LymphoSAT in the SAT Competition 2026

Harrison Green
Carnegie Mellon University
harrisog@andrew.cmu.edu

***Abstract*—LYMPHOSAT[1] is a modular SAT solver consisting of 126 targeted solvers and a fallback derived from KISSAT-PUBLIC 4.0.2, modified to emit VeriPB certificates. We submit both the original LYMPHOSAT solver and a verified version, LYMPHOSAT-VERIFIED, which performs additional model/proof checking after the solver terminates.**

I. OVERVIEW

The great appeal of SAT solvers is that any problem in NP can be efficiently encoded and handed to a *generic* SAT solver, which applies a multitude of tricks and techniques to find a solution more quickly than one might think given the (widely believed) exponential worst-case nature of Boolean satisfiability. Indeed, the role of the SAT Competition itself is to continue to push this frontier further through new techniques, strategies, and heuristics.

The secret is that real-world problems we care about solving (so-called "industrial" problems) contain a significant amount of structure, often permitting a more efficient solution than purely random formulas of the same size. A classic example is the *pigeonhole problem*, which yields exponential-size proofs for traditional resolution-based solvers, yet permits polynomial-time solutions using more specialized techniques [1]. The key challenge is how to identify and exploit these sorts of structures in a general way.

Our observation is that the demand for solvers to be *general* imposes a necessary trade-off: specializing a solver to be fast on pigeonhole problems incurs an overhead on other types of problems where that specialization is wasted. Thus, mainline SAT solvers implement only broadly applicable techniques, leaving a lot of performance on the table for specialized cases.

Recent advancements in large language models (LLMs) have changed the economics of software development, making it feasible to synthesize many targeted solutions (which can take advantage of specialized structure) rather than pooling efforts to produce a single general solution (which necessarily sacrifices performance on specialized cases). For the SAT Competition 2026, we directly exploit this gap by constructing a family of LLM-generated specialized solvers, each of which is optimized to handle a specific family of problems. For the competition, where a solver must handle a wide variety of problems, we merge these solvers into a single modular solver: a classification stage dispatches the problem to the appropriate specialized solver, and a fallback solver (KISSAT-PUBLIC [2]) handles the remaining unmatched problems.

[1]The name derives from *lymphocyte*, the cells that provide us with adaptive immunity, in much the same way our targeted solvers provide us with adaptive, specialized solutions.

II. ARCHITECTURE

Given an input formula (in DIMACS format), LYMPHOSAT first profiles the formula, computing a cheap, structural feature vector consisting of 95 numeric features. This profile summarizes coarse properties of the CNF instance: formula size, clause-length distribution, literal polarity balance, unit/binary/ternary clause prevalence, Horn-like structure, etc. The feature vector is intentionally inexpensive to compute: it is extracted by a single pass over the formula, with bounded sketches for potentially large structures such as binary-clause pairs.

The resulting profile is then passed to a registry of solver matchers. Each matcher is a small predicate over the feature vector that recognizes formulas likely to be well handled by a particular backend solver. For example, a matcher may identify formulas dominated by binary clauses, formulas with strong Horn structure, formulas with positive-block patterns suggestive of exactly-one constraints, or formulas whose size and density make them suitable for an integer-programming backend. Matchers are evaluated in order, and the first successful match selects the corresponding solver.

After selecting a backend, LYMPHOSAT invokes the chosen solver on the original DIMACS input. If the targeted solver returns UNKNOWN, or the formula does not match any matcher, the original formula is passed to the fallback solver, KISSAT-PUBLIC, which has been modified to emit VeriPB certificates.

III. SOLVER DEVELOPMENT

Both *matchers* and *solvers* are synthesized entirely from scratch by a coding agent and validated on withheld validation sets. We use formulas sourced from the Global Benchmark Database [3], which contains problems from a wide range of domains (and is also the source of historical SAT Competition problems).

For each family in the benchmark database, we randomly select 80% of formulas as the *training set* and 20% as the *validation set*. We provide the training set to a coding agent, which is tasked with implementing a specialized solver (from scratch) for the entire family. The agent runs in a sandboxed environment and submits solvers for external evaluation through a provided MCP server. External evaluations run on the combined training and validation sets and are submitted to Google Cloud Batch.[2] Each instance runs with 1 vCPU,

[2]We use `n2-highmem-2` instances with Intel Ice Lake CPUs, mimicking the competition infrastructure for 2026.

8 GB of memory, and a 600-second timeout. Only training-set performance is fed back to the agent for subsequent iterations, while validation-set performance is used to select the final solver. To avoid cheating (by reading comments or filenames), we also strip formula comments and rename files to `formula.cnf` during external evaluation. Solutions are validated by checking the model or validating the proof with VERIPB.

