

Damek Davis and Wotao Yin. A three-operator splitting scheme and its optimization applications. *Set-Valued and Variational Analysis*, 25(4):829–858, 2017.

Alexander Domahidi, Eric Chu, and Stephen Boyd. ECOS: An SOCP solver for embedded systems. In *2013 European Control Conference (ECC)*, pp. 3071–3076. IEEE, July 2013. doi: 10.23919/ecc.2013.6669541. URL http://dx.doi.org/10.23919/ECC.2013.6669541.

Jonathan Eckstein and Dimitri P. Bertsekas. On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1–3): 293–318, 1992.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, et al. POT: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi: 10.1007/s12532-020-00194-3. URL https://doi.org/10.1007/s12532-020-00194-3.

Paul J. Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives, 2024. URL https://arxiv.org/abs/2405.12762.

Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, Apr 2015. ISSN 1573-2894. doi: 10.1007/s10589-014-9687-3. URL http://dx.doi.org/10.1007/s10589-014-9687-3.

Benjamin Halpern. Fixed points of nonexpanding maps. *Bulletin of the American Mathematical Society*, 73(6):957–961, 1967.

Bingsheng He and Xiaoming Yuan. Convergence analysis of primal-dual algorithms for a saddle-point problem: From contraction perspective. *SIAM Journal on Imaging Sciences*, 5(1):119–149, 2012. doi: 10.1137/100814494.

R. Helgason, J. Kennington, and H. Lall. A polynomially bounded algorithm for a singly constrained quadratic program. *Mathematical Programming*, 18:338–343, 1980.

HiGHS. HiGHS solvers, 2026. URL https://ergo-code.github.io/HiGHS/stable/solvers/. Documentation, accessed September 2026.

Qi Huangfu and J. A. Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, Mar 2018. ISSN 1867-2957. doi: 10.1007/s12532-017-0130-5. URL http://dx.doi.org/10.1007/s12532-017-0130-5.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E. Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

Daniel S. Kermany, Michael Goldbaum, Wenjia Cai, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131.e9, 2018.

Donghwan Kim. Accelerated proximal point method for maximally monotone operators. *Mathematical Programming*, 190(1–2):57–87, 2021.

Heechang Kim, Ernest K. Ryu, and Shuvomoy Das Gupta. A domain-specific harness for end-to-end automation of optimization research, 2026. URL https://arxiv.org/abs/2608.07407.

Krzysztof C. Kiwiel. Breakpoint searching algorithms for the continuous quadratic knapsack problem. *Mathematical Programming*, 112(2):473–491, 2008.

M. A. Krasnosel'skiĭ. Two remarks on the method of successive approximations. *Uspekhi Matematicheskikh Nauk*, 10(1):123–127, 1955.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Advances in Neural Information Processing Systems*, 2015.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, 2010. doi: 10.1145/1772690.1772758.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.10450.

Felix Lieder. On the convergence rate of the halpern-iteration. *Optimization Letters*, 15(2):405–418, 2021.

Pierre-Louis Lions and Bertrand Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alvin Cheung, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. EvoX: Meta-evolution for automated discovery, 2026a. URL https://arxiv.org/abs/2602.23413.

Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Matei Zaharia, Alexandros G. Dimakis, and Ion Stoica. SkyDiscover: A flexible, adaptive framework for AI-driven scientific and algorithmic discovery. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS)*, 2026b. doi: 10.1145/3786335.3813221. URL https://skydiscover-ai.github.io/blog.html.

Haihao Lu and Wanyu Zhang. Enhanced PDHG for Linear Programming with Online Preconditioning, 2025. URL https://arxiv.org/abs/2506.17650.

W. Robert Mann. Mean value methods in iteration. *Proceedings of the American Mathematical Society*, 4(3):506–510, 1953.

Mathurin Massias, Alexandre Gramfort, and Joseph Salmon. Celer: a fast solver for the Lasso with dual extrapolation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *PMLR*, 2018.

Nick McGreivy and Ammar Hakim. Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations. *Nature Machine Intelligence*, 6 (10):1256–1269, 2024.

Thomas Moreau, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier, Benjamin Charlier, Mathieu Dagréou, Tom Dupré la Tour, Ghislain Durif, Cassio F. Dantas, Quentin Klopfenstein, Johan Larsson, En Lai, Tanguy Lefort, Benoît Malézieux, Badr Moufad, Binh T. Nguyen, Alain Rakotomamonjy, Zaccharie Ramzi, Joseph Salmon, and Samuel Vaiter. Benchopt: Reproducible, efficient and collaborative optimization benchmarks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2206.13424.

Netlib. Netlib linear programming test problems, 2026. URL https://www.netlib.org/lp/. Accessed: 2026-05-05.

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, 2025. URL https://arxiv.org/abs/2506.13131.

NVIDIA. cuML: GPU-accelerated machine learning, 2026. URL https://github.com/NVIDIA/cuml. Software repository, accessed September 2026.

Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

Nobuhiko Ogura and Isao Yamada. Non-strictly convex minimization over the fixed point set of an asymptotically shrinking nonexpansive mapping. *Numerical Functional Analysis and Optimization*, 23(1–2):113–137, 2002.

Jisun Park and Ernest K. Ryu. Exact optimal accelerated complexity for fixed-point iterations. In *International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17420–17457, 2022.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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, Jan 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL http://dx.doi.org/10.1038/s41586-023-06924-6.

Ernest K. Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 2022.

Roland Schwan, Yuning Jiang, Daniel Kuhn, and Colin N. Jones. PIQP: A proximal interior-point quadratic programming solver. In *Proceedings of the 62nd IEEE Conference on Decision and Control*, 2023.

Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent, 2025. URL https://github.com/codelion/openevolve.

Bartolomeo Stellato, Goran Banjac, Paul J. Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020. doi: 10.1007/s12532-020-00179-2. URL https://doi.org/10.1007/s12532-020-00179-2.

Arya Tschand, Kesavan Ramakrishnan, Alexander Ingare, Simon Guo, Jeffrey Ma, Zishen Wan, Simran Arora, Azalia Mirhoseini, and Vijay Janapa Reddi. Hawkeye: Hardware-aware GPU kernel optimization with minimal supervision, 2026. URL https://www.alphaxiv.org/abs/2608.hawkeye-hardware-aware-gpu-kernel-optimization. Preprint.

Bằng Công Vũ. A splitting algorithm for dual monotone inclusions involving cocoercive operators. *Advances in Computational Mathematics*, 38(3):667–681, 2013.

Rainer Wittmann. Approximation of fixed points of nonexpansive mappings. *Archiv der Mathematik*, 58(5):486–491, 1992.

Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2402.01145.

Haotian Ye, Haowei Lin, Jingyi Tang, Yizhen Luo, Rahul Thapa, Caiyin Yang, Chang Su, Rui Yang, Ruihua Liu, Rundao Li, Zeyu Li, Pengwei Sun, Chong Gao, Dachao Ding, Guangrong He, Miaolei Zhang, Lina Sun, Wenyang Wang, Yuchen Zhong, Zhuohao Shen, Puheng Li, Pan Lu, Bianxiao Cui, Di He, Jianzhu Ma, Junfeng Li, Hexi Baoyin, Yejin Choi, Stefano Ermon, Xiaowen Chu, Tongyang Li, Yuzhi Xu, and James Zou. Structured scaling of AI discovery across diverse scientific domains, 2026. URL https://arxiv.org/abs/2604.19341. Introduces SimpleTES; arXiv:2604.19341.

E. Alper Yıldırım. Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19(3):1368–1391, 2008.

Junzi Zhang, Brendan O'Donoghue, and Stephen Boyd. Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020.

A Proofs for Section 4

Throughout this appendix the standing assumptions of the setup paragraph of Section 4 are in force. Because every metric $H \succ 0$ makes $\langle u, Hv\rangle$ an inner product on a finite-dimensional space, all statements are statements about a Hilbert space, and we work directly in $\|\cdot\|_H$ without rescaling.

