EggMind: LLM-Driven Two-Dimensional Intelligence for Scalable Equality Saturation

Youwei Xiao
shallwe@pku.edu.cn
School of Integrated Circuits
Peking University
Beijing, China

Chenyun Yin
higgs@stu.pku.edu.cn
Peking University
Beijing, China

Yun Liang
ericlyun@pku.edu.cn
School of Integrated Circuits
Peking University
Beijing, China

Abstract

Equality saturation avoids premature commitment to a single, sub-optimal rewrite path in broad compilation problems, yet its practical adoption is limited by uncontrolled e-graph growth due to blind rewriting decisions. We present the EggMind framework to enable intelligent and scalable equality saturation using large language models (LLMs). EggMind introduces a novel *two-dimensional intelligence* approach. First, for the *phase* dimension, EggMind employs LLM-evolved functors to control the equality saturation process and avoid redundancy and suboptimal solutions. Second, for the *case* dimension, EggMind uses inductive learning over solved cases to transfer successful experience to complex, unsolved ones, reducing blind-exploration effort. Combining these two dimensions of intelligence, EggMind provides a feasible method for next-generation optimizers to achieve both scalability and solution superiority in general target domains.

1 Introduction

Equality saturation (EqSat) avoids the early binding decisions made by traditional compilers by exploring a vast space of equivalent programs simultaneously within the e-graph data structure. Supported by high-performance engines such as egg [6] and egglog [14], EqSat has become an increasingly popular paradigm in domains such as vectorization [8, 9], logic synthesis [2, 12], and high-level synthesis [4, 10].

However, in practical deployments, the e-graph scale and algorithmic complexity of the blind EqSat process remain the primary bottlenecks to obtaining optimal solutions within reasonable time and resource constraints. This persists despite recent advancements in e-graph extraction [1, 11], which aim to accelerate the selection of final solutions but cannot mitigate the underlying state-space explosion. Currently, managing such complexity relies on expert-tuned, domain-specific, static strategies [8], lacking the capacity for runtime self-adaptation and generality to new problems.

Adopting LLMs is a natural way to automatically compose effective EqSat strategies for given problems, inspired by the success of LLM-driven compiler optimization [3, 7]. Recently, ASPEN [13] has demonstrated the feasibility of using LLM agents to automate rule proposal and selection for RTL optimization, leveraging real-world PPA feedback to guide the saturation and extraction process. However, these approaches typically operate on a specific domain or rely on design-specific rule discovery, and the key question remains: how to apply LLM techniques to generalized EqSat strategy problems across different phases and problem scales. Naive adoption of LLM agents to make all EqSat decisions consumes many tokens and easily leads to suboptimal solutions and scale explosion.

To address these challenges, we present EggMind, a framework that enables scalable EqSat via two-dimensional intelligence. First, on the phase dimension, EggMind treats EqSat as a runtime control loop, where an LLM evolves compositions of high-level functors (e.g., rule generators and schedulers) based on e-graph signals rather than issuing per-rewrite decisions. Second, on the case dimension, it transfers successful strategy patterns from tractable seed cases to complex instances to bootstrap exploration. We implement this via EqSatL, an inspectable strategy language, and MPActor, a multi-process runtime for orchestrated execution and cross-case knowledge sharing. Ultimately, EggMind provides a scalable, token-efficient paradigm that tames e-graph explosion and transfers optimization expertise across problem scales.

2 Background

2.1 E-Graph and Equality Saturation

An e-graph is a compact data structure that represents large sets of equivalent expressions by grouping them into equivalence classes (e-classes) of alternative e-nodes. Equality saturation (EqSat) starts from an e-graph built from the initial program, and repeatedly applies rewrite rules to add e-nodes and merge e-classes, monotonically growing the e-graph until a fixed point or a resource limit is reached, after which an extraction phase selects the best representative expression given the cost function [6, 14]. By separating exploration from selection, EqSat avoids committing too early, but in practice interacting rules can grow the e-graph rapidly and make saturation expensive, motivating strategy control for scalability [6, 8].

