# MacroDR: A White Paper on Proof-Oriented Inference Engines for Scientific Modeling

## Abstract

MacroDR is a next-generation scientific inference engine for biophysical modeling, built on the principle that **software is a mathematical instrument**. It unifies rigorous proof-oriented design, domain-specific language (DSL) semantics, modular architecture, and provenance-aware reproducibility into a system capable of both immediate research applications and long-term theoretical contributions. This white paper outlines the **vision**, the **conceptual foundations**, the **technical realization**, and the **research roadmap** for MacroDR.

## 1. Introduction

The central motivation behind MacroDR is to bridge the gap between **scientific aspiration** and **software engineering discipline**. Traditional scientific software often accumulates technical debt, ad hoc algorithms, and opaque data handling. MacroDR instead aspires to be a *living demonstration* of **Proof-Oriented Design (POD)**, where each program element embodies a guarantee, and the system’s coherence is its meaning【62†source】.

The research context is ion channel kinetics and Bayesian model evidence. However, MacroDR is designed generically: any stochastic model inference task can benefit from its DSL, safety kernel, and modular structure.

## 2. Vision

MacroDR’s vision can be distilled into four guiding principles:

1. **Correctness by Construction**  
   Every construct encodes its invariants. A Q-matrix is not just a matrix; it is guaranteed nonnegative off-diagonals and row sums to zero【61†source】.
2. **Transparency and Reproducibility**  
   Provenance is tracked as a first-class property. Every simulation, parameter set, or evidence run can be traced to its origin, seed, or measurement【67†source】.
3. **Extendability without Fragility**  
   New commands, models, or algorithms can be added without touching central registries. Invariance is enforced locally, contagiously, without brittle boilerplate【90†source】【91†source】.
4. **Dual-Track Development**  
   Immediate cluster jobs continue to run with the legacy system, while the refactored MacroDR evolves with clean modularity and CI-driven guarantees【66†source】.

## 3. Conceptual Foundations

### 3.1 Proof-Oriented and Contagiously Safe Design

* **Types as Proofs**: Constructing an object is equivalent to proving its invariants. A ScientificSafe<Q> is a proof that Q satisfies stochastic properties【61†source】.
* **Contagion Principle**: Once established, invariants propagate downstream unless explicitly escaped【62†source】.
* **Composite Domains**: To avoid exponential growth of wrapper types, multiple invariants are bundled into composite safety domains【61†source】.

### 3.2 HoTTification

Inspired by Homotopy Type Theory (HoTT), MacroDR treats **types as propositions**. For example:

Current(Instrument=i, Time=t, Channel=c) : Type  
4pA : Current(i=Rigol123, t=2025-09-20T19:45, c=3)

Here the *proposition* is not just “4 picoamperes” but “4 picoamperes measured on channel 3, instrument Rigol123, at time t0, with calibration Y.” Provenance becomes part of the type【67†source】.

### 3.3 Algebraic CLI

Commands are not arbitrary; they form algebraic structures. Running simulate followed by infer and verifying with sav-check corresponds to an approximate identity:

infer(simulate(θ)) ≈ θ

This elevates the CLI into a mathematical contract【62†source】.

### 3.4 Lingua Franca and Boundary Contracts

* **Canonical Data Model**: JSON-like value tree at boundaries, enriched by schemas【112†source】.
* **Contracts**: pre/postconditions checked at CMD surfaces; violations return Maybe\_error instead of propagating silently【111†source】.
* **Vector View**: optional numeric projection ensures hot loops remain efficient【112†source】.

### 3.5 Provenance as First-Class

* **Writer Monad Semantics**: functions remain pure, provenance accumulates externally【67†source】.
* **Compression Strategies**: retain provenance at the level of human decisions (model choice, parameters, seeds) rather than raw multiplications【67†source】.
* **Experiment Layers**: Protocol + Recording = Experiment; transformations yield TExperiment; provenance-preserving views (TExperimentView) allow multiple post-processed perspectives【89†source】.