A.1 Convergence and rate for certified iterations

Proposition 1 (Krasnosel'skiĭ–Mann convergence with a last-iterate rate)**.** Let $H \succ 0$, let T be θ-averaged in $\|\cdot\|_H$ with $\theta \in (0,1)$, and let $\operatorname{Fix} T \neq \emptyset$. Then the iteration $z_{k+1} = Tz_k$ converges, from any z_0, to some $z^\star \in \operatorname{Fix} T$; the residual $\|z_{k+1} - z_k\|_H$ is nonincreasing in k; and, for every $k \geq 0$,

$$\|z_{k+1} - z_k\|_H^2 \;\leq\; \frac{\theta}{(1-\theta)\,(k+1)}\;\operatorname{dist}_H^2(z_0, \operatorname{Fix} T)\,. \tag{4}$$

Proof. Write $T = (1-\theta)I + \theta S$ with S nonexpansive in $\|\cdot\|_H$; all norms in this proof are $\|\cdot\|_H$. Fix any $z^\star \in \operatorname{Fix} T$. Expanding $z_{k+1} = (1-\theta)z_k + \theta S z_k$ with the identity $\|(1-\theta)a + \theta b\|^2 = (1-\theta)\|a\|^2 + \theta\|b\|^2 - \theta(1-\theta)\|a-b\|^2$, at $a = z_k - z^\star$ and $b = Sz_k - z^\star$, and bounding $\|Sz_k - z^\star\| \leq \|z_k - z^\star\|$ by nonexpansiveness gives the Fejér inequality

$$\|z_{k+1} - z^\star\|^2 \;\leq\; \|z_k - z^\star\|^2 - \frac{1-\theta}{\theta}\,\|z_{k+1} - z_k\|^2. \tag{5}$$

The map T is nonexpansive, so $\|z_{k+2} - z_{k+1}\| \leq \|z_{k+1} - z_k\|$. Summing (5) over the first $k+1$ iterations, keeping only the last (smallest) residual, and minimizing over $z^\star$ yields (4). By (5) the iterates are bounded and $\|z_{k+1} - z_k\| \to 0$, so by continuity every cluster point of (z_k) is fixed by T; reusing (5) with such a cluster point as $z^\star$ shows $\|z_k - z^\star\|$ is nonincreasing and vanishes along a subsequence, hence $z_k \to z^\star$. This argument is the Krasnosel'skiĭ–Mann theorem with its modern rate (Ryu & Yin, 2022, Theorem 1); see also Krasnosel'skiĭ (1955), Mann (1953), and Bauschke & Combettes (2017). □

Remark 2 (Monitored residual, normalization, and summability). The runtime monitor reports $\|z_{k+1} - z_k\|_{H_G}$, the step residual of the certified map that is actually iterated. For a genome $\operatorname{relax}_\rho(G')$ the compiled map is $T_G = (1-\rho)I + \rho T_{G'}$, so $z_{k+1} - z_k = \rho\,(T_{G'} z_k - z_k)$: the two residuals differ by the factor ρ, which matters when comparing genomes with different relaxation genes, and (4) divided by ρ^2 bounds the unrelaxed fixed-point residual $\|T_{G'} z_k - z_k\|_{H_G}^2$. Summing (5) also gives the summability estimate $\sum_{k=0}^{\infty} \|z_{k+1} - z_k\|_{H_G}^2 \leq \frac{\theta_G}{1-\theta_G} \operatorname{dist}_{H_G}^2(z_0, \operatorname{Fix} T_G)$.

A.2 Base-scheme certificates

Reference instantiations. The following exact updates fix what the five rows of Table 3 mean for $F = f + g + h \circ L$; $\operatorname{prox}_{\alpha q}$ denotes the exact proximal map of q, and h^* is the Fenchel conjugate of h. A row is admissible only if its stated proximal maps can be evaluated. In particular, writing $g + h \circ L$ as one term does not make its proximal map available. For FBS, $z = x$ and $x^+ = \operatorname{prox}_{\alpha(g + h\circ L)}(x - \alpha \nabla f(x))$. For DRS, set $a = g$, $b = f + h \circ L$ and take $x = \operatorname{prox}_{\alpha a}(z)$, $y = \operatorname{prox}_{\alpha b}(2x - z)$, $z^+ = z + y - x$; the decoder is x. For Davis–Yin, set $a = g$, $b = h \circ L$ and take $x = \operatorname{prox}_{\alpha a}(z)$, $y = \operatorname{prox}_{\alpha b}(2x - z - \alpha \nabla f(x))$, $z^+ = z + y - x$; the decoder is again x. For the primal–dual rows, write $z = (x, u)$ and take

$$\begin{aligned} x^+ &= \operatorname{prox}_{\tau g}\big(x - \tau(\nabla f(x) + L^\top u)\big), \\ u^+ &= \operatorname{prox}_{\sigma h^*}\big(u + \sigma L(2x^+ - x)\big). \end{aligned}$$

This is Condat–Vũ with decoder $\pi(x,u) = x$; PDHG is the specialization in which f is affine. If $f(x) = \langle c, x\rangle + d$, then $\operatorname{prox}_{\tau g}(v - \tau c) = \operatorname{prox}_{\tau(g+f)}(v)$, so the displayed update is a PDHG proximal step. A nonaffine f may instead be absorbed into the primal term only when $\operatorname{prox}_{\tau(g+f)}$ is available exactly and the formulation uses that proximal step.

The fixed points of the first three maps decode to minimizers, and those of the last two are saddle pairs, under the assumptions of Section 4.

Lemma 1 (Base schemes)**.** For each base scheme of Table 3 with positive stepsizes satisfying the gate inequality, the pair (H_G, θ_G) of the table, with $\theta_G = 1/(2 - c_G)$, together with the scheme's standard decoder π_G, is a construction certificate in the sense of Definition 1.

Proof. We first establish every row with the true norm $\|L\|$ in place of ℓ, then account for the conservative bound.

Since ∇f is $(1/L_f)$-cocoercive by the Baillon–Haddad theorem (Baillon & Haddad, 1977), the forward step $I - \alpha\nabla f$ is $(\alpha L_f/2)$-averaged for $\alpha \in (0, 2/L_f)$, and the proximal step is $\frac{1}{2}$-averaged. A composition of θ_1- and θ_2-averaged maps is $(\theta_1 + \theta_2 - 2\theta_1\theta_2)/(1 - \theta_1\theta_2)$-averaged (Ryu & Yin, 2022, Theorem 27), a constant due to Ogura & Yamada (2002); see also Combettes & Yamada (2015). At $\theta_1 = \frac{1}{2}$ and $\theta_2 = c$ it is $1/(2-c)$; this is the FBS row, and the Davis–Yin row is the same constant (Ryu & Yin, 2022, Theorem 28), established by Davis & Yin (2017). DRS is $\frac{1}{2}\big(I + R_{\alpha A}R_{\alpha B}\big)$, a composition of reflected resolvents, hence firmly nonexpansive for every $\alpha > 0$ (Ryu & Yin, 2022, §2.7), a fact going back to Lions & Mercier (1979). For the primal–dual rows, $M_{\tau\sigma} \succ 0$ exactly when $\tau\sigma\|L\|^2 < 1$, and in that metric PDHG (Chambolle & Pock, 2011) is a proximal-point iteration on the saddle subdifferential, hence firmly nonexpansive in $\|\cdot\|_{M_{\tau\sigma}}$ (Ryu & Yin, 2022, §3.3); see also the contraction analysis of He & Yuan (2012). Condat–Vũ (Condat, 2013; Vũ, 2013) is variable-metric forward–backward in the same metric (Combettes & Vũ, 2014), and its forward step is c-averaged in $\|\cdot\|_{M_{\tau\sigma}}$ with $c = \tau L_f/\big(2(1 - \tau\sigma\|L\|^2)\big)$, so composition gives $1/(2-c)$ again (Ryu & Yin, 2022, §3.3). The inequality $\tau L_f/2 + \tau\sigma\|L\|^2 < 1$ is equivalent, for $\tau, \sigma > 0$, to the conjunction of $\tau\sigma\|L\|^2 < 1$ and $c < 1$, so a single inequality delivers both (C1) and a positive-definite metric.

Now let $\ell \geq \|L\|$ be the supplied bound and let c_G denote the tabulated forward-step constant computed from ℓ. The checked inequalities in ℓ imply the corresponding inequalities in $\|L\|$, so $H_G \succ 0$, and the true forward-step constant c computed from $\|L\|$ satisfies $c \leq c_G < 1$, so the compiled iteration is θ-averaged with $\theta = 1/(2-c)$. A θ-averaged map is also θ'-averaged for every $\theta' \in [\theta, 1)$: writing $T = (1-\theta)I + \theta S$ gives $T = (1-\theta')I + \theta' S'$ with $S' = (1-\theta/\theta')I + (\theta/\theta')S$, a convex combination of nonexpansive maps. Since $\theta = 1/(2-c) \leq 1/(2-c_G) = \theta_G < 1$, the pair (H_G, θ_G) certifies (C1), exactly when $\ell = \|L\|$ and conservatively otherwise.