2.2 A Spectrum of Rewriting Techniques

We analyze existing rewriting techniques through a unified lens of how they trade off optimization quality against finite time and resource budgets. We compare traditional

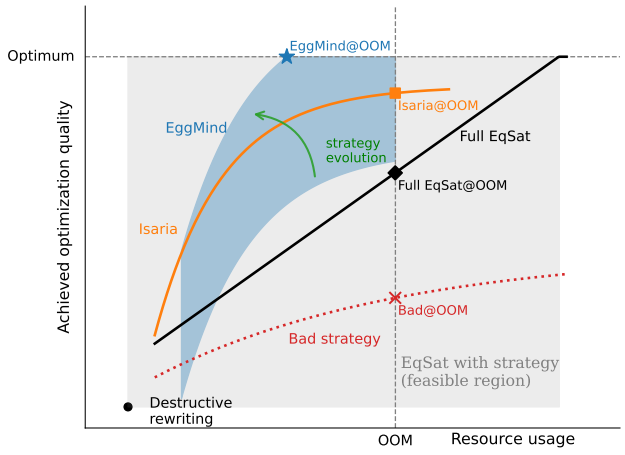

Figure 1. Optimization spectrum for rewriting techniques.

destructive rewriting (e.g., MLIR passes [5]), classic equality saturation, strategy-driven EqSat exemplified by Isaria [8], and EggMind in Figure 1. The x-axis is resource usage (e.g., memory), and the y-axis is achieved optimization quality. The horizontal dashed line denotes the best achievable solution in practice, while finite budgets truncate attainable outcomes, as indicated by the vertical dashed OOM line.

Destructive rewriting appears as a low-resource, low-quality point in the lower-left corner. E-graph–based approaches trace curves because achieved solution quality varies continuously with available budgets. Classic, unguided EqSat can in principle reach the theoretical optimum given sufficient resources, but under realistic constraints it often fails to saturate. Introducing a strategy constrains the rewrite search and restricts attainable outcomes to a feasible region (the shaded rectangle), where solution quality is bounded between the destructive baseline and the theoretical optimum, and resource consumption is bounded between the destructive baseline and full saturation. Different strategies trace different curves within this region: a well-designed, expert-tuned strategy [8] attains substantially higher quality under the same budget (orange curve) but requires domain expertise and transfers poorly across workloads, while a poor strategy may even underperform exhausted EqSat (red curve). EggMind aims to occupy and expand the upper portion of this feasible region by synthesizing and evolving rewriting strategies. Leveraging execution traces and an LLM-based strategist, EggMind dynamically modifies the rewriting strategies both online and across instances. In Figure 1, this behavior is depicted as an evolving blue envelope; the blue star marks EggMind 's operating point under the OOM budget, which approaches expert-level performance while respecting lower time and resource constraints.

Table 1. Functors in an EggMind phase iteration.

#	Functor	Role
1	PREPHASE	Iteration hook: initialize context or state.
2	EVAL	Global termination decision.
3	STEP	Iteration control decision.
4	SCHEDULE	Rewrite rule selection and ordering.
5	UPDATE_RULESET	Active ruleset adaptation.
6	INSPECT	E-graph state query.
7	ANALYZE	Derived metric or property computation.
8	SIMPLIFY	E-graph growth control.
9	SOLVE	Solution extraction decision.
10	POSTPHASE	Iteration hook: finalize records or state.

3 Abstractions for LLM-Based Strategies

EggMind is designed around the principle that effective equality saturation requires explicit, programmable control over rewriting behavior. To this end, EggMind provides two complementary components: runtime-programmable *functors* and an embedded strategy DSL called **EqSatL**. These components decouple optimization strategy from the underlying EqSat engine, transforming EqSat into a manageable control loop for LLMs to evolve.

3.1 Functor Mechanism

EggMind introduces *functors* as the foundational abstraction for runtime control in EqSat-based optimization. A functor encapsulates a side-effect-free decision routine that observes the current e-graph state and emits concrete control actions. By externalizing these decisions, EggMind transforms the standard rewriting loop into an explicit, programmable environment. As summarized in Table 1, EggMind exposes ten specific control points that structure each runtime iteration. These points allow EqSat strategies to react to lightweight runtime signals, such as e-graph growth statistics and cost-improvement trends, enabling adaptive strategy adjustment through LLM guidance at runtime.

3.2 EqSatL DSL

To orchestrate EqSat-based optimization actions into reusable and auditable policies, EggMind introduces **EqSatL**, a lightweight strategy DSL for LLM to compose and evolve. Figure 2 presents EqSatL's declarative three-layer hierarchy.

