**PHASE 1: Formalization of Physical Abstraction**

Goal: Construct a unified symbolic-topological language to describe and reason about physical systems.

**1.1 DSL Design for Physical Abstractions**

Create a DSL syntax (AST or Prolog-like) to describe particles, fields, PDEs, constraints, and laws.

Support for type-checking units (e.g., dimensions of force, entropy).

Build DSL compiler into an intermediate representation (IR) using category theory.

**1.2 Sheaf-Based Structural Encoding**

Encode local laws (PDEs, symmetries, invariants) as sections of sheaves over simplicial complexes.

Design cosheaf duals to represent signal flow, constraint propagation, or causality.

Integrate dynamic updating of sheaf data over evolving spatial-temporal meshes.

**1.3 Graph and Simplicial Complex Backbone**

Construct a dynamic topological space (simplicial complex + time evolution).

Represent particles and fields as nodes/faces; interactions as morphisms.

Embed thermodynamic paths and energy flows in complex structure.

**1.4 Thermodynamic, Quantum, and Causal Logic Layers**

Define a logic of entropy: build a symbolic Lagrangian-Hamiltonian engine with entropy constraints.

Incorporate quantum logic using noncommutative geometric operators and algebra bundles.

Embed a causal structure using directed sheaf morphisms and spacetime constraints.

**1.5 Prototype Engine for Discrete Physical Systems**

Build test DSL-to-simulation compiler for symbolic mass-spring systems.

Validate symbolic PDE reductions, unit checking, and dynamic graph updates.

Benchmark representations against standard physics engines (e.g., MuJoCo, Bullet).

**PHASE 2: Symbolic Physical Representation & PDE Engine**

Goal: Model system dynamics with symbolic, exact, and general PDE representations.

**2.1 Symbolic Tensor PDE Engine**

Implement symbolic solvers for field equations over simplicial domains (e.g., Maxwell, Navier-Stokes).

Extend SymPy with graph tensor calculus: discrete divergence, curl, Laplacian on simplicial graphs.

Support algebraic PDEs from thermodynamics and quantum systems.

**2.2 Invariant and Conservation Law Discovery**

Apply symbolic regression (e.g., PySR) to derive invariants (energy, charge, momentum).

Match discovered invariants to known symmetries via Noether's theorem.

Extend to hybrid symbolic-numeric discovery of unknown conservation laws.

**2.3 Quantum-Classical Hybrid Representations**

Use noncommutative geometry to encode quantum operators as symbolic algebra elements.

Construct semiclassical approximations (WKB, path integrals) from DSL input.

Represent quantum subsystems within classical sheaf logic (fibered Hilbert spaces).

**2.4 Symbolic Thermodynamic PDE Encoding**

Model systems via entropy production equations, Onsager relations, and transport laws.

Add support for symbolic Legendre transforms, Gibbs functions, and internal entropy constraints.

Generate thermodynamically consistent PDEs for diffusion, heat flow, chemical dynamics.

**2.5 Benchmark Systems and PDE Experiments**

Test symbolic PDE representation on canonical systems: harmonic oscillator, double pendulum, Maxwell equations, Ising model.

Compare symbolic/numeric accuracy and generality.

Build symbolic PDE datasets for learning and verification.

**PHASE 3: Symbolic Theorem Proving and Neural-Symbolic AI**

Goal: Link symbolic models to proof systems and machine learning for verifiable inference and model discovery.

**3.1 Transformer-Based Symbolic Inference**

Train physics-aware transformers on DSL + PDE data to simplify, solve, and generate equations.

Incorporate unit-aware embeddings and algebraic invariants into model architecture.

Use attention to highlight terms that match conserved quantities or symmetries.

**3.2 Formal Theorem Proving Integration**

Link symbolic outputs to Lean or Coq theorem provers.

Formally prove conservation laws, solution bounds, and stability conditions.

Automatically generate proof trees for discovered equations.

**3.3 Differentiable Theorem Proving**

Implement gradient-based neural proof guidance for proof completion and conjecture validation.

Use reinforcement learning + logic synthesis for strategy discovery.

Train models on formalized physics axioms, DSL-encoded laws, and derived properties.

**3.4 DSL-to-Proof Verification Pipeline**

Convert DSL statements into logical forms compatible with theorem provers.

Encode field constraints, symmetry assumptions, and boundary conditions into proof language.

Validate each stage of symbolic physics with formal guarantees.

**3.5 Interactive Symbolic Proof Environment**

Build a Jupyter-style interface for entering symbolic physics queries and returning proofs.

Display interactive proof graphs and traceable derivation paths.

Support user-submitted laws, simulations, or conjectures.

PHASE 4: Meta-Learned Control & Game-Theoretic Dynamics

Goal: Synthesize symbolic control policies for general physical systems.

**4.1 Meta-Learning Symbolic Controllers**

Train meta-learners on families of PDEs + constraints to generalize symbolic policies.

Represent control laws as symbolic expressions with parameters tuned by learning.

Use few-shot adaptation for novel systems.

**4.2 Game-Theoretic Control over Physical Graphs**

Model multi-agent interactions using Nash equilibria over physical interaction graphs.

Introduce symbolic game-solving for thermal/economic agents (e.g., energy games).

Implement cooperative/competitive agent strategies from DSL.

**4.3 Thermodynamic Feasibility Constraints**

Ensure all learned policies minimize free energy or obey entropy bounds.

Use symbolic thermodynamic identities as hard constraints during training.

Prove optimality or feasibility of symbolic control laws.

**4.4 PDE-Constrained Reinforcement Learning**

Integrate symbolic PDE constraints into RL environments.

Use symbolic simulators as differentiable environments for physics-aware RL.

Combine RL with theorem-proved constraints and DSL-coded laws.

**4.5 Policy Verification & Explainability**

Translate learned control policy back into DSL + theorem prover format.

Verify stability, convergence, or optimality formally.

Provide symbolic explanations for why a policy works.

**PHASE 5: Simulation, Visualization & Deployment**

Goal: Build an interactive simulation engine and web interface for symbolic physical modeling.

**5.1 Symbolic Simulation Engine**

Develop real-time simulator from DSL + symbolic PDEs.

Include both symbolic evaluation and numerical fallback.

Support interactive field visualization and time evolution.

**5.2 Sheaf and Graph Visualization**

Display evolving sheaf structures, simplicial topologies, causal links.

Use Blender/WebGL to visualize topological features.

Visualize information flow, constraints, and symbolic equation updates.

**5.3 Live DSL Editor + Auto-Prover Interface**

Online interface for editing DSL and compiling to proofs/simulations.

Auto-complete symbolic forms and laws via transformers.

Interface with Coq/Lean proof checker in-browser.

**5.4 Physics Sandbox**

Interactive public physics playground (like PhET, but symbolic).

Export results to LaTeX, Jupyter, or formal theorem repositories.

**5.5 Open Source Publication**

Host full engine and interface on GitHub with MIT license.

Create project site with runnable examples, interactive visualizations, and documentation.