For (C2) and (C3), no bijection between fixed points and solutions is needed; it suffices that fixed points exist and that every fixed point decodes to a solution. For the primal–dual schemes the state is the saddle pair and, because $M_{\tau\sigma} \succ 0$, the fixed points of the (variable-metric) proximal-point and forward–backward iterations are exactly the zeros of the saddle operator, i.e. the saddle set, which is nonempty by assumption; π_G is the primal coordinate (Ryu & Yin, 2022, §3.3). For FBS and Davis–Yin the fixed points are exactly the solutions of the associated inclusion (for Davis–Yin after applying the decoder $\pi_G = \mathrm{prox}_{\alpha g}$), and for DRS every fixed point z satisfies $\mathrm{prox}_{\alpha g}(z) \in \mathcal{X}^\star$ while every solution arises from at least one fixed point; these standard correspondences are (Ryu & Yin, 2022, §2.7). In each case the nonempty saddle set yields a fixed point through the correspondence, giving (C2), and the decoder maps all of $\mathrm{Fix}\, T_G$ into $\mathcal{X}^\star$, giving (C3). □

A.3 Transfer across formulations

Proposition 2 (Formulation transfer)**.** Suppose a transformed composite objective $\widetilde{F}$ satisfies the standing assumptions of Section 4, and an exact recovery map R satisfies $R(\mathrm{argmin}\,\widetilde{F}) \subseteq \mathrm{argmin}\, F$, where F is the original objective in (1). If the gate certifies a base scheme and its relaxation for $\widetilde{F}$, with decoder $\widetilde{\pi}_G$, then the iteration converges in its transformed state space and its limit decodes via $R \circ \widetilde{\pi}_G$ to a minimizer of F.

Proof. Theorem 1 and Corollary 1 applied to $\widetilde{F}$ give convergence to a fixed point whose image under $\widetilde{\pi}_G$ minimizes $\widetilde{F}$. Applying the assumed recovery map places that minimizer in $\mathrm{argmin}\, F$. The gate inequality and metric must be established anew for the transformed operator; an equivalent objective alone does not transfer averagedness. □

When relocation moves a constraint into an indicator term, evaluating the new proximal map requires an exact projection onto that constraint set. Static row scaling or equilibration likewise requires its own equivalence and recovery map. The proposition does not identify an approximate or contract-tested implementation with the exact map.

A.4 Closure rules

Lemma 2 (Relaxation)**.** Let T be θ-averaged in $\|\cdot\|_H$ and let $\rho > 0$ satisfy $\rho\theta < 1$. Then $(1-\rho)I + \rho T$ is $\rho\theta$-averaged in $\|\cdot\|_H$ with $\mathrm{Fix}\big((1-\rho)I + \rho T\big) = \mathrm{Fix}\, T$.

Proof. Writing $T = (1-\theta)I + \theta S$ with S nonexpansive gives $(1-\rho)I + \rho T = (1-\rho\theta)I + \rho\theta S$, which is $\rho\theta$-averaged because $\rho\theta \in (0,1)$, and $\rho > 0$ makes the fixed-point equations equivalent. The decoder and metric are untouched, so the certificate transports as stated in Theorem 1(ii). □

The next lemma is a conditional mathematical rule for blends, not an admission rule of the certified grammar. Its common-fixed-point hypothesis must be established independently; the current syntactic gate conditions alone do not establish it.

Lemma 3 (Blends)**.** Let T_1, T_2 be θ_1- and θ_2-averaged in a common $\|\cdot\|_H$ with $\operatorname{Fix} T_1 \cap \operatorname{Fix} T_2 \neq \emptyset$, and let $\lambda \in (0,1)$. Then $T = \lambda T_1 + (1-\lambda)T_2$ is $\big(\lambda\theta_1 + (1-\lambda)\theta_2\big)$-averaged in $\|\cdot\|_H$ and $\operatorname{Fix} T = \operatorname{Fix} T_1 \cap \operatorname{Fix} T_2$.

Proof. Write $T_i = (1-\theta_i)I + \theta_i S_i$ and set $\theta = \lambda\theta_1 + (1-\lambda)\theta_2$. Then $T = (1-\theta)I + \theta\big(\frac{\lambda\theta_1}{\theta} S_1 + \frac{(1-\lambda)\theta_2}{\theta} S_2\big)$, and the bracket is a convex combination of nonexpansive maps, hence nonexpansive; this is the combination rule of Combettes & Yamada (2015). The inclusion $\operatorname{Fix} T \supseteq \operatorname{Fix} T_1 \cap \operatorname{Fix} T_2$ is immediate. Conversely, let $z \in \operatorname{Fix} T$ and $w \in \operatorname{Fix} T_1 \cap \operatorname{Fix} T_2$. Then $\|z-w\| \le \lambda\|T_1 z - w\| + (1-\lambda)\|T_2 z - w\| \le \|z-w\|$, so both inequalities are equalities and $\|T_i z - w\| = \|z-w\|$ for $i = 1, 2$; the Fejér inequality (5) applied to T_i at z then forces $\|T_i z - z\| = 0$. Hence $z \in \operatorname{Fix} T_1 \cap \operatorname{Fix} T_2$. □

A.5 The Halpern wrapper

Proposition 3 (Halpern genomes)**.** Let T_G carry a certificate (H_G, θ_G, π_G) and set $S_G = I + \theta_G^{-1}(T_G - I)$. Then S_G is nonexpansive in $\|\cdot\|_{H_G}$ with $\operatorname{Fix} S_G = \operatorname{Fix} T_G$, and the compiled iteration

$$z_{k+1} = \lambda_{k+1} z_{\mathrm{anc}} + (1-\lambda_{k+1}) S_G z_k, \tag{6}$$

under any admitted schedule, i.e. $\lambda_k \to 0$, $\sum_k \lambda_k = \infty$, and $\sum_k |\lambda_{k+1} - \lambda_k| < \infty$, converges to $P^{H_G}_{\operatorname{Fix} T_G}(z_{\mathrm{anc}})$, the projection of the anchor onto $\operatorname{Fix} T_G$ in $\|\cdot\|_{H_G}$ (Halpern, 1967; Wittmann, 1992). The canonical schedule $\lambda_{k+1} = 1/(k+2)$ meets all three conditions and, when the wrapper anchors at the initial iterate ($z_{\mathrm{anc}} = z_0$), in addition gives

$$\|S_G z_k - z_k\|^2_{H_G} \;\le\; \frac{4\operatorname{dist}^2_{H_G}(z_0, \operatorname{Fix} T_G)}{(k+1)^2} \tag{7}$$

(Lieder, 2021; Kim, 2021).

Proof. S_G is precisely the nonexpansive map in the averaged decomposition $T_G = (1-\theta_G)I + \theta_G S_G$ of (C1), and $T_G z = z$ if and only if $S_G z = z$ because $\theta_G > 0$. Since $(\mathcal{Z}_G, \langle\cdot, H_G\cdot\rangle)$ is a Hilbert space, the cited Halpern results apply verbatim in $\|\cdot\|_{H_G}$: convergence to the projection of the anchor is Wittmann (1992), and the rate (7) is Lieder (2021). □

Rate (7) is an $O(1/k^2)$ bound on the same squared residual that (4) bounds at $O(1/k)$, and it is exactly optimal in the following sense: Park & Ryu (2022) exhibit a matching lower bound for deterministic fixed-point methods that access S_G only through evaluations, with iterates constrained to the span of the anchor and past evaluations. The claim is optimality within that oracle class, not an unrestricted impossibility statement.

A.6 Safeguarded schedules

Lemma 4 (Summable-error safeguard)**.** Let T be θ-averaged in $\|\cdot\|_H$ with $\operatorname{Fix} T \neq \emptyset$, and let $z_{k+1} = T z_k + e_k$ with $\|e_k\|_H \le \eta_k$ and $\sum_k \eta_k < \infty$. Then (z_k) is quasi-Fejér monotone with respect to $\operatorname{Fix} T$ in $\|\cdot\|_H$ and converges to a point of $\operatorname{Fix} T$.