To create the final composition, we benchmark the fallback solver with the same evaluation environment and select the best per-family solver only if it does not produce any invalid answers and reduces the PAR-2 score relative to the fallback on the validation set; this was the case for 125 of the 189 families.[3]

We experimented with both Claude Opus 4.6 (in Claude Code) and GPT 5.5 (in OpenAI Codex) as our coding agents. Solver development demanded a level of model capability that was only just attainable by recent frontier models; most attempts to use cheaper or open-source models were unsuccessful. Matcher development, on the other hand, was much simpler, and we used GPT 5.4-mini (in Codex) for the vast majority, and GPT 5.5 for remaining cases that demanded a more complex decision boundary.

Total expenditure for the entire experiment was approximately $10,000 USD in LLM credit usage across all providers (at API prices[4]), and approximately $5,000 USD in cloud compute usage.

For both submissions, the only solver base code is the fallback, which is derived from KISSAT-PUBLIC 4.0.2 and modified (with GPT 5.5 in Codex) to emit VeriPB certificates. The 126 specialized solvers, feature matchers, and associated heuristics were produced by coding agents from scratch. Human work consisted of designing the orchestration and architecture (infrastructure implementation was entirely delegated to coding agents) and choosing which agents to run on which families[5] (and when to reattempt if synthesis failed or produced low performance). LYMPHOSAT-VERIFIED uses the same solver composition as LYMPHOSAT and adds a post-processing wrapper for additional model/proof checking.

IV. FINAL COMPOSITION

The final composition consists of an assortment of widely varied strategies and solutions. In many cases, the coding agents discovered highly efficient solutions for specific families: 43 of the solvers reduced the PAR-2 score by more than 90% relative to the fallback solver over a 600-second timeout. Some examples of specific strategies are provided below. Note that these summaries were authored by the coding agents themselves during the development process:

[3]We include one additional specialized solver for a family (GF(3) equations) we submitted to the competition that is not yet present in the benchmark database.

[4]Although many runs were subsidized through LLM subscription plans.

[5]This oversight was largely an issue of budget management; running the best agent multiple times for every family would produce very good results.

Pigeonhole. This solver detects standard PHP, functional PHP (fPHP), ternary PHP (tPHP), and relativized PHP (rPHP) encodings in DIMACS CNF, including shuffled and polarity-flipped variants. For standard/functional PHP, it recovers the bipartite structure using clause length analysis, Union-Find hole grouping, and pairwise AMO detection. For ternary PHP (tPHP with $M = 2N + 1$ pigeons, ternary AMO clauses), it identifies the structure from clause lengths and ternary co-occurrence groups. For rPHP, it detects the variable layout (assignment, selector, y-color variables) from binary implications and pigeon AMO, identifies s-vars via greedy max-clique on co-occurrence graph, and extracts conditional AMO from either 4-literal clauses or ternary auxiliary variable pairs. SAT instances are solved by constructing the canonical assignment for the recovered identity mapping. UNSAT standard PHP uses cutting-planes proofs with inductive AMO derivation. UNSAT rPHP uses extension variables $z = s \wedge y$ via VeriPB red rule, derives pairwise then global AMO per color, and uses a counting argument for contradiction.

Fermat. This solver specializes to the observed PEQNP Fermat factorization encoding. It detects the two little-endian input blocks from the long negative clauses, recovers the hidden odd product constant from the final comparator gadget without using comments or filenames, factors the recovered 64-bit integer with an in-house Miller-Rabin and Pollard Rho implementation, maps a factor pair back to Fermat parameters $a = (u + v)/2$ and $b = (v - u)/2$, then unit-propagates the Tseitin circuit. Before returning SAT, it independently checks every original CNF clause under the complete assignment. Unsupported layouts, constants wider than 63 bits, unfactored values, or any validation failure return UNKNOWN.

XOR-chain. This solver specializes to the xor-chain family. It ignores comments and recovers the formula as complete four-clause 3-XOR gates plus any number of complete two-clause 2-XOR equations. It runs sparse Gaussian elimination over the recovered GF(2) system. If the system is consistent, it emits a complete CNF-checked SAT model. If elimination finds a conflicting dependency, it introduces proof-only full-adder variables for each selected 3-XOR, proves local parity bits with constant-size VeriPB derivations, uses original clauses for selected 2-XORs, sums the selected equations, and derives a direct aggregate VeriPB contradiction by division.

REFERENCES

[1] I. Grosof, N. Zhang, and M. J. Heule, "Towards the shortest drat proof of the pigeonhole principle," *arXiv preprint arXiv:2207.11284*, 2022.

[2] A. Biere, T. Faller, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2025," in *Proceedings of SAT Competition 2025: Solver and Benchmark Descriptions*, ser. Proceedings of SAT Competitions, C. Codel, K. Fazekas, M. J. H. Heule, and M. Iser, Eds., Vienna, 2025, p. 10. [Online]. Available: https://doi.org/10.34726/10379

[3] A. Iser and C. Jabs, "Global Benchmark Database," in *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Chakraborty and J.-H. R. Jiang, Eds., vol. 305. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, pp. 18:1–18:10. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2024.18