## 4. Technical Realization

### 4.1 Architecture & Modules

* **DSL**: Parser, AST, registry; no domain logic【109†source】.
* **CLI**: Thin entrypoint; orchestrates script ingestion【69†source】.
* **CMD**: Command surface with arguments, pre/postconditions, orchestration【111†source】.
* **Core**: Implementations behind CMD; orchestrates Models, Inference, IO, Math【112†source】.
* **Domain Entities**: Canonical types (Experiment, Recording, Parameters)【89†source】.
* **IO**: JSON/CSV serialization, schema validation【112†source】.
* **Models/Inference/Probability/Math**: Deterministic forward simulation, Bayesian likelihood, MCMC, linear algebra, distributions【112†source】.
* **Utils**: Error handling (Maybe\_error), metaprogramming, memoization【112†source】.

### 4.2 Runtime & Type System

* **Grammar**: assignments, function calls, literals【92†source】.
* **Type Deduction**: compile\_expression maps untyped to typed (double, string, identifiers, function calls)【110†source】.
* **Runtime Rules**: Execution via typed\_expression::run, assignment binding in Environment【93†source】.
* **Dispatch**: dynamic\_cast for AST downcasts, postponed visitor pattern【90†source】.
* **Ownership**: unique\_ptr for IR nodes, NodePtr borrow deferred【91†source】.

### 4.3 CLI & DSL Functions

* **CLI Verbs**: run, eval, compile, describe, help【88†source】.
* **DSL Functions**: simulation, likelihood, thermo evidence, experiment loaders, prior construction, utility helpers【94†source】【108†source】.

### 4.4 Testing & CI

* **Unit Tests**: param→Q→param round-trips, expm vs spectral, likelihood finite【93†source】.
* **Property Tests**: score = 0 in expectation; Cov(score) = FIM【66†source】.
* **Regression**: DSL regression tests, CLI smoke tests【68†source】.
* **CI Integration**: build times, template instantiations tracked【68†source】.

### 4.5 Documentation & ADRs

* **spec/**: CLI syntax, DSL grammar, type rules, runtime rules【87†source】.
* **ADR-001**: modules & dependencies【111†source】.
* **ADR-002**: lingua franca【112†source】.
* **Dispatch decision**: dynamic\_cast【90†source】.
* **Ownership decision**: unique\_ptr【91†source】.
* **Roadmap**: M0–M5 phases【113†source】.

## 5. Research Context and Papers

* **Rotational kinetic models** (with Gustavo): extend published work using rotation-coupling【66†source】.
* **Characterization of MacroDR**: present validation (Simulation, Likelihood, Sampling, Evidence, FIM, Score)【66†source】.
* **Cumulative evidence**: theoretical and empirical framework for combining datasets【66†source】.
* **Oocytes + mutants**: macroscopic evidence modeling; mutant perturbations【66†source】.
* **Future**: receptor CIS-LU collaborations, single-channel idealization【66†source】.

## 6. Roadmap

* **M1 Registry Consolidation**: commands extracted from CLI into CMD【113†source】.
* **M2 Core Surfaces**: simulate and load\_experiment stabilized【113†source】.
* **M3 Lingua Franca Pilot**: Recording + Experiment JSON schemas【113†source】.
* **M4 CMD Migration**: likelihood, thermo evidence moved from legacy【113†source】.
* **M5 Performance**: vectorization, CMake target separation【113†source】.

## 7. Conclusion

MacroDR is more than a codebase: it is a **proof-oriented research program**. It embodies the principle that scientific computation must be: - **Safe** (invariants guaranteed), - **Transparent** (provenance explicit), - **Reproducible** (environments persistable), - **Evolvable** (modular architecture), - **Mathematically principled** (HoTTification, algebraic CLI).

The union of audios and documents reveals a coherent trajectory: from philosophical reflection to engineering scaffolding. The system is already serving immediate scientific goals while charting a course toward a general theory of proof-oriented design in computational science.