Proof. For any $z^\star \in \operatorname{Fix} T$, nonexpansiveness gives the one-step inequality $\|z_{k+1} - z^\star\|_H \le \|Tz_k - z^\star\|_H + \|e_k\|_H \le \|z_k - z^\star\|_H + \eta_k$ with $\sum_k \eta_k < \infty$, which is quasi-Fejér monotonicity; in particular $\|z_k - z^\star\|_H \le M := \|z_0 - z^\star\|_H + \sum_j \eta_j$ for every k. Combining the one-step inequality with the Fejér inequality (5) for the exact step gives

$$\|z_{k+1} - z^\star\|^2_H \;\le\; \|z_k - z^\star\|^2_H - \frac{1-\theta}{\theta}\|Tz_k - z_k\|^2_H + 2M\eta_k + \eta_k^2,$$

and summing shows $\sum_k \|Tz_k - z_k\|_H^2 < \infty$, hence $\|Tz_k - z_k\|_H \to 0$. The iterates are bounded, so by continuity some cluster point $z^\star$ is fixed by T; quasi-Fejér monotonicity makes $\|z_k - z^\star\|_H + \sum_{j \geq k} \eta_j$ nonincreasing, so $\|z_k - z^\star\|_H$ converges, and it vanishes along a subsequence, hence $z_k \to z^\star$. This is the classical inexact Krasnosel'skiĭ–Mann analysis (Combettes, 2001; 2004). □

The same mechanism underlies the inexact analysis of Douglas–Rachford and ADMM (Eckstein & Bertsekas, 1992) and safeguarded Anderson acceleration (Zhang et al., 2020), the design template for the switch gene. The guarantee is convergence, not a rate: the budget buys back the limit an aggressive step would otherwise forfeit.

B Related Work Continues

LLM-driven algorithm search. FunSearch and AlphaEvolve search programs using evaluator feedback (Romera-Paredes et al., 2024; Novikov et al., 2025). ReEvo already combines population-based evolution, LLM reflection, and execution feedback for combinatorial-optimization heuristics (Ye et al., 2024); evaluator feedback inside an evolutionary loop is therefore not our contribution. OpenEvolve, ShinkaEvolve, GEPA, AdaEvolve, and EvoX vary the population or proposal policy (Sharma, 2025; Lange et al., 2025; Agrawal et al., 2025; Cemri et al., 2026; Liu et al., 2026a). SkyDiscover supplies infrastructure used here for projection synthesis (Liu et al., 2026b). OptEvolve instead restricts which exact solver schemes may run: a mathematical gate checks sufficient convergence conditions before evaluation, while generated numerical code receives separate tests.

Strong structured baselines. Generic-solver comparisons alone miss coordinate descent and working-set methods for lasso and elastic net (Friedman et al., 2010; Pedregosa et al., 2011; Massias et al., 2018; Bertrand et al., 2022), LIBLINEAR and LIBSVM for SVMs (Fan et al., 2008; Chang & Lin, 2011), L-BFGS-B for smooth bounded problems (Byrd et al., 1995), network simplex for transport (Bonneel et al., 2011; Flamary et al., 2021), and Frank–Wolfe core-set methods for the minimum enclosing ball (Yıldırım, 2008; Lacoste-Julien & Jaggi, 2015). Box–hyperplane projection also has a classical breakpoint algorithm (Helgason et al., 1980; Kiwiel, 2008). These specialists join the field when applicable. Comparisons use a common original-problem acceptance check, and competitors run at their shipped defaults; this measures out-of-the-box performance at a target accuracy, not the best performance after competitor tuning (McGreivy & Hakim, 2024). Beyond generated families, we use established LP, QP, and conic benchmarks, including Netlib and MIPLIB LP relaxations (Netlib, 2026; Gleixner et al., 2021). Benchopt already provides reproducible cross-language, cross-hardware optimization benchmarks (Moreau et al., 2022). OptEvolve-Bench adds a fixed original-problem acceptance check, a comparison field with recognized specialists, and the finest accepted tolerance beside time. When the reference optimum is numerical rather than a proven bound, this is an empirical acceptance check, not a formal optimality certificate. CUTEst and MIPLIB-NL are related nonlinear and language-specified testbeds, respectively, but are not evaluated here (Gould et al., 2015; Li et al., 2026).

C Limitations

OptEvolve does not uniformly beat specialists across every possible problem class. The convergence theorem covers exact updates in the stated convex scheme-and-relaxation grammar, with an admissible formulation, nonempty saddle set, valid norm bound, and exact atoms. It does not certify synthesized projections or floating-point kernels: finite contract tests screen implementations, while original-problem acceptance checks returned points. Search is restricted to implemented schemes and reformulations. Finally, gains depend on workload, hardware, and accounting; large cold wins on generated families do not establish comparable gains for every problem class or one-off solve.

Three limits apply to the claims added by the benchmark and the recovery experiment. The acceptance check is empirical wherever $f^\star$ is a numerical reference rather than a proven bound, so it separates solvers that reach an accuracy from those that do not, and is not a proof of optimality. On the two per-column separable classes the packaged solvers are timed on a subset and reported at their measured per-column rate, which is exact for a separable problem but is an estimate rather than a wall clock, and the unmixing margin rests on it. And in the recovery experiment a uniform random

proposer over the same typed space also assembled a full method and passed every sealed check in one of its two seeds, so those runs show that the exposed grammar makes the method reachable in few trials, not that the language model is what finds it; two seeds per arm cannot separate the two.

D HARDWARE RESULTS

Table 6: Best temporally fused tile (B, T) and its gain over the unfused CUDA step, per device, three repetitions. Only one of the twelve pairwise device/size comparisons selects the same tile.

	RTX 4090		A100		H100	
size	tile	gain	tile	gain	tile	gain
128	$(8, 4)$	$1.619\times$	$(16, 8)$	$2.865\times$	$(16, 8)$	$2.521\times$
256	$(24, 4)$	$1.482\times$	$(32, 8)$	$2.430\times$	$(16, 8)$	$2.330\times$
384	$(24, 4)$	$1.270\times$	$(40, 8)$	$2.098\times$	$(24, 8)$	$1.883\times$
496	$(32, 4)$	$1.337\times$	$(56, 4)$	$1.934\times$	$(32, 8)$	$1.792\times$

E FURTHER CERTIFICATE CONSEQUENCES

Remark 3 (What the certificate buys the search). Because $1/\theta_G = 2 - c_G$ for every base scheme, the certified over-relaxation window is exactly $\rho \in (0, 2 - c_G)$: the textbook interval $(0, 2)$ for the schemes with no forward step, shrinking as the forward step consumes the budget. For Condat–Vũ this window is not a conservative surrogate. Condat (2013, Theorem 3.1) proves convergence of the same relaxed iteration for any constant $\rho_n \in \big(0, \delta\big)$ with $\delta = 2 - \frac{\beta}{2}\big(\frac{1}{\tau} - \sigma\|L\|^2\big)^{-1}$, where $\beta = L_f$ is the Lipschitz constant of the smooth gradient. Multiplying numerator and denominator by τ gives $\delta = 2 - c_\star$, where $c_\star := \frac{\tau L_f}{2(1-\tau\sigma\|L\|^2)}$ is the forward-step constant computed with the exact operator norm. Since $\ell \geq \|L\|$, it follows that $c_\star \leq c_G$ and hence $2 - c_G \leq \delta$. The certified window therefore coincides with Condat's when $\ell = \|L\|$, and is contained in it when $\ell > \|L\|$. The discovered policy of Section 5.2 illustrates this effect. The certified textbook default $(\tau, \sigma) = (0.5, 0.186)$ has $c_G \approx 0.97$ and therefore admits almost no over-relaxation at all, $\rho < 1.03$, whereas the evolved $(\tau, \sigma) = (0.05, 1.0)$ has $c_G = 0.042$ and admits $\rho < 1.96$. Shrinking the primal step does not merely satisfy the gate, it widens the interval the relaxation gene may search in, and the champion sits near its new edge at $\rho = 1.9$, with $\rho\,\theta = 0.97$.

Halpern genomes. A certificate also determines the nonexpansive map used by the Halpern wrapper. The map $S_G = I + \theta_G^{-1}(T_G - I)$ is nonexpansive in $\|\cdot\|_{H_G}$ with $\operatorname{Fix} S_G = \operatorname{Fix} T_G$. The wrapped iteration is not averaged, so Corollary 1 does not apply. Classical Halpern theory instead gives its guarantee using the certificate for T_G. Under the admitted schedules, it converges to $P^{H_G}_{\operatorname{Fix} T_G}(z_{\mathrm{anc}})$, the $\|\cdot\|_{H_G}$-metric projection of the anchor, and the canonical schedule anchored at z_0 drives the squared residual at $O(1/k^2)$, exactly optimal in the black-box oracle model of Park & Ryu (2022) (Proposition 3, Appendix A). Thus, the Halpern gene changes the selected fixed point and the residual rate, while preserving convergence.