Layer 1 (Units) defines the fundamental *optimization atoms*. It covers pattern variables ($?x$), term templates (p), and rewrite rules (r) that are grouped into named **rulesets** (rs). These rulesets form the domain-specific vocabulary of optimization knowledge, while extraction metrics (m) guide the final selection of optimal programs from the e-graph.

Layer 2 (Commands) constitutes a *rewrite algebra* that specifies how these units are applied. By using combinators such as Saturate(rs) for fixpoint application, Repeat(n, c) for bounded loops, and Seq or Union for composition, EqSatL treats scheduling as a first-class algebraic object.

Layer 1: Units (Optimization Atoms)

$?x \in$	PatVars	(Pattern variables, Holes)
$p ::=$	term$(?x, \dots)$	(Term templates)
$r ::=$	$p_1 \rightarrow p_2$ $[g]$	(Rewrite rules with optional guard)
$rs ::=$	$\{r_1, r_2, \dots\}$	(Named rulesets)
$m ::=$	cost_fn(e)	(Extraction metrics)

Layer 2: Commands (Rewrite Algebra)

$c ::=$	`Saturate`(rs)	(Apply until fixpoint)
	| `Repeat`(n, c)	(Bounded iteration)
	| `Seq`$(c_1, \dots)$	(Sequential composition)
	| `Union`$(c_1, \dots)$	(Parallel merge)

Layer 3: Plans (Orchestrated Workflows)

$ph ::=$	Phase(id, c, n)	(Named stage with iteration limit)
$pl ::=$	Plan$(ph, \dots)$	(Concatenated phases)
	| Plan$(\dots, H)$	(With init/cleanup hooks)

Figure 2. Formal syntax of EqSatL's three-layered hierarchy.

Layer 3 (Plans) orchestrates these commands into *structured workflows*. A **Phase** (ph) encapsulates a command with a strict iteration limit, and phases are then concatenated into a **Plan** (pl). This layer also provides hooks (H) for initialization and cleanup, ensuring that complex optimization pipelines remain modular and reproducible.

Because strategies are represented as first-class data rather than imperative glue code, EqSatL provides a constrained, schema-like target for LLM-assisted synthesis. In the EggMind framework, the LLM generates compact EqSatL plans that are statically validated for structural well-formedness and resource bounds before execution. By delegating low-level e-graph management to the DSL runtime and limiting the LLM's authority to the policy layers, EqSatL prevents the LLM from making detailed e-graph action decisions and lets it concentrate on strategy evolution.

4 Two-Dimensional Intelligence

EggMind proposes a two-dimensional intelligence approach for leveraging LLMs to achieve both strong scalability and high solution quality. As illustrated in Figure 3, the x- and y-axes denote the *phase* and *case* dimensions. The EggMind runtime starts from a cold-start strategy and refines its logic across tasks and related cases, converging toward a well-optimized strategy.

4.1 Phase Dimension: Intra-Case Adaptation

Phase-dimension intelligence enables *intra-case adaptation*, transforming equality saturation from an open-loop, exhaustive search into a responsive, closed-loop control system. This dimension governs how EggMind dynamically adjusts its behavior *during* a single optimization run, using real-time evidence to refine its strategy. As established in Section 3.1, each control point is represented by a functor, which may be either fixed or LLM-evolved. Phase-dimension intelligence emerges primarily from the latter: LLM-based functors can update their internal policies online, guided by runtime feedback rather than static pre-programming.

A core design principle of EggMind is bounded authority: the LLM is strictly prohibited from modifying the underlying e-graph engine or injecting unverified rewrite rules, operating instead at the strategy layer where it proposes updates to functor configurations, such as ruleset activation or pruning thresholds, exclusively at phase boundaries. As depicted in Figure 3, an **LLM-based strategist** resides within the optimization pipeline of every case, acting as a high-level controller that evolves functor behaviors by consuming lightweight runtime statistics such as e-graph size, cost-reduction trends, and e-graph analysis. By confining the LLM's agency to policy orchestration while delegating execution to the formal DSL runtime, EggMind achieves a robust synergy between expressive, adaptive search control and the rigorous determinism required for backend correctness.

