SAT Solver Mastery Roadmap

This document outlines a complete, structured guide to mastering SAT solvers. It is designed for someone already proficient in programming who wants to deeply understand the theory, implementation, and research around SAT solving, including DPLL, CDCL, and modern solvers like MiniSAT, Glucose, and NeuroSAT.

# Stage 0: Foundations Review (Optional)

✓ Review Boolean logic (CNF, literals, clauses)  
✓ Review SAT, 3-SAT, NP-completeness  
✓ Understand why SAT is fundamental in computer science  
✓ Optional: Skim Discrete Math refresher (e.g., MIT 6.042J)

# Stage 1: Core Theory — DPLL and CDCL

Topics to Study:  
- DPLL algorithm and backtracking  
- CDCL enhancements: clause learning, conflict analysis  
- Non-chronological backtracking  
- Watched literals, VSIDS, LBD  
- Restart strategies  
  
Resources:  
- Handbook of Satisfiability (Ch. 1–4)  
- SAT Solving for Fun (msoos.org)  
- Marijn Heule’s SAT Solver Lectures  
- Pipatsrisawat's SAT solving notes

# Stage 2: Implement DPLL and CDCL

Implement in Python or C++:  
- Basic DPLL with unit propagation and backtracking  
- CDCL with conflict analysis and clause learning (1st-UIP)  
- Implement restarts and VSIDS  
  
Resources:  
- GitHub: QuMuLab’s python-sat-solver  
- GitHub: benedekrozemberczki/CDCL-SAT-Solver  
- SATLIB benchmark problems  
- DIMACS CNF format

# Stage 3: Study Real Solvers

Solvers to Explore:  
- MiniSAT: Minimal CDCL solver — https://github.com/niklasso/minisat  
- Glucose: LBD clause learning — https://www.labri.fr/perso/lsimon/glucose/  
- CaDiCaL: Modern, clean C++ — https://github.com/arminbiere/cadical  
- Kissat: Highly optimized — https://github.com/arminbiere/kissat  
- MapleSAT: AI-inspired heuristics — GitHub (Maple branch)  
- CryptoMiniSat: XOR-aware — https://github.com/msoos/cryptominisat  
- WalkSAT: Local search (not complete)  
- Z3: SMT solver from Microsoft — https://github.com/Z3Prover/z3  
  
For each:  
- Compile and run on sample CNF  
- Analyze decision heuristics, propagation, clause learning  
- Compare architecture and performance

# Stage 4: Benchmark and Compare Solvers

Tasks:  
- Collect benchmark CNFs (SATLIB, DIMACS)  
- Create test suite: Easy, Medium, Hard 3-SAT  
- Run solvers and compare:  
 - Time  
 - Memory  
 - Number of decisions  
 - Learned clauses  
- Visualize results (e.g., Matplotlib)

# Stage 5: AI + SAT Research

Topics:  
- MapleSAT’s LRB heuristic  
- NeuroSAT: GNN-based solver — https://github.com/dselsam/neurosat  
- Hybrid solvers: combining ML with CDCL  
- Reinforcement Learning for branching  
  
Resources:  
- NeuroSAT paper: https://arxiv.org/abs/1802.03685  
- DeepSAT, L2SAT papers  
- Z3Py for Python-based SMT

# Stage 6: Thesis & Research Ideas

Ideas:  
- Use Genetic Algorithms to evolve solver parameters  
- Combine NeuroSAT predictions with CDCL heuristics  
- Visual SAT solver with live clause learning explanations  
- Hybrid RL + SAT for decision heuristics  
  
Next steps:  
- List possible thesis questions  
- Narrow to 1–2 experimental directions  
- Start prototyping

# Toolbox and Resources

Quick Links:  
- SATLIB benchmarks: http://www.cs.ubc.ca/~hoos/SATLIB/  
- DIMACS CNF format: https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/DIMACS/SATformat.ps  
- pySAT (Python interface): https://pysathq.github.io/  
- SAT Competitions: https://satcompetition.github.io/  
- Z3Py tutorial: <https://ericpony.github.io/z3py-tutorial/guide-examples.htm>  
  
  
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.