Formulations and implementations. Relocating constraints or applying fixed equilibration can change the splitting and hence T_G, but need not change the underlying optimization problem. For each formulation, the gate must check its own scheme inequality with a valid bound on its transformed operator; its decoder must recover a solution of the original problem. The solution-transfer rule is made explicit in Proposition 2 of Appendix A. In particular, an exact projection may serve as a proximal map after relocation; replacing it by synthesized code preserves the theorem only if that code implements the same map. Finite contract tests against a reference projection support admission of the implementation, but do not prove that equality for every input. Precision and fusion also change the implemented computation, not the certificate for the ideal map. The mathematical guarantee must therefore be distinguished from tests of numerical kernels and from the acceptance check on each returned point.

Safeguarded schedules. Corollary 1 assumes exact arithmetic and exact atoms. Returned numerical points are separately checked by the acceptance criterion of Section 5.1; the TV cell also checks its runtime duality gap (Section 3). Free-form schedules, including the switch gene of Section **??**, run outside $\mathcal{G}$ under a summable-error safeguard: the monitor accepts a step $T_G z_k + e_k$ only if $\|e_k\|_{H_G} \leq \eta_k$ for a budget satisfying $\sum_k \eta_k < \infty$. Otherwise, the monitor uses the certified step. This rule preserves convergence to a fixed point (Lemma 4, Appendix A). This extension is not used in the experiments. All reported runs have $e_k = 0$.

For a fixed admissible formulation and exact atoms, every genome admitted from $\mathcal{G}$ thus converges in exact arithmetic. Fitness selects for time to an accepted answer; it does not establish the convergence guarantee.

F METHOD DETAILS

This appendix records the scheme checks, implementation contracts, and routing details underlying Section 3.

F.1 CERTIFIED METHOD MIXING

Mixing methods offers three extensions of increasing aggressiveness. The reported experiments do not rely on them, and the soundness theorem for the base grammar does not automatically cover them.

Averaged blending. A convex combination $\mathrm{avg}(\lambda; T_1, T_2) = \lambda T_1 + (1-\lambda) T_2$ of two averaged operators is averaged with a computable constant (Ryu & Yin, 2022, §2.3), given in closed form by Combettes & Yamada (2015). Lemma 3 in Appendix A gives a certificate rule when both maps share a state and metric and have a nonempty common fixed-point set. A common fixed point is a semantic obligation not established by the present syntactic scheme gate. Thus the lemma describes a conditional extension, not an automatically admitted genome covered by Theorem 1.

Three-operator splitting. Davis–Yin splitting (Davis & Yin, 2017) is the canonical certified fusion of forward–backward and Douglas–Rachford, handling all three blocks of problem (1) simultaneously; it enters the genome as one additional base scheme with its own gated side condition.

Safeguarded switching. A $\mathrm{switch}(\mathrm{trigger}, \mathrm{aggressive}, \mathrm{fallback})$ gene runs an aggressive component (stepsizes outside the certified region, Anderson-type extrapolation, aggressive restarts) under a summable-error safeguard. Relative to the certified fallback step, the accepted perturbation at iteration k must have metric norm at most a prescribed η_k with $\sum_k \eta_k < \infty$; otherwise the fallback step is taken. The resulting inexact iteration therefore inherits the fallback's convergence (Section 4). The hand-designed literature illustrates distinct ways to compose solver decisions: HPR-LP uses Halpern-accelerated semi-proximal Peaceman–Rachford splitting (Chen et al., 2024), whereas PDLP adapts the primal weight of a PDHG-based solver (Applegate et al., 2021). A switch gene would make guarded versions of such decisions searchable; its convergence claim depends on the stated summable-error condition.

Blends and safeguarded switching are promising directions for discovering compositions beyond single schemes. They are not exercised in Section 5; blending requires a common-fixed-point witness or a restricted constructive typing rule before it can be admitted with the guarantee claimed for the formal grammar.

F.2 THE STAGE-0 TYPE-CHECK GATE

For the base-scheme and relaxation grammar of Section 4, the Stage-0 gate checks the convergence side condition before execution: $\alpha \in (0, 2/L_f)$ for FBS and Davis–Yin; any $\alpha > 0$ for DRS; f affine and $\tau\sigma\ell^2 < 1$ for PDHG; $\tau L_f/2 + \tau\sigma\ell^2 < 1$ for Condat–Vũ, where $\ell \geq \|L\|$ is the supplied operator-norm bound (Ryu & Yin, 2022, §2.7 and §3.3); and for the wrappers, $\rho\,\theta < 1$ for over-relaxation of a θ-averaged map, with Halpern anchoring admitted only for nonexpansive iterations. FBS, DRS, and Davis–Yin additionally require the exact proximal maps of their chosen splitting to be available (Appendix A). A genome that satisfies its conditions receives a construction certificate recording the checked inequality, the resulting averagedness constant, and the textbook citation; a

genome that does not is rejected, never silently clamped, and the violated inequality is returned verbatim as feedback to the proposer. The scheme gate is re-run inside every solve; a 20,000-genome fuzz found zero disagreements between that gate and the solver-level checks. The theorem also assumes an admissible formulation and exact atoms: Section 4 distinguishes these mathematical obligations from numerical contract tests.

F.3 Compilation and certificates

An admitted scheme is compiled to a jitted JAX iteration; the mathematical certificate applies to its exact reference update, not automatically to floating-point execution. Genome search never executes LLM-written numerical code. LLM-written code runs only in operator synthesis (Section 3), where a proposed projection is executed only inside its contract harness and enters a solve only after passing it. Beyond the construction certificate (level 1), each solve carries runtime certificates (level 3): the step residual $\|z_{k+1} - z_k\|_{H_G}$ of Section 4 and, for problem (1) with strongly convex f, a rigorous duality gap. For TV denoising the dual value at any u with $\|u\|_\infty \le \lambda$ is $\langle y, D^\top u\rangle - \frac{1}{2}\|D^\top u\|^2$, so $\mathrm{gap}(x, u) \ge 0$ always and termination at relative gap $\le \epsilon$ is a checkable certificate of ϵ-optimality, not a heuristic stopping rule. The benchmark record also stores the original-problem acceptance result.

F.4 Hardware-aware backend genes

A second family of genes controls not what iteration runs but how it is computed: floating-point precision, batching across instances, sparse-matrix format, and kernel-fusion strategy. The construction certificate concerns an exact operator T, not the backend's floating-point behavior. Fusion, precision, and block sizes can change numerical results, so the implementation is checked separately by per-atom tests against a reference backend to tolerance, in the pattern of hardware-aware kernel generation for coding agents (Tschand et al., 2026). This separates two implementation-level checks from the certified core: a safeguard monitor for schedules above the calculus and unit tests for kernels below it. Neither check alone proves the exact-operator assumptions of Section 4.

The target hardware is visible to the search: the device identifier conditions the mutation context, so genomes can specialize to the accelerator they will run on. Kernel fusion matters most exactly where per-iteration work is small and launch overhead dominates, i.e. the microsecond-budget cells; batched execution matters where many instances are solved simultaneously. This paper ships the precision, batching, and fused-versus-naive step genes; autogenerated fused kernels for the atoms are future work.

The fusion tile is a gene. A temporally fused kernel runs T iterations per launch on a $(B + 2T) \times (B+2T)$ tile whose halo is held in shared memory, so the pair (B, T) is a backend gene like precision or batching. Three things make it a gene rather than a constant. Its admissible set is device dependent: a tile whose halo exceeds the device's opt-in shared memory per block cannot launch at all, which removes a third of the grid on a consumer part and none of it on a data-center part. Its correctness is a contract, not an assumption: an admitted tile must reproduce T sequential unfused iterations, which at relaxation $\rho = 1$ means bitwise, so the gene cannot trade accuracy for speed. And its value is measured, not derived: the cost model prices a tile by the per-iteration cost it achieves on the device in question, exactly as it prices an operator, so the tile competes inside equation (8) rather than being fixed ahead of the search. At solve time the device is identified from its reported properties (name, compute capability, multiprocessor count, shared memory per block) and the tile is selected for that device; declining to fuse, that is, running the compiled loop of the array compiler, is always one of the candidates.