4.2 Case Dimension: Inter-Case Learning

EggMind further introduces a case dimension that enables experience reuse across related EqSat instances. Rather than treating each optimization task as an isolated execution, EggMind models optimization as a population of structurally similar cases and uses case-dimension intelligence in two complementary ways: *Strategy induction* for cold start and *Batch learning and transfer* for runtime. The former provides a strong initial policy by synthesizing reusable EqSatL strategies from tractable seed cases, while the latter continuously refines and specializes those policies through aggregated evidence from larger case batches. This division makes the workflow explicit: initialize strategy quality early, then improve robustness and domain fit through iterative cross-case feedback.

Strategy induction. For the "cold start" of a new domain, EggMind offers two initialization paths: an expert-driven specification leveraging predefined domain templates, and strategy induction from small seed cases using an LLM. While the former relies on manual expertise, the latter exemplifies EggMind's case-dimension intelligence by distilling optimization experiences from tractable instances into reusable strategies. The process starts from baseline EqSat runs on small instances and uses the explanation facilities of the egg backend [6] to obtain structured proof traces, exposing which rule interactions are responsible for high-quality solutions. The LLM then acts as a tool-mediated strategist: it queries rule-space statistics, proposes candidate rulesets, and constructs the structural skeleton of phase strategies. EggMind uses a hybrid synthesis workflow that separates structural design from numerical tuning: the LLM determines policy structure, including phase order, ruleset predicates, and loop shape, while an autotuning loop searches iteration limits and loop counts. In this loop, the `Evaluation` tool is

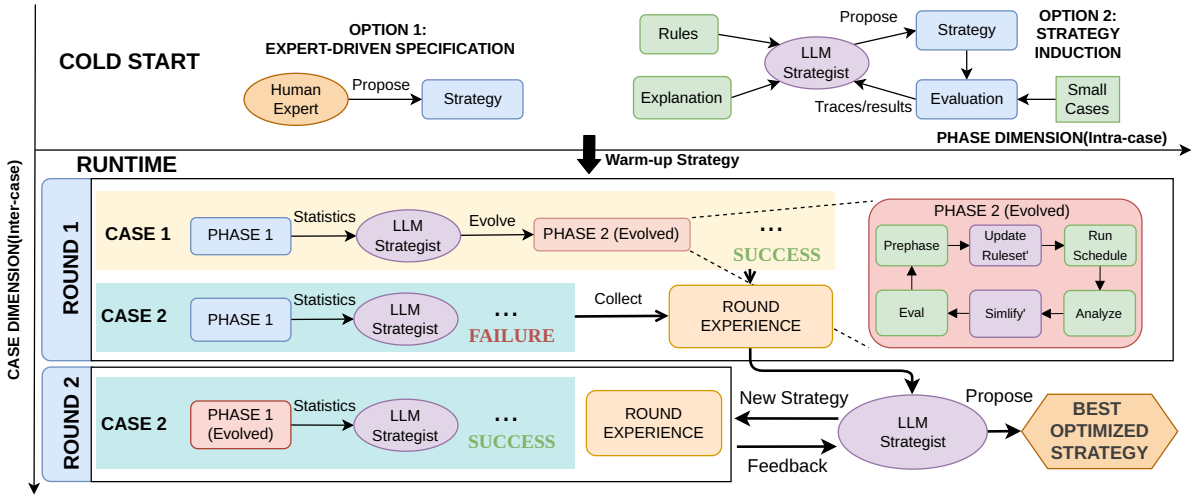

Figure 3. Illustration of *Two-Dimensional Intelligence.*

applied to the small seed cases to filter candidate strategies before transfer: each candidate must satisfy a hard cost constraint (final cost no worse than the baseline) and is further ranked by runtime and memory. The selected policy, which performs best, is then emitted in EqSatL, yielding a typed and executable warm-up strategy artifact that can be directly reused on larger cases.

Batch learning and transfer. Beyond cold start, `EggMind` performs iterative refinement through batch execution over related instances, using a shared container called "Round Experience" that records per-case outcomes, schedule choices, and resource-quality tradeoffs. As results accumulate, the system retrieves similar solved cases using lightweight case features and transfers strategy-level decisions (e.g., phase structure, ruleset selection bias, and pruning aggressiveness) to ongoing or failed runs. Failed cases can then be retried with learned policy variants and adjusted limits, while consistently ineffective policies are pruned from the candidate pool. This workflow specializes strategies to a target domain without manual retuning and keeps transfer lightweight because it reuses control policies rather than e-graph states. In effect, case-dimension learning turns isolated EqSat runs into a feedback process that compounds experience across the batch.