The device is a candidate. Because an exact operator leaves the iteration count N unchanged, the same (formulation, operator) pair on a machine's GPU and on its CPU are two candidates that share one measured N and differ only in setup and per-iteration cost. The router therefore enumerates (formulation, operator, device) triples and prices each device from its own calibration; a machine's CPU is a separate device from its GPU, and a device for which the process has no backend is declined rather than priced from a borrowed measurement. Section G.8 reports where the device, the tile and the precision decision change the winner.

F.5 FORMULATION GENES

The same feasible set admits many splittings of problem (1), and which constraints live in the linear operator L and which live in the proximal step is itself a gene. Relocation removes a constraint family from $h(Lx)$ and enforces it exactly through the projection in g: a single dense row together with the variable bounds it touches (a box intersected with a hyperplane or a slab), a product of disjoint such blocks, or a family of second-order cones. The kept rows stay in L and are optionally equilibrated row by row. A planner reads the candidate families off (P, q, A, l, u) and works under any row and column permutation of the instance, so nothing depends on how a benchmark happens to order its variables. Relocation changes the iteration but not the problem, and the certificate transfers only when the projection is exact; the planner declines when no exact projection exists. The formulations the router can currently choose are the standard form with and without Ruiz equilibration, row relocation with and without row scaling, product relocation under several row-selection rules, and cone relocation. Each transformation must preserve the original solution set under its recovery map and pass the scheme gate with its own operator bound; the formal transfer rule is Proposition 2. Tests of an implemented projection are a separate obligation from its mathematical exactness.

F.6 OPERATOR SYNTHESIS UNDER CONTRACT

A relocated formulation is only as fast as its projection, and when the cost model finds that the projection's per-call time is the binding term it emits a specification rather than a guess: the set (for example $\{x : lo \le x \le hi,\ a^\top x \in [b_{lo}, b_{hi}]\}$), the device, the per-call budget that would change the routing decision, and a contract of input conditions the implementation must handle (mixed-sign a, infinite bounds, degenerate breakpoint ties, very small and very large dimension, and the inputs seen in deployment). An LLM evolves implementations against that specification inside SkyDiscover (Liu et al., 2026b). A candidate implementation is admitted only if it passes the contract tests: agreement with a reference projection on every tested clause. These tests screen numerical code but do not prove that it equals the exact projection on all inputs. Admitted operators are calibrated per device and stored with their provenance. A projection that is fast but fails the contract tests is never used, however good its score.

F.7 THE GENE POOL, THE COST MODEL, AND ROUTING

Evolved components, formulations, measurements, and lessons are kept in an append-only store, the gene pool. The router decomposes the time to a verified answer as

$$T = \text{setup} + N(\text{formulation}) \cdot t(\text{operator}, \text{device}), \tag{8}$$

where the iteration count N is taken from measured evidence for the formulation and never from an operator-norm bound, and the per-iteration cost t and the setup are taken from calibration of the operator on the device. Candidates are the product of formulations, contract-tested projection implementations, and devices, together with external engines, each run in its own interpreter: interior-point and first-order generic solvers (Goulart & Chen, 2024; Schwan et al., 2023; Stellato et al., 2020; O'Donoghue et al., 2016; Domahidi et al., 2013; HiGHS, 2026; Applegate et al., 2021), and structure-specific solvers that are offered only after a detector has recognized their structure in the instance under permutation: coordinate descent for lasso and group lasso (Massias et al., 2018; Bertrand et al., 2022; Pedregosa et al., 2011), network simplex and minimum-cost flow for network linear programs (Bonneel et al., 2011; Flamary et al., 2021), and a Frank–Wolfe core-set method for the minimum enclosing ball (Yıldırım, 2008; Lacoste-Julien & Jaggi, 2015). A candidate without evidence is explored, and its run is charged to the budget; exploration stops once it has cost twice the time to the first verified answer, and the verified incumbent is never discarded. Every answer, whichever engine produced it, is accepted only if it passes the same check on the original problem. An LLM agent drives the system through four actions: solve an instance, evolve an operator against a specification, propose a planner rule, and propose a cost-model term, each admitted only after measurement on held-out instances.

F.8 EVOLUTIONARY LOOP

Search proceeds over generations with elitism. Two proposal engines operate on the identical genome space at identical evaluation budgets: (i) an LLM backend that receives the current archive (best

genomes with their metrics), the mutation menu, and the genome JSON schema, and replies with a patched genome; invalid JSON or a Stage-0 rejection triggers one retry with the error injected, then a fallback to the random proposer; proposals are deterministically seeded and the archive deduplicates by content hash; and (ii) a random-search control that rejection-samples the valid region with log-uniform stepsizes. Fitness is the negated shifted geometric mean (SGM-10) of wall-clock time-to-certificate over the training instances; runs that fail to reach the certificate within the iteration budget score $-\infty$. The loop monitors for degenerate (flat) fitness landscapes and reports them rather than proceeding silently. Final comparisons are made exclusively on held-out instances never seen during search. The backend interface is engine-agnostic by construction, so the evolutionary machinery itself (island scheduling, parent sampling) can be swapped without touching the gate, compiler, or certificates (Liu et al., 2026b).

F.9 Implementation

Solvers are compiled with JAX. The total-variation search of Section 5.2 runs at float64 on the CPU of a single workstation, with Qwen2.5-Coder-1.5B-Instruct served by vLLM on its RTX 4090 as the mutation model. Operator synthesis uses Qwen3.8-27B in FP8 served by vLLM on an H100. Solves are timed on an RTX 4090 workstation and on an H100 80 GB node; baselines and specialists run on the CPU of the machine being measured, in separate interpreters that never share an environment with our solver.

F.10 Proposal policy and adversarial selection.

Which proposer to ask is itself a choice, and it can be learned without touching the model. A disjoint LinUCB contextual bandit (Li et al., 2010) arbitrates the proposal channel, deciding at each step whether to query the model for a complete typed design, mutate one legal gene of the incumbent, or sample a fresh legal design from the same typed space. It is updated from the executed outcome of each candidate and leaves the proposer weights frozen, so it changes where the evaluation budget is spent rather than what the model knows. Selection can also be made adversarial. The manually authored reference program is frozen as the opening opponent with its source withheld from the proposer, and a challenger is promoted only if it certifies every case and then wins a paired contest against the current champion, the original anchor, and a rotating earlier opponent, under randomized contest order, with a bootstrap lower bound above $1.05\times$ and no case regressing by more than 20%; the promoted champion becomes the next opponent and the next parent. On the unmixing class at a fixed budget of eight fresh evaluations, the bandit channel reached its first certified assembly on the fourth candidate using two model calls, against the sixth candidate and six model calls for the frozen loop, so what it buys is queries and evaluations rather than a better solver. We report the completed fixed-budget controls and the seeds in which uniform random search also recovers the method.

G Extended experimental results

This appendix gives the per-family settings, additional comparisons, and measurements summarized in Section 5.

G.1 Evaluation protocol

One acceptance check. Every returned point, from our solvers and from every competitor, is accepted only if its relative constraint violation and its objective gap relative to $1 + |f^\star|$ are both at most the target tolerance, where $f^\star$ is a high-accuracy reference objective computed once and outside every timed region. No solver is credited for its own stopping rule.

A completed field. Generic solvers (Clarabel, PIQP, HiGHS, OSQP, SCS, ECOS, ProxQP, PDLP and cuPDLPx) each run in their own interpreter. Where a structure-specific solver exists it is added: scikit-learn, celer and skglm for lasso and group lasso, LIBSVM and LIBLINEAR for SVMs, network simplex and minimum-cost flow for network LPs, and a Frank–Wolfe core-set method for the minimum enclosing ball.

config	champion genome	train SGM	held-out SGM	held-out fail
default (certified)	`condat_vu` τ=0.5, σ=0.186	0.346s	0.621s	0/3
random-evolved	`condat_vu` τ=0.024, σ=3.70	0.061s	0.073s	0/3
LLM-evolved	`condat_vu` τ=0.05, σ=1.0, ρ=1.9	**0.036s**	**0.073s**	0/3

Table 7: TV-on-OCT cell. Both evolved policies beat the certified textbook default by $\approx 8.5\times$ on held-out clinical images with zero failures. The LLM champion's certificate: $\tau L_f/2 + \tau\sigma\ell^2 = 0.425 < 1$, averagedness $\theta = 0.511$, $\rho\,\theta = 0.97 < 1$.

Timing. Setup is timed on both sides, imports are excluded on both sides, baselines run in fresh processes, and repetitions are paired. Head-to-head speedups (Table 9) time our method in a fresh process with structure detection, formulation build, compilation and solve inside one clock and the persistent compilation cache disabled (the cold first solve), five repetitions, and also report a second solve in the same process (warm). Held-out solve rates come from routing runs with a 20 s budget, where the first visit includes exploring candidates. Solves run on an RTX 4090 workstation and on an H100 80 GB node, and each comparison names its machine.

G.2 Evolution produces solvers, formulations and operators

Parameters and schemes: total-variation denoising. The first cell is anisotropic TV denoising, $\min_x \frac{1}{2}\|x - y\|^2 + \lambda\|Dx\|_1$ with $\lambda = 0.1$, on real clinical retinal optical coherence tomography (OCT) B-scans from the Kermany dataset (Kermany et al., 2018): 128×128 patches, with the raw image as y, 6 training and 3 held-out patches. A solve terminates at relative duality gap $\leq 10^{-4}$ (Section 3). The baseline is the fastest of the four base schemes at textbook, typechecked defaults (Condat–Vũ at 0.346 s training SGM-10). Search runs 10 generations of 8 proposals with elitism 1, and champions are evaluated once on the held-out patches.

The evolved policy is $8.5\times$ faster than the textbook default on held-out images (Table 7), and it is not a standard tuning: it shrinks the primal step $10\times$, grows the dual step $5.4\times$, retreats from the stepsize-condition boundary ($0.9925 \to 0.425$), and spends the freed slack on near-maximal certified over-relaxation ($\rho\,\theta = 0.97$; Remark 1). Invalid proposals were rejected symbolically without consuming solver budget, and no uncertified configuration was executed. The LLM-guided and random runs tie on held-out quality, and because they shared a proposal stream we make no sample-efficiency claim from this cell; Section 5.4 compares against LLM-only evolution instead. This cell measures evolution against the textbook defaults of the same scheme, not against a completed field: no external TV solver is run here, and anisotropic TV-L2 is exactly solvable by parametric max-flow, so the $8.5\times$ is an internal ablation and not a claim about the state of the art for this problem.

Operators: certified synthesis. Table 8 compares implementations of the projection onto a box intersected with a hyperplane, the operator that relocation needs on the budget-constrained families. The best synthesized operator is $10.6\times$ faster than the bisection seed the search started from and $3.16\times$ faster than the textbook breakpoint search on the RTX 4090. A black-box LLM search without the certificate found an operator that is faster still per call, but it fails 16 of the 17 contract clauses (maximum error 9.97) and, used inside a solver, reaches the target on only 5 of 9 unseen instances; the certified operators reach it on 9 of 9, at exactly the iteration count of the reference projection, on both GPUs.

Per-call speed is not end-to-end speed. With the formulation fixed and only the projection swapped, on knapsack_ridge at $n = 12{,}000$ the synthesized operator's cold first solve is $1.16\times$ to $1.29\times$ faster than with a hand-written 12-step Newton projection across tolerances 10^{-3} to 10^{-6}, and $1.27\times$ to $1.56\times$ faster than the textbook breakpoint search on a warm solve. Most of the speedup over generic solvers comes from the formulation: relocation with the hand-written Newton projection is already $3.7\times$ faster than the best engine at 10^{-4}. Operators evolved in a later agent session match the earlier synthesized ones to within about 5%.

Formulations. The formulation gene moves iteration counts far more than any parameter: across the formulations of one capacitated routing LP the iteration count spans $1{,}233\times$, and two formulations

projection onto $\{lo \le x \le hi,\ a^\top x = b\}$	RTX 4090 (μs)	H100 (μs)	certified	unseen instances
bisection, 60 steps (search seed)	395.67	439.09	yes	–
Newton, 12 steps (hand-written)	165.90	113.42	yes	–
breakpoint search (Kiwiel, 2008)	117.88	69.71	yes	–
synthesized, round 1	47.54	59.44	yes	9/9
synthesized, round 2	**37.35**	**33.15**	yes	9/9
black-box LLM search, no certificate	12.04	8.24	no (1/17)	5/9

Table 8: Per-call cost of the box-hyperplane projection, from per-device calibration in the gene pool. "Unseen instances" counts held-out instances on which a solver using the operator reaches the target.

		ours, cold	best engine	speedup, cold first solve			warm
instance (held-out)	route chosen	10^{-4} (s)	10^{-4} (s)	10^{-3}	10^{-4}	10^{-6}	10^{-4}
knapsack_ridge, $n = 12{,}000$	relocation + synthesized op.	1.04	Clarabel 4.69	4.08×	4.49×	5.00×	9.17×
knapsack_ridge, $n = 48{,}000$	relocation + synthesized op.	8.10	Clarabel 204.3	34.98×	25.22×	20.57×	72.79×
robust LP, $n = 5{,}000$	GPU first-order, standard form	1.77	Clarabel 15.71	12.64×	8.86×	5.63×	11.41×
SVM dual, $n = 20{,}000$	relocation + synthesized op.	1.20	Clarabel 1.24	0.84×	1.03×	1.11×	1.97×
budget band, $n = 80{,}000$	relocation + synthesized op.	1.33	PIQP 0.32	0.24×	0.24×	0.25×	0.42×
group lasso, $n = 10{,}000$	cone relocation	1.02	celer 0.57	0.57×	0.56×	0.55×	0.88×

Table 9: The routes the system chose on held-out instances, timed head to head on an RTX 4090. Ours is a fresh process with detection, build, compilation and solve in the clock and no persistent compilation cache (cold), or a second solve in the same process (warm); CUDA context creation, about 0.22 s, is outside the clock. The best engine is the fastest accepted result over every generic solver and every recognized specialist, each run in a fresh process with a 600 s limit. Paired geometric means over five repetitions.

whose operator norms differ by 0.056% differ by $154.7\times$ in iterations. Because the per-iteration cost also changes, the router decides from measured time, not from iterations (Section C).

G.3 Wins against structure-specific solvers

Table 9 times the routes the system chose on held-out instances. The budget-constrained least-squares family has no packaged solver; its natural specialist is projected gradient with an exact projection onto the box and budget constraint, and the system builds that method itself. A cold first solve is $4.1\times$ to $5.0\times$ faster than every available engine at $n = 12{,}000$ and $20.6\times$ to $35.0\times$ faster at $n = 48{,}000$, where Clarabel needs 204 to 245 s and PIQP and ECOS do not finish in 600 s. The margin grows with size: on the same family at tolerance 10^{-4} it is $4.58\times$ at $n = 12{,}000$, $11.37\times$ at 24,000 and $43.09\times$ at 48,000, and at $n = 100{,}000$ no engine returns an answer (Clarabel exceeds a 24 GB memory limit and PIQP returns nothing in 600 s) while ours takes 25.6 s. On the robust LP the evolved route is $12.6\times$ faster at 10^{-3} and $5.6\times$ at 10^{-6}. On three of the six instances the route the system chose loses when timed head to head: the SVM dual is at parity or slower, PIQP is $4\times$ faster on the budget band, and celer is about $1.8\times$ faster on group lasso. The router's earlier cost predictions for these instances were too optimistic, which Section C discusses.

Lasso paths with dense solutions. On a 100-value lasso regularization path, scored per regularization value by the lasso duality gap, with celer and scikit-learn each run as shipped, the batched GPU method wins once the solution support approaches the number of rows. At $n = 10{,}000$ and tolerance 10^{-4} the speedup grows with density from $0.52\times$ to $1.52\times$ to $15.8\times$ (3.28 s against 51.80 s), and at 10^{-6} it is $5.07\times$ (18.27 s against 92.68 s). At $n = 50{,}000$ it is $7.34\times$ at 10^{-4} (29.55 s against 216.92 s), and at 10^{-6} our method certifies all 100 values in 177 s while the best completed specialist run certifies 72 in 641 s.

Other specialist comparisons. On the SVM family at $n_{\mathrm{var}} = 110{,}000$ and tolerance 10^{-6}, LIBLINEAR (Fan et al., 2008) takes 43.0 s on the identical instance, against 10.35 s for our method in the first-reported measurement; under the corrected accounting our margin over Clarabel on this family is $3.78\times$ $[3.67, 3.90]$. On elastic-net logistic regression over the epsilon data set, the method

family	n_{var}	speedup over Clarabel	95% interval
SVM dual	550,000	$7.91\times$	$[6.34, 9.88]$
SVM	110,000	$3.78\times$	$[3.67, 3.90]$
portfolio	500,707	$2.37\times$	$[1.78, 3.16]$

Table 10: Speedup over Clarabel at tolerance 10^{-6}, three seeds, every repetition paying its own setup. Families appear here only where Clarabel is the strongest available engine. Huber, lasso and the minimum enclosing ball are excluded because their structure-specific solvers are faster than our method (Section C), and the control family is excluded because OSQP, already in our own field, is faster than Clarabel at that size.

is $2.28\times$ $[1.73, 3.01]$ faster than the GPU specialist cuML (NVIDIA, 2026) on ten of ten penalty settings for a repeated solve, and $0.06\times$ on a first solve with an empty compilation cache.