5 Implementation

We implement `EggMind` as a lightweight orchestration layer on top of `egglog` [14], which serves as the rewriting engine and focuses on providing the minimal runtime mechanisms required to support adaptive strategies and inter-case learning.

5.1 MPActor Runtime

`MPActor` is the runtime substrate that supports the exploratory and adaptive execution model required by `EggMind` without modifying the core EqSat engine. It treats each EqSat run as an actor with an explicit lifecycle and encapsulated state, allowing multiple executions to be managed, paused, and resumed independently. To enable safe experimentation with strategies, MPActor provides snapshot and rollback functionality over EqSat state. This allows the system to evaluate alternative functor configurations or phase transitions without committing to irreversible rewrite sequences. In addition, MPActor supports fork and join semantics, enabling multiple strategies or batched cases to be explored in parallel.

5.2 Prototype Status and Limitations

The current `EggMind` prototype implements programmable functors, the EqSatL strategy DSL, and the MPActor runtime, and supports inductive cold start from small instances as well as batch execution on related cases. LLM interaction is currently external to the runtime, synthesizing EqSatL strategies from extracted explanations of EqSat states. Strategy validation relies on execution-based feedback rather than formal guarantees, and transfer quality depends on the representativeness of seed cases. Addressing these limitations and conducting a comprehensive empirical evaluation is ongoing work.

6 Conclusion

This paper presents `EggMind`, a work-in-progress framework that introduces two-dimensional intelligence for equality saturation through adaptive runtime control and cross-case learning. By decoupling scheduling strategies from the EqSat engine and treating optimization as a reusable process rather than an isolated run, `EggMind` outlines a path toward more robust and scalable EqSat workflows.

References

[1] Yaohui Cai, Kaixin Yang, Chenhui Deng, Cunxi Yu, and Zhiru Zhang. 2025. Smoothe: Differentiable e-graph extraction. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1020–1034.

[2] Chen Chen, Guangyu Hu, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2025. E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11133110

[3] Hongzheng Chen, Alexander Novikov, Ngân Vũ, Hanna Alam, Zhiru Zhang, Aiden Grossman, Mircea Trofin, and Amir Yazdanbakhsh. 2026. Magellan: Autonomous Discovery of Novel Compiler Optimization Heuristics with AlphaEvolve. arXiv:2601.21096 [cs.AI] https://arxiv.org/abs/2601.21096

[4] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. 2024. SEER: Super-Optimization Explorer for High-Level Synthesis using E-graph Rewriting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1029–1044. https://doi.org/10.1145/3620665.3640392

[5] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[6] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (Jan. 2021), 1–29. https://doi.org/10.1145/3434304

[7] 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. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv:2506.13131 [cs.AI] https://arxiv.org/abs/2506.13131

[8] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '24, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 19–34. https://doi.org/10.1145/3617232.3624873

[9] Alexa VanHattum, Rachit Nigam, Vincent T. Lee, James Bornholt, and Adrian Sampson. 2021. Vectorization for digital signal processors via equality saturation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 874–886. https://doi.org/10.1145/3445814.3446707

[10] Youwei Xiao, Yuyang Zou, and Yun Liang. 2025. SkyEgg: Joint Implementation Selection and Scheduling for Hardware Synthesis using E-graphs. arXiv:2511.15323 [cs.PL] https://arxiv.org/abs/2511.15323

[11] Jiaqi Yin, Zhan Song, Chen Chen, Yaohui Cai, Zhiru Zhang, and Cunxi Yu. 2025. e-boost: Boosted E-Graph Extraction with Adaptive Heuristics and Exact Solving. *arXiv preprint arXiv:2508.13020* (2025).

[12] Jiaqi Yin, Zhan Song, Chen Chen, Qihao Hu, and Cunxi Yu. 2025. BoolE: Exact Symbolic Reasoning via Boolean Equality Saturation. *arXiv preprint arXiv:2504.05577* (2025).

[13] Niansong Zhang, Chenhui Deng, Johannes Maximilian Kuehn, Chia-Tung Ho, Cunxi Yu, Zhiru Zhang, and Haoxing Ren. 2025. ASPEN: LLM-Guided E-Graph Rewriting for RTL Datapath Optimization. *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)* (2025), 1–9. https://api.semanticscholar.org/CorpusID:281043445

[14] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. http://arxiv.org/abs/2304.04332 arXiv:2304.04332 [cs].