Across the held-out set. Over 28 held-out instances spanning QP, LP, SOCP, Maros–Mészáros, Netlib and CBLIB, the routed system solves 27 within a 20 s budget, against 26 for the best-per-instance oracle over the completed field and 25 for the best single engine (Clarabel). On the 25 instances both solve, it is faster on 13 and the oracle on 12, with a geometric-mean speedup of $1.29\times$ on repeat visits and $0.035\times$ on first visits, when exploration dominates.

G.4 Against general-purpose solvers at scale

Against a general-purpose interior-point solver, the evolved formulations win on every family in Table 10, and every interval excludes 1. The wins are size dependent: the SVM dual crosses over between $n_{\text{var}} = 38{,}500$ ($0.79\times$) and $55{,}000$ ($1.45\times$), and portfolio between $100{,}316$ ($0.92\times$) and $250{,}500$ ($1.57\times$).

G.5 Against LLM-only evolution

The LLM-only arm evolves an operator with the same LLM and the same search engine, but without the certificate, the gene pool or the router, and uses it in a fixed pipeline. It solves 18 of 28 held-out instances against 27 for the full system, and its geometric-mean repeat solve is $23\times$ slower (Table 5). Its first visit is slightly faster because it has a single candidate to run. The faster uncertified operator passes only 1 of 17 contract clauses and reaches the target on 5 of 9 unseen instances, against 9 of 9 for the contract-checked operators (Table 8). LLM token counts were not logged per arm, so the two arms are matched on search iterations, not on tokens.

G.6 Choosing CPU or GPU

The router's device choices follow instance structure and size. It sends sparse lasso to a CPU coordinate-descent specialist (celer, a routed cost of 11.9 ms at $n = 5{,}000$), small Maros–Mészáros and Netlib instances to a CPU interior-point method (about 1 ms), and large structured QPs and robust LPs to evolved GPU methods (Table 9). On MIPLIB LP relaxations, 20 of the 22 instances solved within budget are routed to CPU interior-point engines and 2 to our first-order formulations. The payoff of a formulation also depends on the device: on the capacitated routing LP the simplex formulation's advantage is $5.6\times$ on the GPU and $45.0\times$ on the CPU, and one SVM instance loses at $0.41\times$ on the H100 and wins at $1.22\times$ on the RTX 4090 because the dominant work is single-threaded host code and the H100 node has the weaker single thread.

Table 11 shows that the winning pair of device and precision changes with the accuracy target. At $n = 10{,}000$ and 10^{-4} the consumer RTX 4090 in float32 is the fastest of the four (1.63 s). The H100's float64 advantage over the RTX 4090 is $1.6\times$, far below the ratio of their float64 throughput and below their $3.3\times$ memory-bandwidth ratio. On this cross-validation workload the CPU specialists are faster still, which is the routing decision the system should and does make.

The kernel a device should run is also a per-device decision. Sweeping 21 temporally fused CUDA tiles against XLA's compiled loop on an RTX 4090 and an H100, the best tile differs at five of six problem sizes, the fused kernel is $1.15\times$ to $1.66\times$ faster than XLA on the 4090 and $1.95\times$ to $2.74\times$

device, precision	tolerance 10^{-4}	10^{-6}	10^{-8}
H100, float64	23.01 s	**45.60 s**	**76.88 s**
H100, float32	**9.69 s**	never	never
RTX 4090, float64	35.64 s	71.77 s	121.83 s
RTX 4090, float32	16.52 s	never	never

Table 11: The same batched lasso cross-validation workload ($n = 50{,}000$, 5 folds, 100 regularization values) on two devices and two precisions. float32 stalls at a relative gap of about 2×10^{-5}, so the winning pair changes with the target.

faster on the H100, and carrying one device's choice to the other costs $1.06\times$ to $1.63\times$. A selector that reads the CUDA device properties reproduces each machine's choice from one table with no recompile, and the kernel is bitwise identical to the unfused iteration at relaxation 1 on both devices.

The device is a priced candidate, and the crossover is measurable. A candidate is a triple (formulation, operator, device): the same formulation on this machine's GPU and on its CPU are two candidates, sharing one measured iteration count and priced from each device's own calibration, with a device this process has no backend for declined rather than guessed at. Measuring our method on both devices for all 28 held-out instances, four repetitions, the iteration count is device independent as the cost model assumes (median ratio 1.0000), and the two machines disagree about which device to use: at tolerance 10^{-3} the GPU is faster on 18 instances and the CPU on 2 on the RTX 4090, against 14 and 6 on the H100.

What predicts the crossover is not the instance size. Ranked by $\mathrm{nnz}(A)$, 13 of 27 instances fall on the wrong side of any threshold, because relocation removes rows from the operator the iteration actually applies. Ranked by the work one iteration does, $\mathrm{nnz}(L) + n$, the CPU has the cheaper iteration up to about $14{,}400$ and the GPU from about $22{,}400$, and on the RTX 4090 no instance falls on the wrong side of that band. Below it the GPU is launch bound by up to $600\times$ per iteration; above it the GPU wins by up to $142\times$. Setup does not discriminate between the devices, which is why several small instances tie on total time despite a large per-iteration gap.

With device selection enabled the router picks the faster device on 14 of the 15 decidable instances, and on 9 of 9 when it chose the same formulation and operator the ground truth timed, so only the device differs; its one mistake costs $1.37\times$. The honest accounting is that this buys little on this workload: repeat-visit time improves by 2% while first-visit wall time rises $1.70\times$ from the extra exploration, and 22 of the 28 instances already route to a CPU engine by recognition rather than by the device axis. The cost model is biased low on both devices, predicting $0.75\times$ of measured time on the GPU and $0.63\times$ on the CPU, but the bias is common enough that it still orders the two devices correctly on 15 of 15 instances, which is what the decision requires.

G.7 Generated families: wins, losses, and routing

Table 12 gives cold first-solve comparisons with the fastest accepted engine at shipped defaults in the completed field. Budget-constrained least squares has no packaged specialist; OptEvolve builds a relocated formulation and synthesized projection that beat the available generic engines. The same router misjudges some families: PIQP and celer win on the budget band and group lasso. A large-scale comparison against Clarabel also favors evolved formulations on SVM dual and portfolio instances, but only where Clarabel is the strongest available engine; the qualified comparisons and intervals are in Appendix G.

held-out instance	fastest accepted engine	ours, cold (s)	speedup
budget LS, $n = 12{,}000$	Clarabel	1.04	$4.49\times$
budget LS, $n = 48{,}000$	Clarabel	8.10	$25.22\times$
robust LP, $n = 5{,}000$	Clarabel	1.77	$8.86\times$
budget band, $n = 80{,}000$	PIQP	1.33	$0.24\times$
group lasso, $n = 10{,}000$	celer	1.02	$0.56\times$

Table 12: Chosen routes versus the fastest accepted shipped-default engine at 10^{-4}, RTX 4090 host; cold first solve with no persistent compile cache, five paired repetitions. Speedups are paired geometric means; other tolerances, intervals, and the SVM route at parity are in Table 9, Appendix G. CUDA context creation (about 0.22 s) is excluded.

G.8 Choosing CPU or GPU

The router defers to CPU specialists for sparse lasso and many small LPs, but can choose an evolved GPU iteration on large structured problems. Precision and kernel fusion change device rankings: on the tested lasso cross-validation workload float32 does not reach 10^{-6}. In an earlier six-size OCT sweep on an RTX 4090 and H100, the best temporally fused CUDA tile differs at five sizes and runs $1.15\times$–$2.74\times$ faster than the XLA loop; the separate three-device sweep above compares fusion with an unfused CUDA step. These kernel measurements do not transfer directly to full-solve gains: selecting CPU versus GPU improves repeat-visit time by only 2% and raises first-visit time $1.70\times$ on the held-out field. Device and precision measurements are in Appendix G.